ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems

Peter Kogge, Editor & Study Lead
Keren Bergman
Shekhar Borkar
Dan Campbell
William Carlson
William Dally
Monty Denneau
Paul Franzon
William Harrod
Kerry Hill
Jon Hiller
Sherman Karp
Stephen Keckler
Dean Klein
Robert Lucas
Mark Richards
Al Scarpelli
Steven Scott
Allan Snavely
Thomas Sterling
R. Stanley Williams
Katherine Yelick

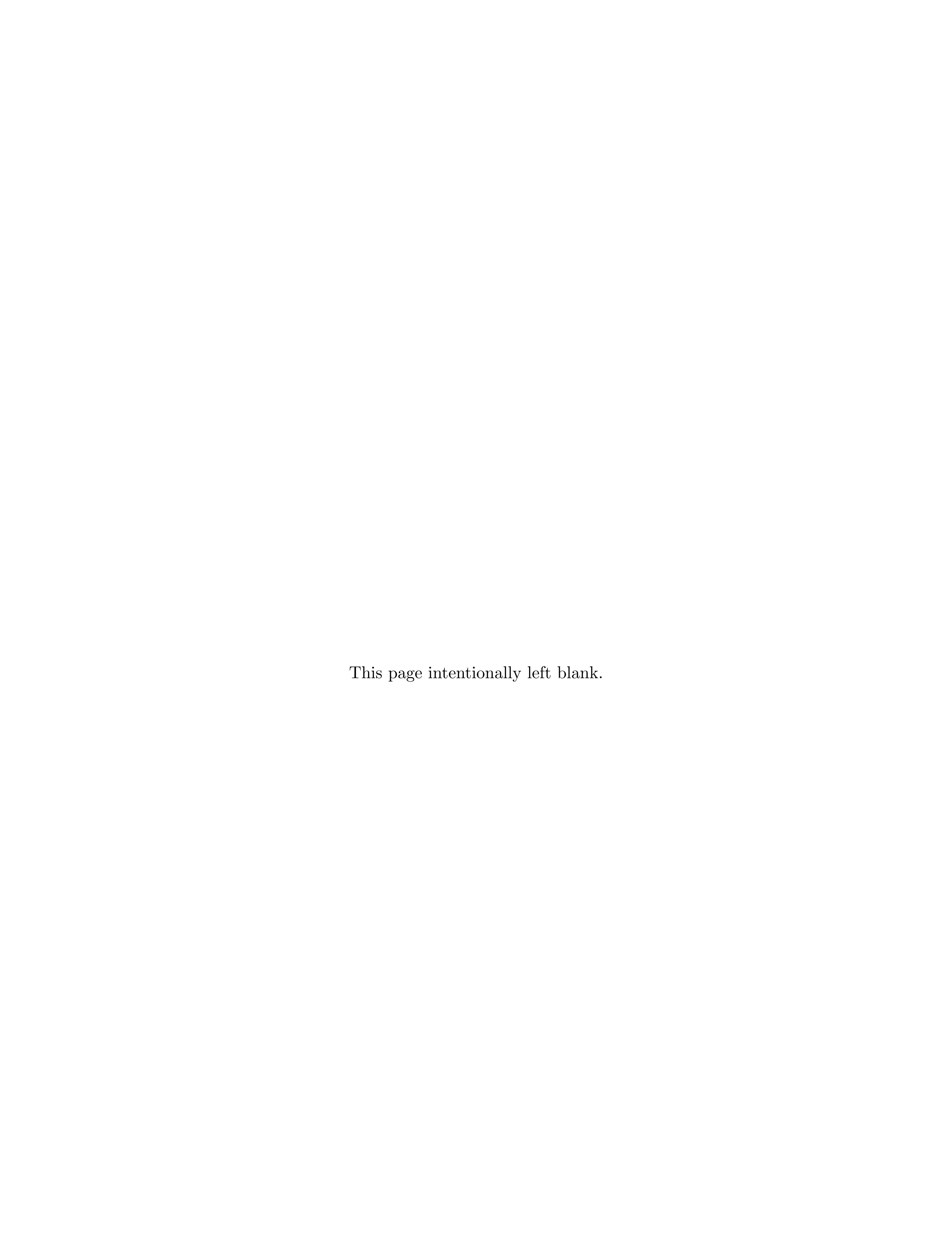
September 28, 2008

This work was sponsored by DARPA IPTO in the ExaScale Computing Study with Dr. William Harrod as Program Manager; AFRL contract number **FA8650-07-C-7724.** This report is published in the interest of scientific and technical information exchange and its publication does not constitute the Government's approval or disapproval of its ideas or findings

NOTICE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

APPROVED FOR PUBLIC RELEASE, DISTRIBUTION UNLIMITED.


This page intentionally left blank.

DISCLAIMER

The following disclaimer was signed by all members of the Exascale Study Group (listed below):

> I agree that the material in this document reflects the collective views, ideas, opinions and findings of the study participants only, and not those of any of the universities, corporations, or other institutions with which they are affiliated. Furthermore, the material in this document does not reflect the official views, ideas, opinions and/or findings of DARPA, the Department of Defense, or of the United States government.

Keren Bergman
Shekhar Borkar
Dan Campbell
William Carlson
William Dally
Monty Denneau
Paul Franzon
William Harrod
Kerry Hill
Jon Hiller
Sherman Karp
Stephen Keckler
Dean Klein
Peter Kogge
Robert Lucas
Mark Richards
Al Scarpelli
Steven Scott
Allan Snavely
Thomas Sterling
R. Stanley Williams
Katherine Yelick

This page intentionally left blank.

FOREWORD

This document reflects the thoughts of a group of highly talented individuals from universities, industry, and research labs on what might be the challenges in advancing computing by a thousand-fold by 2015. The work was sponsored by DARPA IPTO with Dr. William Harrod as Program Manager, under AFRL contract #FA8650-07-C-7724. The report itself was drawn from the results of a series of meetings over the second half of 2007, and as such reflects a snapshot in time.

The goal of the study was to assay the state of the art, and not to either propose a potential system or prepare and propose a detailed roadmap for its development. Further, the report itself was assembled in just a few months at the beginning of 2008 from input by the participants. As such, all inconsistencies reflect either areas where there really are significant open research questions, or misunderstandings by the editor. There was, however, virtually complete agreement about the key challenges that surfaced from the study, and the potential value that solving them may have towards advancing the field of high performance computing.

I am honored to have been part of this study, and wish to thank the study members for their passion for the subject, and for contributing far more of their precious time than they expected.

Peter M. Kogge, Editor and Study Lead
University of Notre Dame
May 1, 2008.

This page intentionally left blank.

Contents

1 Executive Overview **1**

2 Defining an Exascale System **5**
2.1 Attributes 5
2.1.1 Functional Metrics 5
2.1.2 Physical Attributes 6
2.1.3 Balanced Designs 6
2.1.4 Application Performance 7
2.2 Classes of Exascale Systems 8
2.2.1 Data Center System 8
2.2.2 Exascale and HPC 9
2.2.3 Departmental Systems 9
2.2.4 Embedded Systems 10
2.2.5 Cross-class Applications 11
2.3 Systems Classes and Matching Attributes 12
2.3.1 Capacity Data Center-sized Exa Systems 12
2.3.2 Capability Data Center-sized Exa Systems 13
2.3.3 Departmental Peta Systems 14
2.3.4 Embedded Tera Systems 14
2.4 Prioritizing the Attributes 14

3 Background **17**
3.1 Prehistory 17
3.2 Trends 18
3.3 Overall Observations 19
3.4 This Study 19
3.5 Target Timeframes and Tipping Points 20
3.6 Companion Studies 20
3.7 Prior Relevant Studies 21
3.7.1 1999 PITAC Report to the President 21
3.7.2 2000 DSB Report on DoD Supercomputing Needs 21
3.7.3 2001 Survey of National Security HPC Architectural Requirements 21
3.7.4 2001 DoD R&D Agenda For High Productivity Computing Systems 22
3.7.5 2002 HPC for the National Security Community 22
3.7.6 2003 Jason Study on Requirements for ASCI 23
3.7.7 2003 Roadmap for the Revitalization of High-End Computing 23
3.7.8 2004 Getting Up to Speed: The Future of Supercomputing 24

3.7.9 2005 Revitalizing Computer Architecture Research 24
3.7.10 2006 DSB Task Force on Defense Critical Technologies 25
3.7.11 2006 The Landscape of Parallel Computing Research 25

4 Computing as We Know It 27
4.1 Today's Architectures and Execution Models 27
4.1.1 Today's Microarchitectural Trends 27
4.1.1.1 Conventional Microprocessors 28
4.1.1.2 Graphics Processors 28
4.1.1.3 Multi-core Microprocessors 28
4.1.2 Today's Memory Systems 29
4.1.3 Unconventional Architectures 30
4.1.4 Data Center/Supercomputing Systems 31
4.1.4.1 Data Center Architectures 31
4.1.4.2 Data Center Power 32
4.1.4.2.1 Mitigation 33
4.1.4.3 Other Data Center Challenges 33
4.1.5 Departmental Systems 34
4.1.6 Embedded Systems 34
4.1.7 Summary of the State of the Art 35
4.2 Today's Operating Environments 35
4.2.1 Unix 36
4.2.2 Windows NT Kernel 37
4.2.3 Microkernels 37
4.2.4 Middleware 38
4.2.5 Summary of the State of the Art 38
4.3 Today's Programming Models 38
4.3.1 Automatic Parallelization 40
4.3.2 Data Parallel Languages 40
4.3.3 Shared Memory 41
4.3.3.1 OpenMP 42
4.3.3.2 Threads 43
4.3.4 Message Passing 44
4.3.5 PGAS Languages 45
4.3.6 The HPCS Languages 46
4.4 Today's Microprocessors 47
4.4.1 Basic Technology Parameters 47
4.4.2 Overall Chip Parameters 49
4.4.3 Summary of the State of the Art 53
4.5 Today's Top 500 Supercomputers 53
4.5.1 Aggregate Performance 53
4.5.2 Efficiency 54
4.5.3 Performance Components 54
4.5.3.1 Processor Parallelism 55
4.5.3.2 Clock 56
4.5.3.3 Thread Level Concurrency 56
4.5.3.4 Total Concurrency 57
4.5.4 Main Memory Capacity 59

5 Exascale Application Characteristics **61**
5.1 Kiviat Diagrams . . . 61
5.2 Balance and the von Neumann Bottleneck . . . 62
5.3 A Typical Application . . . 63
5.4 Exascale Application Characteristics . . . 65
5.5 Memory Intensive Applications of Today . . . 66
5.5.1 Latency-Sensitive Applications . . . 66
5.5.2 Locality Sensitive Applications . . . 68
5.5.3 Communication Costs - Bisection Bandwidth . . . 69
5.6 Exascale Applications Scaling . . . 71
5.6.1 Application Categories . . . 71
5.6.2 Memory Requirements . . . 72
5.6.3 Increasing Non-Main Memory Storage Capacity . . . 73
5.6.3.1 Scratch Storage . . . 73
5.6.3.2 File Storage . . . 73
5.6.3.3 Archival Storage . . . 73
5.6.4 Increasing Memory Bandwidth . . . 74
5.6.5 Increasing Bisection Bandwidth . . . 74
5.6.6 Increasing Processor Count . . . 74
5.7 Application Concurrency Growth and Scalability . . . 75
5.7.1 Projections Based on Current Implementations . . . 75
5.7.2 Projections Based on Theoretical Algorithm Analysis . . . 78
5.7.3 Scaling to Departmental or Embedded Systems . . . 80
5.8 Applications Assessments . . . 81
5.8.1 Summary Observations . . . 81
5.8.2 Implications for Future Research . . . 82

6 Technology Roadmaps **85**
6.1 Technological Maturity . . . 86
6.2 Logic Today . . . 87
6.2.1 ITRS Logic Projections . . . 87
6.2.1.1 Power and Energy . . . 88
6.2.1.2 Area . . . 88
6.2.1.3 High Performance Devices . . . 89
6.2.1.4 Low Operating Voltage Devices . . . 89
6.2.1.5 Limitations of Power Density and Its Effect on Operating Frequency 90
6.2.2 Silicon Logic Technology . . . 92
6.2.2.1 Technology Scaling Challenges . . . 92
6.2.2.2 Silicon on Insulator . . . 93
6.2.2.3 Supply Voltage Scaling . . . 94
6.2.2.4 Interaction with Key Circuits . . . 96
6.2.3 Hybrid Logic . . . 97
6.2.4 Superconducting Logic . . . 100
6.2.4.1 Logic Power and Density Comparison . . . 101
6.2.4.1.1 Cooling Costs . . . 102
6.2.4.2 The Memory Challenge . . . 102
6.2.4.3 The Latency Challenge . . . 102
6.2.4.4 The Cross-Cryo Bandwidth Challenge . . . 102

6.3 Main Memory Today . . . 103
6.3.1 The Memory/Storage Hierarchy . . . 103
6.3.2 Memory Types . . . 104
6.3.2.1 SRAM Attributes . . . 104
6.3.2.2 DRAM Attributes and Operation . . . 106
6.3.2.3 NAND Attributes and Operation . . . 107
6.3.2.4 Alternative Memory Types . . . 108
6.3.2.4.1 Phase Change Memory . . . 108
6.3.2.4.2 SONOS Memory . . . 108
6.3.2.4.3 MRAM . . . 109
6.3.3 Main Memory Reliability - Good News . . . 109
6.3.3.1 Trends in FIT Rates . . . 110
6.3.3.2 Immunity to SER . . . 111
6.3.3.3 Possible Issue: Variable Retention Time . . . 111
6.3.3.3.1 Causes . . . 111
6.3.3.3.2 Effects . . . 112
6.3.3.3.3 Mitigation . . . 112
6.3.4 The Main Memory Scaling Challenges . . . 113
6.3.4.1 The Performance Challenge . . . 113
6.3.4.1.1 Bandwidth and Latency . . . 113
6.3.4.1.2 Tradeoffs . . . 113
6.3.4.1.3 Per-pin Limitations . . . 114
6.3.4.2 The Packaging Challenge . . . 114
6.3.4.3 The Power Challenge . . . 115
6.3.4.3.1 Module Power Efficiency . . . 115
6.3.4.3.2 Cell Power . . . 116
6.3.4.4 Major Elements of DRAM Power Consumption . . . 116
6.3.4.4.1 DRAM Operating Modes . . . 117
6.3.4.4.2 DRAM Architecture . . . 118
6.3.4.4.3 Power Consumption Calculations . . . 119
6.3.5 Emerging Memory Technology . . . 120
6.4 Storage Memory Today . . . 122
6.4.1 Disk Technology . . . 122
6.4.1.1 Capacity . . . 123
6.4.1.2 Power . . . 123
6.4.1.3 Transfer Rate and Seek Time . . . 125
6.4.1.4 Time to Move a Petabyte . . . 125
6.4.1.5 Cost . . . 126
6.4.2 Holographic Memory Technology . . . 126
6.4.3 Archival Storage Technology . . . 127
6.5 Interconnect Technologies . . . 127
6.5.1 Strawman Interconnect . . . 128
6.5.1.1 Local Core-level On-chip Interconnect . . . 128
6.5.1.2 Switched Long-range On-chip Interconnect . . . 128
6.5.1.3 Supporting DRAM and CPU Bandwidth . . . 129
6.5.1.4 Intramodule Bandwidth . . . 129
6.5.1.5 Intermodule Bandwidth . . . 129
6.5.1.6 Rack to Rack Bandwidth . . . 129

6.5.2 Signaling on Wire . . . 130
6.5.2.1 Point-to-Point Links . . . 130
6.5.2.1.1 On-Chip Wired Interconnect . . . 130
6.5.2.1.2 Off-chip Wired Interconnect . . . 130
6.5.2.1.3 Direct Chip-Chip Interconnect . . . 131
6.5.2.2 Switches and Routers . . . 132
6.5.3 Optical Interconnects . . . 133
6.5.3.1 Optical Point to Point Communications . . . 134
6.5.3.2 Optical Routed Communications . . . 136
6.5.4 Other Interconnect . . . 137
6.5.5 Implications . . . 137
6.6 Packaging and Cooling . . . 139
6.6.1 Packaging . . . 139
6.6.1.1 Level 1 Packaging . . . 140
6.6.1.2 Level 2 Packaging . . . 141
6.6.2 Cooling . . . 141
6.6.2.1 Module Level Cooling . . . 142
6.6.2.2 Cooling at Higher Levels . . . 144
6.7 System Resiliency . . . 144
6.7.1 Resiliency in Large Scale Systems . . . 145
6.7.2 Device Resiliency Scaling . . . 147
6.7.3 Resiliency Techniques . . . 147
6.7.4 Checkpoint/Rollback . . . 149
6.8 Evolution of Operating Environments . . . 150
6.9 Programming Models and Languages . . . 151
6.9.1 The Evolution of Languages and Models . . . 151
6.9.2 Road map . . . 152

7 Strawmen: Where Evolution Is and Is Not Enough **153**
7.1 Subsystem Projections . . . 153
7.1.1 Measurement Units . . . 153
7.1.2 FPU Power Alone . . . 154
7.1.3 Core Energy . . . 155
7.1.4 Main Memory from DRAM . . . 156
7.1.4.1 Number of Chips . . . 156
7.1.4.2 Off-chip Bandwidth . . . 157
7.1.4.3 On-chip Concurrency . . . 157
7.1.5 Packaging and Cooling . . . 158
7.1.6 Non-Main Memory Storage . . . 162
7.1.7 Summary Observations . . . 163
7.2 Evolutionary Data Center Class Strawmen . . . 164
7.2.1 Heavy Node Strawmen . . . 164
7.2.1.1 A Baseline . . . 164
7.2.1.2 Scaling Assumptions . . . 165
7.2.1.3 Power Models . . . 166
7.2.1.4 Projections . . . 166
7.2.2 Light Node Strawmen . . . 170
7.2.2.1 A Baseline . . . 170

7.2.2.2 Scaling Assumptions 172
7.2.2.3 Power Models 173
7.2.2.4 Projections 174
7.3 Aggressive Silicon System Strawman 175
7.3.1 FPUs 176
7.3.2 Single Processor Core 178
7.3.3 On-Chip Accesses 179
7.3.4 Processing Node 181
7.3.5 Rack and System 182
7.3.5.1 System Interconnect Topology 182
7.3.5.2 Router Chips 184
7.3.5.3 Packaging within a rack 184
7.3.6 Secondary Storage 184
7.3.7 An Adaptively Balanced Node 185
7.3.8 Overall Analysis 186
7.3.9 Other Considerations 186
7.3.10 Summary and Translation to Other Exascale System Classes 187
7.3.10.1 Summary: Embedded 189
7.3.10.2 Summary: Departmental 189
7.3.10.3 Summary: Data Center 190
7.4 Exascale Resiliency 190
7.5 Optical Interconnection Networks for Exascale Systems 191
7.5.1 On-Chip Optical Interconnect 192
7.5.2 Off-chip Optical Interconnect 192
7.5.3 Rack to Rack Optical Interconnect 195
7.5.4 Alternative Optically-connected Memory and Storage System 196
7.6 Aggressive Operating Environments 198
7.6.1 Summary of Requirements 198
7.6.2 Phase Change in Operating Environments 199
7.6.3 An Aggressive Strategy 199
7.6.4 Open Questions 200
7.7 Programming Model 202
7.8 Exascale Applications 202
7.8.1 WRF 203
7.8.2 AVUS 203
7.8.3 HPL 204
7.9 Strawman Assessments 204

8 Exascale Challenges and Key Research Areas 207
8.1 Major Challenges 209
8.1.1 The Energy and Power Challenge 209
8.1.1.1 Functional Power 210
8.1.1.2 DRAM Main Memory Power 211
8.1.1.3 Interconnect Power 212
8.1.1.4 Secondary Storage Power 212
8.1.2 The Memory and Storage Challenge 212
8.1.2.1 Main Memory 212
8.1.2.2 Secondary Storage 213

8.1.3 The Concurrency and Locality Challenge . . . 214
8.1.3.1 Extraordinary Concurrency as the Only Game in Town . . . 214
8.1.3.2 Applications Aren't Going in the Same Direction . . . 216
8.1.4 The Resiliency Challenge . . . 217
8.2 Research Thrust Areas . . . 218
8.2.1 Thrust Area: Exascale Hardware Technologies and Architecture . . . 219
8.2.1.1 Energy-efficient Circuits and Architecture In Silicon . . . 220
8.2.1.2 Alternative Low-energy Devices and Circuits for Logic and Memory 222
8.2.1.3 Alternative Low-energy Systems for Memory and Storage . . . 222
8.2.1.4 3D Interconnect, Packaging, and Cooling . . . 223
8.2.1.5 Photonic Interconnect Research Opportunities and Goals . . . 224
8.2.2 Thrust Area: Exascale Architectures and Programming Models . . . 225
8.2.2.1 Systems Architectures and Programming Models to Reduce Communication . . . 225
8.2.2.2 Locality-aware Architectures . . . 225
8.2.3 Thrust Area: Exascale Algorithm and Application Development . . . 227
8.2.3.1 Power and Resiliency Models in Application Models . . . 228
8.2.3.2 Understanding and Adapting Old Algorithms . . . 228
8.2.3.3 Inventing New Algorithms . . . 229
8.2.3.4 Inventing New Applications . . . 229
8.2.3.5 Making Applications Resiliency-Aware . . . 230
8.2.4 Thrust Area: Resilient Exascale Systems . . . 230
8.2.4.1 Energy-efficient Error Detection and Correction Architectures . . . 230
8.2.4.2 Fail-in-place and Self-Healing Systems . . . 230
8.2.4.3 Checkpoint Rollback and Recovery . . . 231
8.2.4.4 Algorithmic-level Fault Checking and Fault Resiliency . . . 231
8.2.4.5 Vertically-Integrated Resilient Systems . . . 231
8.3 Multi-phase Technology Development . . . 232
8.3.1 Phase 1: Systems Architecture Explorations . . . 232
8.3.2 Phase 2: Technology Demonstrators . . . 232
8.3.3 Phase 3: Scalability Slice Prototype . . . 232

A Exascale Study Group Members 235
A.1 Committee Members . . . 235
A.2 Biographies . . . 235

B Exascale Computing Study Meetings, Speakers, and Guests 245
B.1 Meeting #1: Study Kickoff . . . 245
B.2 Meeting #2: Roadmaps and Nanotechnology . . . 246
B.3 Special Topics Meeting #1: Packaging . . . 246
B.4 Meeting #3: Logic . . . 247
B.5 Meeting #4: Memory Roadmap and Issues . . . 248
B.6 Special Topics Meeting #2: Architectures and Programming Environments . . . 249
B.7 Special Topics Meeting #3: Applications, Storage, and I/O . . . 250
B.8 Special Topics Meeting #4: Optical Interconnects . . . 250
B.9 Meeting #5: Report Conclusions and Finalization Plans . . . 251

C Glossary and Abbreviations 253

List of Figures

4.1 Three classes of multi-core die microarchitectures. 29
4.2 Microprocessor feature size. 48
4.3 Microprocessor transistor density. 48
4.4 Microprocessor cache capacity. 49
4.5 Microprocessor die size. 50
4.6 Microprocessor transistor count. 50
4.7 Microprocessor V_{dd}. 51
4.8 Microprocessor clock. 51
4.9 Microprocessor chip power. 52
4.10 Microprocessor chip power density. 52
4.11 Performance metrics for the Top 10 supercomputers over time. 53
4.12 Efficiency for the Top 10 supercomputers while running Linpack. 54
4.13 Processor parallelism in the Top 10 supercomputers. 55
4.14 Clock rate in the Top 10 supercomputers. 56
4.15 Thread level concurrency in the Top 10 supercomputers. 57
4.16 Total hardware concurrency in the Top 10 supercomputers. 58
4.17 Memory capacity in the Top 10 supercomputers. 58

5.1 Predicted speedup of WRF “Large”. 63
5.2 Time breakdown of WRF by operation category. 64
5.3 Application functionalities. 65
5.4 Predicted speedup of AVUS to latency halving. 66
5.5 Predicted speedup of WRF to latency halving. 67
5.6 Predicted speedup of AMR to latency halving. 67
5.7 Predicted speedup of Hycom to latency halving. 67
5.8 Spatial and temporal locality of strategic applications. 68
5.9 Performance strategic applications as a function of locality. 70
5.10 Growth of communications overhead. 70
5.11 WRF performance response. 75
5.12 AVUS performance response. 76
5.13 HPL performance response. 76
5.14 WRF with log(n) communications growth. 78
5.15 AVUS with log(n) communications growth. 79
5.16 Future scaling trends 82

6.1 ITRS roadmap logic device projections 87
6.2 Relative change in key power parameters 90
6.3 Power-constrained clock rate 91

6.4 Technology outlook . . . 92
6.5 Simple transport model . . . 93
6.6 Transistor sub-threshold leakage current and leakage power in recent microprocessors 93
6.7 Technology outlook and estimates . . . 94
6.8 Frequency and power scaling with supply voltage . . . 95
6.9 Sensitivities to changing V_{dd}. . . . 96
6.10 Technology scaling, V_t variations, and energy efficiency. . . . 97
6.11 Hybrid logic circuits . . . 99
6.12 CPU and memory cycle time trends. . . . 103
6.13 ITRS roadmap memory density projections. . . . 105
6.14 DRAM cross section. . . . 106
6.15 Programmed and un-programmed NAND cells. . . . 107
6.16 DRAM retention time distribution. . . . 109
6.17 Memory module RMA results. . . . 110
6.18 Variable retention time as it affects refresh distribution. . . . 112
6.19 Industry memory projections. . . . 113
6.20 Reduced latency DRAM. . . . 114
6.21 Center-bonded DRAM package. . . . 114
6.22 Commodity DRAM module power efficiency as a function of bandwidth. . . . 116
6.23 Commodity DRAM voltage scaling. . . . 117
6.24 Block diagram of 1Gbit, X8 DDR2 device. . . . 118
6.25 DDR3 current breakdown for Idle, Active, Read and Write. . . . 119
6.26 Nanoscale memory addressing. . . . 121
6.27 Nanoscale memory via imprint lithography . . . 122
6.28 Disk capacity properties. . . . 123
6.29 Disk power per Exabyte. . . . 124
6.30 Disk transfer rate properties. . . . 124
6.31 Disk price per GB. . . . 125
6.32 Interconnect bandwidth requirements for an Exascale system. . . . 128
6.33 Comparison of 3D chip stacking communications schemes. . . . 131
6.34 Entire optical communication path. . . . 133
6.35 Modulator approach to integrated optics. . . . 135
6.36 Representative current and future high-end level 1 packaging. . . . 139
6.37 Estimated chip counts in recent HPC systems. . . . 144
6.38 Scaling trends for environmental factors that affect resiliency. . . . 146
6.39 Increase in vulnerability as a function of per-socket failure rates. . . . 148
6.40 Projected application utilization when accounting for checkpoint overheads. . . . 149

7.1 Projections to reach an Exaflop per second. . . . 154
7.2 Energy per cycle for several cores. . . . 155
7.3 DRAM as main memory for data center class systems. . . . 156
7.4 Memory access rates in DRAM main memory. . . . 158
7.5 Potential directions for 3D packaging (A). . . . 160
7.6 Potential directions for 3D packaging (B). . . . 161
7.7 Potential directions for 3D packaging (C). . . . 161
7.8 A typical heavy node reference board. . . . 164
7.9 Characteristics of a typical board today. . . . 165
7.10 Heavy node strawman projections. . . . 167

7.11 Heavy node performance projections. 168
7.12 Heavy node GFlops per Watt. 169
7.13 Power distribution in the light node strawman. 172
7.14 Light node strawman performance projections. 174
7.15 Light node strawman Gflops per watt. 175
7.16 Aggressive strawman architecture. 177
7.17 Possible aggressive strawman packaging of a single node. 181
7.18 Power distribution within a node. 182
7.19 The top level of a dragonfly system interconnect. 183
7.20 Power distribution in aggressive strawman system. 186
7.21 Chip super-core organization and photonic interconnect. 192
7.22 Gateway functional block design. 193
7.23 Super-core to super-core optical on-chip link. 194
7.24 Optical system interconnect. 195
7.25 A possible optically connected memory stack. 197

8.1 Exascale goals - Linpack. 208
8.2 Critically of each challenge to each Exascale system class. 209
8.3 The power challenge for an Exaflops Linpack. 211
8.4 The overall concurrency challenge. 215
8.5 The processor parallelism challenge. 216
8.6 Future scaling trends present DARPA-hard challenges. 217
8.7 Sensitivities to changing V_{dd}. 221

List of Tables

2.1 Attributes of Exascale class systems. 12
2.2 Attributes of Exascale class systems. 15

4.1 Power distribution losses in a typical data center. 32

5.1 Summary applications characteristics. 81

6.1 Some performance comparisons with silicon. 98
6.2 2005 projection of potential RSFQ logic roadmap. 101
6.3 Area comparisons of various memory technologies. 104
6.4 Memory types and characteristics. 108
6.5 Commodity DRAM operating current. 117
6.6 Projected disk characteristics. 122
6.7 Energy budget for optical modulator. 136
6.8 Summary interconnect technology roadmap. 138
6.9 Internal heat removal approaches. 142
6.10 External cooling mechanisms. 143
6.11 Root causes of failures in Terascale systems. 145
6.12 BlueGene FIT budget. 146

7.1 Non-memory storage projections for Exascale systems. 162
7.2 Light node baseline based on Blue Gene. 171
7.3 Summary characteristics of aggressively designed strawman architecture. 176
7.4 Expected area, power, and performance of FPUs with technology scaling. 176
7.5 Expected area, power, and performance of FPUs with more aggressive voltage scaling. 178
7.6 Energy breakdown for a four FPU processor core. 178
7.7 Power budget for strawman multi-core chip. 180
7.8 Area breakdown of processor chip. 181
7.9 Power allocation for adaptive node. 185
7.10 Exascale class system characteristics derived from aggressive design. 188
7.11 Failure rates for the strawman Exascale system. 190
7.12 Checkpointing overheads. 191
7.13 Optical interconnect power parameters. 193
7.14 Optical on-chip interconnect power consumption. 194
7.15 Optical system interconnect power consumption. 196

8.1 The relationship between research thrusts and challenges. 210

A.1 Study Committee Members. 235

Chapter 1

Executive Overview

This report presents the findings and recommendations of the **Exascale Working Group** as conducted over the summer and fall of 2007. The objectives given the study were to understand the course of mainstream computing technology, and determine whether or not it would allow a 1,000X increase in the computational capabilities of computing systems by the 2015 time frame. If current technology trends were deemed as not capable of permitting such increases, then the study was also charged with identifying where were the major challenges, and in what areas may additional targeted research lay the groundwork for overcoming them.

The use of the word "Exascale"[1] in the study title was deliberately aimed at focusing the group's attention on more than just high end, floating point intensive, supercomputers ("exaflops" machines), but on increasing our ability to perform computations of both traditional and emerging significance at across-the-board levels of performance. The study thus developed as a more precise goal the understanding of technologies to accelerate computing by 1,000X for three distinct classes of systems:

- **data center-sized systems**, where the focus is on achieving 1,000 times the performance of the "petaflop" class systems that will come on line in the next few years, and for more than just numeric intensive applications.
- **departmental-sized systems** that would allow the capabilities of the near-term Petascale machines to be shrunk in size and power to fit within a few racks, allowing widespread deployment.
- **embedded systems** that would allow something approximating a "Terascale" rack of computing such as may be found in the near-term Petascale systems to be reduced to a few chips and a few ten's of watts that would allow deployment in a host of embedded environments.

Clearly, if done right, a technology path that permits Terascale embedded systems would allow variants of that to serve in Petascale departmental systems, and then in turn grow into larger Exascale supercomputers. The study group recognizes that even though the underlying technologies may be similar, when implementing real systems the actual mix of each technology, and even the architecture of the systems themselves may be different. Thus it was not our focus to design such systems, but to develop a deep understanding of the technological challenges that might prohibit their implementation.

[1] The term "Exa" is entering the national dialog through discussions of total Internet bandwidth and on-net storage - see for example *Unleashing the 'Exaflood'* in the Feb. 22, 2008 Wall Street Journal.

In total, the study concluded that there are four **major challenges** to achieving Exascale systems where current technology trends are simply insufficient, and significant new research is absolutely needed to bring alternatives on line.

- The **Energy and Power Challenge** is the most pervasive of the four, and has its roots in the inability of the group to project any combination of currently mature technologies that will deliver sufficiently powerful systems in any class at the desired power levels. Indeed, a key observation of the study is that it may be easier to solve the power problem associated with base computation than it will be to reduce the problem of transporting data from one site to another - on the same chip, between closely coupled chips in a common package, or between different racks on opposite sides of a large machine room, or on storing data in the aggregate memory hierarchy.
- The **Memory and Storage Challenge** concerns the lack of currently available technology to retain data at high enough capacities, and access it at high enough rates, to support the desired application suites at the desired computational rate, and still fit within an acceptable power envelope. This information storage challenge lies in both main memory (DRAM today) and in secondary storage (rotating disks today).
- The **Concurrency and Locality Challenge** likewise grows out of the flattening of silicon clock rates and the end of increasing single thread performance, which has left explicit, largely programmer visible, parallelism as the only mechanism in silicon to increase overall system performance. While this affects all three classes of systems, projections for the data center class systems in particular indicate that applications may have to support upwards of a billion separate threads to efficiently use the hardware.
- A **Resiliency Challenge** that deals with the ability of a system to continue operation in the presence of either faults or performance fluctuations. This concern grew out of not only the explosive growth in component count for the larger classes of systems, but also out of the need to use advanced technology, at lower voltage levels, where individual devices and circuits become more and more sensitive to local operating environments, and new classes of aging effects become significant.

While the latter three challenges grew out of the high end systems, they are certainly not limited to that class. One need only look at the explosive grow of highly multi-core microprocessors and their voracious appetite for more RAM.

The study's recommendations are thus that significant research needs to be engaged in four major research thrusts whose effects cross all these challenges:

1. Co-development and optimization of **Exascale Hardware Technologies and Architectures**, where the potential of new devices must be evaluated in the context of new architectures that can utilize the new features of such devices to solve the various challenges. This research includes development of energy efficient circuits for communication, memory, and logic; investigation of alternative low-energy devices; development of efficient packaging, interconnection, and cooling technologies; and energy efficient machine organizations. Each of these research areas should be investigated in the context of all Exascale system classes, and metrics should be in terms of energy efficiency realized in this context.
2. Co-development and optimization of **Exascale Architectures and Programming Models**, where the new architectures that arise out of either the device efforts of the previous

thrust or those needed to control the explosive growth in concurrency must be played out against programming models that allow applications to use them efficiently. This work includes developing locality-aware and communication-efficient architectures to minimize energy consumed through data movement, and developing architectures and programming models capable of handling billion-thread concurrency.

3. Co-development of **Exascale Algorithms, Applications, Tools, and Run-times**, where substantial alternatives are needed for programmers to develop and describe, with less that heroic efforts, applications that can in fact use the new architectures and new technology capabilities in efficient enough ways that permit true "Exascale" levels of performance to be sustained.

4. Development of a deep understanding of how to architect **Resilient Exascale Systems**, where the problems in both sheer hardware and algorithmic complexity, and the emergence of new fault mechanisms, are studied in a context that drives the development of circuit, subsystem, and system-level architectures, and the programming models that can exploit them, in ways that allows Exascale systems to provide the kind of dependable service that will make them technically and economically viable.

Further, such research must be very heavily inter-disciplinary. For example, power is the number one concern, but research that focuses only on low-power devices is unlikely to solve the systemic power problems the study encountered. Co-optimization of devices and circuits that employ those devices must be done within the framework of innovative micro and system architectures which can leverage the special features of such devices in the most efficient ways. An explicit and constant attention must be made to **interconnect**, namely the technologies by which one set of functions exchange data with another.

As another example, research that leads to increasing significantly the density of memory parts will clearly help reliability by reducing part count; it may also reduce power significantly by reducing the number of off-chip interfaces that need to be driven, and reduce the number of memory controller circuits that must be incorporated elsewhere.

Finally, if the appropriate technologies are to be developed in time to be considered for 2015 deployments, then, given the level of technical maturity seen by the study group in potential emerging technologies, a three phase research agenda is needed:

1. A **System Architecture Exploration Phase** to not only enhance the maturity of some set of underlying device technologies, but to do enough of the higher level system architecture and modeling efforts to identify how best to employ the technology and what the expected value might be *at the system level.*

2. A **Technology Demonstration Phase** where solutions to the "long poles" in the challenges are developed and demonstrated in an isolated manner.

3. A **Scalability Slice Prototyping Phase** where one or more of the disparate technologies demonstrated in the preceding phase are combined into coherent end-to-end "slices" of a complete Exascale system that permits believable scaling to deployable systems. Such slices are not expected to be a complete system in themselves, but should include enough of multiple subsystems from a potential real system that the transition to a real, complete, system integration is feasible and believable.

The rest of this report is organized as follows:

- Chapter 2 defines the major classes of Exascale systems and their attributes.
- Chapter 3 gives some background to this study in terms of prior trends and studies.
- Chapter 4 discuss the structure and inter-relationship of computing systems today, along with some key historical data.
- Chapter 5 concentrates on developing the characteristics of applications that are liable to be relevant to Exascale systems.
- Chapter 6 reviews the suite of relevant technologies as we understand them today, and develops some roadmaps as to how they are likely to improve in the future.
- Chapter 7 takes the best of known currently available technologies, and projects ahead what such technologies would yield at the Exascale. This is particularly critical in identifying the holes in current technology trends, and where the challenges lie.
- Chapter 8 summarizes the major challenges, develops a list of major research thrusts to answer those challenges, and suggests a staged development effort to bring them to fruition.

Chapter 2

Defining an Exascale System

The goal of this study was not to design Exascale systems, but to identify the overall challenges and problems that need special emphasis between now and 2010 in order to have a technology base sufficient to support development and deployment of Exascale-class systems by 2015. However, to do this, it is first necessary to define more carefully those classes of systems that would fall under the Exascale rubric. In this chapter, we first discuss the attributes by which achievement of the label "Exascale" may be claimed, then the classes of systems that would lever such attributes into "Exascale" systems.

2.1 Attributes

To get to the point of being able to analyze classes of Exascale systems, we first need a definition of what "Exascale" means. For this study, an Exascale system is taken to mean that one or more key attributes of the system has 1,000 times the value of what an attribute of a "Petascale" system of 2010 will have. The study defined three dimensions for these attributes: functional performance, physical attributes, and application performance. Each is discussed below.

2.1.1 Functional Metrics

We define a functional metric as an attribute of a system that directly measures some parameter that relates to the ability of the system to solve problems. The three major metrics for this category include:

- **Basic computational rate**: the rate at which some type of operation can be executed per second. This includes, but is not limited to:
  - **flops**: floating point operations per second.
  - **IPS**: instructions per second.
  - and (remote) **memory accesses** per second.
- **Storage capacity**: how much memory is available to support holding different forms of the problem state at different times. Specific metrics here include the capacities of various parts of the storage hierarchy:
  - **Main memory**: the memory out of which an application program can directly access data via simple loads and stores.

  - **Scratch Storage**: the memory needed to hold checkpoints, I/O buffers, and the like during a computation.
  - **Persistent Storage**: the storage to hold both initial data sets, parameter files, computed results, and long-term data used by multiple independent runs of the application.

- **Bandwidth**: the rate at which data relevant to computation can be moved around the system, usually between memory and processing logic. Again, a variety of specialized forms of bandwidth metrics are relevant, including:

  - **Bisection bandwidth**: for large systems, what is the bandwidth if we partition the system in half, and measure the maximum that might flow from one half of a system to the other.
  - **Local memory bandwidth**: how fast can data be transferred between memories and closest computational nodes.
  - **Checkpoint bandwidth**: how fast copies of memory can be backed up to a secondary storage medium to permit roll-back to some fixed point in the program if a later error is discovered in the computations due to hardware fault.
  - **I/O bandwidth**: the rate at which data can be extracted from the computational regions of the system to secondary storage or visualization facilities where it can be saved or analyzed later.
  - **On chip bandwidth**: how fast can data be exchanged between functional units and memory structures within a single chip.

2.1.2 Physical Attributes

The second class of attributes that are relevant to an Exascale discussion are those related to the instantiation of the design as a real system, primarily:

- Total power consumption
- Physical size (both area and volume)
- Cost

Since this is primarily a technology study, cost is one metric that will for the most part be ignored.

While peak values for all the above are important, from a technology vantage, it proved to be more relevant to use these metrics as denominators for ratios involving the functional metrics described above. Further, for power it will also be valuable to focus not just on the "per watt" values of a technology, but a "per joule" metric. Knowing the rates per joule allows computation of the total energy needed to solve some problem; dividing by the time desired gives the total power.

2.1.3 Balanced Designs

While individual metrics give valuable insight into a design, more complete evaluations occur when we consider how designs are **balanced**, that is how equal in efficiency of use is the design with respect to different high-cost resources. This balance is usually a function of application class.

For example, some systems based on special purpose processor designs such as Grape[97] may be "well-balanced" for some narrow classes of applications that consume the whole machine (such as multi-body problems), but become very inefficient when applied to others (insufficient memory, bandwidth, or ability to manage long latencies). Other systems such as large scale search engines may be built out of more commodity components, but have specialized software stacks and are balanced so that they perform well only for loads consisting of large numbers of short, independent, queries that interact only through access to large, persistent databases.

Classically, such balance discussions have been reduced to simple ratios of metrics from above, such as "bytes to flops per second" or "Gb/s per flops per second." While useful historical vignettes, care must be taken with such ratios when looking at different scales of applications, and different application classes.

2.1.4 Application Performance

There are two major reasons why one invests in a new computing system: for solving problems not previously solvable, either because of time to solution or size of problem, or for solving the same kinds of problems solved on a prior system, but faster or more frequently. Systems that are built for such purposes are known as capability and capacity systems respectively. The NRC report *Getting Up to Speed* ([54] page 24) defines these terms more formally:

> The largest supercomputers are used for **capability** or turnaround computing where the maximum processing power is applied to a single problem. The goal is to solve a larger problem, or to solve a single problem in a shorter period of time. Capability computing also enables the solution of problems that cannot otherwise be solved in a reasonable period of time (for example, by moving from a two-dimensional to a three-dimensional simulation, using finer grids, or using more realistic models). Capability computing also enables the solution of problems with real-time constraints (e.g. intelligence processing and analysis). The main figure of merit is time to solution.
>
> Smaller or cheaper systems are used for **capacity** computing, where smaller problems are solved. Capacity computing can be used to enable parametric studies or to explore design alternatives; it is often needed to prepare for more expensive runs on capability systems. Capacity systems will often run several jobs simultaneously. The main figure of merit is sustained performance per unit cost.

Capacity machines are designed for throughput acceleration; they accelerate the rate at which certain types of currently solvable applications can be solved. Capability machines change the spectrum of applications that are now "solvable," either because such problems can now be fit in the machine and solved at all, or because they can now be solved fast enough for the results to be meaningful (such as weather forecasting).

We note that there is significant fuzziness in these definitions, especially in the "real-time" arena. A machine may be called a capability machine if it can take a problem that is solvable in prior generations of machines (but not in time for the results to be useful), and make it solvable (at the same levels of accuracy) in some fixed period of time where the results have value (such a weather forecasting). A machine that solves the same problems even faster, or solves multiple versions concurrently, and still in real-time, may be called a capacity machine. It is not uncommon for today's capability systems to become tomorrow's capacity systems and newer, and even more capable, machines are introduced.

Also, some machines may have significant aspects of both, such as large scale search engines, where the capability part lies in the ability to retain significant indexes to information, and the capacity part lies in the ability to handle very large numbers of simultaneous queries.

Although there is significant validity to the premise that the difference between a capacity and a capability machine may be just in the "style" of computing or in a job scheduling policy, we will continue the distinction here a bit further mainly because it may affect several system parameters that in turn affect real hardware and related technology needs. In particular, it may relate to the breadth of applications for which a particular design point is "balanced" to the point of being economically valuable. As an example, it appears from the strawmen designs of Chapter 7 that silicon-based Exascale machines may be significantly memory-poor in relation to today's supercomputing systems. This may be acceptable for some large "capability" problems where the volume of computation scales faster than required data size, but not for "capacity" applications where there is insufficient per-node memory space to allow enough copies of enough different data sets to reside simultaneously in the machine in a way that can use all the computational capabilities.

2.2 Classes of Exascale Systems

To reiterate the purpose of this study, it is to identify those technology challenges that stand in the way of achieving initial deployment of Exascale systems by 2015. The approach taken here for identifying when such future systems may be called "Exascale" is multi-step. First, we partition the Exascale space based on the gross physical characteristics of a deployed system. Then, we identify the functional and application performance characteristics we expect to see needed to achieve "1,000X" increase in "capability." Finally, we look at the ratios of these characteristics to power and volume as discussed above. Then by looking at these ratios we can identify where the largest challenges are liable to arise, and which class of systems will exhibit them.

In terms of overall physical size and power, we partition the 2015 system space into three categories:

- "Exa-sized" data center systems,
- "Peta-sized" departmental computing systems,
- and "Tera-sized" Embedded systems.

It is very important to understand that these three systems are not just a 1,000X or 1,000,000X scaling in all parameters. Depending on the application class for each level, the individual mix of parameters such as computational rates, memory capacity, and memory bandwidth may vary dramatically.

2.2.1 Data Center System

For this study, an exa-sized **data center system** of 2015 is one that roughly corresponds to a typical notion of a supercomputer center today - a large machine room of several thousand square feet and multiple megawatts of power consumption. This is the class system that would fall in the same footprint as the Petascale systems of 2010, except with 1,000X the capability. Because of the difficulty of achieving such physical constraints, the study was permitted to assume some growth, perhaps a factor of 2X, to something with a maximum limit of 500 racks and 20 MW for the computational part of the 2015 system.

2.2.2 Exascale and HPC

In **high-end computing** (i.e. **supercomputing** or **high-performance computing**), the major milestones are the emergence of systems whose aggregate performance first crosses a threshold of 10^{3k} operations performed per second, for some k. Gigascale (10^9) was achieved in 1985 with the delivery of the Cray 2. Terascale (10^{12}) was achieved in 1997 with the delivery of the Intel ASCI Red system to Sandia National Laboratory. Today, there are contracts for near-Petascale (10^{15}) systems, and the first will likely be deployed in 2008. Assuming that progress continues to accelerate, one might hope to see an Exascale (10^{18}) system as early as 2015.

For most scientific and engineering applications, Exascale implies 10^{18} IEEE 754 Double Precision (64-bit) operations (multiplications and/or additions) per second (**exaflops**[1]). The **High Performance Linpack** (**HPL**) benchmark[118], which solves a dense linear system using LU factorization with partial pivoting, is the current benchmark by which the community measures the throughput of a computing system. To be generally accepted as an Exascale system, a computer must exceed 10^{18} flops (1 exaflops) on the HPL benchmark. However, there are critical Defense and Intelligence problems for which the operations would be over the integers or some other number field. Thus a true Exascale system has to execute a fairly rich instruction set at 10^{18} operations per second lest it be simply a special purpose machine for one small family of problems.

A truly general purpose computer must provide a balance of arithmetic throughput with memory volume, memory and communication bandwidth, persistent storage, etc. To perform the HPL benchmark in a reasonable period of time, an Exascale system would require on the order of 10 petabytes (10^{16} Bytes) of main memory. Such a system could credibly solve a small set of other problems. However, it would need at least another order-of-magnitude of additional main memory (10^{17} bytes) to address as broad a range of problems as will be tackled on near-Petascale systems in the next year or so. Amdahl's rule of thumb was one byte of main memory per operation, but in recent years systems have been deployed with less (0.14 to 0.3), reflecting both the diverging relative costs of the components as well as the evolving needs of applications. Chapter 5 tries to get a better handle on this as we scale algorithms up.

Finally, an Exascale system must provide data bandwidth to and from at least local subsets of its memory at approximately an exabyte per second. Section 5.6 discusses sensitivities of many current algorithms to bandwidth.

To store checkpoints, intermediate files, external data, results, and to stage jobs, an Exascale system will need at least another order-of-magnitude (i.e., 10^{18} Bytes) of persistent storage, analogous to today's disk arrays, with another 10-100X for file storage. Section 5.6.3 explores these numbers in more detail.

2.2.3 Departmental Systems

The discussion in the previous was oriented towards the requirements for a leadership-class Exascale system deployed to support a handful of national-scale, capability jobs. To be economically viable, the technology developed for such a system must also be utilized in higher volume systems such as those deployed for departmental-scale computing in industry and government. To the casual eye, the biggest difference will be in the physical size of the departmental systems, which will only fill a few racks. Power density will be a critical aspect of these systems, as many customers will want them to be air cooled. Others may only have building chilled water available. Thus, a peta-sized **departmental system** of 2015 would be one whose computational capabilities would match

[1]We denote a computation involving a total of 10^{18} floating point operations as an **exaflop** of computation; if they are all performed in one second then the performance is one **exaflops**.

roughly those of a 2010 Petascale data center-sized system, but in the form factor of a departmental computing cluster - perhaps one or two racks, with a maximum power budget of what could be found in reasonable computer "cluster" room of today - perhaps 100-200KW at maximum. In a sense this is around 1/1,000th in both capability and size of the larger systems.

The principle differentiation between the Exascale data center systems and departmental systems will be the need to support a large set of third-party applications. These span a broad range from business applications like Oracle to engineering codes like LS-DYNA[32]. These are very large, sophisticated applications that demand the services of a full featured operating system, usually a derivative of UNIX, not a micro-kernel as may suffice on the early Exascale systems. The operating system will need to support virtualization and to interact with a **Grid**[45] of external systems as the department will likely be part of a much larger, geographically distributed enterprize.

To support a broad range of such mainstream applications, departmental systems composed of Exascale components will require a proportionately larger main memory and persistent store. It's not uncommon today to see systems such as the SGI Altix delivered with only four Intel Itanium CPUs yet a terabyte of main memory. Latency and bandwidth to the memory hierarchy (DRAM and disk) is already a problem today for mainstream applications, and probably cannot be tapered as aggressively as it likely will be on a leadership, Exascale system.

The third party applications whose availability will be critical to the success of departmental scale systems in 2015 run on up to 100 processors today. Only a handful of scaling studies or heroic runs exceed that. Petascale departmental systems will likely have 10^5, perhaps even 10^6 threads in them. Extending enough of today's applications to this level of parallelism will be a daunting task. In the last fifteen years, commercial software developers have struggled to transition their codes from one CPU (perhaps a vector system) to O(1000) today. They will have to expand this scalability by another three orders-of-magnitude in the next decade. This will require breakthroughs not only in computer architecture, compilers and other software tools, but also in diverse areas of applied mathematics, science, and engineering. Not amount of system concurrency can overcome an algorithm that does not scale well. Finally, mainstream applications can not be expected to have evolved to the point where they can adapt to faults in the underlying system. Thus mean-time-to failure comparable to today's large-scale servers will be necessary.

2.2.4 Embedded Systems

One of the motivations for the development of parallel computing systems in the late 1980's and early 1990's was to exploit the economies of scale enjoyed by the developers of mainstream, commodity components. At the time, commodity meant personal computers and servers. Today, the volume of electronics created for embedded systems, such as cell phones, dwarfs desktop and servers, and this trend will likely continue. Thus its increasingly clear that high-end computing will need to leverage embedded computing technology. In fact, the blending of embedded technology and HPC has already begun, as demonstrated by IBM's Blue Gene family of supercomputers, and HPC derivatives, such as the Roadrunner at Los Alamos National Labs, using the STI Cell chip (which has its origins in game systems).

If one hopes to use Exascale technology in embedded systems, not just leadership and departmental scale systems, then there are a number of additional issues that must be addressed. The first is the need to minimize size, weight, and power consumption. While embedded systems are often very aggressive in their use of novel packaging technology (e.g. today's DRAM stacks), they cannot similarly exploit the most aggressive liquid cooling technology which is often available in large machine rooms.

Historically embedded processors traded lower performance for greater power efficiency, whereas

processors designed for large servers maximized performance, and power was only constrained by the need to cool the system. At Exascale, this will no longer be the case. As this study documents, limiting power consumption will be a major issue for Exascale systems. Therefore, this major difference in the design of embedded and server systems may very well disappear. This is not to say that the two processor niches will merge. Servers will need to be engineered as components in much larger systems, and will have to support large address spaces that will not burden embedded systems.

Embedded systems, especially those developed for national security applications, often operate in hostile environment conditions for long periods of time, during which they cannot be serviced. Space is the obvious example, and to be useful in such an environment, one must be able to extend Exascale technology to be radiation hard. The architecture must be fault tolerant, and able to degrade gracefully when individual components inevitably fail. It is interesting to speculate that this distinction between the fault tolerance design standards for today's embedded vs. enterprise systems may disappear as we approach Exascale since smaller geometries and lower voltage thresholds will lead to components that are inherently less reliable even when deployed in normal, well controlled environments.

As with departmental scale systems, the biggest difference between pioneering embedded Exascale systems and their contemporary embedded systems will likely be the software. Whereas most scientific and engineering applications operate on double precision values, embedded applications are often fixed point (there is no need to carry more precision than the external sensors provide) or, increasingly, single precision. The applications and hence the operating system will often have hard real time constraints. The storage required to hold application is often limited, and this bounds the memory footprint of the operating systems and the application.

2.2.5 Cross-class Applications

While the three classes were discussed above in isolation, there are significant applications that may end up needing all three at the same time. Consider, for example, a unified battlefield **persistent surveillance** system that starts with drones or other sensor platforms to monitor a wide spectrum of potential information domains, proceeds through local multi-sensor correlation for real-time event detection, target development, and sensor management to post-event collection management and forensic analysis, where tactical and strategic trends can be identified and extracted.

A bit more formally, a recent DSB report[18] (page 103) defines persistent surveillance as follows:

> The systematic and integrated management of collection processing, and customer collaboration for assured monitoring of all classes of threat entities, activities and environments in physical, aural or cyber space with sufficient frequency, accuracy, resolution, precision, spectral diversity, spatial extent, spatial and sensing diversity and other enhanced temporal and other performance attributes in order to obtain the desired adversary information, even in the presence of deception.

Implementing such systems may very well require Exascale embedded systems in a host of different platforms to do radar, video, aural, or cyber processing, analysis, and feature extraction; Exascale departmental systems may be needed to integrate multiple sensors and perform local event detection and sensor management, and larger Exascale data center class systems are needed for overall information collection and deep analysis.

	Attributes				
	Aggregate Computational Rate	**Aggregate Memory Capacity**	**Aggregate Bandwidth**	**Volume**	**Power**
Exa Scale Data Center Capacity System relative to 2010 Peta Capacity System					
Single Job Speedup	1000X flops	Same	1000X	Same	Same
Job Replication	1000X flops	up to 1000X	1000X	Same	Same
Exa Scale Data Center Capability System relative to 2010 Peta Capability System					
Current in Real-Time	1000X flops, ops	Same	1000X	Same	Same
Scaled Current Apps	up to 1000X flops, ops	up to 1000X	up to 1000X	Same	Same
New Apps	up to 1000X flops, ops, mem accesses	up to 1000X - with more persistence	up to 1000X	Same	Same
Peta Scale Department System relative to 2010 Peta HPC System					
	Same	Same	Same	1/1000	1/1000
Tera Scale Embedded System relative to 2010 Peta HPC System					
	1/1000	1/1000	1/1000	1/1 million	1/1 million

Table 2.1: Attributes of Exascale class systems.

2.3 Systems Classes and Matching Attributes

The following subsections discuss each of these target system, with Table 2.1 summarizing how the various metrics might inter-relate. The header row for each class includes a reference to the type of system used as a baseline in estimating how much on an increases in the various attributes are needed to achieve something that would be termed an Exascale system. Attribute columns with the value "same" should be interpreted as being close to (i.e. perhaps within a factor of 2) of the same numbers for the 2010 reference point.

2.3.1 Capacity Data Center-sized Exa Systems

A 2015 data center sized capacity system is one whose goal would be to allow roughly 1,000X the production of results from the same applications that run in 2010 on the Petascale systems of the time. In particular, by "same" is meant "approximately" (within a factor of 2 or so) the same data set size (and thus memory capacity) and same application code (and thus the same primary metric of computational rate as for the reference 2010 Petascale applications - probably still flops).

There are two potential variants of this class, based on how the increase in throughput is achieved: by increasing the computational rate as applied to a single job by 1,000X, or by concurrently running multiple (up to 1,000) jobs through the system at the same time. In either case the aggregate computational rates must increase by 1,000X. For the single job speedup case, bandwidths probably scale linearly with the 1,000X in rate, but total memory capacity will stay roughly flat at what it will be in 2010.

For systems supporting concurrent jobs, the total memory capacity must scale with the degree of concurrency. While most of the lower levels of bandwidth must scale, it may be that the system-

wide bandwidth need not scale linearly, since the footprint in terms of racks needed for each job may also shrink.

2.3.2 Capability Data Center-sized Exa Systems

A 2015 data center sized capability system is one whose goal would be to allow solution of problems up to 1,000 times "more complex" than solvable by a 2010 peta capability system. In contrast to capacity machines, these systems are assumed to run only one application at a time, so that sizing the various attributes need reflect only one application. There are least three variants of such systems:

- Real-time performance: where the same application that runs on a peta system needs to run 1,000X faster to achieve value for predictive purposes. As with the capacity system that was speedup based, computational rate and bandwidth will scale, but it may be that memory capacity need not.

  In this scenario, if the basic speed of the computational units does not increase significantly (as is likely), then new levels of parallelism must be discovered in the underlying algorithms, and if that parallelism takes a different form than the current coarse-grained parallelism used on current high end systems, then the software models will have to be developed to support that form of parallelism.

- Upscaling of current peta applications: where the overall application is the same as exists for a 2010 peta scale system, but the data set size representing problems of interest needs to grow considerably. There the meaning of a 1000X increase in computing may have several interpretations. A special case is the weak-scaling scenario in which a fixed amount of data is used per thread; if the computation is linear time in the data size, then this corresponds to a 1000x increase in memory capacity along with computation and bandwidth. Another obvious one may mean solving problem sizes that require 1,000X the number of computations in the same time as the maximum sized problems did on the reference peta system. Here computational and bandwidths scale by 1,000X, and memory must scale at least as fast enough to contain the size problems that force the 1,000X increase in computation. This memory scaling may thus vary from almost flat to linear in the performance increase. Intermediate numbers are to be expected, where, for example, 4D simulations may has an $N^{3/4}$ law, meaning that 1,000X in performance requires $1000^{3/4} = 178$X the memory capacity.

  A second interpretation of a 1000X gain in computation is that there may be some product between increase in performance and increase in problem size (i.e. storage). Thus, for example, a valid definition of an Exascale system may be one that supports data sets 100X that of today, and provides 10X more computation per second against its solution, regardless of how long the total problem execution takes.

- New applications: where the properties of the desired computation looks nothing like what is supportable today. This includes the types of operations that in the end dominate the computational rate requirements (instead of flops, perhaps integer ops or memory accesses), the amount and type of memory (very large graphs, for example, that must persist in memory essentially continuously), to bandwidth (which might even explode to become the dominant performance parameter). These new application might very well use algorithms that are unknown today, along with new software and architecture models.

2.3.3 Departmental Peta Systems

Another version of an exa sized system might be one that takes a 2010 peta scale system (encompassing 100s' of racks) and physically reduces it to a rack or two that may fit in, and be dedicated to, the needs of a small part of an organization, such as a department. This rack would thus in 2015 have the computational attributes of a 2010 peta system in the footprint of a 2010 tera scale systems.

If the goal is to shrink a 2010 peta scale system, then the overall computational rate and memory capacity at an absolute level, are unchanged from the 2010 numbers. The power and volume limits, however, must scale by a factor of about 1/1000. Also, while the aggregate internal system bandwidth is unchanged from the 2010 system, reducing the physical size also means that where the bandwidth requirements must be met changes. For example, going to a 1-2 rack system means that much more of the 2015 inter-rack data bandwidth must be present within a single rack.

2.3.4 Embedded Tera Systems

The final exa technology-based system in 2015 would be a system where the computational potential of a 2010 tera scale system is converted downwards to a few chips and a few tens of watts. This might be the basis for a tera scale workstation of PC, or a tera scale chip set for embedded applications. The latter has the most relevance to DoD missions, and is least likely to be achieved by commercial developments, so it will be the "low end" of the Exascale systems for the rest of this report.

Much of the discussion presented above for the departmental sized system is still relevant, but with all the bandwidth now supported between a handful of chips.

If we still reference a peta scale system as a starting point, this means that the key attributes of rate, capacity, and bandwidth all decrease by a factor of 1,000, and the volume and power ratios by a factor of 1 million. This, however, may not be very precise, since the application suite for a tera-sized system that fits in say an aircraft will not be the same suite supported by a Petascale system. This is particularly true for memory, where scaling down in performance may or may not result in a different scaling down in memory capacity

2.4 Prioritizing the Attributes

If as is likely, achieving factors of 1,000X in any particular dimension are liable to be the most difficult to achieve, then looking at which systems from the above suite have the most 1,000X multipliers is liable to indicate both which system types might be the most challenging, and which attributes are liable to be the most relevant to the greatest number of systems.

Table 2.2 summarizes the metrics from the prior chart when they are ratioed for "per watt" and "per unit volume." At the bottom and on the left edge the number of attribute entries that are "1,000X" are recorded. From these, it is clear that the capability, departmental, and embedded systems seem to have the most concurrent challenges, and that the computational rate and bandwidth "per watt" and "per volume" are uniformly important. The only reason why memory capacity doesn't rise to the same level as these other two is that we do not at this time fully understand the scaling rules needed to raise problems up to the exa level, although the analysis of Chapter 5 indicates that significantly more than is likely in our strawmen of Chapter 7 is likely.

Regardless of this analysis, however, the real question is how hard is it to achieve any of these ratios, for any of the systems, in the technologies that may be available.

	Attributes						# of 1000X Ratios
	Comp. Rate	**Memory Cap.**	**BW**	**Comp. Rate**	**Memory Cap.**	**BW**	
Exa Scale Data Center Capacity System relative to 2010 Peta Capacity System							
Single Job Speedup	1000X	Same	1000X	1000X	Same	1000X	4
Job Replication	1000X	up to 1000X	1000X	1000X	up to 1000X	1000X	6
Exa Scale Data Center Capability System relative to 2010 Peta Capability System							
Current in Real-Time	1000X	Same	1000X	1000X	Same	1000X	4
Scaled Current Apps	up to 1000X	up to 1000X	up to 1000X	up to 1000X	up to 1000X	up to 1000X	6
New Apps	up to 1000X	up to 1000X	up to 1000X	up to 1000X	up to 1000X	up to 1000X	6
Peta Scale Department System relative to 2010 Peta HPC System							
	1000X	1000X	1000X	1000X	1000X	1000X	6
Tera Scale Embedded System relative to 2010 Peta HPC System							
	1000X	1000X	1000X	1000X	1000X	1000X	6
# of 1000X ratios	7	5	7	7	5	7	

Table 2.2: Attributes of Exascale class systems.

Chapter 3

Background

This chapter provides a bit of background about trends in leading edge computational systems that are relevant to this work.

3.1 Prehistory

In the early 1990s, it became obvious that there was both the need for, and the potential to, achieve a trillion (10^{12}) floating point operations per second against problems that could be composed by dense linear algebra. This reached reality in 1996 when the ASCI Red machine passed 1 **teraflops**[1] (a teraflop per second) in both peak[2] and sustained[3] performance.

In early 1994, more than two years before the achievement of a teraflops, an effort was started[129] to define the characteristics of systems (the device technologies, architectures, and support software) that would be needed to achieve one thousand times a teraflops, namely a **petaflops** - a million billion floating point operations per second, and whether or not there were real and significant applications that could take advantage of such systems. This effort triggered multiple government-sponsored research efforts such as the **HTMT** (**Hybrid Technology Multi-Threaded**)[47] and **HPCS** (**High Productivity Computing Systems**)[4] programs, which in turn helped lead to peak petaflops systems within a year of now, and sustained petaflops-level systems by 2010.

At the same time as this explosion at the high end of "classical" scientific computing occurred, an equally compelling explosion has occurred for applications with little to do with such floating point intensive computations, but where supporting them at speed requires computing resources rivaling those of any supercomputer. Internet commerce has led to massive **server systems** with thousands of processors, performing highly concurrent web serving, order processing, inventory control, data base processing, and data mining. The explosion of digital cameras and cell phones with rich multi-media capabilities have led to growing on-line, and heavily linked, object data bases with the growing need to perform real-time multimedia storage, searching, and categorization. Intelligence applications after 9/11 have developed to anticipate the actions of terrorists and rogue states.

Thus in a real sense the need for advanced computing has grown significantly beyond the need for just flops, and to recognize that fact, we will use the terms **gigascale**, **Terascale**, **Petascale**, etc to reflect such systems.

---

[1] In this report we will define a **gigaflop**, **teraflop**, etc to represent a billion, trillion, etc floating point operations. Adding an "s" to the end of such terms will denote a "per second" performance metric

[2] **Peak performance** is a measure of the maximum concurrency in terms of the maximum number of relevant operations that the hardware could sustain in any conditions in a second

[3] **Sustained performance** is a measure of the number of relevant operations that a real application can execute on the hardware per second

In addition to this drive for high-end computing, equally compelling cases have been building for continually increasing the computational capabilities of "smaller" systems. Most of the non "top 10" of the "Top 500" systems are smaller than world-class systems but provide for an increasing base of users the performance offered by prior years supercomputing. The development of **server blades** has led to very scalable systems built out of commercial microprocessor technology. In a real sense, updates to the prior generation of Terascale computing has led to the Petascale computing of the high end, which in turn is becoming the technology driver for the next generation of broadly deployed Terascale computing.

In the embedded arena, the relentless push for more functionality such as multi-media, video, and GPS processing, in smaller packages (both for personal, industrial and scientific uses) has become a linchpin of our modern economy and defense establishment, with a growing need to allow such packages to function in a growing suite of difficult environmental conditions. Thus just as we have moved to gigascale laptops, PDAs, etc with today's technologies, the trend is clearly to migrate into Terascale performance in similar sizes.

3.2 Trends

At the same time the above events were happening, a series of trends have emerged that has cast some doubt as to the ability of "technology development as usual" to provide the kind of leap that drove Terascale and Petascale computing. These include:

- Moore's Law, if (correctly) interpreted as doubling the number of devices per unit of area on a chip every 18-24 months, will continue to do so.
- Moore's Law, if (incorrectly) interpreted as of doubling performance every 24 months, has hit a power wall, where clock rates have been essentially flat since the early 2000s.
- Our ability to automatically extract from serial programs more operations out of normal programs to perform in parallel in the hardware has plateaued.
- Our ability to hide the growing memory latency wall by increasingly large and complex cache hierarchies has hit limits in terms of its effectiveness on real applications.
- Memory chip designs are now being driven by the need to support large amounts of very cheap nonvolatile memory, ideally with medium high bandwidth but where read and write latencies are almost irrelevant.
- Our ability to devise devices with finer and finer feature sizes is being limited by lithography, which seems to be stuck for the foreseeable future with what can be formed from 193 nm deep ultraviolet light sources.
- The cost of designing new microprocessors in leading edge technologies has skyrocketed to the point where only a very few large industrial firms can afford to design new chips.
- Funding for new research in computer architecture to look for alternatives to these trends has declined to the point where senior leaders in the field have raised alarms[100], and major research communities organized studies outlining the problem and why it should be addressed[12]. The traditional model of single-domain research activities where hardware and software techniques are explored in isolation will not address the current challenges.

3.3 Overall Observations

While it took something approaching 16 years to get from the first serious discussions of Petascale computers (1994) to their emergence as deployable systems (expected in 2010), the technologies, architectures, and programming models were all remarkably foreseen early on[129], and in retrospect bear a real family resemblance to the early Terascale machines. Looking forward to another similar three-order jump in computational capabilities (termed here as **Exascale** systems), several observations emerge:

- There is a continuing need for critical applications to run at much higher rates of performance than will be possible even with the Petascale machines.
- These applications are evolving into more complex entities than those from the Terascale era.
- This need for increasing computing capability is more than just as absolute numbers, but also for reducing the size of the systems that perform today's levels of computing into smaller and smaller packages.
- Technology is hitting walls for which there is no visible viable solutions, and commercial pressures are not driving vendors in directions that will provide the advanced capabilities needed for the next level of performance.
- There is a dearth of advanced computer architecture research that can leverage either existing or emerging technologies in ways that can provide another explosive growth in performance.
- Our programming methodologies have evolved to the point where with some heroics scientific codes can be run on machines with tens' of thousands of processors, but our ability to scale up applications to the millions of processors, or even port conventional "personal" codes to a few dozen cores (as will be available soon in everyone's laptops) is almost non-existent.

Given this, and given the clear need for deploying systems with such capabilities, it seems clear that without explicit direction and support, even another 16 years of waiting cannot guarantee the emergence of Exascale systems.

3.4 This Study

The project that led to this report was started by DARPA in the spring of 2007, with the general objective of assisting the U.S. government in exploring the issues, technical limitations, key concepts, and potential enabling solutions for deploying Exascale computing systems, and ensuring that such technologies are mature enough and in place by 2015, not a decade later.

A contract was let to the Georgia Institute of Technology to manage a study group that would cover the technological spectrum. A study chair, Dr. Peter Kogge of the University of Notre Dame, was appointed, and a group of nationally recognized experts assembled (Appendix A). The study group's charter was to explore the issues, technical limitations, key concepts, and potential solutions to enable Exascale computing by addressing the needed system technologies, architectures, and methodologies.

The key outcome of the study was to develop an open report (this document) that identifies the key roadblocks, challenges, and technology developments that must be tackled by 2010 to support potential 2015 deployment. A subsidiary goal was to generate ideas that will permit use of similar technologies for Terascale embedded platforms in the same time frame.

Over the summer and fall of 2007, almost a dozen meetings were held at various venues, with a large number of additional experts invited in for discussions on specific topics (Appendix B).

3.5 Target Timeframes and Tipping Points

A target time for the development of Exascale technologies and systems was chosen as 2015. This was chosen as an aggressive goal because it represented about 1/2 of the time required to get to Petascale from the first workshop in 1994, and mirrors the approximate time period required to formulate and execute the HPCS program from start to finish.

This target of 2015 included both technologies and systems as goals. Achieving the former would mean that all technologies would be in place by 2015 to develop and deploy any class of Exascale system. Achieving the latter means that systems of some, but not necessarily all, Exascale classes (perhaps embedded or departmental) with Exascale levels of capabilities (as defined here) would be possible.

For such 2015 deployments to be commercially feasible, the underlying technologies must have been available long enough in advance for design and development activities to utilize them. Thus, a secondary time target was placed at 2013 to 2014 for base technologies out of which product components could be designed.

Finally, in looking back at the Petascale development, even though there was a significant effort invested in the early years, it wasn't until the early 2000s' that it became clear to commercial vendors that Petascale systems were in fact going to be commercially feasible, even if it wasn't obvious what the architecture or exact designs would be. We refer to such a time as the **tipping point** in terms of technology development, and thus set as a study goal the task of defining what emerging technologies ought be pushed so that an equivalent tipping point in terms of their potential usefulness for Exascale systems would be reached by 2010. At this point, it is hoped that industry can see clearly the value of such technologies, and can begin to pencil in their use in systems by 2015.

3.6 Companion Studies

Concurrent with this study were two other activities with significant synergism. First is a DoE-sponsored study entitled "Simulation and Modeling at the Exascale for Energy, Ecological Sustainability and Global Security (E3SGS)"[4], whose goal is to set the stage for supercomputer-class Exascale applications that can attack global challenges through modeling and simulation. A series of town-hall meetings was held in order to develop a more complete understanding of the properties of such applications and the algorithms that underly them.

These meetings focused primarily on Exascale applications and possible algorithms needed for their implementation, and not on underlying system architecture or technology.

Second is a workshop held in Oct. 2007 on "Frontiers of Extreme Computing," which represented the third in a series of workshops[5] on pushing computation towards some sort of ultimate limit of a **zettaflops** (10^{21}) flops per second, and the kinds of technologies that might conceivably be necessary to get there. While this series of workshops have included discussions of not just applications, but also device technologies and architectures, the time frame is "at the limits" and not one as "near term" as 2015.

[4]http://hpcrd.lbl.gov/E3SGS/main.html
[5]http://www.zettaflops.org/

3.7 Prior Relevant Studies

For reference, during the time from 1994 through the present, a long series of studies have continued to build consensus into the importance and justification for advanced computing for all classes of systems from embedded to supercomputing. The sections below attempt to summarize the key findings and recommendations that are most relevant to this study. Since the focus of this study is mostly technical, findings and recommendations discussed below will be largely those with a technical bent: investment and business environment-relevant factors are left to the original reports and largely not covered here.

3.7.1 1999 PITAC Report to the President

The February 1999 *PITAC Report to the President - Information Technology Research: Investing in Our Future*[75] was charged with developing "future directions for Federal support of research and development for information technology."

The major findings were that Federal information technology R&D investment was inadequate and too heavily focused on near-term problems. In particular relevance to this report, the study found that high-end computing is essential to science and engineering research, it is an enabling element of the United States national security program, that new applications of high-end computing are ripe for exploration, that US suppliers of high-end systems suffer from difficult market pressures, and that innovations are required in high-end systems and application-development software, algorithms, programming methods, component technologies, and computer architecture.

Some major relevant recommendations were to create a strategic initiative in long-term information technology R&D, to encourage research that is visionary and high-risk, and to fund research into innovative computing technologies and architectures.

3.7.2 2000 DSB Report on DoD Supercomputing Needs

The October 2000 Defense Science Board *Task Force on DoD Supercomputing Needs*[17] was charged with examine changes in supercomputing technology and investigate alternative supercomputing technologies in the areas of distributed networks and multi-processor machines.

The Task Force concluded that there is a significant need for high performance computers that provide extremely fast access to extremely large global memories. Such computers support a crucial national cryptanalysis capability. To be of most use to the affected research community, these supercomputers also must be easy to program.

The key recommendation for the long-term was to invest in research on critical technologies for the long term. Areas such as single-processor architecture and semiconductor technology that are adequately addressed by industry should *not* be the focus of such a program.Areas that should be invested in include architecture of high-performance computer systems, memory and I/O systems, high-bandwidth interconnection technology, system software for high-performance computers, and application software and programming methods for high-performance computers.

3.7.3 2001 Survey of National Security HPC Architectural Requirements

The June 2001 *Survey and Analysis of the National Security High Performance Computing Architectural Requirements*[46] was charged with determining if then-current high performance computers that use commodity microprocessors were adequate for national security applications, and also was there a critical need for traditional vector supercomputers.

One key finding was that based on interviews conducted, commodity PCs were providing useful capability in all of 10 DoD-relevant application areas surveyed except for the cryptanalysis area. Another was that while most big applications had scaled well onto commodity PC HPC systems with MPI, several had not, especially those that must access global memory in an irregular and unpredictable fashion.

A summary of the recommendations was to encourage significant research into the use of OpenMP on shared-memory systems, and establish a multifaceted R&D program to improve the productivity of high performance computing for national security applications.

3.7.4 2001 DoD R&D Agenda For High Productivity Computing Systems

The June 2001 *White Paper DoD Research and Development Agenda For High Productivity Computing Systems*[40] was charged with outlining an R&D plan for the HPCS program to revitalize high-end computer industry, providing options for high-end computing systems for the national security community and, developing improved software tools for a wide range of computer architectures. The key findings were that:

- The increasing imbalance among processor speed, communications performance, power consumption, and heat removal results in high-end systems that are chronically inefficient for large-scale applications.

- There exists a critical need for improved software tools, standards, and methodologies for effective utilization of multiprocessor computers. As multi-processor systems become pervasive throughout the DoD, such tools will reduce software development and maintenance - a major cost driver for many Defense system acquisitions.

- Near-elimination of R&D funding for high-end hardware architectures has resulted in a dramatic decrease in academic interest, new ideas, and people required to build the next generation high-end computing systems.

The recommendations were that the attention of academia and industry needed to be drawn to high bandwidth/low latency hierarchical memory systems using advanced technologies, develop highly scalable systems that balance the performance of processors, memory systems, interconnects, system software, and programming environments, and address the system brittleness and susceptibility of large complex computing systems.

3.7.5 2002 HPC for the National Security Community

The July 2002 *Report on High Performance Computing for the National Security Community*[41] was charged with supporting the Secretary of Defense in submitting a development and acquisition plan for a comprehensive, long-range, integrated, high-end computing program to Congress.

The key finding was that the mix of R&D and engineering programs lacked balance and coordination and was far below the critical mass required to sustain a robust technology/industrial base in high-end supercomputing, with requirements identified as critical by the national security community, especially improved memory subsystem performance and more productive programming environments, not being addressed. Another relevant finding was that then current communication interfaces for moving data on and off chips throttled performance.

The key recommendation was to restore the level and range of effort for applied research in fundamental HEC concepts, and apply it nearly evenly across seven general research areas: systems architectures; memory subsystems; parallel languages and programmer tools; packaging/power/thermal management; interconnects and switches; storage and input/output; and novel computational technologies (exclusive of quantum computing).

3.7.6 2003 Jason Study on Requirements for ASCI

The October 2003 Jason study on *Requirements for ASCI*[126] was charged with identifying the distinct requirements of NNSA's stockpile stewardship program in relation to the hardware procurement strategy of the ASCI program.

The two most relevant findings were that a factor of two oversubscription in ASCI Capacity systems was projected to potentially worsen in the foreseeable future, and that future calculations were estimated to take 125X in memory and 500X computations per zone to handle opacity calculations, 40X memory for reactive flow kinetics, and 4X in memory and performance for Sn transport.

Besides increasing ASCI's capability machines, this report also recommended continuing and expanding investments in computational science investigations directed toward improving the delivered performance of algorithms relevant to ASCI.

3.7.7 2003 Roadmap for the Revitalization of High-End Computing

The June 2003 *Workshop on The Roadmap for the Revitalization of High-End Computing*[119] (**HECRTF**) was charged with developing a five-year plan to guide future federal investments in high-end computing.

The overall finding was that short-term strategies and one-time crash programs were unlikely to develop the technology pipelines and new approaches required to realize the Petascale computing systems needed by a range of scientific, defense, and national security applications.

Specific findings and recommendations covered several areas. In enabling technologies, key areas needing work included the management of power and improvements in interconnection performance, the bandwidth and latency among chips, boards, and chassis, new device technologies and 3D integration and packaging concepts, a long-term research agenda in superconducting technologies, spintronics, photonic switching, and molecular electronics, and system demonstrations of new software approaches.

Specific findings and recommendations in architecture were split between **COTS**-based (Custom Off the Shelf) and **Custom**. For the former the emphasis was on exploiting scarce memory bandwidth by new computational structures, and increasing memory bandwidth across the board. For the latter, expected performance advantages of custom features were forecast of between 10X and, programmability advantage of twofold to fourfold. Beyond a relatively near-term period, the report concluded that continued growth in system performance will be derived primarily through brute force scale, advances in custom computer architecture, and incorporation of exotic technologies, and that a steady stream of prototypes were needed to determine viability.

In terms of runtime and operating systems, the study found that the then-research community was almost entirely focused on delivering capability via commodity-leveraging clusters, that for HEC systems the two should merge and need to incorporate much more dynamic performance feedback, and that the lack of large-scale testbeds was limiting such research. In addition, the study concluded that as system sizes increased to 100,000 nodes and beyond, novel scalable and high-performance solutions would be needed to manage faults and maintain both application and system operation.

In terms of programming environments and tools, the study concluded that the most pressing scientific challenges will require application solutions that are multidisciplinary and multiscale, requiring an interdisciplinary team of scientists and software specialists to design, manage, and maintain them, with a dramatic increase in investment to improve the quality, availability, and usability of the software tools that are used throughout an application's life cycle.

3.7.8 2004 Getting Up to Speed: The Future of Supercomputing

The November 2004 National Academy report *Getting Up to Speed The Future of Supercomputing*[54] summarizes an extensive two-year study whose charge was to examine the characteristics of relevant systems and architecture research in government, industry, and academia, identify key elements of context, examine the changing nature of problems demanding supercomputing and the implications for systems design, and outline the role of national security in the supercomputer market and the long-term federal interest in supercomputing.

The key findings were that the peak performance of supercomputers has increased rapidly in the last decades, but equivalent growth in sustained performance and productivity had lagged. Also, the applications described in Section 3.7.3 were still relevant, but that about a dozen additional application areas were identified, with performance needs often 1000X or more would be needed - just over the next five years. Finally, while commodity clusters satisfied the needs of many supercomputer users, important applications needed better main memory bandwidth and latency hiding that are available only in custom supercomputers, and most users would benefit from the simpler programming model that can be supported well on custom systems.

The key recommendations all dealt with the government's necessary role in fostering the development of relevant new technologies, and in ensuring that there are multiple strong domestic suppliers of both hardware and software.

3.7.9 2005 Revitalizing Computer Architecture Research

A 2005 CRA Conference on *Grand Challenges in Computer Architecture*[72] was charged with determining how changes in technology below 65 nm are likely to change computer architectures, and to identify what problems are likely to become most challenging, and what avenues of computer architecture research are likely to be of most value. To quote the report: "Separating computing and communication is no longer useful; differentiating between embedded and mainstream computing is no longer meaningful. Extreme mobility and highly compact form factors will likely dominate. A distributed peer-to-peer paradigm may replace the client-server model. New applications for recognition, mining, synthesis, and entertainment could be the dominant workloads."

The findings of the workshop was that there were four specific challenges whose solutions would potentially have a huge impact on computing in 2020:

1. A "featherweight supercomputer" that could provide 1 teraops per watt in an aggressive 3D package suitable for a wide range of systems from embedded to energy efficient data centers.

2. "Popular parallel programming models" that will allow significant numbers of programmers to work with multi-core and manycore systems.

3. "Systems that you can count on" that provide self-healing and trustworthy hardware and software.

4. "New models of computation" that are not von Neumann in nature, and may better leverage the properties of emerging technologies, especially non-silicon, in better ways for new and emerging classes of applications.

3.7.10 2006 DSB Task Force on Defense Critical Technologies

The March 2006 the *Report on Joint U.S. Defense Science Board and UK Defence Scientific Advisory Council Task Force on Defense Critical Technologies*[18] was charged with examine five transformational technology areas that are critical to the defense needs of the US and UK, including Advanced Command Environments, Persistent Surveillance, Power sources for small distributed sensor networks, High Performance Computing, and Defence Critical Electronic Components.

They key finding on the HPC side were that there are a number of applications that cannot be solved with sufficient speed or precision today, that the high performance needs of national security will not be satisfied by systems designed for broader commercial market, and that two new memory-intensive applications are becoming especially important: knowledge discovery and integration and image and video processing.

The key recommendations on the HPC side were to make HPCS a recurring program, and invest in technologies especially for knowledge discovery including new very large memory-centric systems, programming tools, system software, improved knowledge discovery algorithms that can run unattended, new inference engines, support for rapid, high productivity programming, appropriate metrics, and open test beds that permit research community to explore and evaluate without revealing national securing information.

The recommendations on knowledge discovery and video processing were buttressed by the findings and recommendations on **Persistent Surveillance**. In particular a recommendation was to establish a US persistent surveillance effort.

3.7.11 2006 The Landscape of Parallel Computing Research

The report *The Landscape of Parallel Computing Research: A View From Berkeley*[11] summarized a two-year effort by researchers at the University of California at Berkeley to discuss the effects of recent emergence of multi-core microprocessors on highly parallel computing, from applications and programming models to hardware and evaluation.

Their major finding was that multi-core was unlikely to be the ideal answer to achieving enhanced performance, and that a new solution for parallel hardware and software is desperately needed. More detailed findings included that

- It is now memory and power that are the major walls.
- As chips drop below 65 nm feature sizes, they will have higher soft and hard error rates that must be accounted for in system design.
- Both instruction level parallelism and clock rates have reached points of diminishing returns, and conventional uniprocessors will no longer improve in performance by 2X every 18 months.
- Increasing explicit parallelism will be the primary method of improving processor performance.
- While embedded and server computing have historically evolved along separate paths, increasingly parallel hardware brings them together both architecturally and in terms of programming models.

A key output of the report was the identification of 13 benchmark dwarves that together can delineate application requirements in a way that allows insight into hardware requirements. In terms of hardware projections, the report suggests that technology will allow upwards of a 1000 simple and perhaps heterogeneous cores on a die (**manycore**), but that separating DRAM from CPUs as is done today needs to be seriously revisited. Further, interconnection networks will become of increasing importance, both on and off chip, with coherency and synchronization of growing concern. In addition, more attention must be payed to both dependability and performance and power monitoring to provide for **autotuners**, which are software systems that automatically adapt to performance characteristics of hardware, often by searching over a large space of optimized versions. New models of programming will also be needed for such systems.

Chapter 4

Computing as We Know It

This chapter discuss the structure and inter-relationship of computing systems today, including architecture, programming models, and resulting properties of applications. It also includes some historical data on microprocessor chips, on the leading supercomputer systems in the world over the last 20 years, and on web servers which make up many of the departmental sized systems of today.

4.1 Today's Architectures and Execution Models

This section should overview today's deep memory hierarchy-based system architectures using hot multi-core processing chips, dense DRAM main memory, and spinning, disk-based, secondary memory.

4.1.1 Today's Microarchitectural Trends

Contemporary microprocessor architectures are undergoing a transition to simplicity and parallelism that is driven by three trends. First, the instruction-level parallelism and deep pipelining (resulting in high clock rates) that accounted for much of the growth of single-processor computing performance in the 1990s has been *mined out*. Today, the only way to increase performance beyond that achieved by improved device speed is to use explicit parallelism.

Second, the constant field scaling that has been used to give "cubic"[1] energy scaling for several decades has come to an end because threshold voltage cannot be reduced further without prohibitive subthreshold leakage current. As a result, power is *the* scarce resource in the design of a modern processor, and many aggressive superscalar techniques that burn considerable power with only modest returns on performance have been abandoned. Newer processors are in many cases simpler than their predecessors to give better performance per unit power.

Finally, the increase in main memory latency and decrease in main memory bandwidth relative to processor cycle time and execution rate continues. This trend makes memory bandwidth and latency the performance-limiting factor for many applications, and often results in sustained application performance that is only a few percent of peak performance. Many mainstream processors now adopt aggressive latency hiding techniques, such as out-of-order execution and explicit multi-threading, to try to tolerate the slow path to memory. This, however, has an element of self-defeatism, since they often results in worse than linear increase in the complexity of some ba-

[1] The energy per device shrinking as the 3rd power of the reduction in the device's basic feature size

sic structure (such as comparators associated with a load-store buffer)to contributes a worse than linear growth in power for a less than linear increase in performance.

4.1.1.1 Conventional Microprocessors

Conventional high-end microprocessors aim for high single-thread performance using speculative superscalar designs. For example, the AMD K8 Core microarchitecture includes a 12 stage pipeline that can execute up to 3 regular instructions per cycle. The complex out-of-order instruction issue, retirement logic, and consumes considerable power and die area; only about 10% of the core die area (not including L2 caches) is devoted to arithmetic units.

At the other end of the spectrum, Sun's Niagara 2 chip[110] includes 8 dual-issue cores, each of which supports up to 8-way multi-threading. The cores are simpler, with in-order execution, sacrificing single-thread performance for the benefit of greater thread-level parallelism. These simpler processors are more efficient, but still consume considerable area and energy on instruction supply, data supply, and overhead functions.

To amortize the high overheads of modern processors over more operations, many processors include a *short vector* instruction set extension such as Intel's SSE. These extensions allow two or four floating point operations to be issued per cycle drawing their operands from a wide (128b) register file. Emerging processors such as Intel's Larrabee take this direction one step further by adding a wider vector processor.

4.1.1.2 Graphics Processors

Performance in contemporary **graphics processing units** (**GPUs**) has increased much more rapidly than conventional processors, in part because of the ease with which these processors can exploit parallelism in their main application area. Modern GPUs incorporate an array of programmable processors to support the programmable shaders found in graphics APIs. For example, the Nvidia GForce 8800 includes an array of 128 processors each of which can execute one single-precision floating-point operation each cycle.

The programmability and high performance and efficiency of modern GPUs has made them an attractive target for scientific and other non-graphics applications. Programming systems such as Brook-GPU [22] and Nvidia's CUDA [111] have evolved to support general purpose applications on these platforms. Emerging products such as AMD's *fusion* processor are expected to integrate a GPU with a conventional processor to support general-purpose applications.

4.1.1.3 Multi-core Microprocessors

In the early 2000s' the confluence of limitations on per chip power dissipation and the flattening of our ability to trade more transistors for higher ILP led to a stagnation in single-core single-thread performance, and a switch in chip level microarchitectures to die with multiple separate processing cores on them. This switch has occurred across the board from general purpose microprocessors to DSPs, GPUs, routers, and game processors, and has taken on the generic names of **multi-core**, **manycore**, or **chip-level multi-processing** (**CMP**).

The rise of multiple cores on a single die has also introduced a new factor in a die's microarchitecture: the interconnect fabric among the cores and between the cores and any external memory. There are currently at least three different classes[86] of such die-level interconnect patterns, Figure 4.1:

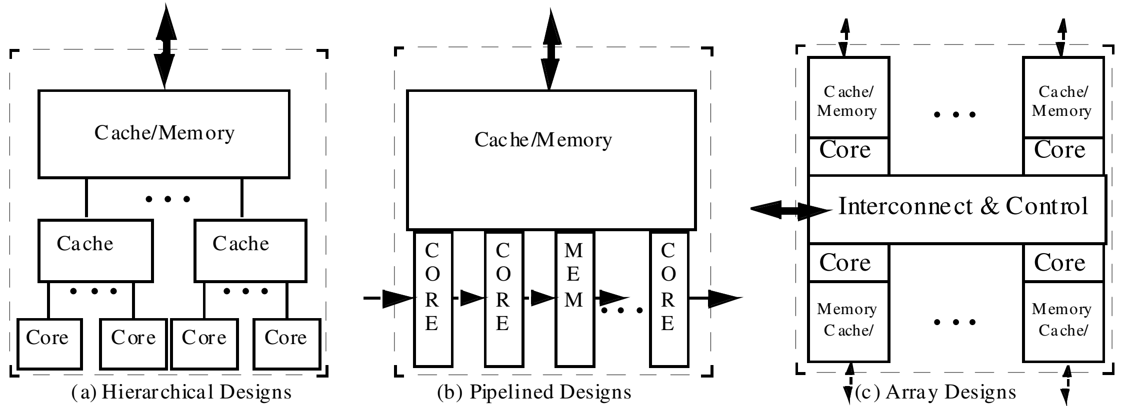


Figure 4.1: Three classes of multi-core die microarchitectures.

- **Hierarchical**: where cores share multi-level caches in a tree-like fashion, with one or more interfaces to external memory at the top of the tree. Most general purpose microprocessors fall into this category.
- **Pipelined**: where off-chip inputs enter one core, which then may communicate with the "next" core, and so on in a pipelined fashion, and where the "last" core then sends processed data off-chip. On-chip memory may be either as "stage" in its own right, associated with a unique core, or part of a memory hierarchy as above. Many router and graphics processors follow this microarchitecture, such as the Xelerator X10q[25], which contains 200 separate cores, all arranged in a logically linear pipeline.
- **Array**: where each core has at least some local memory or cache, may or may not share a common memory, and may communicate with other cores either with some sort of X-Y routing or through a global crossbar. Examples of designs for 2 or 3D X-Y interconnect include both EXECUBE[85][144] and Intel's **Teraflops Research Chip**[149]; examples of crossbar interconnect include Sun's Niagara 2 chip[110] and IBM's **Cyclops**[8]; examples of reconfigurable tiles include **RAW**[146].

4.1.2 Today's Memory Systems

Today's general purpose computer systems incorporate one or more **chip multiprocessors** (**CMPs**) along with external DRAM memory and peripherals. Each processor chip typically falls into the hierarchical arrangement as discussed above, and combines a number of processors (currently ranging from 2-24) along with the lower levels of the memory hierarchy. Each processor typically has its own **level-1** (**L1**) data and instruction caches. Some CMPs also have private L2 caches, while others share a banked L2 cache (interleaved by cache block) across the processors on the chip. Some CMPs, such as the Itanium-2^{TM}, have large (24MB) on-chip shared level-3 caches. Some manufacturers (Sun and Azul) provide some support for transactional memory functionality.

The on-chip processors and L2 cache banks are connected by an on-chip interconnection network. In the small configurations found today, this network is typically either a bus, a crossbar, or a ring. In the larger configurations expected in the near future, more sophisticated on-chip networks will be required.

The main memory system is composed of conventional (DDR2) DRAM chips packaged on small daughter cards often called variants of **DIMMs** (**Dual Inline Memory Modules**). A

small number of data bits leave the DIMM from each chip, usually 4, 8 or 9 bits. In most cases, the data outputs of these chips connect directly to the processor chip, while in other cases they connect to a **north bridge** chip that is connected to the processor via its **front-side bus**. In both cases a **memory controller** is present either on the CPU, the north bridge, or as a separate chip, and whose purpose is to manage the timing of accesses to the DRAM chips for both accesses and for the refresh operations that are needed to keep their contents valid.

In today's cache-based memory systems, the processor reads and writes in units of words, but all transfers above the level of the L1 caches take place in units of cache lines which range from 32B to 128B. When there is a miss in the L1 cache, an entire line is transferred from L2 or from main memory. Similarly when a dirty line is evicted from a cache, the entire line is written back. Particularly with the larger granularity cache lines, such a line oriented memory system reduces the efficiency of random fine-grained accesses to memory.

Pin bandwidth of modern DRAMs are about 1Gb/s using the **Stub Series Terminated Logic** (**SSTL**) signaling standard. Modern commodity processors use interfaces ranging from 8B to 32B wide to provide from 8 to 32GB/s of main memory bandwidth. This per pin rate is relatively low because of the technology used on the DRAMs, and because of power consumption in the interface circuits. In addition, typical DIMM4 may be paralleled up to 4 ways to increase the memory capacity without expanding the pin count on the microprocessor. This increases the capacitance on the data pins, thus making higher rate signalling more difficult.

One new technique to increase the data rate from a multi-card memory system is called **Fully Buffered DIMM** (**FB-DIMM**), and uses a higher speed serialized point-to-point data and control signalling protocol from the microprocessor side to an **Advanced Memory Buffer** (**AMB**) chip on each FB-DIMM memory card. This AMB then talks to the on-card DRAM parts to convert from their parallel data to the protocol's serialized format. The higher speed of the protocol is possible because each link is uni-directional and point-to-point, and multiple FB-DIMM cards are arranged in a daisy-chain fashion with separate links going out and coming back. Additional bandwidth is gained because this arrangement allows accesses to be pipelined both in terms of outgoing commands and returning data.

A typical contemporary microprocessor has latencies of 1 cycle, 20 cycles, and 200 cycles to the L1 cache, L2 cache, and external DRAM respectively. Memory systems such as FB-DIMM actually increase such latencies substantially because of the pipelined nature of the card-to-card protocol. These latencies make the penalty for a cache miss very high and motivate the use of latency-hiding mechanisms.

4.1.3 Unconventional Architectures

Overcoming the memory wall has been a major emphasis of alternative architectures that have begun to crop up, especially with the emergence of multi-core architectures that place extra pressure on the memory system. Multi-threading has become more acceptable, with dual threads present in many main-line microprocessors, and some chips such as Sun's Niagara 2™chip[110] support even more. Cray's **MTA™**[132] and **XMT™**[44] processors were designed from the outset as multi-threaded machines, with in excess of 100 threads per core supported.

Stream processors such as the IBM Cell[57] and SPI Storm-1™ are an emerging class of architecture that has arisen to address the limits of memory latency and bandwidth. These processors have explicitly managed on-chip memories in place of (or in addition to) the conventional L2 cache. Managing the contents of these memories explicitly (vs. reactively as with a cache) enables a number of aggressive compiler optimizations to hide latency and improve the utilization of scarce bandwidth. Explicit management of the memory hierarchy, however, places additional burdens on

the application programmer and compiler writer.

Processing-In-Memory (**PIM**) chips reduce the latency of a memory access, and increase the effective bandwidth by using more of the bits that are read from a dense, usually DRAM, memory array at each access by placing one or more processors on the chip with the memory. A variety of different processor architectures and interconnect patterns have been used, ranging from arrangements of more or less conventional cores on **EXECUBE**[85], **Cyclops**[8] and Intel's **Teraflops Research Chip**[149], "memory chips" with embedded processing such as **DIVA**[60] and **YUKON**[83], and **VIRAM**[88] - where multiple vector units are integrated onto a DRAM.

In addition, a new class of **reconfigurable multi-core architectures** are entering evaluation where a combination of large and small grained concurrency is exploited by tiles of cores that may be reconfigured on an algorithm by algorithm basis. These include **TRIPS**[123] and **RAW**[146].

4.1.4 Data Center/Supercomputing Systems

While no clear definition exists, **supercomputing systems** are commonly defined as the most computationally powerful machines available at any time. Since the mid 1990s, all supercomputers have been designed to be highly scalable, and thus a given system design may be scaled down or up over a broad range. A typical supercomputer today may sell in configurations costing O($1M) to O($100M).

4.1.4.1 Data Center Architectures

Supercomputing systems fall into two broad classes: clusters of conventional compute nodes, and custom systems. Custom systems are machines where the individual compute nodes are designed specifically for use in a large scientific computing system. Motivated by demanding scientific applications, these nodes typically have much higher memory bandwidth than conventional processors, very high peak and sustained arithmetic rates, and some form of latency hiding mechanism — often vectors as in the Earth Simulator. These custom processing nodes are connected by an interconnection network that provides access to a global shared memory across the machine. Global memory bandwidth is usually a substantial fraction of local memory bandwidth (e.g. 10%) and global memory latency is usually only a small multiple of local memory latency (e.g. 4x).

Clusters of conventional nodes are, as the name suggests, conventional compute nodes connected by an interconnection network to form a cluster. While most such nodes may use "mass-market" chips, there are significant examples of specially designed chips that leverage prior processor designs (such as the Blue Gene/L systems) Further, most such compute nodes are themselves SMPs, either as collections of microprocessors, or increasingly multi-core processors, or both. These in fact make up the majority of systems on the Top 500.

The individual nodes typically have separate address spaces and have cache-based memory systems with limited DRAM bandwidth and large cache-line granularity. The nodes are connected together by an interconnection network that can range from conventional 1G or 10G ethernet to a cluster-specific interconnect such as **Infiniband**[2], **Myrinet**[3], **Quadrics**[4], or even custom high bandwidth interconnect such as the SeaStar[21] used in the Red Storm and derivatives in the Cray XT line. Global bandwidth is usually a small fraction of local bandwidth and latencies are often several microseconds (with much of the latency in software protocol overhead).

[2] Infiniband standard can be found at http://www.infinibandta.org/
[3] Myrinet standards can be found at http://www.myri.com/open-specs/
[4] http://www.quadrics.com

Conversion Step	Efficiency	Delivered	Dissipated	Delivered	Dissipated
AC In		1.00W		2.06W	
Uninterruptible Power Supply (UPS)	88%	0.88W	0.12W	1.81W	0.25W
Power Distribution Unit (PDU)	93%	0.82W	0.06W	1.69W	0.13W
In Rack Power Supply Unit (PSU)	79%	0.65W	0.17W	1.33W	0.35W
On Board Voltage Regulator (VR)	75%	0.49W	0.16W	1.00W	0.33W
Target Logic		0.49W	0.49W	1.00W	1.00W

Table 4.1: Power distribution losses in a typical data center.

Another distinguishing characteristic for the interconnect is support addressing extensions that provide the abstraction of a globally addressable main memory (as in the **pGAS** model). Several interconnect protocols such as Infiniband and SeaStar provide such capabilities in the protocol, but using them effectively requires support for large address spaces that, if not present in the native microprocessor, in turn may require specialized network interface co-processors.

4.1.4.2 Data Center Power

Perhaps the single biggest impact of scale is on system power. Table 4.1 summarizes the different steps in converting wall power to power at the logic for a typical data center today as projected by an Intel study[7]. For each step there is:

- an "Efficiency" number that indicates what fraction of the power delivered at the step's input is passed to the output,
- a "Delivered" number that indicated how must of the power (in watts) is passed on to the output if the power on the line above is presented as input,
- and a "Dissipated" power (again in watts) that is lost by the process.

There are two sets of the Delivered and Dissipated numbers: one where the net input power is "1," and one where the power delivered to the logic is "1." The net efficiency of this whole process is about 48%, that is for each watt taken from the power grid, only 0.48 watts are made available to the logic, and 0.52 watts are dissipated as heat in the power conversion process.

A similar white paper by Dell[38] gives slightly better numbers for this power conversion process, on the order of about 60%, that is for each watt taken from the power grid, 0.6 watts are made available to the logic, and 0.4 watts are dissipated as heat in the power conversion process.

This same source also bounds the cost of the **HVAC** (heating, ventilating, and air conditioning) equipment needed to remove the heat dissipated by the data center at about 31% of the total data center power draw. This represents an energy tax of about 52% on every watt dissipated in either the logic itself or in the power conditioning steps described above.

Since the focus on this report is on analyzing the power dissipation in a potential Exascale data center, these numbers roll up to an estimate that for every watt dissipated by the electronics, somewhere between 0.7 and 1.06 watts are lost to the power conditioning, and between 0.88 and 1.07 watts are lost to cooling. Thus the ratio between the power that may be dissipated in processing logic and the wattage that must be pulled from the grid is a ratio of between 2.58 and 3.13 to 1.

As a reference point, Microsoft is reported[5] to be building a data center near Chicago with an initial power source of up to 40 MW. Using the above numbers, 40 MW would be enough for about 13 to 15 MW of computing - about 2/3 of the Exascale data center goal number.

4.1.4.2.1 Mitigation Mitigation strategies are often employed to help overcome the various losses in the electricity delivery system. For example:

- UPS: At these power levels, many data centers choose to forgo UPS protection for the compute nodes and only use the UPS to enable graceful shutdown of the disk subsystem (a small fraction of the overall computer power dissipation). In the case where the UPS is used for the entire data center, DC power is an increasingly studied option for reducing losses. The primary source of losses within a UPS are the AC-DC rectification stage used to charge the batteries, and then the DC-to-AC inversion stage used to pull power back off of the batteries at the other end (which will again be rectified by the compute node power supplies). DC power cuts out an entire layer of redundant DC-AC-DC conversion. Below 600VDC, the building electrical codes and dielectric strengths of electrical conduits and UL standards are sufficient, so most of the existing AC electrical infrastructure can be used without modification. Care must be taken to control arc-flashes in the power supply connectors. In addition, the circuit design modifications required to existing server power supplies are minimal. Using DC power, one can achieve ¿95% efficiency for the UPS.

- PDUs: Using high-voltage (480VAC 3-phase) distribution within the data center can reduce a layer of conversion losses from PDUs in the data center and reduce distribution losses due to the resistivity of the copper wiring (or allow savings in electrical cable costs via smaller diameter power conduits).

- PSUs: The Intel and Dell studies highlight the power efficiency typically derived from low-end 1U commodity server power supplies (pizza-box servers). Using one large, and well-engineered power supply at the base of the rack and distributing DC to the components, power conversion efficiencies exceeding 92% can be achieved. Power supplies achieve their best efficiency at 90% load. When redundant power supplies are used, or the power supplies are designed for higher power utilization than is drawn by the node, the power efficiency drops dramatically. In order to stay at the upper-end of the power conversion efficiency curve one must employ circuit-switching for redundant feeds, or N+1 redundancy (using multiple power supply sub-units) rather than using the typical 2N redundancy used in modern commodity servers. Also, the power supply should be tailored to the actual power consumption of the system components rather than over-designing to support more than one system configuration.

The overall lesson for controlling power conversion losses for data centers is to consider the data center as a whole rather than treating each component as a separate entity. The data-center is the computer and power supply distribution chain must be engineered from end-to-end starting with the front-door of the building. Using this engineering approach can reduce power-distribution losses to ¡ 10% of the overall data center power budget.

4.1.4.3 Other Data Center Challenges

Regardless of the class of system, supercomputers face several unique challenges related to scale. These include reliability, administration, packaging, cooling, system software scaling and performance scaling. A large supercomputer may contain over 100 racks, each containing over 100

[5] http://www.data centerknowledge.com/archives/2007/Nov/05/microsoft_plans_500m_illinois_data_center.html

processor chips, each connected to tens of memory chips. Reliability is thus a major concern, and component vendors targeting the much larger market of much smaller-scale machines may not be motivated to drive component reliability to where it needs to be for reliable operation at large scales.

Density is important in supercomputing systems, both because machine room floor may be at a premium, and also because higher density reduces the average length of interconnect cables, which reduces cost and increases achievable signaling rates. High density in turn creates power and cooling challenges. Current systems can dissipate several tens of kilowatts per rack, and several megawatts of total power.

Careful attention must be paid to all aspects of system administration and operation. System booting, job launch, processor scheduling, error handling, file systems, networking and miscellaneous system services must all scale to well beyond what is needed for the desktop and server markets. Removing sources of contention and system jitter are key to effective scaling. Finally, the applications themselves must be scaled to tens of thousands of threads today, and likely millions of threads within the next five to ten years.

4.1.5 Departmental Systems

Today's departmental systems fall into two main categories. There are symmetric multiprocessors (SMPs) that have one large, coherent main memory shared by a few 64-bit processors such as AMD's Opterons™, IBM's Power 6™ and z series, Intel's Xeon™and Itanium™, and Sun's UltraSparc™T-series (also known as Niagara). The main memory is usually large, as is the accompanying file system. Its not uncommon to see systems with a terabyte of memory and only a handful of CPUs. These systems are often used in mission critical applications such as on-line transaction processing. Thus they offer their users higher availability than systems based on PC components. Most importantly, they provide a graceful transition path to concurrent execution for commercial applications whose software architecture goes back earlier than the mid-1990s, and hence do not support a partitioned address space.

The second category of departmental systems is composed of clusters of PC boxes (often called **Beowulf** systems), and at first glance resemble smaller versions of the supercomputers discussed above. The main difference is that the departmental clusters generally do not include relatively expensive components like the custom high-speed interconnects found in the supercomputers. Such clusters also tend to run the Linux operating system, and are generally used to host a job mix consisting of large ensembles of independent jobs that each use only a small number of CPUs.

4.1.6 Embedded Systems

Embedded systems is a term that covers a broad range of computers from ubiquitous hand held consumer items like cellular phones to Defense reconnaissance satellites. In fact, the number of CPUs manufactured for embedded applications dwarfs those for desktops and servers. Embedded systems tend to be optimized for lower power consumption (AA battery lifetime is a key metric) which in turn allows for higher packaging density. Flash memories are used to provide a modest volume of persistent memory, and interconnection is often provided via ad hoc wireless networks.

Embedded systems usually run small, real-time operating systems. These are more sophisticated than the micro-kernels on some supercomputers, but do not support the broad range of features expected by desktop or server applications. Today's embedded applications are generally written in C. Historically, they were different than the applications seen in desktops and servers, but with the new generation of hand-held Web interfaces (e.g. Apple's iPhone™), the end user application

space may tend to become indistinguishable.

4.1.7 Summary of the State of the Art

The state of the art in computer architecture today is dominated by a 50 year reliance on the von Neumann model where computing systems are physically and logically split between memory and CPUs. Memory is “dumb,” with the only functionality being able to correlate an “address” with a particular physical set of storage bits. The dominant execution model for CPUs is that of a sequential thread, where programs are divided into lists of instructions that are logically executed one at a time, until completion, and each operation applies to only a relatively small amount of data. Higher performance architectures invest a great deal of complexity in trying to place more instructions in play concurrently, without the logical model of sequential execution being violated, especially when exceptions may occur.

Two technological developments have impacted this model of computer architecture. First is the **memory wall**, where the performance of memory, in terms of access time and data bandwidth, has not kept up with the improvement in clock rates for the CPU designs. This has led to multi-level caches and deep memory hierarchies to try to keep most of the most relevant data for a program close to the processing logic. This in turn has given rise to significantly increased complexity when multiple CPUs attempt to share the same memory, and collaborate by explicitly executing separate parts of a program in parallel.

The second technological development that has impacted architecture is the rise of power as a first class constraint (the **power wall**), and the concomitant flattening of CPU clock rates. When coupled with the memory wall, performance enhancement through increasing clock rates is no longer viable for the mainstream market. The alternative is to utilize the still growing number of transistors on a die by simple replication of conventional CPU cores, each still executing according to the von Neumann model. Chips have already been produced with a 100+ cores, and there is every expectation that the potential exists for literally thousands of cores to occupy a die. The memory wall, however, is still with us, with a vengeance. Regardless of processor chip architectures, more cores usually means more demand for memory access, which drives a requirement for more memory bandwidth. Unfortunately, current technology sees a flattening of the number of off-chip contacts available to a processor chip architect, and the rate at which these contacts can be driven is tempered by power concerns and the technology used to make memory chips - one chosen for density first, at low fabrication cost. High performance interfaces in such an environment are difficult.

As Section 4.1.3 describes, there are attempts today to modify this von Neumann model by blurring in various ways the distinction between memory “over here” and processing logic “over there.” To date, however, the penetration into the commercially viable sector has been nearly nil.

4.2 Today’s Operating Environments

An **operating environment** is the environment in which users run programs. This can range from a graphical user interface, through a command line interface, to a simple loader, run-time scheduler, memory manager, and API through which application programs can interact with system resources. Conventional software computing environments supporting high performance computing platforms combine commercial or open source node local operating system software and compilers with additional middle-ware developed to coordinate cooperative computing among the myriad nodes comprising the total system (or some major partition thereof). For technical computing, local node operating systems are almost exclusively based on variants of the Unix operating

system introduced initially in the mid to late 1970s. While a number of successful commercial Unix OS are provided by vendors, most notably IBM (AIX), Sun (Solaris), and HP (HPUX), the dominant operating system in today's production level high performance computing machines is the Open Source Unix variant, Linux. In addition, a recent offering by Microsoft is providing a Windows based HPC environment. On top of this largely Unix based node local foundation is additional "middleware" for user access, global system administration, job management, distributed programming, and scheduling.

In addition to these heavy-weight operating systems is a class of lightweight kernels. These have been employed on some of the largest supercomputers including the IBM BlueGene/L system (its own microkernel) and the Cray XT series (Compute Node Linux - a trimmed down Linux).

Further, most system actually employ nodes with a mix of operating environments, with service or I/O nodes typically running a more complete offering to provide more complete functionality.

Finally, **Catamount** is a tailored HPC run-time that has run on several HPC systems, such as Red Storm and Cray's XT3 and XT4. Each is discussed briefly below.

4.2.1 Unix

Several key attributes of the Unix kernel make the system highly customizable and portable, especially as realized in Linux. At the core of the various Linux distributions, the modular monolithic kernel is a contributing factor to the success of Linux in HPC. The Linux kernel is layered into a number of distinct subsystems and the core ties these various modules together into the kernel.

The main subsystems of the Linux kernel are:

- **System Call Interface** (**SCI**): This is an architecture dependent layer that provides the capability of translating user space function calls to kernel level calls.

- **Process management**: This provides ability for active threads to share CPU resources based on a predefined scheduling algorithm. The Linux kernel implements the **O(1) Scheduler** that supports symmetric multiprocessing.

- **Memory Management**: Hardware memory is commonly virtualised into 4KB chunks known as **pages**, with the kernel providing the services to resolve these mapping. Linux provides management structures such as the **slab allocator** to manage full, partially used and empty pages.

- **File system management**: Linux uses the **Virtual File System interface** (**VFS**) (a common high level abstraction supported by various file systems) to provide an access layer between the kernel SCI and VFS-supported file systems.

- **Network subsystem**: The kernel utilizes the sockets layer to translate messages into the TCP(/IP) and UDP packets in a standardized manner to manage connections and move data between communicating endpoints.

- **Device Drivers**: These are key codes that help the kernel to utilize various devices, and specify protocol description so that the kernel (SCI) can easily access devices with low overheads. The Linux kernel supports dynamic addition and removal of software components (dynamically loadable kernel modules).

4.2.2 Windows NT Kernel

Windows Server 2003 has a hybrid kernel (also known as **macrokernel**) with several emulation subsystems run in the user space rather than in the kernel space.

The **NT** kernel mode has complete access to system and hardware resources and manages scheduling, thread prioritization, memory and hardware interaction. The kernel mode comprises of executive services, the microkernel, kernel drivers, and the hardware abstraction layer.

- **Executive**: The executive service layer acts as an interface between the user mode application calls and the core kernel services. This layer deals with I/O, object management, security and process management, local procedural calls, etc through its own subsystems. The executive service layer is also responsible for cache coherence, and memory management.

- **MicroKernel**: The microkernel sits between the executive services and the hardware abstraction layer. This system is responsible for multiprocessor synchronization, thread and interrupt scheduling, exception handling and initializing device drivers during boot up.

- **Kernel-mode drivers**: This layer enables the operating system to access kernel-level device drivers to interact with hardware devices.

- **Hardware Abstraction Layer (HAL)**: The HAL provides a uniform access interface, so that the kernel can seamlessly access the underlying hardware. The HAL includes hardware specific directives to control the I/O interfaces, interrupt controllers and multiple processors.

4.2.3 Microkernels

A microkernel is a minimal run-time which provides little or no operating system services directly, only the mechanisms needed to implement such services above it.

Compute Node Linux[152] (**CNL**) is an example of one such that has its origins in the SUSE SLES distribution.

The microkernel for BlueGene/L[108] was designed to fit in small memory footprints, and whose major function would be to support scalable, efficient execution of communicating parallel jobs.

Catamount[152] is an independently-developed microkernel based operating system that has two main components: the **quintessential kernel** and a **process control thread**. The process control thread manages physical memory and virtual addresses for a new process, and based on requests from the process control thread the quintessential kernel sets up the virtual addressing structures as required by the hardware. While Catamount supports virtual addressing it does not support virtual memory. The process control thread decides the order of processes to be executed and the kernel is responsible for flushing of caches, setting up of hardware registers and running of the process. In essence the process control thread sets the policies and the quintessential kernel enforces the policies. Catamount uses large (2 MB) page sizes to reduce cache misses and TLB flushes.

- **Quintessential Kernel**: The **Q. Kernel** (**QK**) sits between the process control thread (**PCT**) and the hardware and performs services based on requests from PCT and user level processes. these services include network requests, interrupt handling and fault handling. if the interrupt or fault request is initiated by the application then the QK turns over the control to PCT to handle those requests. . The QK handles privileged requests made by PCT such as running processes, context switching, virtual address translation and validation.

- **Process Control Thread**: The PCT is a special user-level process with read/write access to the entire memory in user-space and manages all operating system resources such as process loading, job scheduling and memory management. QKs are non communicating entities, while the PCTs allocated to an application can communicate to start, manage and kill jobs.
- **Catamount System Libraries**: While QK and PCT provide the mechanisms to effect parallel computation, Catamount libraries provide applications access to harness their mechanisms. Catamount system libraries comprise of Libc, libcatamount, libsysio and libportals, Applications using these services have to be compiled along with the Catamount system libraries.

4.2.4 Middleware

Additional system software is provided to simplify coordination of the distributed resources of the scalable MPPs and commodity clusters most prevalent today. Tool sets such as **PBS**, the **Maui scheduler**, and **Loadleveler** provide such system wide environments. Management of secondary storage from parallel systems is provided by additional file system software such as the open source **PVFS** and Lustre systems and the IBM proprietary GPFS. Although Lustre is open source, is it supported and maintained in large part by Sun.

4.2.5 Summary of the State of the Art

The vast majority of operating systems on current the Top-500 list are Unix or its derivatives and the majority of these are Linux in various forms. Unix provides a well understood and time tested set of services and interfaces. Systems today are ensembles of compute subsystems with the majority organized as distributed memory collections. The operating environments reflect this division by providing node operating systems with supporting middleware to coordinate their scheduling and allocation to user workloads. Therefore, as the physical system is a collection of multiprocessor or multicore nodes integrated via one or more system interconnect networks, so the logical system is a collection of Unix or Linux domains managing distinct sets of cores integrated by a scheduling layer. A separate but important piece of the operating environment is the mass storage system including the file system. Parallel file systems manage large numbers of disk drives often with multiple controllers each servicing a number of spindles. An emerging trend in operating environments is the use of lightweight kernels to manage some or all of the compute nodes. The original environment on the Cray XT3 was the Catamount lightweight kernel system. The IBM Blue Gene system has mixed mode environments with a lightweight kernel on each of its compute nodes managed by an I/O node hosting a Linux system. The challenge for the immediate future is to develop advanced Unix and Linux versions that can support up to hundreds of cores per shared memory node. But over the longer term, lightweight kernel systems may prove valuable for systems comprising millions of small cores such as the Cell architecture.

4.3 Today's Programming Models

A **programming model** is comprised of a set of languages and libraries that define the programmer's view of a machine. Examples of programming models for parallel machines include traditional sequential languages combined with a message passing layer or a thread library, parallel languages like UPC or HPF, and sequential languages combined with parallelizing compilers. A single machine may support multiple programming models that are implemented through compiler translation and

run-time software layers. In a broad sense, each parallel programming model combines a control model that specifies how the parallelism is created and managed, and a communication model that allows the parallel entities to cooperate somehow by sharing data.

An **execution model** is distinct from the programming model and consists of the physical and abstract objects that comprise the evolving state of the computation. It defines how the physical and abstract objects that actually perform the computation and track its results interact. An execution model ties together all layers of the vertical functionality stack from the application and algorithms by means of the programming language and compiler, the run-time and operating system, and goes down to the system wide and local micro-architecture and logic design. The execution model conceptually binds all elements of a computing system in to a single coordinated cooperating computing corporation. An execution model may consist of a fixed set of thread or process states, each with its own program stack, and a set of message queues and other state information needed to communicate between the threads. A programming model may be very similar to its underlying execution model, although in some cases they can also be quite different. For example, a data parallel languages like HPF provide the illusion of an unbounded set of lightweight processors that execute each statement in lock step, while compilers for distributed memory architectures convert this data parallel code into a smaller set of physical threads that correspond to the number of physical processors; the threads execute independently and communicate with explicit message passing.

The execution model determines how a program executes, whereas the programming model determines how it is expressed.

As computing technology has evolved it has enabled ever more powerful, often increasingly parallel, computer architectures. Innovations in execution models have included vector and SIMD, dataflow and systolic, message passing and multi-threaded, with a diversity of variations of each. Such execution models have differed in the form of parallelism explicitly exploited, while taking advantage of others intrinsic to the micro-architecture structures such as ILP, pipelines, and prefetching. The programming models have similarly evolved, starting with serial languages (with automatically parallelization) on the vector execution model, data parallel languages with SIMD execution, and functional languages on dataflow.

The **communicating sequential processes**[65] (**CSP**) execution model (essentially a formalism of message passing) has dominated much of the last two decades and drives many of the largest systems today including the IBM Blue Gene, Cray XTn, and commodity Linux clusters. Execution models based on threads communicating through shared variables are often used on smaller machines with cache coherent shared memory, and a hybrid model may be used on machines that are built from as clusters of shared memory nodes, especially on the IBM Power architectures with a relatively large number of processor cores per node. Performance experience with the hybrid model is mixed, and in many cases programmers use message passing processes even on individual cores of a shared memory node.

Users select programming models based on a number of factors, including familiarity, productivity, convenience, performance, compatibility with existing code, and portability across current and future machines. Today's Terascale codes are written predominantly in the MPI Message Passing Interface using a traditional sequential language like Fortran, C or C++. Mixed language applications are also quite common, and scripting languages like Python are often used in putting together complex application workflows built from multiple programming models.

Machine architectures may enable newer and (by some criteria) better programming models, but there is a great deal of inertia in existing models that must be overcome to move a significant fraction of the user community. Historically, this has happened when the execution model changes significantly due to underlying architectural changes, and the programming model of choice is either

unsupported or offers reduced performance capabilities. When vector machines were the dominant high end architecture, they were programmed primarily with annotated sequential languages, such as Fortran with loop annotations. When large-scale SIMD machines were popular (e.g. the CM-2 and Maspar), data parallel languages like CMFortran, *Lisp, and C* were the preferred model. These models were quickly overtaken by message passing models as distributed memory machines including low-cost Beowulf clusters became the architecture of choice.

In each case, there were efforts to move the previous model to an execution model that could be supported on the new machines. Attempts to move automatic parallelism from the vector execution model to shared memory threads was not successful in practice, in spite of significant investment and many positive research results on the problem. Motivated by the success of vector compiler annotations, OpenMP was designed to provide a similar programming model for the shared memory execution model and this model is successful, although the annotations are significantly more complex than earlier vector annotations. The HPF language effort was initiated to take a data parallel model to message passing hardware, but performance was not considered adequate for larger scale machines. Similarly, despite many efforts to provide shared memory programming on a message passing execution model, this is not used in practice today. Partitioned Global Address Space languages like UPC and Co-Array Fortran have seen some success in limited application domains, because they give programmers control over data layout and local, but they are missing the kind of broad adoption that MPI enjoys.

Somewhere between Petascale and Exascale computing, due in large part to the increased exposure of on-chip parallelism, the MPI model may find its limit, as a separate process with its own address space may be too heavyweight for individual cores. Each process requires its own process state and message buffers, in addition to replicas of user level data structures that are needed by multiple processes. As the number of cores per chip increases, either hybrid parallelism or some new model is likely to take hold.

4.3.1 Automatic Parallelization

Automatic parallelization has long been the holy grail of parallel computing, allowing programmers to take advantage of parallel hardware without modifying their programs. There are many examples of successful automatic parallelization in the research literature, especially for programs dominated by arrays and loops and for machines with a modest number of processors and hardware-supported shared memory. Parallelizing compilers have been challenged by the use of languages like C that rely heavily on pointers, since they make the necessary dependence analysis difficult to impossible. The shift towards distributed memory machines in parallel computing further complicated the automatic parallelization problem, since on top of parallelization, compilers needed to automatically partition data structures across the compute node memory. Again, while progress was made on this problem for some applications, it soon became clear that programmers needed to provide the partitioning information to make this practical, and the user community, impatient for an immediate solution, instead switched to message passing.

4.3.2 Data Parallel Languages

Data parallel languages in their purest form have serial semantics, which means that every execution for a given program and input (broadly construed to include all environment information, such as random number seeds and user input) will produce the same result. Operationally, one can see all behavior of the data parallel code using a deterministic serial machine execution. Data parallel languages are distinguished from conventional serial languages through their use of operations and

assignments over aggregate data types, typically arrays. The following code fragment shows some data parallel operations arrays `A` and `B`:

```
A = B;
B(1:N-1) = 2*B(1:N-1) - shift(B(2:N),{-1}) - shift(B(0:N-2,{1}));
A = B;
```

The first statement assigns all of the elements in `B` to the corresponding elements at the same indices in `A`. The set of assignments within the statement can be done in any order, and because any serial or parallel order is guaranteed to give the same result, one can treat the statement semantically as if it were serial, e.g., the elements are assigned left to right. The second statement is more interesting, because it has side effects on array `B` that also appears on the right-hand-side. Under data parallel semantics, the entire right-hand-side is evaluated to produce an array, which we can think of as being stored in a temporary array, and that array is then assigned to the left-hand side. This preserves the serial semantics, since the side effects happen only after the expression evaluation is complete. This particular expression shows a relaxation operation (3-point stencil) in which the interior of array `B`, indicated by `B(1:N-1)`, is updated with after scaling all the values by 2 and subtracting the left and right neighboring values.

There is an implicit barrier between statements, meaning that one statement cannot appear to execute before the previous has completed, although an optimizing compiler may regroup and reorder instructions as long as those reorderings do not introduce a change in semantic behavior. A compiler for the above code might divide `A` and `B` into contiguous blocks, use one per processor and allocate a "ghost region" around the blocks of `B` to allow for space to hold copies of neighboring values. The code could be a Single Program Multiple Data style code with one thread per processor, which performs a local assignment of the local elements of `A` to `B`, followed by communication with processing that own neighboring blocks to fill in ghost values of `B`, followed by a local update of `B` and then (without communication or synchronization) another assignment of the elements of `B` to `A`.

Data parallel languages were quite popular on SIMD architectures like the Connection Machine (CM-2) and Maspar, where the fine-grained parallelism was supported in hardware. Once clusters began dominating high end computing, data parallel languages became less popular, because the compilation problem described above was challenging in the general case. The HPF language effort was designed to address this problem by adding "layout" specifications to arrays, so that the compiler could more easily determine how to break up the problem, and the computation would follow the data using an "owner computes" rule. The HPF effort had two conflicting agendas, one being support of general computational methods, including sparse, adaptive, and unstructured, while also providing performance that was competitive with message passing style programs. This created significant langauge and compiler challenges, and the application community and U.S. funding agencies soon became impatient for results. There are notable examples of successful HPF applications, including two on the Earth Simulator System in Japan, which benefited from both architectural support in the form of vectors processing nodes and a significant sustained compiler effort that lasted longer after funding for HPF within the U.S. had dried up.

4.3.3 Shared Memory

The **Shared Memory Model** reflects a paradigm where parallel threads of computation communicate by reading and writing to shared variables, and there is, implicitly, a uniform cost to accessing such shared variables. The data parallel model uses shared variables to communicate,

but the term “shared memory terminology” is used here to capture programming models with an explicit form of parallelism, such as user-defined threads. The uniform cost model distinguished shared memory models from the partitioned global address space models, where there is an explicit notion of near and far memory in the programming langauge.

4.3.3.1 OpenMP

OpenMP is an API that supports parallel programming on shared memory machines. The API is a defined by a set of compiler directives, library routines, and environment variables. These are implemented as a set of extensions to Fortran, C, and C++, which are added as comments to the base language to simplify maintenance of programs that work in both a pure serial mode without OpenMP compiler support and in parallel with such support. OpenMP was developed by an consortium of major computer hardware and software vendors with the goal of addressing the technical computing community, and it currently runs on most shared memory parallel machines. There are many features of OpenMP to support a variety of parallelization patterns, but the most common is the parallel loop. The parallel loop annotations can also include mechanisms to distinguish shared and private variables, to load balance the iterations across processors, and to perform operations like reductions, which create dependencies across loop iterations and therefore require special attention to avoid data races in the generated code. OpenMP offers more general form of parallelism besides loops, in which independent tasks (threads) operate on shared variables. OpenMP programs are typically translated to an execution layer in which threads read and write variables that live in shared memory.

OpenMP is generally considered an easier programming model than message passing, both antidotally and in user studies.[66] There is no programmer control over data layout within OpenMP, which is key to its programming simplicity, but a limitation in scalability. There have been research efforts to run OpenMP on a software shared memory execution layer on top of distributed memory hardware, and efforts to expand OpenMP to directly support clusters, but in practice OpenMP is used only on cache-coherent shared memory hardware. Many of the largest systems today are built from shared memory nodes, and several application teams have developed hybrid versions of their codes that combine MPI between nodes and OpenMP within nodes. To date, the performance results for such hybrid codes have been mixed when compared to a flat message passing model with one process per core. While the OpenMP code avoids the costs of message construction, buffering, matching and synchronization, it often performs worse than message passing due to a combination of false sharing, coherence traffic, contention, and system issues that arise from the difference in scheduling and network interface moderation for threads as compared to processes. Well optimized OpenMP can outperform MPI on shared memory, and with sophisticated compiler and runtime techniques, it has also been demonstrated to be competitive on cluster hardware.[151] But the history of shared memory programming and message passing programming cannot be ignored: despite an expectation that hybrid programming would become popular on machines built as a distributed memory network of shared memory nodes, it has not.

There is speculation that OpenMP will see gains in popularity with multi-core systems, since most multi-core architectures today support cache-coherent shared memory. The on-chip caching should exhibit lower overheads than on multi-socket SMPs today, and the programming convenience of OpenMP continues to make it attractive for programmers newly introduced to parallelism. Whether the hybrid model will become popular for Exascale programs will depend primarily on how the performance and memory scaling tradeoffs are resolved with large number of cores per chip; the momentum in this case is in favor of a flat MPI model, which avoids the need to use two different forms of parallelism in a single application.

4.3.3.2 Threads

Many modern operating environments support the POSIX threads (PThreads), or a similar threading interface that is specific to a given operating system. library. This library allows for a simple thread creation mechanism as well as mutex locks, conditional signal mechanisms, and shared memory maps. A major advantage of this model, and in particular the POSIX API, is that it is available on a vast number of shared memory platforms and (unlike OpenMP) does not require compiler support. A thread library includes mechanisms to create and destroy threads, as well as locks and other synchronization mechanisms to protect accesses to shared variables. Variables allocated in the program heap are typically available to other threads, and a common programming idiom is to package the shared state in a single structure and pass a pointer to that state to each thread.

There are various special cases of threading that arise in specific languages. In several cases the language-based implementations uses a lexically scoped threads mechanism, such as cobegin/coend, in which the lifetime of a thread is visible in the program text. The POSIX model, in contrast, gives a the thread handle a the creation point, and that thread may be terminated any point in the program where the handle is available. The paired mechanisms are simpler to reason about for formally and informally, but are less expressive and require some language and compiler support to enforce the pairing. For example, the Cilk language extends C with threads and requires a compiler to translate the language syntax into lower level system threads, such as PThreads. As another example of language support for threading, the Java mechanism uses its object-orientation to support threads, whereby a class is created to contain application-specific thread state that is derived from a standard thread class, and methods on that class allow the threads to be created and managed.

An important distinction in thread systems is whether the threads are fairly scheduled in the face of long-running computations within a single thread. Fair scheduling ensures that all threads make progress independent of the number of hardware cores or hardware-supported threads that are available, and in general fairness requires some kind of preemption to ensure that one or more threads cannot dominate all available resources and leave other threads stalled. Fair scheduling is important is if programs use spin locks or more subtle forms of shared memory synchronization and the number of threads exceeds hardware resources. For example, if there are two types of worker threads in a system, one producing work and the other sharing it, and those threads synchronized by accessing shared queues, then consumer threads may starve producers if they continually poll for work by reading the shared queues but are never descheduled.

Fairness and preemption comes with a cost, though, since each thread can have a significant amount of state stored in registers, which must be explicitly saved at a preemption point, and caches, which may be slowly saved as the preempting thread refills the cache. An alternate model is cooperating threading, in which the programmer is informed that threads may run to completion and need only give up control over processor resources at explicit synchronization or yield points. The Cilk language takes advantage of this by allowing programmers to specify a larger number of threads that are executed simply as either function calls or as separate threads depending on resource availability. In this way, a Cilk thread is a logical thread that only translates to a more expensive physical thread when hardware is available to hold the thread state. In this case the previous producer-consumer programmer or programs with spin locks are not supposed to work. Cooperating threading models have advantages in performance by avoiding preemption at points where locality is critical, e.g., in the middle of a dense matrix operation; they also have memory footprint advantages, since thread state may be created lazily when processors become available to run them.

4.3.4 Message Passing

The **Message Passing Interface**indexMessage Passing Interface (**MPI**) is a specification of a set of library routines for message-passing. These routines handle communication and synchronization for parallel programming using the message-passing model. MPI is targeted for distributed memory machines but can also be used effectively on shared memory systems. The MPI interface was derived from a number of independent message passing interfaces that had been developed by hardware vendors and application groups to address the growing interest on distributed memory hardware in the early 90s. In this sense, MPI was the standardization of a popular programming model, rather than the introduction of a new model. Its success can be attributed to this preexisting popularity for the model, to its high performance and scalability across shared memory and distributed memory platforms, and its wide availability, which includes highly portable open source implementations like MPICH and OpenMPI.

MPI has mechanisms to send and receive contiguous blocks of memory, and while there are higher level mechanism to allow the implementation to pack and unpack non-contiguous data structures, this is typically done by programmers of higher level libraries and applications for performance considerations. Synchronous send and receive operations can easily result in deadlock, if two processes try to simultaneously send to each other. MPI therefore supports asynchronous messages, which allow send operations to complete even if the remote process is unavailable to receive, and asynchronous receive operations which can available buffer copying by having the user level target data structure allocated before the message arrives.

An important feature of MPI besides its point-to-point messages are collective communication operations which allow a collection of processes to perform a single operation such as a broadcast, reduction, or all-to-all communication operation. These operations can be built on top of send and received, but can also use optimized execution models or architectures, such as a special network. Collectives in MPI need not be globally performed across all processors, but sub-domains can be identified using the notion of a communicator which is defined over a subset of physical processes. This can be used to subdivide a single computation, e.g., perform reductions across rows of a matrix, and to compose together programs that were written independently. The latter is especially important with the growing interest on multi-physics simulations, in which separate models, such as an ocean and wind dynamics model in a climate simulation, are combined to model complex physical phenomenon.

MPI-2 is a second version of the MPI standard, which adds support for support for one-sided communication, dynamic processes creation, intercommunicator collective operations, and expanded IO functionality (the MPI-IO portion of the interface). While many of the MPI-2 is supported in most implementations, in particular MPI-IO, support for, and optimization of, one-sided communication has been slower. As noted above, the ubiquity of MPI relies on open source implementations, and interconnect vendors may start with these open source version and optimize certain features of the implementation.

MPICH is an implementation of the MPI-1 specification, while MPICH2 supports the expanded MPI-2 interface. MPICH is one of the oldest and most widely used MPI implementations, and benefits from continued, active development as well as wide scale usage in research and production environments. **OpenMPI** is a relatively new implementation of the MPI-2 specification. It is an open source project maintained by academic, research, and industry partners. The core development team was composed of members of many different MPI implementations, and it represents a consolidation of their experience and expertise. The focus for OpenMPI has been on increased reliability. Specific features include support for dynamic process spawning, network and process fault tolerance, and thread safety.

4.3.5 PGAS Languages

A **Partitioned Global Address Space** (PGAS) combines some of the features of message passing and shared memory threads. Like a shared memory model, there are shared variables including arrays and pointer-based structures that live in a common address space, and are accessible to all processes. But like message passing, there the address space is logically "partitioned" so that a particular section of memory is viewed as "closer" to one or more processes. In this way the PGAS languages provide the necessary locality information to map data structure efficiently and scalably onto both shared and distributed memory hardware. The partitioning provides different execution and performance-related characteristics, namely fast access through conventional pointers or array indexes to nearby memory and slower access through global pointers and arrays to data that is far away. Since an individual process may directly read and write memory that is near another process, the global address space model directly supports one-sided communication: no participation from a remote process is required for communication. Because PGAS languages have characteristics of both shared memory threads and (separate memory) processes, some PGAS languages use the term "thread" while others use "process." The model is distinguishable from shared memory threads such as POSIX or OpenMP, because the logical partitioning of memory gives programmers control over data layout. Arrays may be distributed at creation time to match the access patterns that will arise later and more complex pointer-based structures may be constructed by allocating parts in each of the memory partitions and linking them together with pointers.

The PGAS model is realized in three decade-old languages, each presented as an extension to a familiar base language: **Unified Parallel C** (UPC)[31] for C; **Co-Array Fortran** (CAF)[112] for Fortran, and Titanium[160] for Java. The three PGAS languages make references to shared memory explicit in the type system, which means that a pointer or reference to shared memory has a type that is distinct from references to local memory. These mechanisms differ across the languages in subtle ways, but in all three cases the ability to statically separate local and global references has proven important in performance tuning. On machines lacking hardware support for global memory, a global pointer encodes a node identifier along with a memory address, and when the pointer is dereferenced, the runtime must deconstruct this pointer representation and test whether the node is the local one. This overhead is significant for local references, and is avoided in all three languages by having expression that are statically known to be local, which allows the compiler to generate code that uses a simpler (address-only) representation and avoids the test on dereference.

These three PGAS languages used a static number of processes fixed at job start time, with identifiers for each process. This Single Program Multiple Data (SPMD) model results in a one-to-one mapping between processes and memory partitions and allows for very simple runtime support, since the runtime has only a fixed number of processes to manage and these typically correspond to the underlying hardware processors. The languages run on shared memory hardware, distributed memory clusters, and hybrid architectures. On shared memory systems and nodes within a hybrid system, they typically use a thread model such as Pthreads for the underlying execution model.

The distributed array support in all three languages is fairly rigid, a reaction to the implementation challenges that plagued the High Performance Fortran (HPF) effort. In UPC distributed arrays may be blocked, but there is only a single blocking factor that must be a compile-time constant; in CAF the blocking factors appear in separate "co-dimensions;" Titanium does not have built-in support for distributed arrays, but they are programmed in libraries and applications using global pointers and a built-in all-to-all operation for exchanging pointers. There is an ongoing tension in this area of language design between the generality of distributed array support and the desire to avoid significant runtime overhead.

Each of the languages is also influenced by the philosophy of their base serial language. Co-Array Fortran support is focused on distributed arrays, while UPC and Titanium have extensive support for pointer-based structures, although Titanium also breaks from Java by adding extensive support for multidimensional arrays. UPC allows programmers to deconstruct global pointers and to perform pointer arithmetic such a incrementing pointers and dereferencing the results. Titanium programs retain the strong typing features of Java and adds language and compiler analysis to prove deadlock freedom on global synchronization mechanisms.

4.3.6 The HPCS Languages

As part of the Phase II of the DARPA HPCS Project, three vendors—Cray, IBM, and SUN—were commissioned to develop new languages that would optimize software development time as well as performance on each vendor's HPCS hardware being developed over the same time period. Each of the languages — Cray's Chapel[6],/indexChapel IBM's X10[7], and Sun's Fortress[8]—provides a global view of data (similar to the PGAS languages), together with a more dynamic model of processes and sophisticated synchronization mechanisms.

The original intent of these languages was to exploit the advanced hardware architectures being developed by the corresponding vendors, and in turn to be particularly well supported by these architectures. However, in order for these languages to be adopted by a broad sector of the community, they will also have to perform reasonably well on other parallel architectures, including the commodity clusters on which much parallel software development takes place. (And, in turn, the advanced architectures will have to run "legacy" MPI programs well in order to facilitate the migration of existing applications.)

The goal of these language efforts was to improve the programmability of HPC systems. This included both lowering the barrier to entry for new parallel programmers and making experienced programmers more productive. The HPCS languages all provide high level language support for abstraction and modularity using object-orientation and other modern language features, augmenting these with novel ideas for creating massive amounts of parallelism. The HPCS languages share some characteristics with each other and with the PGAS languages: the use of global name space, explicit representation of localities, and syntactic distinction of local and global data access. They all differ from the previously-described PGAS languages because they allow for dynamic parallelism and (in some cases) data parallelism, rather than a static number of threads. These languages require sophisticated compiler and runtime techniques, and in each case the vendors have developed at least prototype implementations that demonstrate the feasibility of implementation, although not necessarily at scale. Work on the implementations and analysis of productivity and performance benefits of the languages is ongoing.

Chapel is a parallel programming language developed by Cray. It supports data parallel programming and builds on some of the successful results from the ZPL languages. Chapel is not an extension of an existing serial languages, but addresses some of the limitations of the serial languages in addition providing parallelism support. A goal of Chapel is to provide better abstractions for separating algorithmic logic and data structure implementation and to provide programmers with a global view of the computation, rather than programming on a per-thread basis. Chapel uses data parallelism: data is distributed over memory partitions known as locales, and execution location can be controlled. Chapel is designed with features of the Cray's Cascade system in mind, including hardware support for a global address space, but is currently implemented

[6]http://chapel.cs.washington.edu/
[7]http://domino.research.ibm.com/comm/research_projects.nsf/pages/x10.index.html
[8]http://projectfortress.sun.com/Projects/Community/

on top of the GASNet communication layer and using a source-to-source translation system which make it portable across many existing machines.

Fortress is a parallel programming language designed at Sun. Like Chapel, Fortress uses a new syntax and semantics, rather than building on an existing serial language. Fortress uses a shared global address space but generalizes is though a hierarchical notion of partitioning, which could prove useful on machines with hierarchical parallelism and for composing separately-written modules. In an effort to expose maximum parallelism in applications, Fortress loops and argument evaluations in function calls are both parallel by default. The focus is on extensibility, and there is a novel type system for building libraries and allowing them to be analyzed and optimized by the compiler. The language takes advantage of the support for modularity and extensibility by implementing a small core langauge directly and then supporting a large set of standard libraries, in the same spirit as the Java libraries. Including the libraries, Fortress also provide support for matrices and vectors, with static checking for properties, such as physical units or boundary conditions, included.

X10 is an extension of the Java programming language. Java was chosen as a base language for its type safety features, object-oriented style, and familiarity among developers. X10 supports distributed object-oriented programming. The target architecture for X10 are low and high end systems comprised of multi-core SMP nodes. With X10, the memory is logically partitioned into locales and data is explicitly distributed by programmers to those locales. Computations can be explicitly assigned to particular locales which is implemented as remote function invocation. The first implementation was done for shared memory with shared task queues, but a implementation on top of the LAPI communication layer is also under development.

4.4 Today's Microprocessors

In this section we review briefly the state of current microprocessor chips which form the basis of all classes of computing from embedded to supercomputers. In most cases, historical data is combined with data from the ITRS Roadmap[13] to show trends, and more importantly, where trends break. The Roadmap data goes back to 2004 to overlap with the historical data and give an indication of how accurate the ITRS projections have been in the past.

4.4.1 Basic Technology Parameters

Perhaps the single parameter that most drives the semiconductor industry in general and the microprocessor vendors in particular, is the continued decline in the **feature size** of the transistors that make up the logic and storage circuits of a microprocessor. Figure 4.2 diagrams this parameter over time, with historical data from real microprocessors (labeled at the time of their first release) combined with projections from the ITRS roadmap for transistors to be used in logic circuits. As the trend line shows, the leading edge devices have been improving by a factor of about 0.88 per year. Today, leading edge production is being done at the 65 nm node, with experimental chips being produced at around 40 nm feature size.

A related characteristic is Figure 4.3, the density of transistors on a CMOS die over time, measured in millions of transistors per sq. mm. There are two distinct trend lines in this figure: that for the ITRS projections at a CAGR of 1.28 per year, and the historical trend of a higher 1.5 per year. A possible reason for this discrepancy is the huge growth in transistor-dense cache structures during the time of the single core microprocessors. This is buttressed by Figure 4.4 which shows a historical CAGR of 1.82 - significantly in excess of the growth in transistor density. How this will continue into the era of multi-core designs is unclear.

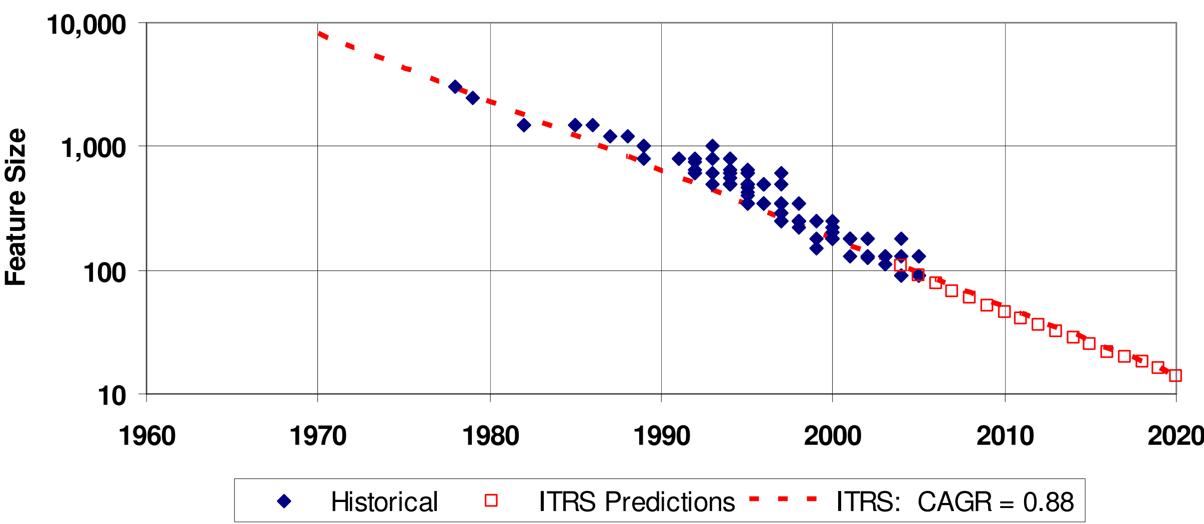


Figure 4.2: Microprocessor feature size.

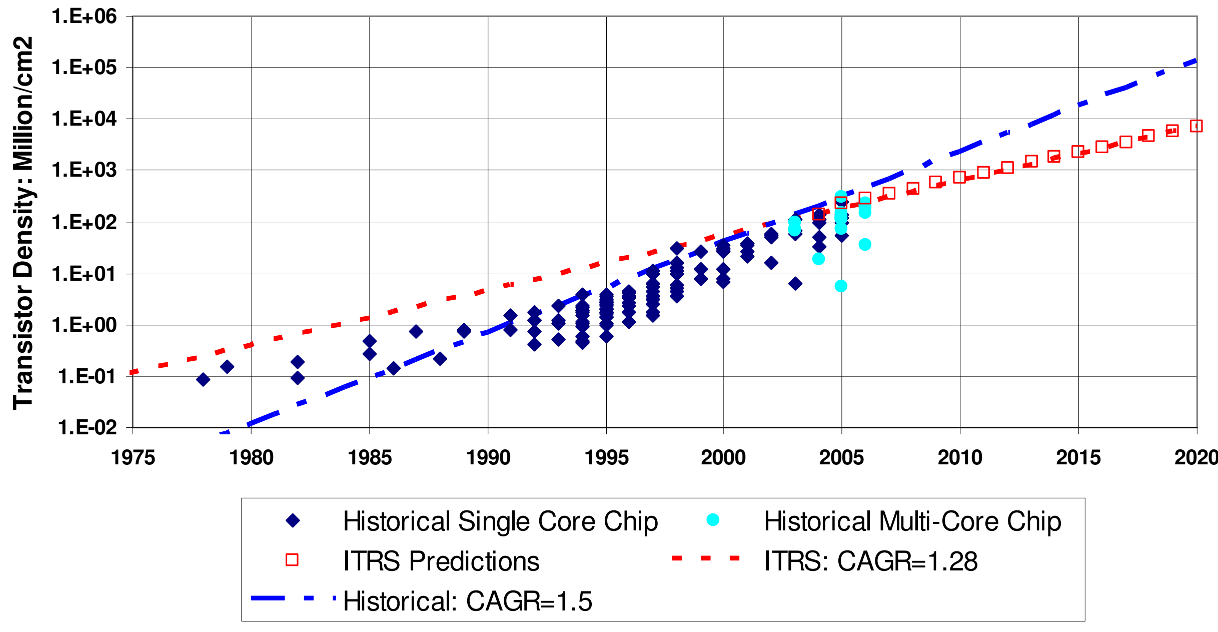


Figure 4.3: Microprocessor transistor density.

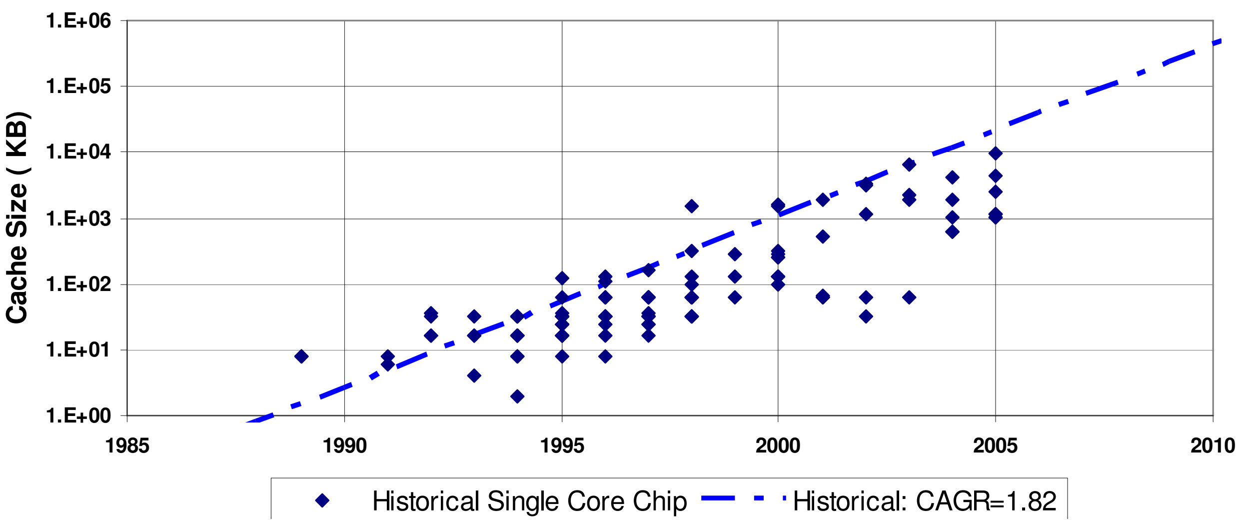


Figure 4.4: Microprocessor cache capacity.

4.4.2 Overall Chip Parameters

Two top level related chip-level parameters are the actual die size used for the chip (Figure 4.5) and the actual number of transistors on that die (Figure 4.6). The former exhibited a continual rise in size until 1995, after which the maximum die size varies, and is projected to continue to vary, in the 200-33 sq. mm range. The latter is pretty much a direct product of the die size of Figure 4.5 and the density of Figure 4.3, and an interpretation is similar to that for those figures.

Of more direct importance to this study is the voltage level used for the microprocessors (Figure 4.7). The V_{dd} curve shows a **constant voltage** at 5 volts until about 1993, when **constant field scaling** was employed and V_{dd} could be decreased very rapidly. As will be discussed later, this decrease balanced several other factors that affected the power of the die, and allowed faster chips to be deployed. After about 2000, however, this rapid decrease slowed, mainly because of minimums in the threshold voltages of CMOS transistors. Looking into the future, V_{dd} is projected to flatten even more, with a relatively tight range between what is used for high performance parts and that used for low power embedded parts.

Likewise, the decrease in transistor feature size led directly to faster transistors, which in turn led to increasing clock rates, as pictured in Figure 4.8. Up through the early 2000s' the historical clock rate increased at a CAGR of 1.3X per year. After 2004, the actual parts produced stagnated at around 3 GHz, below even a decreased ITRS projection.

It is important to note that these ITRS clock projections were computed as a maximum number assuming that the logic stays fixed at a 12 gate per pipeline stage delay, and the clock is raised up to a rate commensurate with the intrinsic improvement in the individual transistor delay. As will be discussed later, for power reasons, the actually implemented clock rates are now considerably less than this curve.

Finally, Figures 4.9 and 4.10 give both the power dissipated per die and the power density - the power dissipated per unit area. Both numbers went through a rapid increase in the 1990s, and then hit a limit. The maximum power that could be cooled at reasonable expense for a die destined for a high volume market was in the order of 100+ watts. Once this was reached, something had to be done to reduce the power, and that was done by reducing the clock - regardless of how fast the individual devices could go.

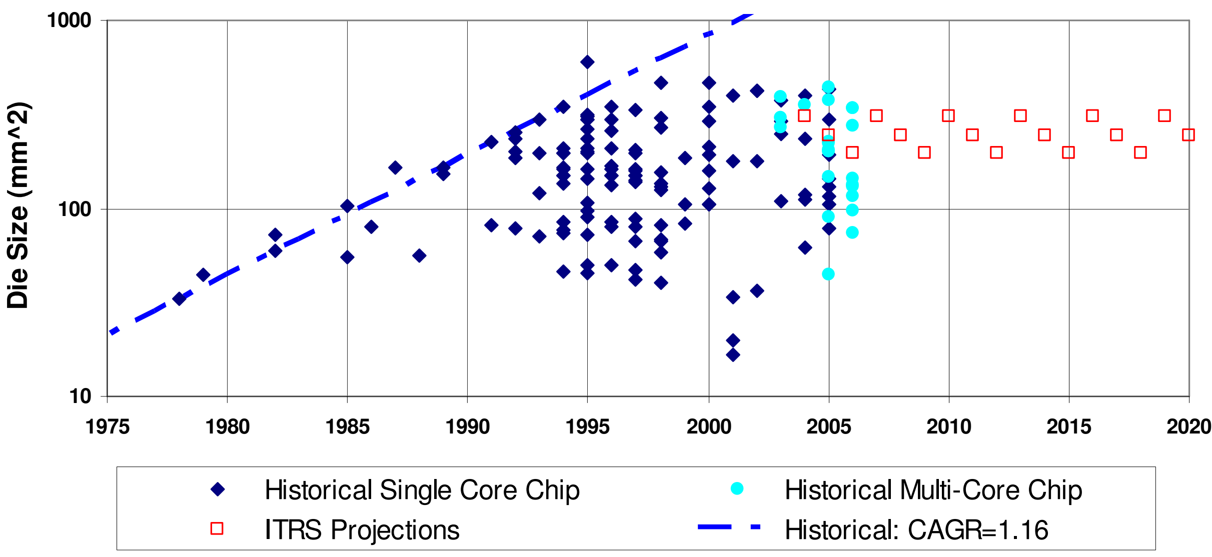


Figure 4.5: Microprocessor die size.

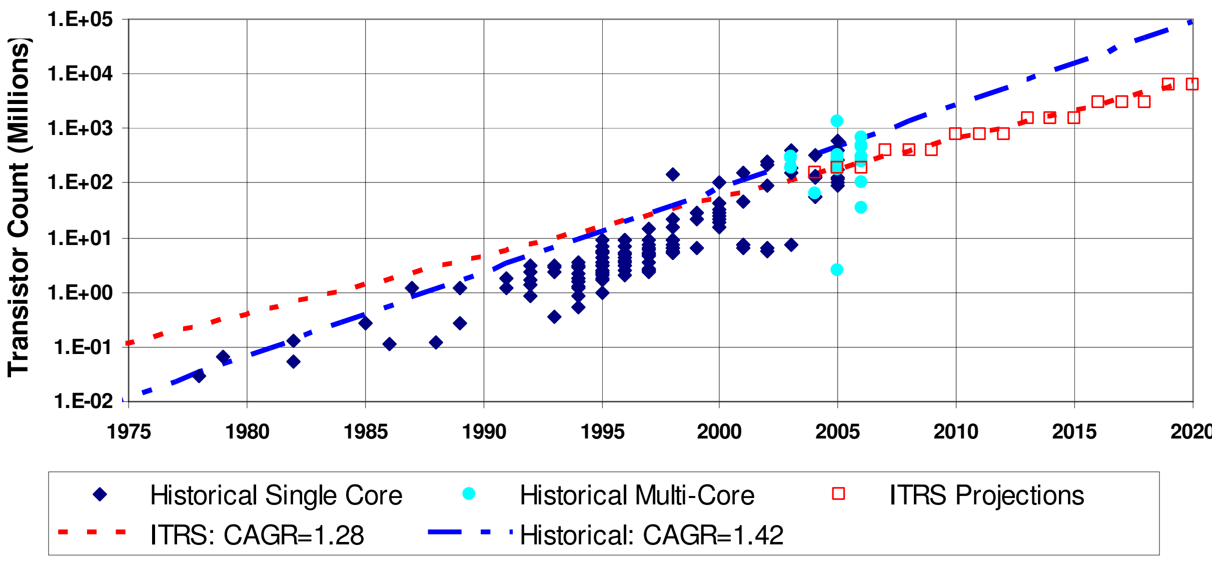


Figure 4.6: Microprocessor transistor count.

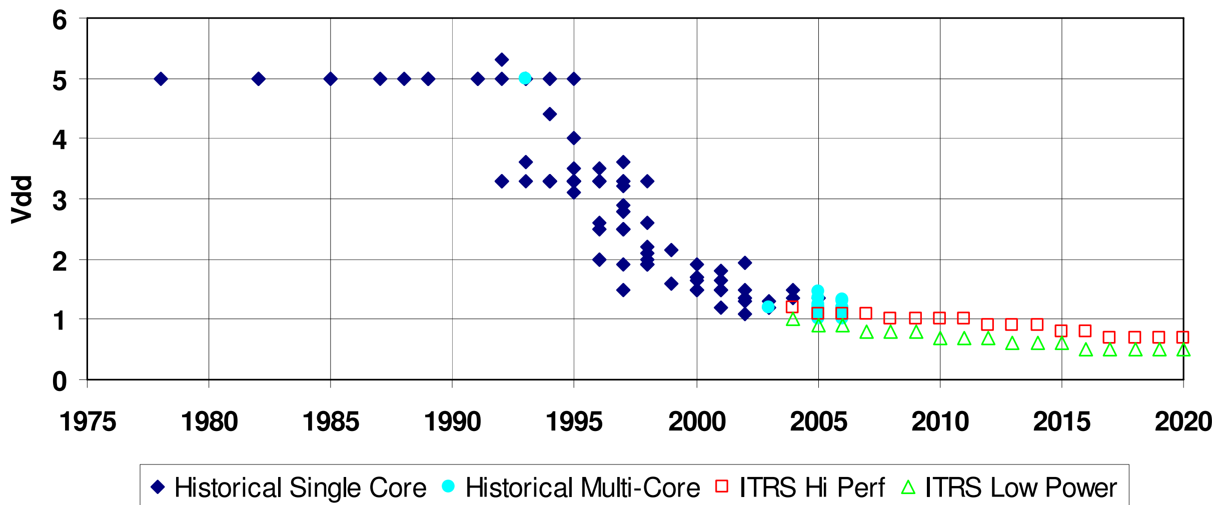


Figure 4.7: Microprocessor V_{dd}.

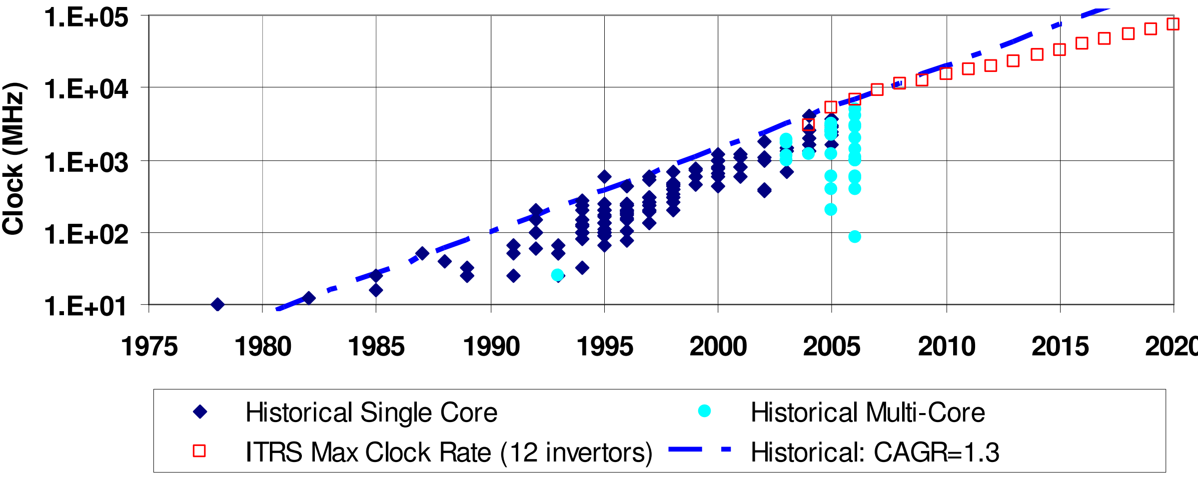


Figure 4.8: Microprocessor clock.

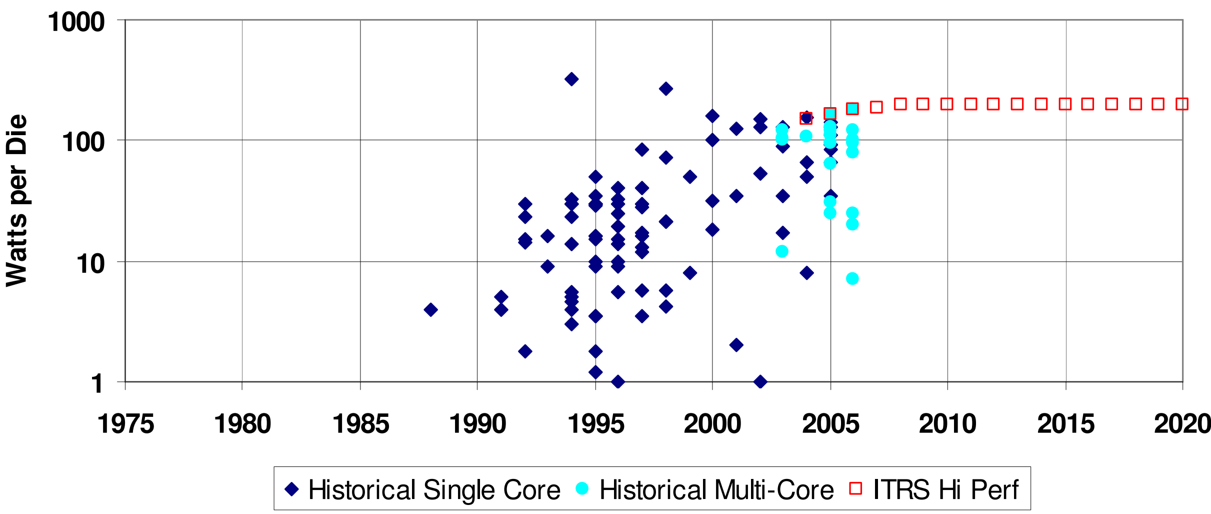


Figure 4.9: Microprocessor chip power.

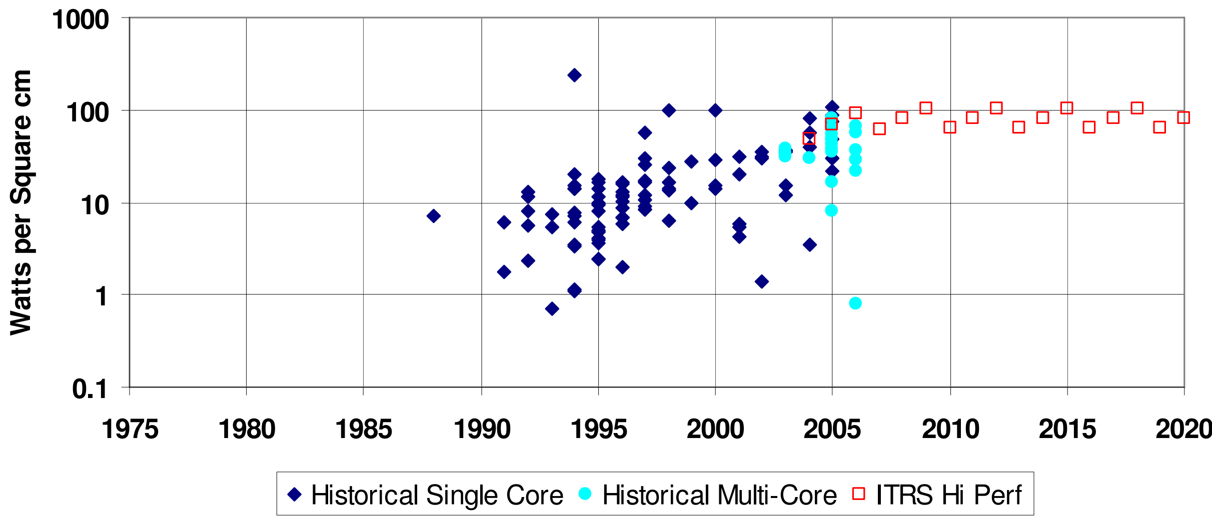


Figure 4.10: Microprocessor chip power density.

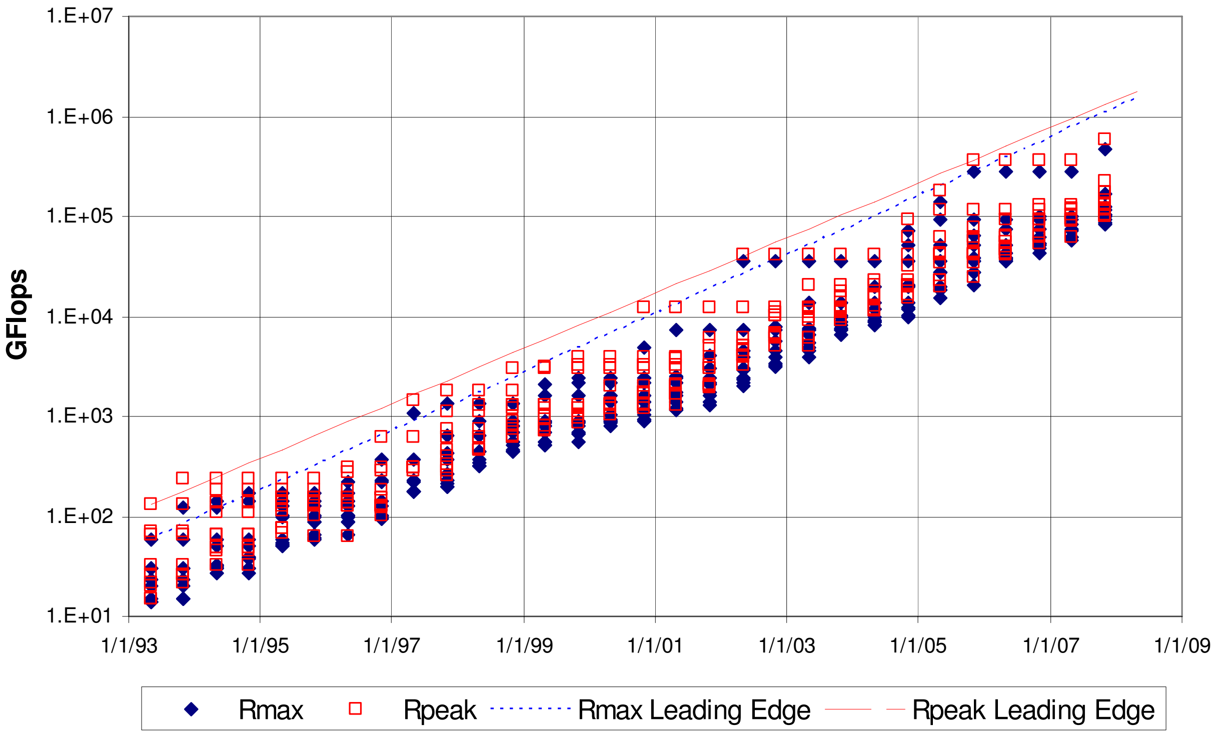


Figure 4.11: Performance metrics for the Top 10 supercomputers over time.

4.4.3 Summary of the State of the Art

Microprocessor design has ridden Moore's Law successfully for decades. However, the emergence of the power wall has fundamentally changed not only the microarchitecture of microprocessors (with the rise of multi-core as discussed in Section 4.1.1), but also their actual computational rates (as evidenced by the flattening of clock rates). These trends will continue.

4.5 Today's Top 500 Supercomputers

The **Top 500**[9] is a list of the top 500 supercomputers ("data-center sized") in the world as measured by their performance against the **Linpack** dense linear algebra benchmark, with a metric of floating point operations per second (flops). It has been updated every 6 months (June and November) since 1993, and as such can give some insight into the characteristics and trends of one class of both data center scale hardware systems and high end floating point intensive applications.

4.5.1 Aggregate Performance

Figure 4.11 gives two performance metrics for the top 10 from each list since 1993: $\mathbf{R}_{peak}$ is the theoretical peak performance of the machine in gigaflops, and $\mathbf{R}_{max}$ is the best measure floating point count when running the Linpack benchmark. The top 10 were chosen for study, rather than all 500, because they tend to represent the best of the different architectural tracks without introducing a lot of duplication based on replication size alone. Even so, as can be seen, the spread in performance is uniformly about an order of magnitude.

[9] http://www.top500.org/

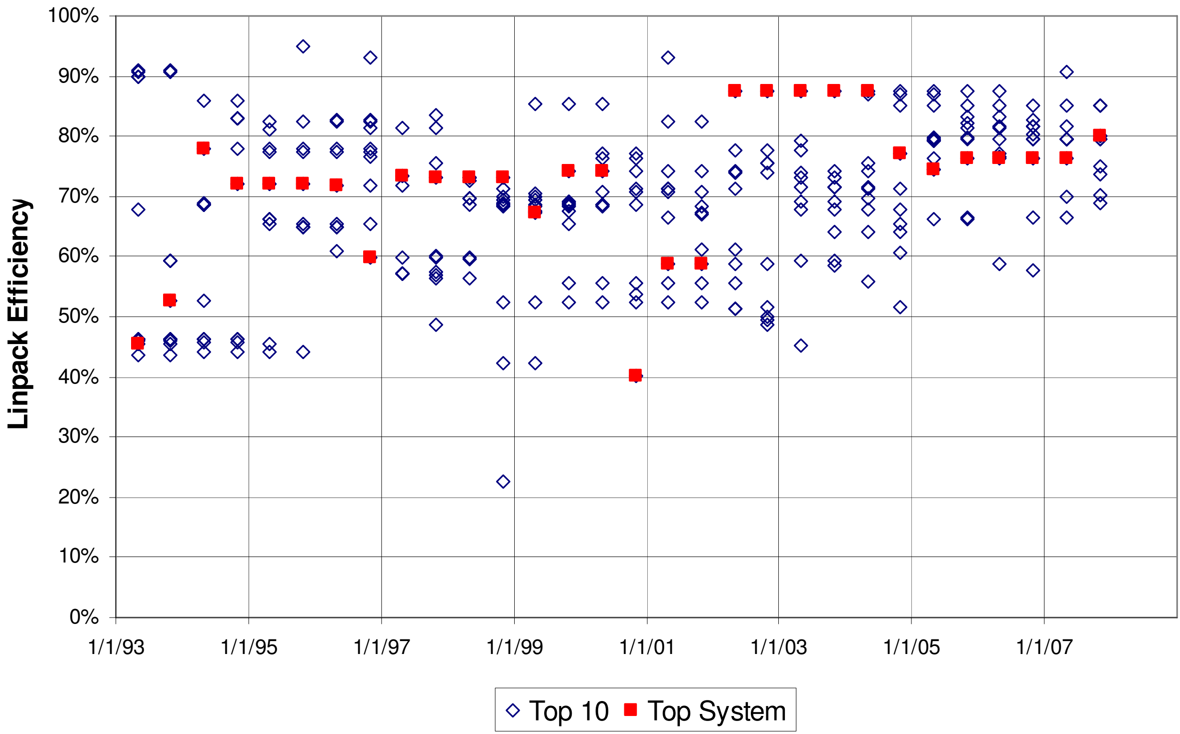


Figure 4.12: Efficiency for the Top 10 supercomputers while running Linpack.

As can be seen, the CAGR for R_{peak} is about 1.89 and for R_{max} it is 1.97. This translates into about a 10X growth every 41-43 months, or 1000X in between 10 and 11 years. If 2010 is nominally the year of the first sustained petaflops R_{peak}, then if these rates were sustainable, it will be 2020 for the first exaflops machine.

4.5.2 Efficiency

We define **efficiency** as the ratio of the number of useful operations obtained from a computing system per second to the peak number of operations per second that is possible. Figure 4.12 graphs this metric as the ratio of R_{max} to R_{peak}, with the top system in each list indicated as a square marker. The major observations from this chart are that:

- the range has historically been between 40 and 90
- there has been a tightening of the lower end over the last 5 years,
- the efficiency of the top performing system has not always been the highest, but has been in a tighter range from 70 to 90

4.5.3 Performance Components

The performance of these machines can be expressed as the product of three terms:

- **Parallelism**: the number of separate "processor" nodes in the system, each nominally capable of executing a separate thread of execution (none of the machines on the Top 500 are to date multi-threaded).

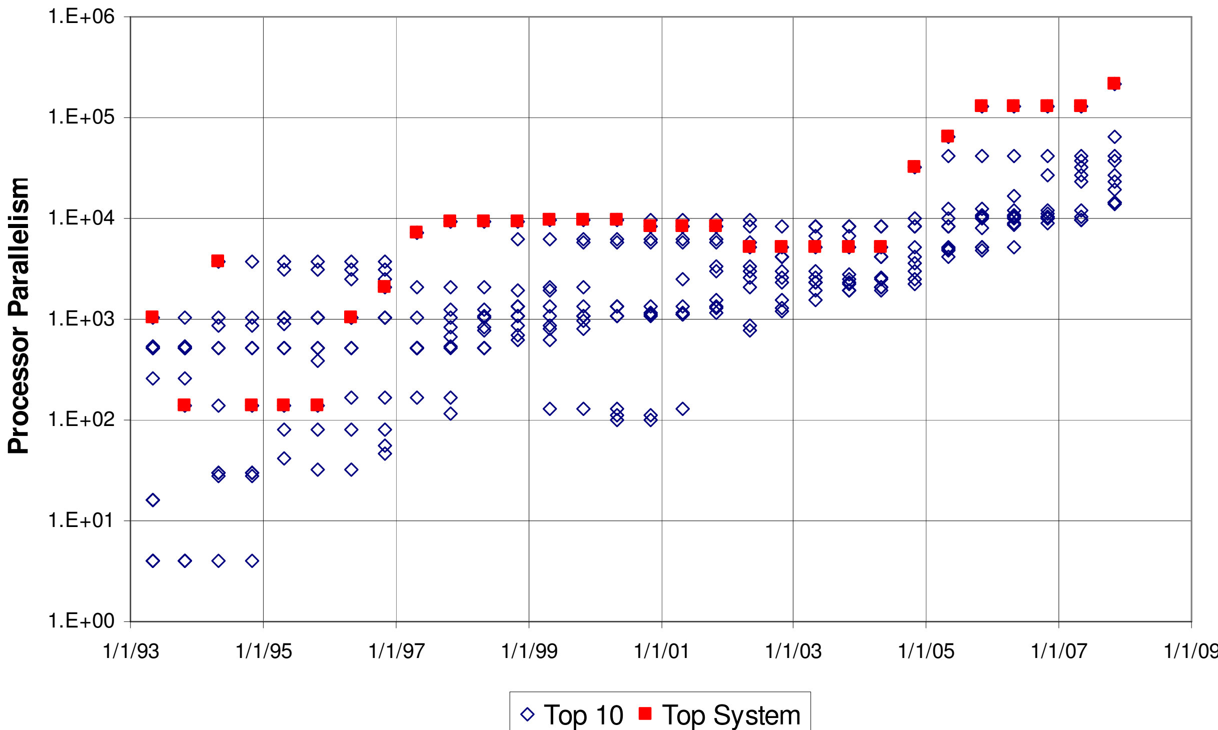


Figure 4.13: Processor parallelism in the Top 10 supercomputers.

- **Clock**: the rate at which operations can be performed in any single processor.
- **Thread Level Concurrency (TLC)**: this is equivalent to the number of separate operations of interest (floating point for these Linpack-based kernels) that can be executed per cycle.

We note that if we look at the activities in each machine cycle, then the total number of processors times the maximum TLC gives some sense to the "overall concurrency" in a system, that is the total number of separate hardware units capable of parallel operation.

Each of these topics is discussed separately below. However, the clear take away from them is that since 1993, the performance gains have been driven primarily by brute force parallelism.

4.5.3.1 Processor Parallelism

Parallelism is the number of distinct threads that makes up the execution of a program. None of the top systems to date have been multi-threaded, so each processor as reported in the Top 500 list corresponds to a single thread of execution, or in modern terms a "single core." Figure 4.13 then graphs this number in each of the top 10 systems over the entire time frame, with the top system highlighted.

The key observation is that the top system tended to lead the way in terms of processor parallelism, with the period of 1993 to 1996 dominated by systems in the 100 to 1000 region, 1996 through 2003 in the 10,000 range, and 2004 to now in the 100,000 range.

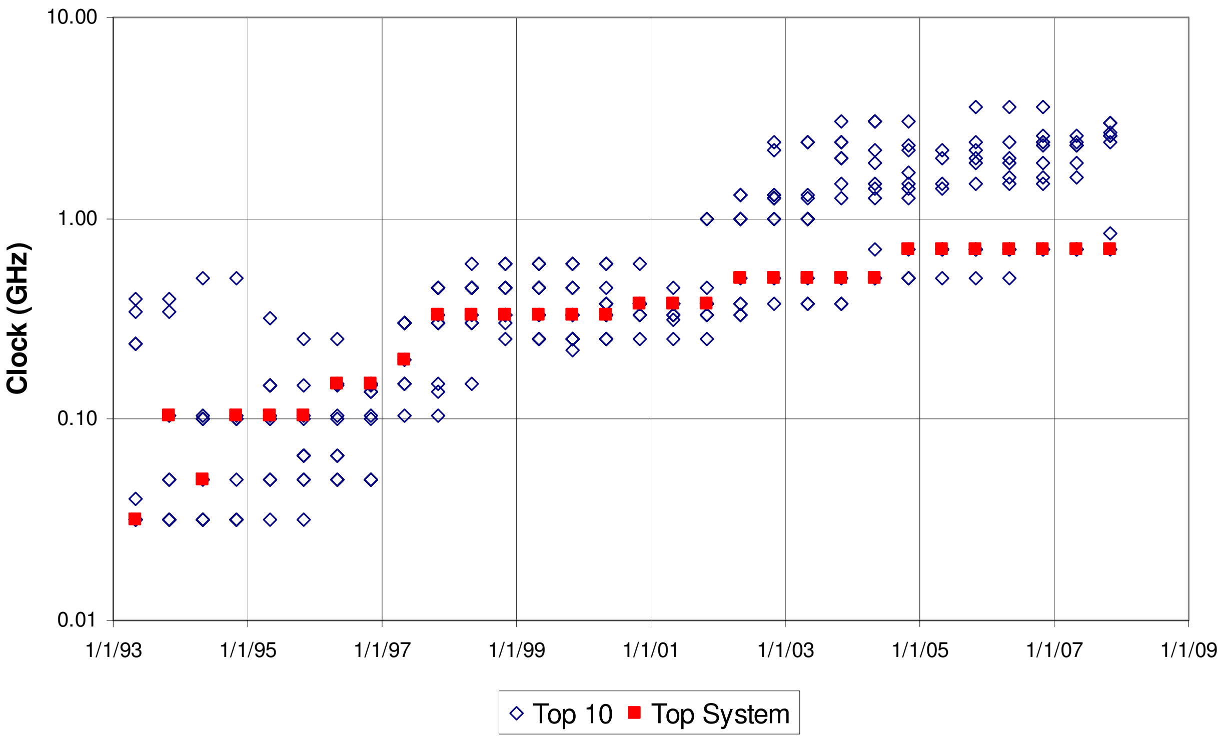


Figure 4.14: Clock rate in the Top 10 supercomputers.

4.5.3.2 Clock

The **clock** is the rate at which operations can be performed in any single processor. Figure 4.14 graphs this for the top 10 systems, with the top system highlighted as before. Here the growth rates are much different that for parallelism. While the highest clock rates for Top 10 systems have in general been in line with the best of then-leading edge technology, the clock rate growth for the top system has been extremely modest, with a range of from 500 to 700 MHz for well over a decade.

4.5.3.3 Thread Level Concurrency

Thread Level Concurrency (**TLC**) is an attempt to measure the number of separate operations of interest that can be executed per cycle. For the Top 500 to date the operation of interest has been floating point operations (based on the Linpack-based kernels). It is computed as the performance metric (R_{max} or R_{peak}) divided by the product of the number of processors and the clock rate as given in the lists.

TLC is meant to be similar to the **Instruction Level Parallelism** (**ILP**) term used in computer architecture to measure the number of instructions from a single thread that can either be issued per cycle within a microprocessor core (akin to a "peak" measurement), or the number of instructions that are actually completed and retired per second (akin to the sustained or "max" numbers of the current discussion). Figure 4.15 graphs both the peak and the max for the top 10 systems, with the top system highlighted as before.

As can be seen, these numbers reflect the microarchitecture of underlying processors. Those systems with peak numbers on the 16 to 32 range have for the most part been vector machines. Virtually all of the rest have been 4 or less, both in max and peak, and correspond to more or less

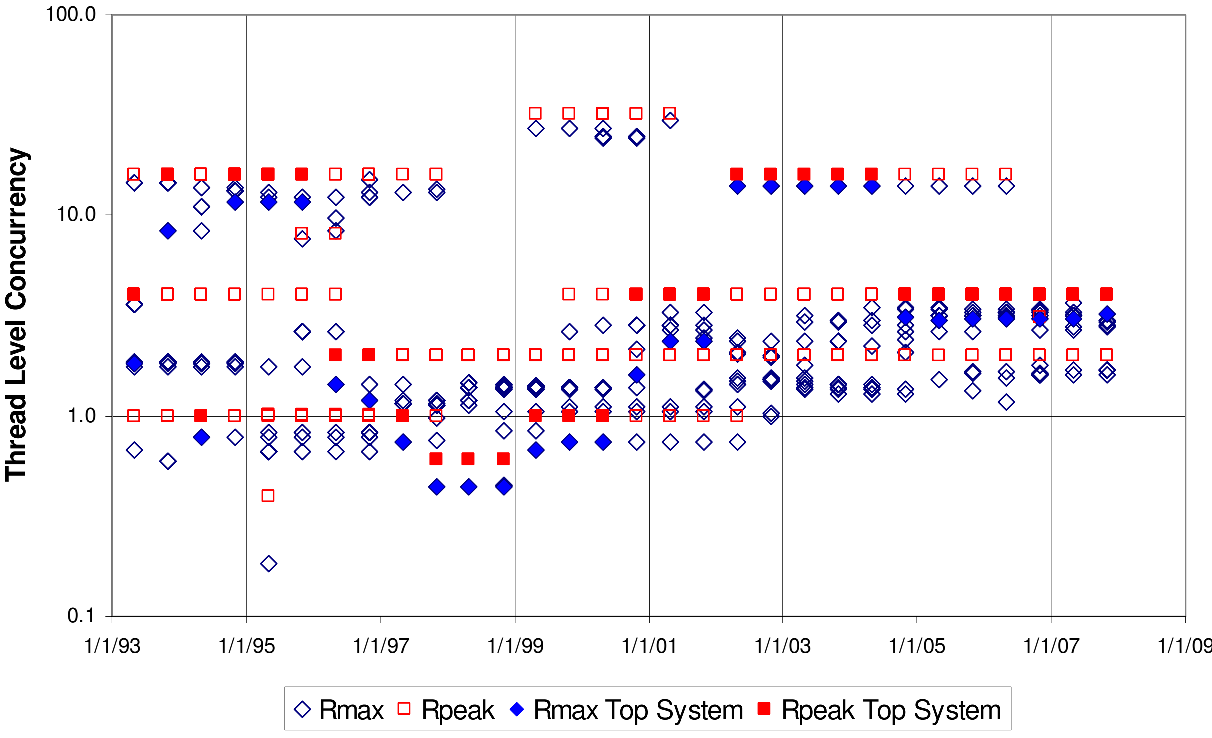


Figure 4.15: Thread level concurrency in the Top 10 supercomputers.

conventional microprocessor designs with just a very few floating point units, each often counted as being capable of executing up to two operations per cycle (a **fused multiply-add**).

With the exception of the Earth Simulator, virtually all of the top systems over the last decade have been these relatively low TLC designs.

4.5.3.4 Total Concurrency

Total concurrency is the total number of separate operations of interest that can be computed in a system at each clock cycle. For the Top 500 these operations are floating point, and such a measure thus reflects (within a factor of 2 to account for fused multiply-add) the total number of distinct hardware units capable of computing those operations.

For our Top 10 list, this metric can be computed as the number of processors times the peak TLC. Figure 4.16 graphs these numbers, with the top system highlighted as before. Unlike just the processor parallelism of Section 4.5.3.1 (which is very stair-stepped), and the clock of Section 4.5.3.2 and the TLC of Section 4.5.3.3 (which have at best very irregular trends), the Top 1 system tends to ride at the very top of the curve, and to advance in a monotonic fashion. To see this more clearly, a trend line is included that touches the transitions of the top system almost perfectly. The CAGR for this line is about 1.65, meaning that a 10X increase in concurrency is achieved every 4.5 years, and a 1000X in 13.7 years. We note that this CAGR is equivalent to about 2.17X every 18 months, which is somewhat above Moore's Law, meaning that the rate in increase in separate computational units is increasing faster than the number that can be placed on a die, implying that relatively more chips are being added to the peak systems of each succeeding generation.

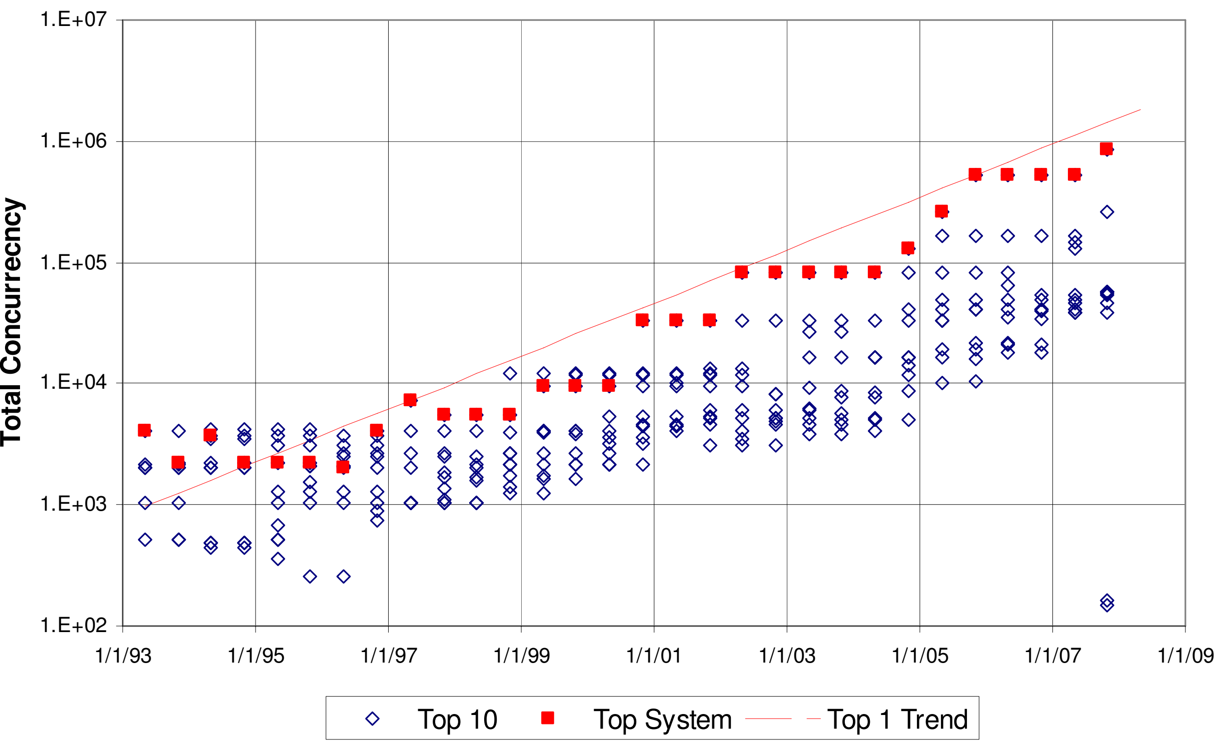


Figure 4.16: Total hardware concurrency in the Top 10 supercomputers.

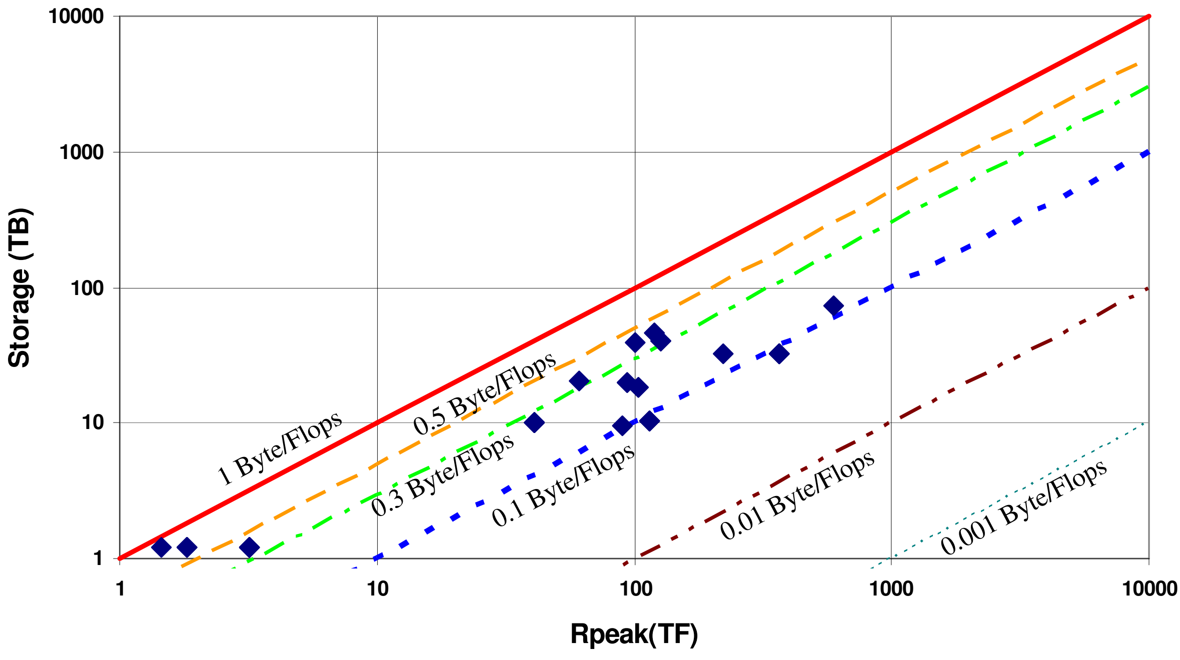


Figure 4.17: Memory capacity in the Top 10 supercomputers.

4.5.4 Main Memory Capacity

Another important aspect of such machines that has a direct affect on the actual problems that can be placed on them is the amount of directly addressable memory that is available to hold the data. Figure 4.17 plots main memory capacity versus R_{peak} for a set of the most recent top 10 machines. Also included are lines of "bytes per flops."

As can be seen, virtually all the recent machines are clustered in the 0.1 to 0.5 bytes per flops range. If these trends continue, we would thus expect that the near term Petascale systems may have main memory capacities in the 100-500TB range.

When we look at future machines, two points may be worth considering. First, most of the current Top 10 machine architectures are clusters that employ message passing. Thus there may be some extra memory consumed in duplicate programs, and in data space for data exchange (i.e. "ghost node" data) that might not be present in a shared address space machine. Second, the x-axis in Figure 4.17 is R_{peak}, not R_{max}. The latter reflects at least an executing program, and not just a simple"100% busy FPU" calculation. With real efficiencies in the 70-90%, the storage per "useful" flop is actually somewhat higher than shown.

Chapter 5

Exascale Application Characteristics

This chapter attempts to develop a characterization of a class of applications that are liable to be significant to Exascale systems. Because of our long history of difficulties inherent in porting such applications to the largest computing systems, and because the middle departmental class of Exascale systems is still at the scale as today's largest, the bulk of the discussion will be on applications relevant to data center class systems.

At these scales, the overall performance of a system is a complex function of many parameters of the system's design, such as logic clock speeds, latencies to various memory structures, and bandwidths. Our discussion will attempt to analyze effects on performance due to changes in one or multiple parameters.

Section 5.1 first defines a graphical representation that will be used for exploring these effects. Section 5.2 describes the concepts of balance and the von Neumann bottleneck in terms of applications and performance. Section 5.3 then describes a typical application of significance today, and how a machine that is "balanced" in its design parameters might behave as those parameters are changed. Section 5.4 briefly discusses how different classes of applications may be composed of more basic functionality. Section 5.5 then performs an analysis of several strategic applications of today that are particularly sensitive to memory system parameters.

Section 5.6 then focuses on understanding what it means to "scale performance" of applications by 1000X. Section 5.7 then looks at several applications and how they in fact scale to higher levels of performance.

Section 5.8 then summaries what this all means to a potential Exascale program.

5.1 Kiviat Diagrams

The performance of real applications on real computers is a complex mapping between multiple interacting design parameters. The approach used here to describe such interactions is through use of **Kiviat diagrams**, or "radar plots." In such diagrams a series of radial axes emanate from the center, and a series of labeled concentric polygonal grids intersect these axes. Each axis represents a performance attribute of the machine that might be individually improved, such as peak flops, cache bandwidth, main memory bandwidth, network latency etc. Each polygon then represents some degree of constant performance improvement (usually interpreted here as a "speedup") of the application relative to some norm, with '1" (the degenerate polygon at the center) representing a baseline performance of the machine with no modifications. The units of the axis are normalized improvement.

A dark lined polygon drawn on this diagram then represents the effects on application per-

formance resulting from improving the design parameters associated with each axis by some fixed amount, such as 2X. Thus by moving from axis to axis, and seeing where this dark line lies in relation to the labeled grid polygons, one can tell the relative effect of each component taken in isolation. Vertices of this dark polygon that are further out from the origin than other vertices thus correspond to attributes where the relative change has a larger effect on overall performance.

In many cases, some axes are labeled for combinations of parameters. In such cases, the resultant measurement is performance when *all* those parameters are simultaneously improved by the same amount.

5.2 Balance and the von Neumann Bottleneck

The term "balanced design" refers to a design where some set of resources, which individually "cost" about the same, are used at about the same levels of efficiency, so that adding more of one resource without adding the same amount of the others adds very little to overall system performance. In terms of computing, such balancing acts typically revolve around the computational versus memory access resources. The von Neumann bottleneck, i.e. limited data transfer rate between the processor and memory compared to the amount of memory, has been a performance limiter since the inception of the digital computer. From Wikipedia: the term "von Neumann bottleneck" was coined by John Backus in his 1977 ACM Turing award lecture. According to Backus:

> Surely there must be a less primitive way of making big changes in the store than by pushing vast numbers of words back and forth through the von Neumann bottleneck. Not only is this tube a literal bottleneck for the data traffic of a problem, but, more importantly, it is an intellectual bottleneck that has kept us tied to word-at-a-time thinking instead of encouraging us to think in terms of the larger conceptual units of the task at hand. Thus programming is basically planning and detailing the enormous traffic of words through the von Neumann bottleneck, and much of that traffic concerns not significant data itself, but where to find it.

This bottleneck is exacerbated by modern architectural trends, and is particularly irksome in scientific computing that consists primarily in evaluating mathematical expressions exemplified by a simple example: $A = B + C$. To carry out this calculation, the computer must fetch the arguments B and C into the processor from wherever they reside in the memory, then carry out the mathematical operation, then store the result A back into memory; unfortunately the fetching and storing steps can take several of orders of magnitude longer than the mathematical operation (+) on today's processors.

This imbalance is growing as an indirect result of Moore's Law, which states that the density of transistors on a chip doubles every 18 months or so. The resulting smaller logic and shorter signaling distances has primarily been used to enable ever higher processor clock frequencies, with resulting faster processors for carrying out mathematical operations, while the absolute distance to memory off-chip remains about the same, and thus the *relative* time to access this data (time measured in processor clock cycles per access) becomes greater with every turn of Moore's Law. This phenomenon has been termed "**red shift**" because, analogous to our expanding universe, computers seem to recede in relative distance to their own local memory and storage, and each other, with every turn of Moore's Law. Another implication of red shift is that modern computers spend most of their time moving data, rather than performing mathematical operations, when running today's memory intensive applications. We also observe that more and more applications

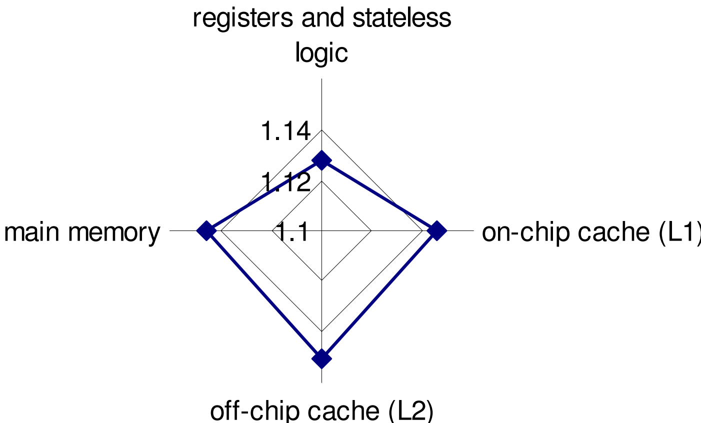


Figure 5.1: Predicted speedup of WRF "Large".

become memory-bound in the same sense (i.e. they spend the greatest percentage of their time moving data).

As a historical context: the Cray XMP[157], the world's fastest supercomputer in 1983-1985, was a "balanced machine" in that it could perform 2 fetches and 1 store from main memory, as well as up to 2 floating-point mathematical operation, per cycle. Thus operations such as vector inner products could run at the maximum memory rate, and yet still utilize 100% of the floating point units capabilities.

In contrast, today's fastest supercomputer, the IBM BG/L[48], can accomplish 4 floating-point mathematical operations in 1 (much shorter) cycle yet requires around 100 such cycles to fetch just 1 argument from memory, at a rate of somewhere between 0.5 and 0.8 bytes per cycle if the data must come from local memory. This explains why the XMP would spend about 50% of its time moving data on a memory intensive code, but BG/L may spend 99% of its time moving data when executing the same code.

5.3 A Typical Application

Let us consider a "typical" scientific computation of today. As depicted in Figure 5.1, Datastar, SDSC's IBM Power4-based supercomputer, is approximately a "balanced" machine for the **WRF** (**Weather Research and Forecasting**) benchmark when run on 256 CPUs with various system parameters are doubled relative to IBM Power4 on a well-known input (the HPCMO "Large" test case). This balance comes from the observation that the predicted speedup up is about the same factor (to one decimal place of precision) if any one of the following occur:

1. the arithmetic logic and registers are clocked faster, by a factor of 2.
2. the latency to the on-chip L1 cache is halved without improving anything else,
3. the latency to the off-chip L2 cache is halved, again exclusive of other improvements,
4. the latency to main memory halved.

As can be seen by the dark polygon of Figure 5.1, the relative performance improvement for any one of these individual improvements is within a percent or two of only 14%. The reason for

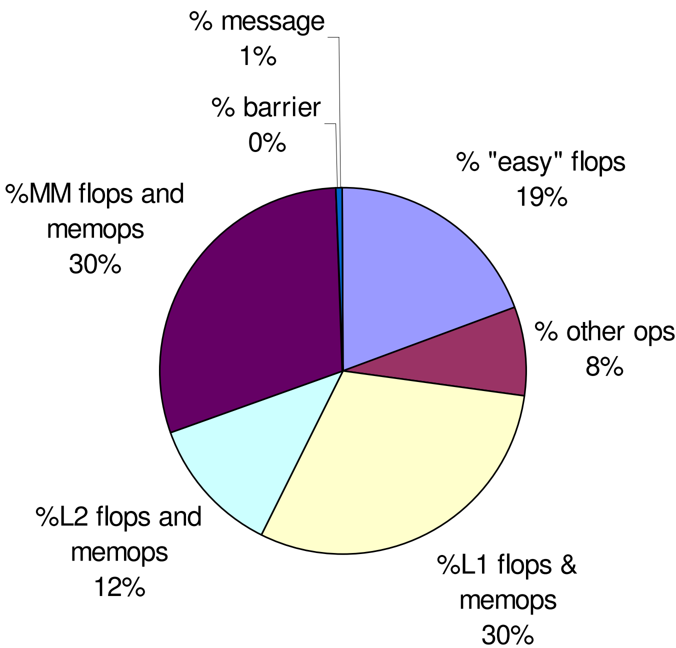


Figure 5.2: Time breakdown of WRF by operation category.

this balance is clear from consulting Figure 5.2 below which shows the time fraction spent by WRF in executing operations in different categories on DataStar, defined as follows:

- “Easy” flops that reuse data in registers (there is no associated memory fetch to get in the way of their completion).
- “Other ops” include address arithmetic, branch compares and etc.
- “L1 flops and memops” (memory operations) are either fetches from L1, or flops that cannot be issued until such a fetch completes.
- “L2 flops and memops” are either fetches from L2, or flops that cannot be issued until such a fetch completes.
- “MM flops and memops” are either fetches from main memory, or flops that cannot be issued until such a fetch completes.
- “Barrier MPI” are MPI global barriers for synchronization (minuscule contribution for WRF).
- “Message MPI” are MPI messages involving data exchange.

Note that the “easy” + “other” pie slice is approximately the same size as the L1 and MM slices while L2 is slightly smaller. An almost exactly balanced machine for this problem would result if DataStar’s L2 latency were a little longer (approximately 50 cycle) and then the L2 slice would be about the same size as the others. Also note that WRF is not spending a high percentage of its time doing MPI. In general, we would not consider a machine or application that is spending a large fraction of its time doing communications, or one that would benefit as much from improving communications as from improving the processor and memory, as “balanced” because we think of communications as being pure overhead (that is, something to be minimized). And usually, for

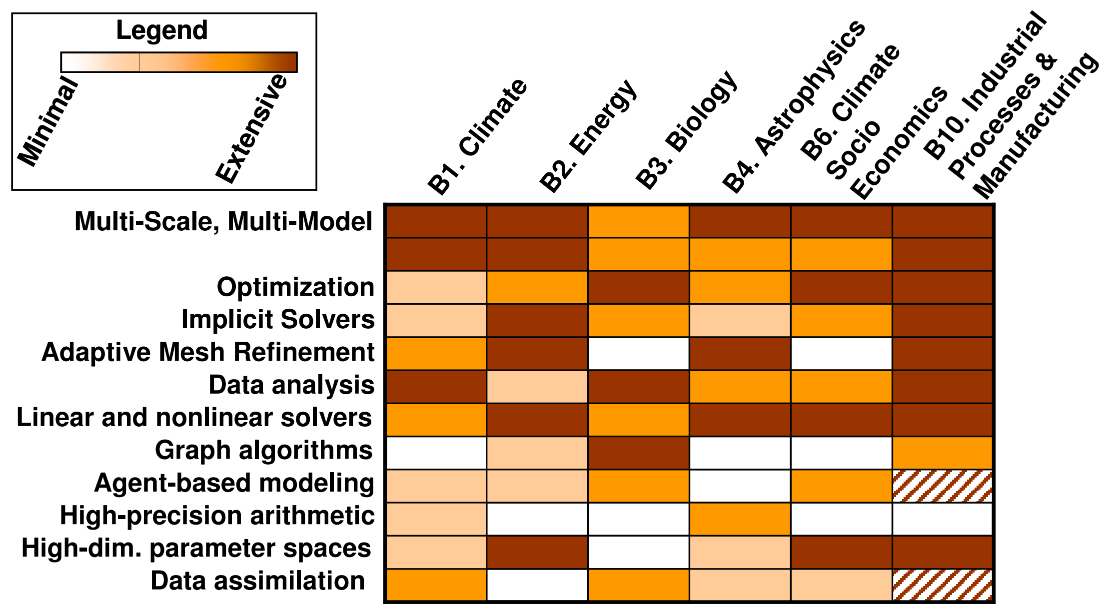


Figure 5.3: Application functionalities.

a parallel application such as WRF, if communications time is the same or greater than compute time in the other categories, then the application is past the knee of its scaling curve and should just be run on fewer processors (this may be not possible if the problem is too large to fit in the physical memory of a reduced number of processors).

By Amdahl's Law (which limits the rate of return for speeding up just one part of a multi-part task), Moore's Law, even if it continues to hold, will not speed up memory-bound applications such as WRF by more than a few percent. For example the MM flops and memops section of the pie chart would not shrink, unless new technologies are applied to improve or mitigate red shift. If doubling of transistor density per chip is just devoted to speeding up mathematical operations that are already 100 times faster than argument fetching and storing in code that (like the example) has about an equal mix of both, then not much performance improvement of memory intensive applications will be gained.

In either case it is clear that widening, mitigating, or eliminating the Von Neumann Bottleneck must be a thrust of research to enable Exascale computing as it lies in the path to increasing calculation speed by 1000x.

5.4 Exascale Application Characteristics

Exascale applications, particularly for the departmental and data center classes of problems, are liable to be rather complex in their structure, with no single overriding attribute that governs their characteristics. To get some insight into this, Figure 5.3[1] looks into several classes of applications taken from [54], and what kinds of lower level functionalities would be found in them. In this figure, the columns represent classes of applications and the rows represent different types of algorithms that might be employed in one form or another. The degree of shading in each box represents the degree to which such algorithms play an important part of the application.

[1] Figure courtesy of D. Koester from [84]

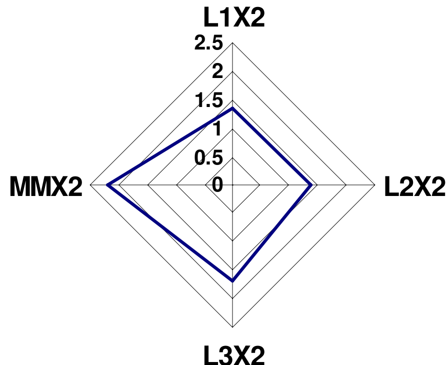


Figure 5.4: Predicted speedup of AVUS to latency halving.

The key take-away from this is that the days of very regular, simplistic, applications are over, and that applications of the future will involve a richer and more diverse suite of underlying algorithms, all of which must perform well on an underlying machine architecture in order for a system to deliver truly significant and efficient performance.

The following sections delve in more detail into some of these applications and algorithms.

5.5 Memory Intensive Applications of Today

Of all the performance parameters discussed earlier, memory latency in all its forms is often the dominant one. Applications for which this is particularly true are called **memory intensive**, and are discussed here.

5.5.1 Latency-Sensitive Applications

Figure 5.4 through 5.7 depict Kiviat diagrams predicting the performance speedup of several strategic applications due to a halving of latency to different steps in the target machine's memory hierarchy (thus shrinking the length of the bottleneck). These applications cover a spectrum of strategic science uses:

- **AVUS** (Air Vehicle Unstructured Solver) is a CFD code used in airplane design,
- **WRF** is a weather prediction code mentioned above,
- **AMR** (**Adaptive Mesh Refinement**) is a benchmark representative of many domains of computational science,
- **Hycom** is an ocean modeling code.

These applications or their variants are run in a production mode by federal agencies including DoD, DoE, NSF, and NOAA, and consume tens of millions of supercomputer hours annually. They are not chosen for this study solely because they are memory intensive but because they are scientifically and strategically important.

The speedups graphed above in Figure 5.4 through 5.7 are forecast by performance models relative to an existing base system (a 2048 processor IBM Power4 based supercomputer). Each radial axis of these graphs represents the predicted speedup of halving the latency (in Power4

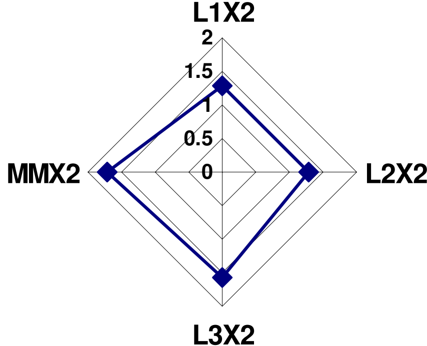


Figure 5.5: Predicted speedup of WRF to latency halving.

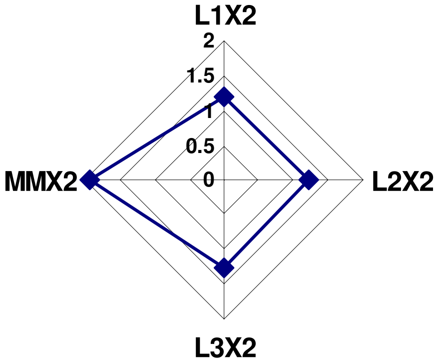


Figure 5.6: Predicted speedup of AMR to latency halving.

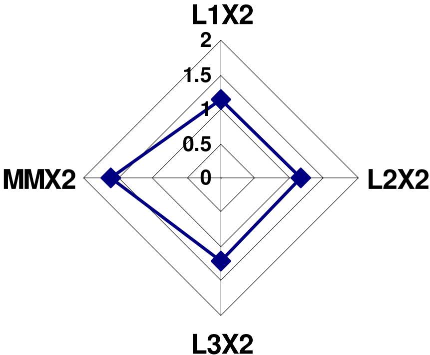


Figure 5.7: Predicted speedup of Hycom to latency halving.

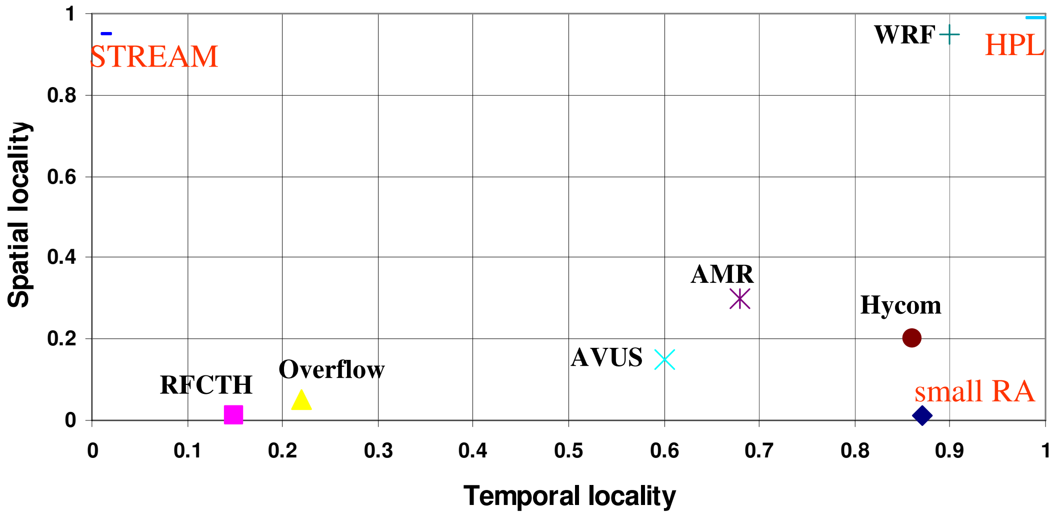


Figure 5.8: Spatial and temporal locality of strategic applications.

cycles) of memory accesses to successive levels of the memory hierarchy ("L1" is the level 1 cache, "L2" is the level 2 cache, etc. "MM" is main memory). The units of each axis are predicted speedup of the application's total run-time relative the base system's speed if the base machine were to be improved in just that dimension. For example, a value of 1 on an axis would represent no predicted speedup of the application for halving the latency to that level of the memory hierarchy while a value of 2 would predict the most speedup one could reasonably expect from halving the latency of that level of the memory hierarchy (and imply the application is only doing memory references to that level of the memory hierarchy).

Of note is that all these applications are forecast to benefit more from decreasing main memory latency than from decreasing L1 (on chip) latency. So Moore's law, even if it continues to hold, will not help these applications much unless it is exploited in a way different from the ways it has been historically.

5.5.2 Locality Sensitive Applications

The performance of memory intensive applications such as the above on today's machines depends mostly on the application's spatial and temporal locality and the ability of the machine to take advantage of those to mitigate the effect of the von Neumann Bottleneck. Here by **locality** we mean the likelihood of a memory reference will be in some sense "local" to some prior memory access. **Spatial locality** is the tendency of a computation to access memory locations that are contiguous by address location to prior references - these addresses are then amenable to prefetching which can improve performance by hiding latency when they are accessed. **Temporal locality** is the tendency of a computation to access memory locations that it has already accessed - these addresses are amenable to caching which can improve performance by storing the data in small, near-to-the-processor memories which can be accessed with reduced latency.

Figure 5.8 shows spatial and temporal locality of a number of strategic applications (including those of Figures 5.4 2 through 5.7) where spatial and temporal locality are assigned a numeric score in the range [0,1] as defined in [155], with a score of 1 being the highest possible locality (every reference is local to some previous one) and a score of 0 being the least (there is no locality correlation of any kind).

To help put these applications into perspective with respect to locality, they are plotted in Figure 5.9 along with a suite of smaller kernels (highlighted):

- **HPL**, a small-footprint, cache-friendly benchmark with near maximal temporal locality used to generate the Top500 list,
- **STREAM**, a unit-stride benchmark with near perfect spatial locality but no reuse,
- **small Random Access** (**RA**), a benchmark with almost no locality whatever.

HPL, for comparison, runs very efficiently on modern machines where it gets "balanced" performance, that is, it usually achieves about 1 cycle for a memory reference to on-chip L1 and thus can run at near theoretical peak floating-point issue rate.

STREAM gets reasonably good performance on modern machines as it uses a whole cache line and benefits from prefetching; when a large-memory STREAM test case is run its performance is limited by bandwidth (rather than latency) to main memory.

Small Random Access performs poorly on most modern machines, at about the latency of main memory (in the neighborhood of greater than 100 cycles per memory operation). This is despite the fact that it fits in cache like HPL, but jumps around without striding through cache lines.

Real applications fall between these kernel extremes both in locality and in performance as shown in Figure 5.9 where performance in terms of something proportional to "cycles per instruction" (CPI)/ondex(CPI) is plotted on the Z axis. Higher numbers on this Z axis correspond to "lower performance." As can be seen from this Figure, applications with higher locality are "easier" for today's machines to run fast; they average lower cycles per operation by either doing more memory references in nearer caches (reducing latency) or prefetching data from future addresses (mitigating latency by Little's Law).

Unfortunately, however, there are strategic applications that are even more memory intensive than those graphed in Figures 5.4 through 5.7 such as **Overflow**, a CFD code run by NASA, and **RFCTH**, a blast physics code. These perform but poorly on all of today's machines.

As a general rule of thumb: these applications require a minimum of 0.5GB of local main memory for each 1 GFlops of processor (and some such as Overflow and RFCTH require more than 2x that).

Also as a general rule the amount of level 2 and level 3 cache required by these applications corresponds inversely to their temporal locality score (see reference for details). For example WRF is relatively cache-friendly code and can get by today with an L2 cache size of 1MB while Overflow and RFCTH need at least a 12 MB L3 cache (and would perform better if it were a 12 MB L2).

5.5.3 Communication Costs - Bisection Bandwidth

Generally speaking as depicted in Figure 5.10, the fraction of time spent in communications increases for these applications as a fixed problem size is solved with more processors. In such cases, increasing the number of processors results in smaller amounts of data that is "close to" each processor. In other words an increase in concurrency is offset by a decrease in locality; and at some point diminishing returns are reached.

A current standard metric for such communication costs is **bisection bandwidth**- if a system is arbitrarily divided into two halves, the bisection bandwidth is the minimum bandwidth that is still guaranteed to be available between these two halves. How well this metric holds up in the future will depend on the characteristics of the applications and the patterns of how different sites of computation must communicate with other sites, such as:

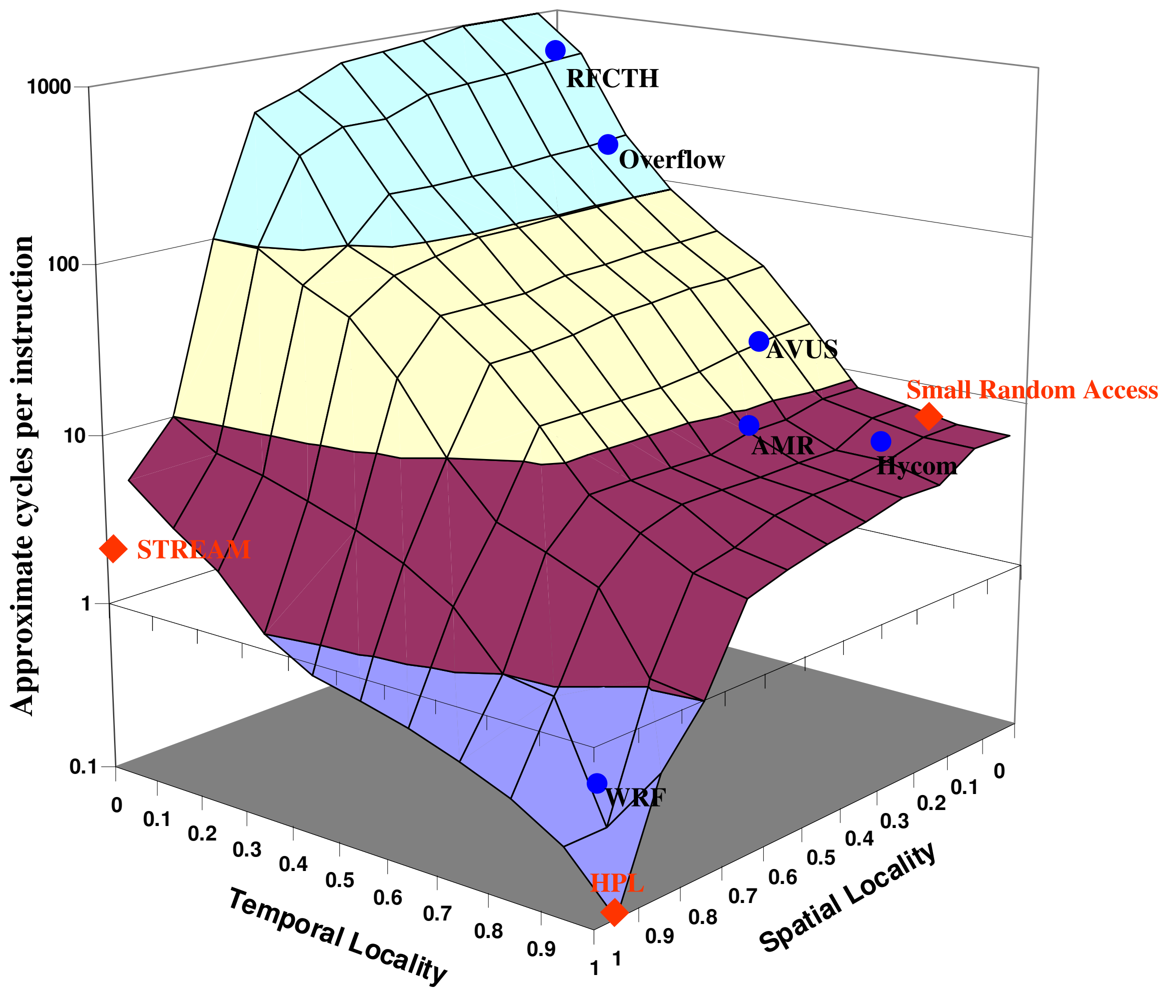


Figure 5.9: Performance strategic applications as a function of locality.

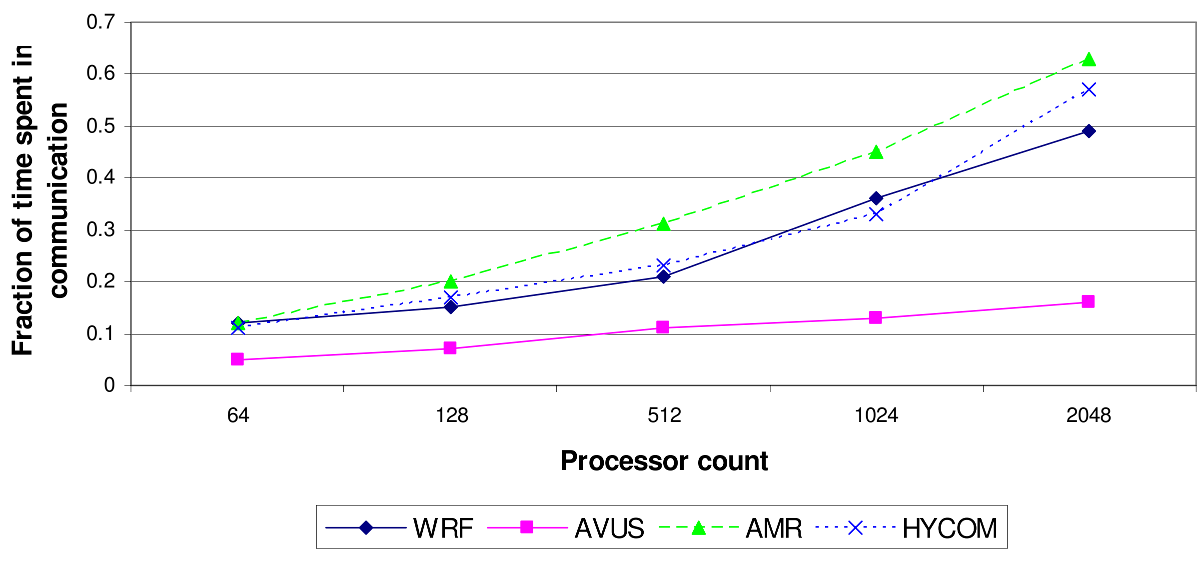


Figure 5.10: Growth of communications overhead.

- scientific and engineering codes such as PDEs and 3D meshes:
  - structured grids involve nearest neighbor communications,
  - unstructured grids where indirection through link tables is the norm,
  - adaptive mesh refinements where a fair amount of data must be moved around the machine at unpredictable times to rebalance and load-balance the application.
- multi-scale and multi-model applications that act like amalgams of multiple "single algorithm" applications as above, but must do so simultaneously, meaning that any sort of regularize mapping between data and processors becomes very difficult, and the aggregate bandwidth patterns become "random."
- new applications such as those involving large unstructured graphs may simply be impossible to partition to minimize communication for long periods of time.
- as discussed in Section 6.7.4, the growing need to "copy memory" for application-level data checkpointing to increase the resiliency of the system to internal faults.

5.6 Exascale Applications Scaling

The applications discussed in the previous section require around 0.5GB per Gflops of computation when run on today's high end systems. How such numbers, and similar parameters for other applications, will change as we move towards Exascale is crucial to sizing the complexity and basic architecture of Exascale computers, and is the subject of this section.

5.6.1 Application Categories

To start such discussions, we consider four categories of applications in terms of their "scalability," and ordered in approximate ease of implementation:

I. Those applications that would solve the same problem as today but with a 1000x more data points. This represents using the "same physics" and the "same algorithm," but over larger surfaces, and is normally called "weak scaling." In this category might be, for example, global weather models at sub 1km resolution (effectively solving a WRF calculation at the granularity used today to forecast hurricanes but at the breadth of the entire earth's atmosphere rather than over the Gulf alone).

II. Those that could solve the same problem as today but 1000x faster. In this category for example might be real-time CFD to stabilize a physically morphable airplane wing (effectively recalculating every few seconds an AVUS calculation that now takes hours). Another example would be running an operational weather forecast of a hurricane storm track in minutes rather than days (thus improving advanced warning).

III. Those would that would solve the same problem, at the same size, as today, but with 1000x more time steps. In this category for example might be local weather models at climatic timescales (effectively solving a WRF calculation at the size today used for medium-term weather forecasting but projecting out for centuries).

IV. Those that would solve the same problem as today but at 1000x more resolution (usually requiring increased physics and chemistry in every cell). In this category might be for example global ocean and tide models such as Hycom to include micro-features such as wave refraction and coastal turbulent mixing.

Category I applications are those that often go by the term "embarrassingly parallel" where it is obvious how to partition instances up into almost arbitrarily many pieces that can all be computed in parallel. Thus systems with more parallelism in them can in fact be used by such applications for near linear increases in performance.

Categories II, III, and IV are all versions of **strong scaling**, with category II being the conventional version, but category IV being the most challenging because of the need to model more science with more computation. In all of these categories, the relationship between parallelism and performance is highly non-linear.

Clearly Exascale applications could have aspects of any combination of the above; for example one could perform a calculation at 100x resolution and 10x more time-steps, for a total of 1000x more computation than is possible today.

5.6.2 Memory Requirements

We next want to consider three aspects of the **memory footprint** of Exascale applications:

1. their total memory size requirements,
2. the size of working sets (**locality clusters**) in their data that in turn would determine the sizes of local main memory and local caches they require, and
3. the latency and bandwidth requirements they would place on each level of the memory hierarchy, including global interconnect.

As to applications in category I above, the total memory footprint would increase 1000x but the sizes and latencies of the local memory hierarchy would not need to change relative to today. However, to enable scalability and efficient deployment of more processors, the interconnect between processors would have to improve in latency, or trade latency for bandwidth, in proportion to some function of the topology (i.e. square root of 1000 $\approx$ 32-fold for a torus) to preserve the performance of global data communications and synchronization.

As to category II above, the total memory footprint would not change. But latency would have to be improved 1000x to each level of the memory hierarchy unless a combination of code tuning and machine features could unlock sufficient parallelism in memory references to cover some of the additional latency requirements via increased bandwidth[2]. In that case it would be the bandwidth that would need to be improved proportionally.

Category III requirements are the same as for category II.

Category IV applications are similar to category I unless the higher resolution problem is highly parallelizable (and this depends on the nature of the science problem on a case-by-case basis) in which case it resembles more category II. In other words, if additional resolution or additional physics in the local calculation does not improve coarse-grained task parallelism, then improved latency (or improved local memory bandwidth and ILP) is the only way to increase memory performance. If on the other hand, additional resolution or physics results in greater task parallelism,

[2]Bandwidth can be traded for latency by Little's Law

then one can achieve increased performance by adding more processors that each look like today's, a la category II.

Thus we have four categories of potential Exascale applications, and several dimensions of Exascale memory technology to consider that could speed them up 1000x. Futuristic AVUS is perhaps an exemplar of Category II, variants of WRF could be category I, II or III, and extensions to Hycom of category IV.

5.6.3 Increasing Non-Main Memory Storage Capacity

Historically, storage beyond the DRAM of main memory has fallen into several categories:

- **scratch storage** used for both checkpointing and for intermediate data sets needed for "out of core" solvers.
- **file storage** used for named input and output files that will persist beyond the lifetime of the application execution.
- **archival storage** used for long term storage of data.

5.6.3.1 Scratch Storage

Scratch storage in the past has been primarily driven by the size of main memory and the need to periodically checkpoint (as discussed in Section 6.7.4. To date its implementation has been as a large number of disk drives.

In the future, however, there will be an increasing need for larger scratch storage uses, such as for applications that dynamically construct data derived models, that have such large memory needs that "out of core" algorithms are necessary, and in capturing performance monitoring data that is used by the system to do dynamic load-balancing and performance tuning.

Thus, it should be expected that such storage needs will grow on the order of 10 to 100X the size of main memory.

5.6.3.2 File Storage

File storage represents persistent data sets that have real value to the end users (either as input files or summary outputs), and may be shared among different applications that run at different times. In addition, as systems become larger, at least a subset of the performance monitoring data mentioned in the scratch storage discussion may need to be saved, along with fault data, to look for trends that may require either repartitioning of future application runs or scheduling down time for maintenance.

Besides the capacity issue, there is also a significant **metadata** problem with file storage, where metadata relates to the directory information needed to both locate individual files in the system, and monitor the state of those files. Today, there are many applications that even on sub petaflops machines keep literally hundreds of thousands of files open at a time, and that number is growing.

At the end of the day, file system size is thus also at least linear with main memory, with perhaps a 1000X multiplier present for Petascale systems.

5.6.3.3 Archival Storage

The size of archival storage is highly dependent on the application and the problem size. Traditionally, it has been driven by both checkpoint/restart as above, and by requirements where

subsets of such "checkpoints" represent time points in the computation that are to be used in 3D visualizations. The latter terms will certainly grow rapidly as Category III applications become important.

Further, as bad as the metadata problem is for file storage, it is even worse for archival storage, especially as the number of time steps per application run increases.

Even within current supercomputers where the computational assets are unchanging, the growth in archival storage needs has been significant, with 1.7 to 1.9 CAGR observed.

A recent study[55] about capturing data from NASA's **EOS/DIS** (Earth Observing System/-Data Information System) not only analyzer the cost efficiency of several approaches before suggesting tape farms, but also introduced two additional classes of metrics: the number of (kilo-/mega/giga) objects servable from such a system per second, and (more interestingly) the number of "scans" of the entire data set that can be done per day. This latter term becomes critically important as data mining and similar utilities become important.

In summary, such needs signal storage requirements that are easily in the 100X main memory category.

5.6.4 Increasing Memory Bandwidth

Increasing memory bandwidth to the local processor would most benefit Category II, and Category III applications. A machine that can do this would have to have radical technology advances to improve local memory bandwidth such as 3D stacked memory chips with higher speed signalling.

5.6.5 Increasing Bisection Bandwidth

Increasing bisection bandwidth becomes very expensive as the system grows in size, and may be a strong function of the maximum scale of the system and target application class. For example, an Exascale departmental system that will execute variations of today's applications (i.e. largely Category I and II) should probably have a bisection bandwidth comparable to that provided by the current 3D system topologies, but scaled to a petaflops. For example, a current XT4 supporting a 40x32x24 topology at 318 Tflops has a bisection bandwidth of 19.4TB/s. Simply scaling each dimension by a factor of 2 (to 80x64x48), and scaling the flops to 2.4 Pflops would thus require about 80 TB/s. For other Petascale applications, bisection bandwidth may be approaching the HPCS goals of 0.5 to 3.2 PB/s. Thus, overall, machines will become useful with bisection bandwidths in the O(50 TB/s) to O(1 PB/s) range.

As will be discussed later, Exascale applications that are destined for execution on the data center class systems may scale differently. While an "existence proof" application (Category I) may be runnable on systems with bandwidths in the O(1 PB/s) range, more realistic applications may have to grow with the memory bandwidth (reflecting more random data exchange patterns), and thus soar into the O(10 PB/s) to O(1 EB/s) range.

5.6.6 Increasing Processor Count

Increasing the number of processors (and local memory subsystems) would most benefit applications in category I. A machine that can do this would have to have radical advances in scalability with perhaps hundreds of millions of processors.

Category IV may require a combination of improved local memory bandwidth and increased number of processors and local memories.

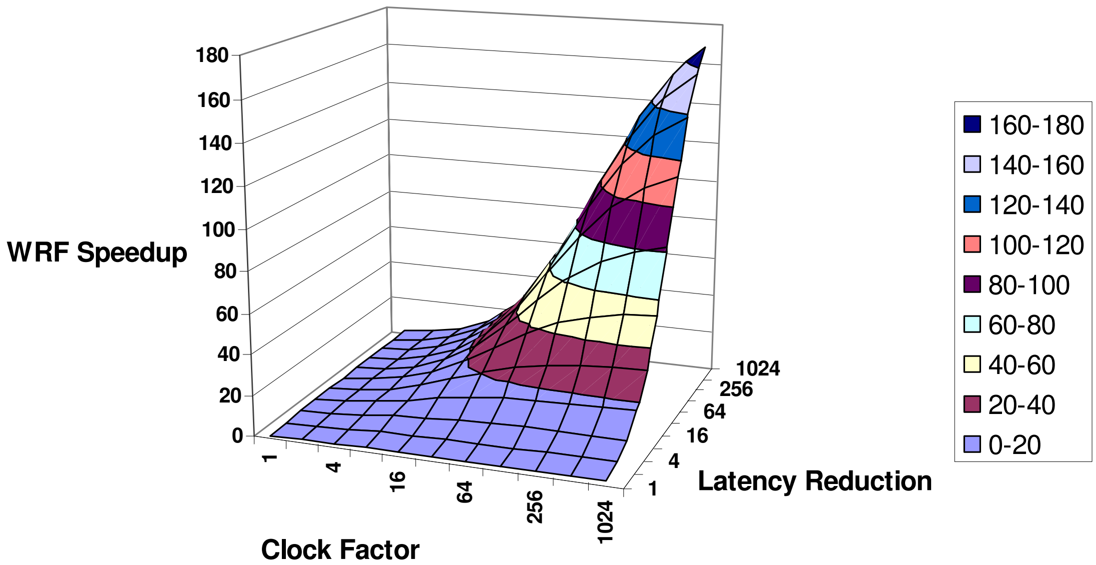


Figure 5.11: WRF performance response.

5.7 Application Concurrency Growth and Scalability

Quantifying what it means to be '1000x harder" has aspects in:

- Performance Growth
- Storage Growth
- Interconnect Bandwidth Growth
- Scratch Storage Growth
- File System Growth

All of these will grow more or less than 1000x, often independently. This section looks at the process of making such extrapolations, looking in particular for non-linear characteristics.

5.7.1 Projections Based on Current Implementations

For projecting to Exascale, we focus on three applications from Figure 5.9 that could be considered "Easy," "Hard," and "Harder" from a memory spatial and temporal locality perspective: HPL, WRF, and AVUS. We note that in the nomenclature defined above, WRF is a Category I, AVUS a Category II application, and WRF is potentially a Category III application. For these projections we ignore Category IV because its requirements for memory and number of processors fall somewhere in between Category I and Category II.

Figures 5.11 through 5.13 depict the performance response surface as predicted by the Convolution Method[130][131], and give forecast speedup for WRF, AVUS, and HPL as a function of increasing on-chip operations rates such as flops (X axis), or decreasing latency to memory at L2 and main memory level inclusive (Y axis). These convolutions are hardware agnostic, and do not specify how such improvements would be obtained (as for example whether by adding more

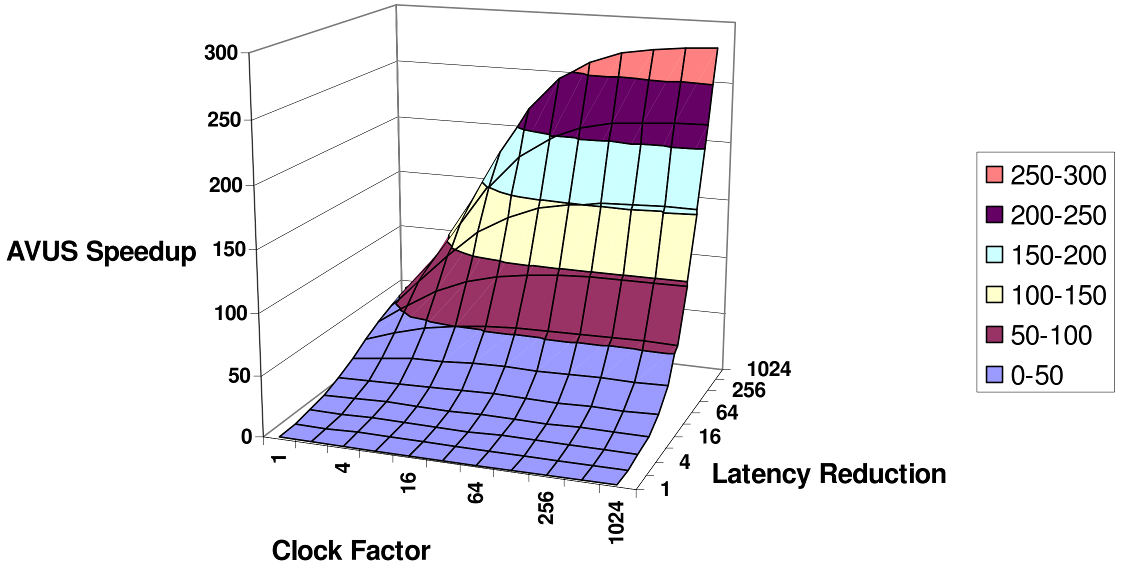


Figure 5.12: AVUS performance response.

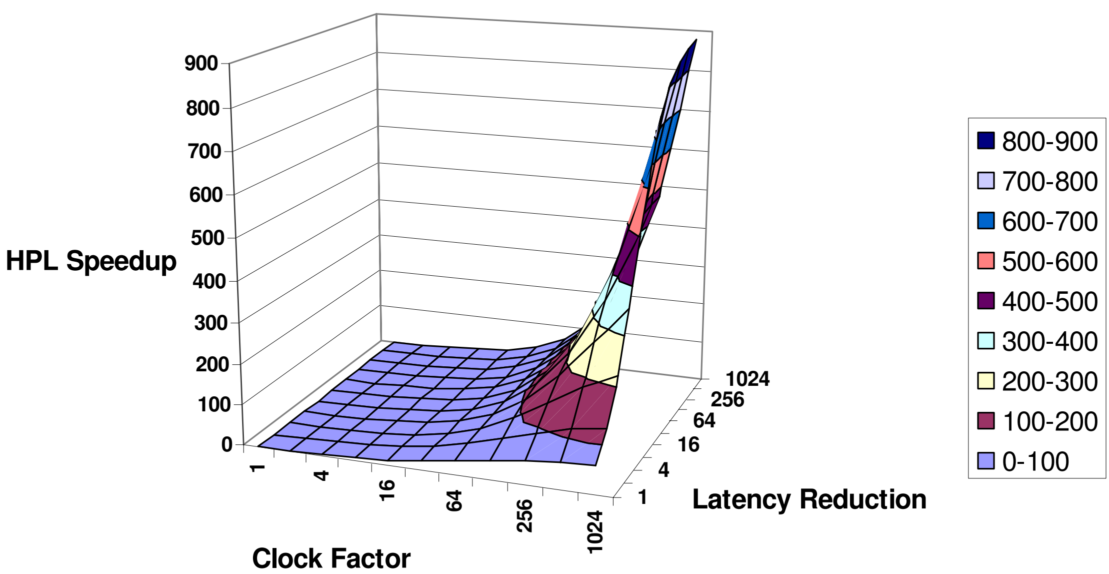


Figure 5.13: HPL performance response.

processors or by speeding up the clocking rate or otherwise improving the processors and/or memory). They simply assume that the machine has some way of doing work in the above operation categories faster.

However, an underlying assumption in Figures 5.11 through 5.13 is that communications overhead is a constant. (This perhaps overly optimistic assumption will be restricted later below). Note even so, for example in Figure 5.2, that an initially small fraction of communications becomes the bottleneck ultimately (by Amdahl's Law); thus even with 1024-fold improvement of both flops and memory, WRF can only be sped up about 180-fold according to the predictions. If communications latencies also could be improved by the same factors, then the WRF performance response would still have the same shape as Figure 5.2 but would reach higher levels of speedup.

Further, if one can trade bandwidth for latency then one could relabel the Y axis of Figure 5.2 and the subsequent figures as "memory bandwidth" rather than "1/memory latency." However the ability to trade bandwidth for latency tolerance via Little's Law is a complex function of the machine and the application. One needs to know for example, "how much inherent instruction-level parallelism (ILP) is in the application?," and "how much hardware support for in-flight instructions is in the hardware?." This is a level of detail not incorporated in the convolutions shown here. Thus Figure 5.2 and subsequent figures could be considered the most optimistic performance results obtainable by improving memory bandwidth rather than latency under the assumption that one could use all that extra bandwidth to tolerate latency better (and real performance would probably be less for that approach). Convolutions could of course be carried out under more pessimistic assumptions, but that is reserved for future work.

The "balanced" nature of WRF is apparent again from Figure 5.11. The performance response is symmetric for improving flops or memory latency. Once one is improved the other becomes the bottleneck.

By contrast, comparing Figure 5.12 for AVUS to Figure 5.11 for WRF above, we see that AVUS is a much more memory intensive code. Indeed, one must improve memory latency (or perhaps bandwidth, see discussion above) by an order-of-magnitude to unlock measurable improvements due to improving flops rate. One could even improve flops three orders of magnitude and get less than 50-fold AVUS speedup if the memory subsystem is not also sped up. On the other hand, AVUS does even less communications that WRF and so speeds up better for the same memory improvements - high-bandwidth processors would help AVUS, and other such memory intensive applications, a great deal.

Comparing Figure 5.13 for HPL to Figures 5.11 and 5.12 demonstrates what a pathologically useless benchmark HPL is as a representative for broader classes of applications. Its peak is overly optimistic (as it has almost no communications) and its lack of memory latency or bandwidth demands is not representative of more robust calculations such as WRF and AVUS. Never-the-less, even HPL will only speed up two orders of magnitude for three orders of magnitude improvement in flops if some improvement of memory latency (or possibly bandwidth) is not also accomplished.

Figures 5.11 through 5.13 then are most straightforwardly interpreted as applying to fixed problem size and processor/memory count, where it is reasonable to assume that communications does not grow in an absolute sense, but just as a fraction of total run-time due to Amdahl's Law.

Now let us consider the case of weak scaling, that is, building a system out to more processors and memories to enable more aggregate flops and bandwidth, and also making the problem bigger. Both of these approaches are not unreasonable to consider for WRF and AVUS as there are indeed larger problems (whole Earth atmosphere, full jet plane under maneuver) that can approach Exascale and still be scientifically interesting. Figures 5.11 through 5.13 still make sense if we interpret "speedup" to mean doing more work in a fixed time. But again recall the assumption that communications time is a constant. Once again, for this to be true, it would seem to imply

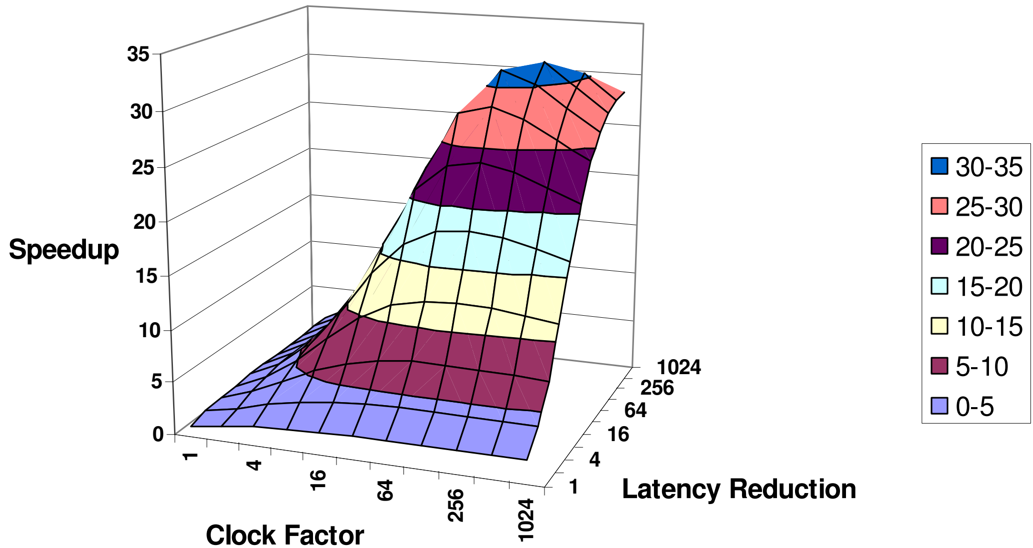


Figure 5.14: WRF with log(n) communications growth.

we can trade increased bandwidth of the larger machine to cover increasing latency. Whether this is true would again depend on the application and the architecture.

Figures 5.14 and 5.15 then give more pessimistic performance response surfaces of WRF and AVUS if communications cost instead grows as a logarithmic function of improving flops or memory latency/bandwidth (i.e. increasing problem size in the weak-scaling approach). One would have to improve network latencies, or trade bandwidth for latency, at a rate better than log(cpu count) to be able to get more than about 1 order of magnitude performance improvement reasonable for scaling out by 1000X one of today's machines 1000X.

Figures 5.14 and 5.15 then seem to argue for the importance of improving processor and memory performance, especially to benefit Category II and higher, not just scaling out huge systems for Category I, to enable a broad spectrum of important applications to operate at Exascale.

5.7.2 Projections Based on Theoretical Algorithm Analysis

A Petascale calculation of today, such as for the WRF projections in the previous section, is an operational hurricane forecast. It requires both ultra-high-resolution of gradients across the eye-wall boundaries (at ≈1 km or less), and representation of the turbulent mixing process correctly (at ≈10 m or less). Today's Petascale hurricane forecasting computation might have the following parameters:

- a 100 kilometer square outer-most domain at 10 meter horizontal grid spacing and 150 vertical levels,
- a 15-billion cell inner-most 10 meter nested domain,
- with a model time step of 60 milliseconds

Such a computation consumes about 18 machine hours per simulated day at a sustained petaflop/second on 100,000 processors and takes up about 100 MB per task of data not counting buffers, executable size, OS tax etc. (10 TB of main memory for application in aggregate). The computation generates 24 1.8 terabyte data sets, or 43.2 TB per simulation day if hourly output of

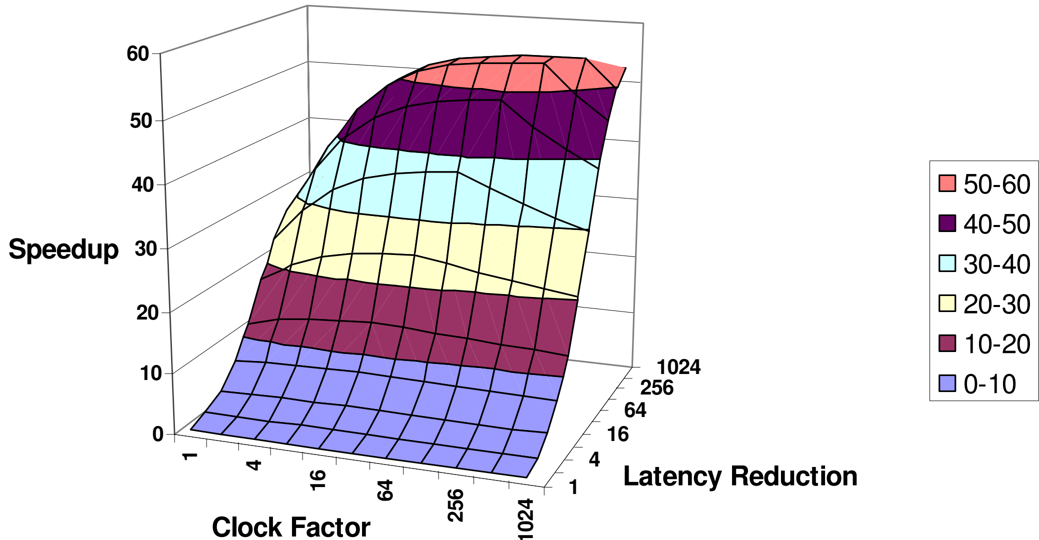


Figure 5.15: AVUS with log(n) communications growth.

30 three-dimensional fields is performed. At an integration rate of 18 machine hours per simulated day at a sustained petaflop, the average sustained output bandwidth required is 700 MB/second.

If we project a Category II Exascale WRF calculation (with a goal to track a single storm 1000X faster) then no increase in physical memory size or I/O bandwidth is needed (though floating-point issue and memory bandwidth have to improve by 1000X). This would allow forecasts of storm tracks with lead time of days or even weeks instead of only hours or minutes, and would greatly improve disaster preparedness lead time.

A different Petascale calculation of today is a WRF “nature run” that provide very high-resolution “truth” against which more coarse simulations or perturbation runs may be compared for purposes of studying predictability, stochastic parameterization, and fundamental dynamics. Modeled is an idealized high resolution rotating fluid on the hemisphere to investigate scales that span the k-3 to k-5/3 kinetic energy spectral transition. Computational requirements for a 907x907 grid with 101 levels, resolution at 25km, time step at 30s on 100,000 processors, is 100 MB main memory per task (= 10 TB of main memory) and outputs about 100 TB per simulation day, or a sustained 1K MB/s I/O.

Another Category I Exascale WRF calculation would be to perform a full nature run of l hemisphere of earth at sub 1km resolution. This would capture small features and “the butterfly effect” - large perturbations at long distance driven by small local effects. This very challenging calculation at a sustained Exaflop would then require 10,000 GB = 10 PB of main memory. I/O requirements would also go up 1000x.

If we project a Category III Exascale WRF hurricane calculation (track 10 storms of interest at once as part of a simulation of a portion of the hurricane season), and predict each storm track 100X faster, then memory requirements scale up 10X and I/O 10X.

In other domains of science such as mantle physics, modeling earthquakes with multiple scenarios in short term disaster response or evolution of fault on geological timescale, is a “capacity” (Category I) problem, while modeling the mantle of earth as a ”living thing” coupled to crust for understanding tectonic plate system evolution is a “capability” (Category II) problem.

In biology, protein folding very long sequences, interactive protein docking, and calculating multiple drug interactions is a ”capacity” (Category I) problem, while modeling membranes, organs,

organisms, and even going the other direction to cell modeling at molecular level of detail is a "capability" Category II problem.

5.7.3 Scaling to Departmental or Embedded Systems

Pioneering Exascale applications, like those first expected to run at Petascale in the next year, will likely be a handful of specialized scientific codes. These applications will be very narrowly designed to solve specific problems. They will be selected for their ability to scale to billions of concurrent operations without having to solve currently intractable problems such as partitioning unstructured grids. They might relax normal synchronization constraints in order to avoid the resulting bottlenecks. With heroic programming efforts, they will execute out of the smallest, fastest levels of the memory hierarchy, minimizing the number of concurrent operations needed to cover main memory latency.

For there to be a viable market for Petascale departmental systems, there will have to be viable commercial applications to run on them. Such Petascale commercial applications will need to sustain millions of concurrent operations, a daunting task when today, only a handful of commercial codes can sustain even O(1000) concurrent operations on a few hundred multi-issue CPUs. Thus the developers of such codes will share many of the same challenges as those developing Exascale applications for the nation, how does one add four orders-of-magnitude to the level of concurrency one struggles to sustain today. Other issues such as debugging at this scale or tolerating errors in increasingly unreliable systems will also be common to developers of both Exascale and departmental scale programs.

There will also be significant differences in the challenges facing commercial Petascale software. Rather than solve a handful of problems of national interest, they will have to solve a broad range of engineering and business problems. Rather than be small scale versions of Exascale scientific codes, they will be Petascale versions of codes that also run on Terascale desk top systems. As such, they will enjoy large numbers of users and hence be economically viable. However, their developers will have to address a broader range of problems, including unstructured grids and more robust numerical algorithms. Such applications today are millions of lines of code, and represent an investment in labor that cannot be repeated, but rather must be evolve into the future. Thus any new programming languages or features must be backward compatible to be accepted into commercial software.

Petascale application will also put demands on departmental systems that make them different than simply smaller Exascale systems. Petascale departmental systems will have to provide full featured operating systems, not micro-kernels. They will have to efficiently process large ensembles of end-user applications, not merely a handful of heroic jobs. Where Petascale scale systems do not have enough main memory, their applications will go "out-of-core,"requiring a disproportionately larger and higher performing file system to hold the balance of their state.

Scaling to embedded systems will also be an issue for Exascale applications. Just as today one sees applications such as sPPM running on everything from the largest supercomputers (BG/L) to Playstations, one can expect the same in the next decade, as users try to exploit any computing system available to them. Their will be additional challenges however, making embedded applications more challenging than their departmental counterparts. Embedded codes often involve similar computational kernels, but they tend to be implemented with specialized software. This reflects both physical constraints such as volume and power, which reduce the size and performance of the processors and memory, as well as novel software environments, such as real-time operating systems with less functionality than those expected on departmental servers.

	Departmental Class		Data Center Class	
	Range	"Sweet Spot"	Range	"Sweet Spot"
Memory Footprint				
System Memory	O(100TB) to O(1PB)	500 TB	O(1PB) to O(1EB)	50 PB
Scratch Storage	O(1PB) to O(100PB)	10 PB	O(100PB) to O(100EB)	2 EB
Archival Storage	>O(100PB) to O(100PB)	100 PB	>O(100EB)	100 EB
Communications Footprint				
Local Memory Bandwidth and Latency	Expect low spatial locality			
Global Memory Bisection Bandwidth	O(50TB/S) to O(1PB/s)	1PB/s	O(10PB/s) to O(1EB/s)	200PB/s
Global Memory Latency	Expect limited locality			
Storage Bandwidth	Will grow at faster rate than system peak performance or system memory growth			

Table 5.1: Summary applications characteristics.

5.8 Applications Assessments

5.8.1 Summary Observations

In terms of overall system characteristics, Table 5.1 attempts to summarize a first cut at how the different memory, storage, and bandwidth considerations might play out for both a Departmental class Exascale system and a Data Center class. It should be stressed that these numbers are relatively speculative right now, and can only be validated when extensive work on real Exascale applications is performed.

For a bit more detail on the internal structure of Exascale applications and how they may achieve Exascale performance, Figure 5.16 attempts to summarize some of the trends observed in terms of applications over time. The vertical axis refers to locality - how much information about one memory reference can be used to improve the latency of accessing a future one. The horizontal axis refers to how much concurrency is present.

The contents of the graph refer to classes of applications. Looking backwards in time, many early "high end" numeric applications used simple 3D regular grids where the data could be positioned precisely before hand, and techniques such as red-black ordering provided potential levels of parallelism that approached the number of grid points.

Moving forward to today, grids have become much more irregular, with a great deal of auxiliary table look-ups needed to account for detailed modeling. The result has been some more potential for concurrency (more points), but significantly decreased locality. This locality becomes even worse when dynamic mesh refinement is used to change the grid dynamically.

Another class of applications are highly non-numeric, and have significantly different characteristics, particularly in locality, than the first class. Searching a simple one-dimensional linked list, for example, has almost no possible concurrency if no pre-computed pointers to intermediate nodes

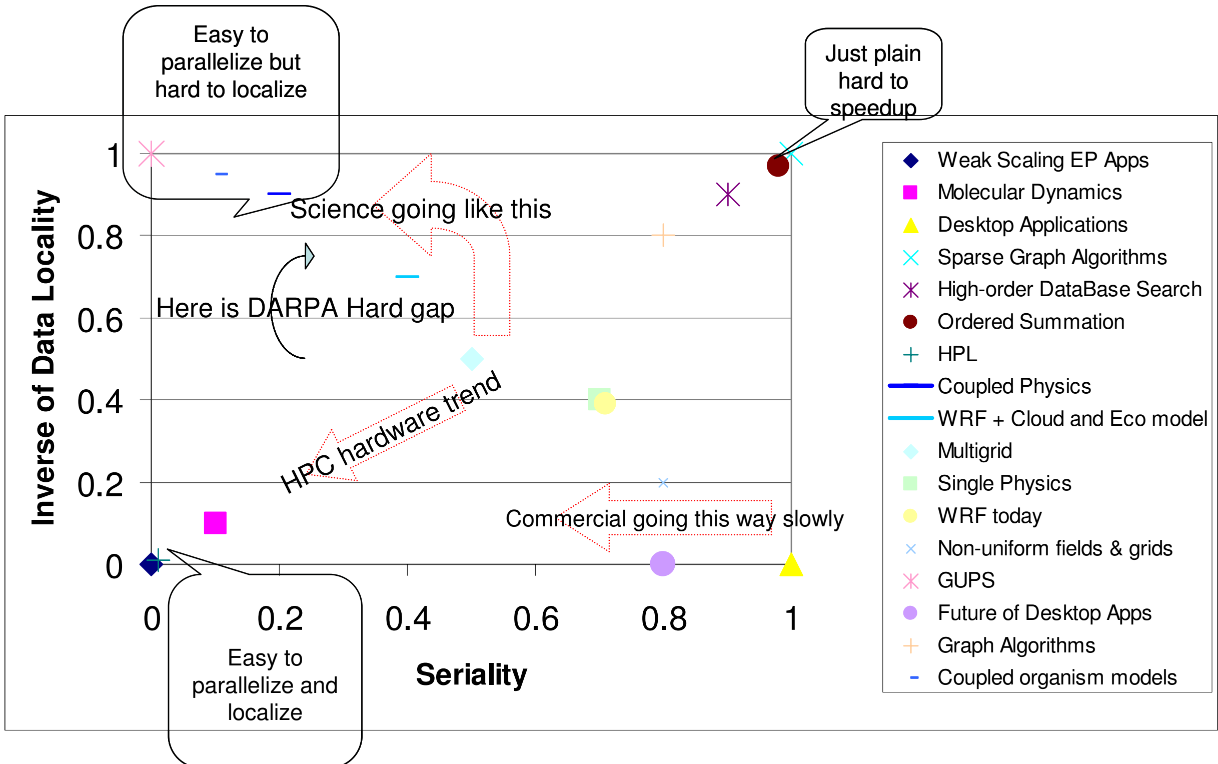


Figure 5.16: Future scaling trends

exist to allow searches to jump into the middle of the list. It also has potentially very low locality if each node of the list ends up in different memories.

Going forward to more relevant applications, searching large distributed graphs has as little locality as the 1D search, but with a parallelism limited by the diameter of the graph. Finally, applications similar to GUPS have potentially huge amounts of parallelism, but again little or no locality.

5.8.2 Implications for Future Research

The discussions in this chapter lead fairly directly to several research directions that would directly impact the ability of applications to take advantage of Exascale hardware:

- Develop additional hardware and architectural techniques to mitigate the impact of poor locality within an application.
- Provide more hardware and architectural hooks to control the memory hierarchy, and provide programming metaphors and APIs that allow an application to express how it wants to control locality of data within this hierarchy.
- Through compiler technology, program transformation techniques, or address reordering techniques, find or create improved locality automatically.
- For applications of direct interest to DoD, foster algorithm work that focuses on both parallelism and locality.

- Develop algorithmic and programming techniques that can tolerate poor locality, such as increasing asynchronicity in communications, and more prefetch opportunities.

In addition, in support of nearly all of these research directions, it appears reasonable to develop suites of tools that can provide estimates of "upside potentials" of current and emerging codes, including:

- tools to analyzes existing codes for dependencies and sequences of memory access addresses,
- tools to "data mine" the outputs of the above to look for and identify "patterns,"
- tools that allow such patterns to be translated into forms that can be used by systems to implement these patterns in ways that increase performance,
- tools that analyze the codes to provide estimates of "oracle" parallelism opportunities, and transform that into estimates of scaling when run on parallel hardware.

Chapter 6

Technology Roadmaps

This chapter reviews the suite of relevant technologies as we understand them today. This includes both those that are well-established and in current use in today's systems, and those that are just emerging. For each such technology we discuss:

- Its fundamental operation and expected target usage.
- The key metrics by which to judge its progress.
- Its current state of maturity.
- For the key metrics both current values, physical limits as we understand them today, and a roadmap as to how improvements are projected to occur with current funding and focus, with an emphasis on the time between now and 2015.
- The fundamental reliability of subsystems built from such technologies.
- Particular areas where additional research may prove valuable in accelerating progress towards reaching the limits of the technology.

The key technology areas reviewed in the sections below include:

- Section 6.2: technology from which logic and computational functions may be constructed.
- Section 6.3: technology from which the primary memory used in computers is constructed.
- Section 6.4: technology from which mass store such as file systems is constructed.
- Section 6.5: interconnect technology that permits different computing sites to communicate with a single computing structure.
- Section 6.6: technology with which combinations of chips from the above categories (especially logic and main memory) may be packaged together and cooled.
- Section 6.7: techniques used today to improve the overall resiliency of a computing system.
- Section 6.8: the (largely software) technologies for managing the operating environment of the computing systems.
- Section 6.9: software technologies for extracting the parallelism and generating the application code.

6.1 Technological Maturity

Gauging the maturity of a technology is an important aspect of performing any projections such as done in this study, especially when time frames to get to deployable systems are important. In this report we use a metric developed by NASA termed "Technology Readiness Levels," which has multiple levels as follows (taken from [99]):

1. Basic principles observed and reported
2. Technology concept and/or application formulated
3. Analytical and experimental critical function and/or characteristic proof-of concept
4. Component and/or breadboard validation in laboratory environment
5. Component and/or breadboard validation in relevant environment
6. System/subsystem model or prototype demonstration in a relevant environment
7. System prototype demonstration in a real environment
8. Actual system completed and "flight qualified" through test and demonstration
9. Actual system "flight proven" through successful operations

Such levels address the maturity of a technology in terms of how fully it has been developed. However, it is important to distinguish such levels from the funding categories often used in describing development budgets used to bring such technologies to practice:

- 6.1 Basic Research
- 6.2 Applied Research
- 6.3 Advanced Technology Development
- 6.4 Demonstration and Validation
- 6.5 Engineering Manufacturing Development
- 6.6 Management Support
- 6.7 Operational Systems Development

It was the sense of the study group that for Exascale projects constructed from technologies currently at TRL levels 1 and 2 most closely corresponded to 6.1 projects, TRL levels 3 and 4 corresponded to 6.2, and TRL levels 5 and 6 to 6.3. Once technologies have become more mature that TRL 6, they are no longer "immature," and need to stand on their own commercial value for continued development and deployment.

In these terms, the goal of the study is thus to determine where technologies projected to be "mature" in the 2013-2014 time frame by normal development will be inadequate to support Exascale systems, and whether or not there is the potential for supporting the accelerated development of new, currently "immature," technologies that may bridge the gap.

	Units	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020
Feature Size	**nm**	**90**	**78**	**68**	**59**	**52**	**45**	**40**	**36**	**32**	**28**	**25**	**22**	**20**	**18**	**16**	**14**
Logic Area	relative	1.00	0.80	0.63	0.51	0.39	0.32	0.25	0.20	0.16	0.12	0.10	0.08	0.06	0.05	0.04	0.03
SRAM Area	relative	1.00	0.78	0.61	0.48	0.38	0.29	0.23	0.18	0.14	0.11	0.09	0.07	0.06	0.04	0.03	0.03
50/50 Area	relative	1.00	0.79	0.62	0.49	0.38	0.30	0.24	0.19	0.15	0.12	0.09	0.07	0.06	0.05	0.04	0.03
High Performance Devices																	
Delay	ps	0.87	0.74	0.64	0.54	0.51	0.40	0.34	0.29	0.25	0.21	0.18	0.15	0.13	0.11	0.10	0.08
Average Device Capacitance	relative	1.00	0.87	0.76	0.66	0.58	0.50	0.44	0.40	0.36	0.31	0.28	0.24	0.22	0.20	0.18	0.16
Circuit speedup: 1/delay	relative	1.00	1.18	1.36	1.61	1.71	2.18	2.56	3.00	3.48	4.14	4.83	5.80	6.69	7.91	8.70	10.88
ITRS Max Clock	relative	1.00	1.30	1.78	2.11	2.38	2.90	3.39	3.86	4.42	5.45	6.42	7.63	8.75	10.22	12.00	14.05
Vdd	volts	1.10	1.10	1.10	1.00	1.00	1.00	1.00	0.90	0.90	0.90	0.80	0.80	0.70	0.70	0.70	0.70
Vdd/Vt	ratio	5.64	6.55	6.67	6.10	4.22	6.62	6.85	6.08	5.39	5.49	4.82	4.10	3.50	3.48	3.41	3.37
Power Density @ Circuit Speedup	relative	1.00	1.29	1.65	1.77	2.13	2.95	3.95	4.19	5.52	7.41	7.54	10.19	10.17	13.91	17.12	23.47
Power Density @ Max Clock	relative	1.00	1.43	2.17	2.31	2.97	3.93	5.23	5.39	7.00	9.75	10.01	13.39	13.30	17.98	23.61	30.33
Energy/Operation	relative	**1.000**	**0.867**	**0.756**	**0.542**	**0.478**	**0.413**	**0.367**	**0.268**	**0.238**	**0.208**	**0.147**	**0.129**	**0.090**	**0.081**	**0.072**	**0.063**
Low Operating Power Devices																	
Delay	ps	1.52	1.33	1.17	1.03	0.90	0.79	0.79	0.61	0.53	0.47	0.41	0.36	0.32	0.28	0.24	0.21
Circuit speedup: 1/delay	relative	0.57	0.65	0.74	0.84	0.97	1.10	1.10	1.43	1.64	1.85	2.12	2.42	2.72	3.11	3.63	4.14
Vdd	volts	0.90	0.90	0.80	0.80	0.80	0.70	0.70	0.70	0.60	0.60	0.60	0.50	0.50	0.50	0.50	0.50
Vdd/Vt	ratio	3.13	2.97	2.81	2.95	2.90	3.10	3.00	3.03	2.33	2.40	2.39	2.10	2.09	2.07	2.06	2.03
Power Density @ Circuit Speedup	relative	0.38	0.48	0.48	0.59	0.77	0.73	0.83	1.21	1.16	1.47	1.86	1.66	2.11	2.79	3.64	4.56
Energy/Operation	relative	**0.669**	**0.580**	**0.400**	**0.347**	**0.306**	**0.202**	**0.180**	**0.162**	**0.106**	**0.093**	**0.083**	**0.051**	**0.046**	**0.041**	**0.037**	**0.032**
Note: units of "relative" represent values normalized to those of the 2005 high performance technology																	

Figure 6.1: ITRS roadmap logic device projections

6.2 Logic Today

Perhaps the most mature technology that is in use today for logic and memory is CMOS silicon. This section discusses the outlook for this technology through the end of the next decade in two ways: in summary form as projected by the ITRS Roadmap, and then in terms of the inherent device physics as seen by industry-leading sources. The latter is covered in two pieces: silicon-related and non-silicon technologies.

6.2.1 ITRS Logic Projections

The 2006 ITRS Roadmap[13] projects the properties of silicon CMOS logic through the year 2020, and is used as a standard reference for future trends. This section overviews some of the more general projections as they may relate to Exascale logic sizings for 2015 developments. It is important to note, however, that these projections are for "business as usual" CMOS silicon, and do not represent potential alternative silicon device families.

For logic there are two kinds of devices that are most relevant: those designed for modern leading-edge high performance microprocessors, and those designed for low operating power where clock rates may be sacrificed. The differences lie primarily in the threshold voltages of the devices, and the operating conditions of typical circuits (V_{dd} and clock rate).

For reference, Figures 4.2 through 4.10 combine both historical data with their matching projections from ITRS for a variety of key parameters. Figure 6.1 then summarizes numerically some projections that are most relevant to this report. The columns represent years from 2005 through the end of the roadmap in 2020. For each year, the "feature size" (Metal 1 half-pitch) is listed.

The table itself is divided into three sub-tables: area-related, speed and power for the high performance devices, and similar speed and power for the low operating power devices. In this figure, any row marked as being in "relative" units represents the actual values derived from the roadmap, but normalized to the equivalent numbers for 2005 high performance devices. Thus the key results have been normalized to industry-standard high performance 90nm CMOS technology.

The years highlighted in green represent the values most relevant to chips that might be employed in 2015 in real systems.

6.2.1.1 Power and Energy

Throughout all the rows in Figure 6.1, power consumed (and thus dissipated) for some fixed subsystem of logic is computed using the formula (leakage power is ignored for now):

$$Power_per_subsystem = Capacitance_of_subsystem * Clock * V_{dd}^2 \quad (6.1)$$

Dividing this by the area of the subsystem yields the **power density**, or power dissipated per unit area:

$$Power_density = (Capacitance_of_subsystem/area_of_subsystem) * Clock * V_{dd}^2 \quad (6.2)$$

or

$$Power_density = Capacitance_per_unit_area * Clock * V_{dd}^2 \quad (6.3)$$

Also, canceling out the clock term in Equation 6.1 yields not the power of a circuit but (at least for pipelined function units) an **energy dissipated per machine cycle** which, if the subsystem is pipelined is the same as the **energy per operation**, and which will prove useful in later work:

$$Energy_per_operation = Capacitance_per_unit_area * V_{dd}^2 \quad (6.4)$$

It is instructive to express the capacitance of a subsystem as:

$$Capacitance_of_subsystem = Capacitance_per_device * \#_of_devices \quad (6.5)$$

which yields another variant of Equation 6.2:

$$Power_density = (Capacitance_per_device * \#_of_devices/area_of_subsystem) * Clock * V_{dd}^2 \quad (6.6)$$

However, the ratio of device count to area is exactly transistor density as discussed above, and thus we can rewrite this as:

$$Power_density = Capacitance_per_device * Transistor_density * Clock * V_{dd}^2 \quad (6.7)$$

Finally, we can also invert this last equation to yield one that indicates at what maximum clock frequency a chip could run at in order to stay at or below some power density limit:

$$Clock = Power_density/(Capacitance_per_device * Transistor_density * V_{dd}^2) \quad (6.8)$$

6.2.1.2 Area

The first three rows of Figure 6.1 reflect the effect of the technology on **transistor density** - the number of transistors per unit area on a die. In the rows, these numbers are expressed as relative "area" factors, that is by what factor would a circuit designed in 90nm technology shrink if it were simply recast in succeeding year's technology, with no enhancements or scaling. There are separate rows for circuits that are pure logic, pure SRAM arrays, and circuits with a 50/50 mix of logic and SRAM.

The key take-away is that for 2015 deployment, using ITRS projections we can assume that a core designed initially for 90nm would shrink between a factor of 6 and 8 in area, allowing up to 6 to 8 times more of the same complexity cores on a die that in 2005.

6.2.1.3 High Performance Devices

The high performance device sub-table of Figure 6.1 represents the mainstay of modern CMOS microprocessor technology. The first row gives the **intrinsic delay** of an N-type device, with the third representing its reciprocal relative to the 2005 number. Thus a number of 3.48 in 2013, for example, implies that the same circuit using 2013 devices would be capable of running 3.48 times faster than in 2005.

The column labeled "ITRS Max Clock" represents clock rate growths projected by ITRS for a pipeline stage involving 12 invertors in a chain. This number grows somewhat faster than the second row, due to other circuit effects. The choice of a 12 invertor stage was made for historical reasons based on microprocessors designed for the maximum possible clock rate (called **super-pipelining**), regardless of microarchitectural performance. It is important to understand that such "short pipe stages" have been found since then to be inefficient when used in modern superscalar microprocessors, where long pipe lengths exacerbate significant data hazards and bottlenecks that negatively impact performance. In addition, power and clock rate are proportional, so higher clock rates also raise power dissipation. Thus, for several years the microprocessor industry has retreated from super-pipelining in favor or more efficient but lower clock rate microarchitectures. Thus this row should be taken as an indication of absolutely maximum upper end potential, not expected practice.

The row labeled "Vdd" represents the main operating voltage projected for use by circuits using these devices. The row below this gives the ratio between V_{dd} and the main threshold voltage of the projected devices. Over the time period of interest, this ratio is about 5.5, meaning that there is sufficient voltage for multiple devices to be stacked, and still have good operating margins.

Given these numbers, the row labeled "Power density @ Max Clock" represents the relative change in power dissipated per unit area on a die when the V_{dd} is as marked, and the clock rate is the maximum projected by ITRS. The numbers for 2013-2014 indicate that if we simply tiled dies on that time with multiple copies of a 2005 core, the die would dissipate 7-10X as much power per unit area - far more than is assumed coolable today.

The row labeled "Power Density @ Circuit Speedup" is a similar calculation but assuming circuits speed up only as much as the devices do. The growth in power dissipation, however, is still significant.

The last row in this section reflects the "energy per operation" relative to a 90nm circuit, in units of pico joules (pJ) where 1 pJ equals 10^{-12} joules. Since this is "per operation" the clock rate in Equation 6.1 is irrelevant. Thus a circuit that took X pJ per operation in 2005 will, in 2013-2014 technology, take 1/4 to 1/5 of that.

Also we note again that numerically, if performing some operation takes $X\ pJ$ of energy, then computing 1 "exa" (10^{18}) of them in one second consumes $X\ MW$ of power.

6.2.1.4 Low Operating Voltage Devices

The final section of Figure 6.1 reflects projections for devices designed to be placed into lower power circuits. The devices are designed to have a higher threshold voltage (less leakage) and run at a lower V_{dd}. Thus the circuit families cannot be as complex in terms of stacking devices, there is less margin, and the circuit is significantly slower than the high performance one, but the power density and energy per operations are significantly improved, by a factor of two in energy per operation.

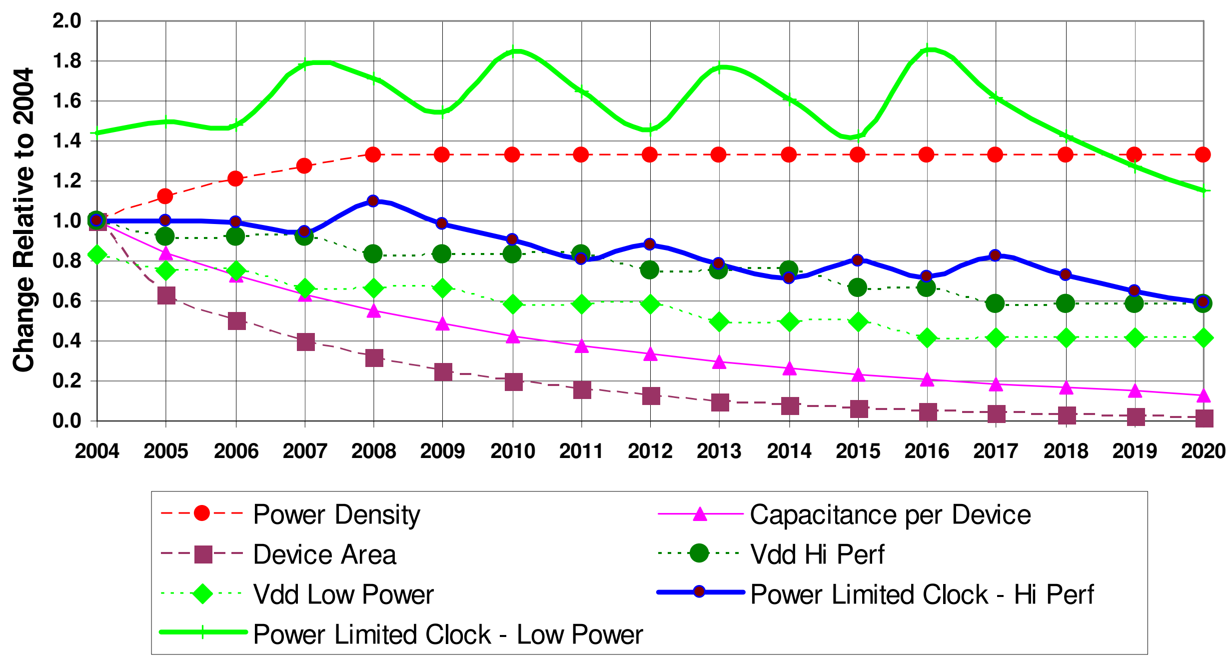


Figure 6.2: Relative change in key power parameters

6.2.1.5 Limitations of Power Density and Its Effect on Operating Frequency

Figures 4.3, 4.7, and 4.8 diagram the density, V_{dd}, and clock parameters in the power equation 6.7. These includes both historical data (from both single core chips and the newer multi-core chips), and ITRS extrapolations. Figures 4.9 and 4.10 then graph similarly observed and projected maximum power dissipated per chip, and the equivalent power density. These graphs yield the following key takeaways:

- Transistor area (density) continues to drop (increase) as the square of the feature size. This tends to increase the overall power of a chip because more transistors are present.
- V_{dd} for the dominant high performance logic families has essentially flattened after a long period of significant reductions, with minimal projected future decrease. Thus decreasing power by reducing voltage has just about run its course.
- Capacitance per device continues to decrease approximately linearly with feature size. This both allows the transistors to run faster, and reduces the power per device.
- After a multi-decade run-up, sometime in the early 2000's, the clock rate actually implemented in real chips began to lag behind that which was possible due to the inherent improvement in the transistors delay characteristics. In fact, a plateauing around 3GHz occurred at the upper end.

The reason for the last of these observations is clear from Figure 4.9 - the absolute power dissipated by a typical chip reached a threshold of around 100 watts above which it is uneconomical to cool. At this point, the only knob available to designers to limit this was operational frequency. Thus the flattening of clock rate.

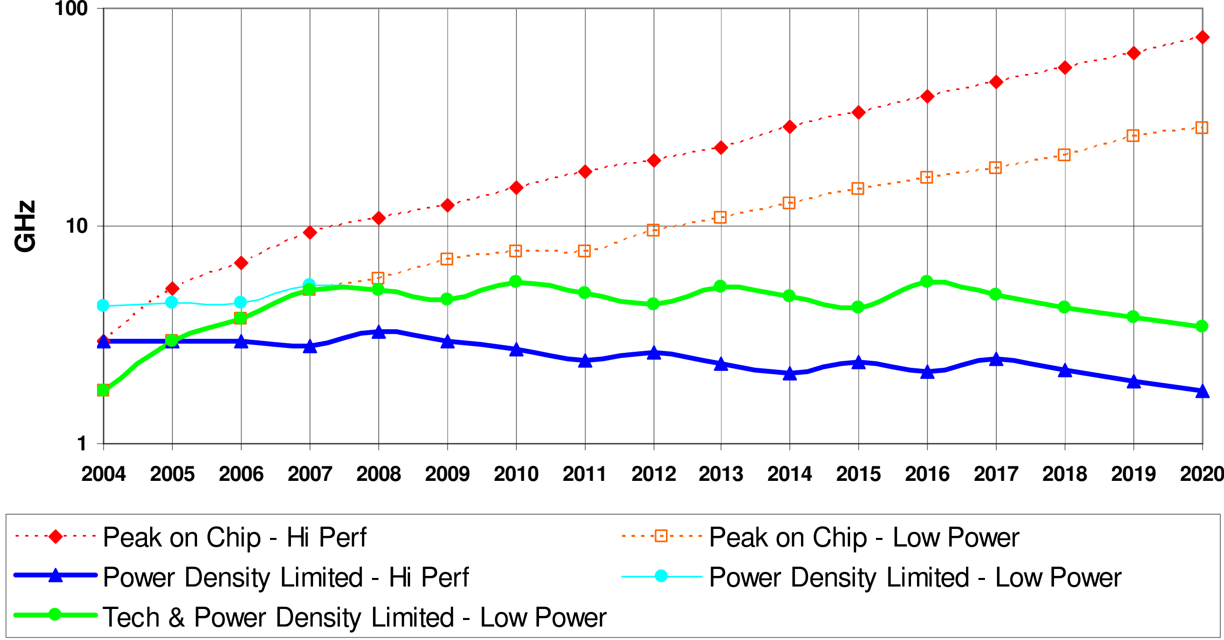


Figure 6.3: Power-constrained clock rate

An instructive exercise to explore what might happen in the future is to use these ITRS roadmap projections to forecast what kind of growth in operating frequency is thus likely to be observed. To do this we assume that chips of the future will look essentially like today in terms of mix of transistors. This is in fact more or less will happen if the current ground swell to multi-core processor chips continues unabated, with future logic chips representing a tiling of copies of what look like today's core.

Figure 6.2 does this by referencing the ITRS trends to those values of 2004, the approximate year when clock rates peaked. The curves include the maximum power density that can be cooled, the range of V_{dd} that is still available, the transistor density (the reciprocal of this is graphed to keep the numbers in the range of the others), and the capacitance per device. Then, using Equation 6.8 we can determine a clock rate that will just support the maximum possible cooling capacity. Figure 6.3 converts this relative clock curve to an absolute clock. Also included for reference is the peak clock that is possible based on the transistor delay.

These curves give a somewhat astonishing conclusion: with a design-as-usual mindset, microprocessors of the future will not only *not* run faster than today, they will actually decline in clock rate.

The same sort of analysis can be done if instead of the high performance logic we assume the low operating power form discussed in Section 6.2.1.4. Figure 6.2 includes a separate curve for the V_{dd} in this case, as does Figure 6.3.

The conclusion from this chart is equally enlightening - again the clock rates are way below the absolute possible based on the transistor performance. However, because of the lowered voltage, very shortly the maximum clock that is sustainable at the maximum power dissipation actually becomes *higher* than that for the supposed high performance logic. This may be an indication that the low voltage logic is perhaps a better choice for Exascale than conventional logic.

High Volume Manufacturing	2008	2010	2012	2014	2016	2018	2020	2022
Technology Node (nm)	45	32	22	16	11	8	6	4
Integration Capacity (BT)	8	16	32	64	128	256	512	1024
Delay Scaling	>0.7			~1?				
Energy Scaling	~0.5			>0.5				
Transistors	Planar			3G, FinFET				
Variability	High			Extreme				
ILD	~3			towards 2				
RC Delay	1	1	1	1	1	1	1	1
Metal Layers	8-9	0.5 to 1 Layer per generation						

Figure 6.4: Technology outlook

6.2.2 Silicon Logic Technology

The ITRS data of the prior section focused on general trends; given the challenges involved, it is instructive to understand the physics behind these trends, and how they affect the kinds of circuits and performance metrics that make up the trends. The following subsections discuss the underlying scaling challenges, the potential for new processes such as SOI, and what this means to logic and associated high speed memory circuits.

6.2.2.1 Technology Scaling Challenges

We begin with the technology outlook presented in Figure 6.4. Transistor integration capacity is expected to double each generation and there are good reasons to believe that it will be on track. Logic and circuit delay scaling, however, has already slowed down, and is expected to slow down even further, approaching a constant. Energy scaling too has slowed down, so transistor architecture will have to change to something other than todays planar architecture, and thus the variability in transistors will become even worse that what it is today. In a nutshell, you will get transistor integration capacity in the future (the first major benefit), but not the same performance, and the energy/power reduction.

Figure 6.5 is a simplified transport model describing the scaling challenges. Thinner gate dielectric (gate oxide) is better since it results in higher gate capacitance, creating higher charge volume. But gate oxide scaling has reached a limit due to tunneling, causing excessive gate leakage. High-K gate dielectric is a solution, but for just a generation or two, since it too has to scale down and will reach the scaling limit.

Lower threshold voltage (V_t) is desired for higher current, but it causes excessive source to drain sub-threshold leakage, that is why, we suspect that V_t scaling too has reached the limit. Mobility engineering, such as straining, to improve the drive current will continue, but with diminishing return.

To summarize, due to gate dielectric scaling and V_t scaling slowing down, the supply voltage

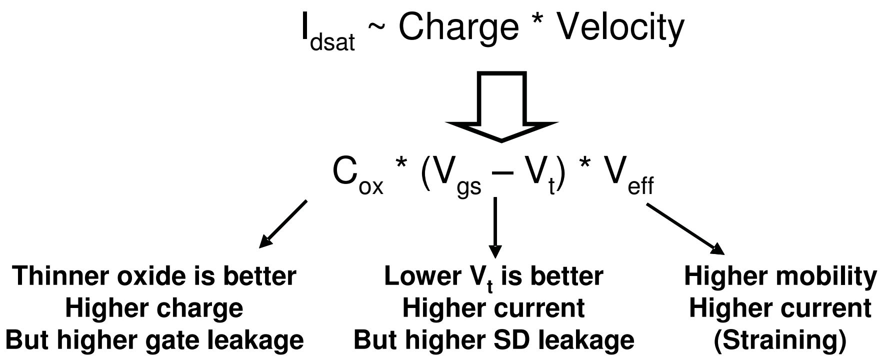


Figure 6.5: Simple transport model

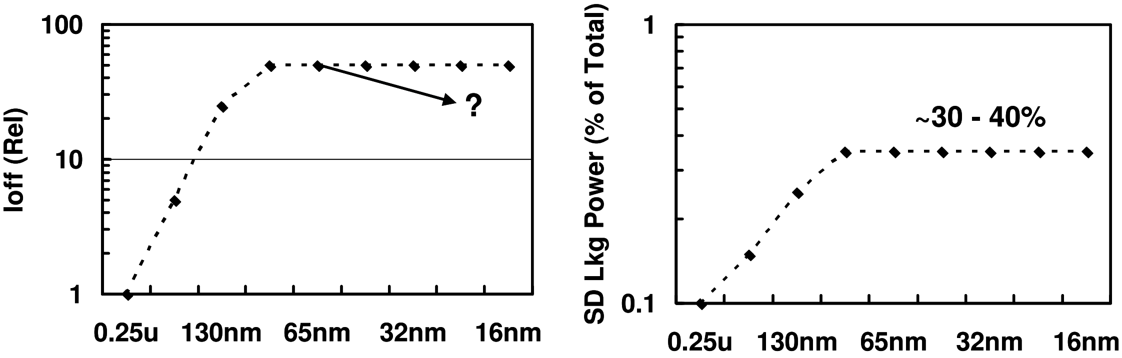


Figure 6.6: Transistor sub-threshold leakage current and leakage power in recent microprocessors

scaling too will slow down, transistor performance increase will slow down, and active energy/power reduction will slow down.

Figure 6.6 graphs increase in sub-threshold leakage current in successive past technology generations, and corresponding increases in the leakage power as a percentage of total power. The slope of the non-flat part of the I_{off} curve is often called the **sub-threshold slope**,, and the flatter the curve, the less that future generations of technology will have to worry about leakage. Also included are extensions to the future. Notice that the transistor leakage power increased exponentially to deliver the necessary performance, but now it is staying constant, and may even decrease to keep full chip leakage power under control.

Figure 6.7 shows predictions of technology and attributes for the timeframe of Exascale systems as envisioned here.

In terms of summary projections for future designs in silicon such as in Section 7.3, it is thus reasonable to assume that leakage power will account for 30% of the total chip power.

6.2.2.2 Silicon on Insulator

There is a common belief in the technical community that something drastic can be done in the process technology to reduce power further, and the most commonly cited solution is **SOI** (**Silicon on Insulator**) technology, where the transistor is built not on a conductive substrate (as in **bulk silicon**), but as a film on an insulating oxide. This technology comes in two flavors: **partially depleted**, and **fully depleted**. In partially depleted SOI, the silicon film is thick and only part

Tech Node	Units	32nm	22nm	16nm	Comments
High Volume		2010	2012	2014	
Vdd	Volts	1	0.95	0.9	Vdd scaling slowed down
Delay Scaling		1	-15%	-10%	Delay scaling slowed down (estimated)
FO4 Delay	ps	10	8.5	7.65	Estimate
Logic density	MT/mm2	1	1.5	2.2	1.5X, limited by design rule complexity and interconnects
Cache (SRAM) density	MB/mm2	0.2	0.34	0.54	Includes tags, ECC, etc 1.6X limited by stability
Logic Cdyn (αC/Tran)	nf/MT	0.2	0.16	0.13	0.8X scaling due to variability, etc.
Cache Cdyn (αC/MB)	nf/MB	0.09	0.06	0.04	0.8X scaling

Figure 6.7: Technology outlook and estimates

of the body of the transistor is depleted (empty of free carriers) during inversion, as opposed to fully depleted SOI where the silicon film is very thin and the region below the whole body of the transistor is depleted of free carriers.

There are two major reasons for the claims about power savings in SOI, namely reduced source-drain junction capacitance, and better sub-threshold slope. We will examine each of these separately.

It is true that the source-drain junction capacitance is lower in SOI due to shallow junctions, which do not extend all the way into the substrate as in the case of bulk transistors. But, bulk transistors, on the other hand, use compensating implants to reduce this capacitance. Moreover, source-drain junction capacitance contributes relatively little to the overall power consumption. Therefore, the benefit of lower junction capacitance in SOI has limited or no impact on power in comparison to modern logic CMOS.

Better sub-threshold slope does permit reducing the threshold voltage of a transistor, providing higher performance for the same leakage power, or lower leakage power for the same transistor performance. Partially depleted SOI and bulk transistors both have comparable sub-threshold slope, and thus comparable performance and leakage power. Fully depleted SOI, on the other hand, shows improved short-channel effects, and much better sub-threshold slope. Research and development activities are already underway in the industry to exploit this. For example, **FINFETs** or **Tri-gate transistors** are inherently fully depleted, and will exhibit this benefit. The power and energy benefit is estimated to be in the 10% to 20% range, and does not constitute orders of magnitude improvement desired for Exascale.

6.2.2.3 Supply Voltage Scaling

This section considers the use of **supply voltage scaling** to reduce power and energy, with benefits as well as design challenges. This is perhaps the biggest available lever that is currently available in conventional silicon technology.

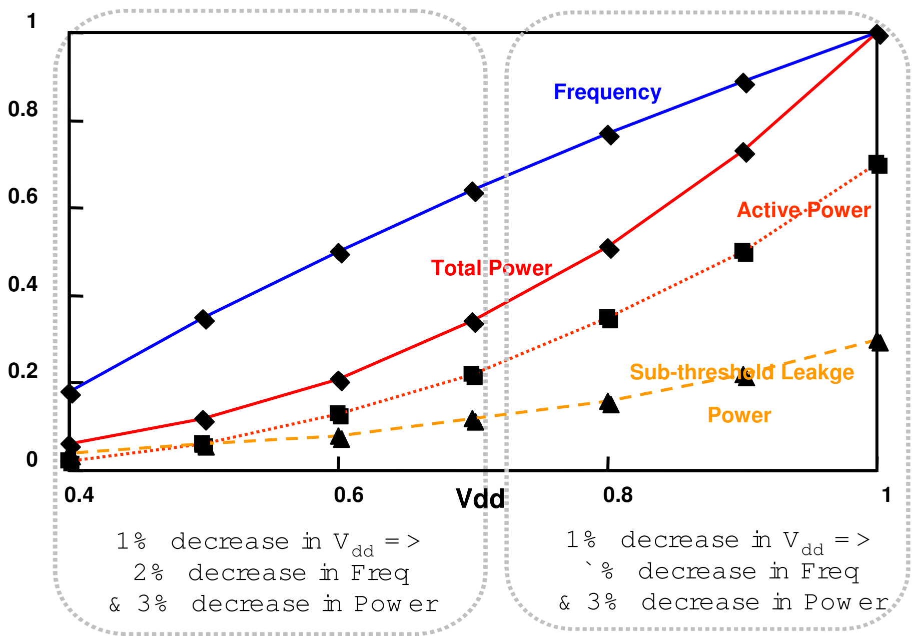


Figure 6.8: Frequency and power scaling with supply voltage

As background consider some circuit that is designed to run at some maximum clock rate when powered by some nominal V_{dd}. As discussed in Section 6.2.1.1, the circuit power is proportional to the square of this voltage, so decreasing it has a significant effect. However, as V_{dd} is decreased, neither the RC characteristics nor the threshold voltage of the transistors making up the circuit change much, meaning that it takes longer for a signal (say from a 0 to a 1) to reach the threshold of the transistors that it is driving. Thus to maintain correct operation of the circuit, the clock must be decreased appropriately. While this decreases the performance of the circuit, it also approximately proportionately decreases the power even further. However, the reduction in performance is lower than the savings in power and energy.

This benefit is shown quantitatively in Figure 6.8, based on an analytical model. As supply voltage is reduced, frequency reduces, but so do the active and leakage power. At higher supply voltage (much above the threshold voltage of a transistor), reduction in frequency is almost linear with voltage. Thus a 1% drop in V_{dd} requires a 1% drop in clock (and thus performance), but saves 3% in active power (the power reduction is cubic). As the supply voltage is lowered even further (close to threshold voltage of the transistor), peak frequency must decrease somewhat faster. Power savings, however, is still higher than frequency loss. Here a 1% drop in V_{dd} requires a 2% drop in clock, but still yields a 3% drop in active power.

Figure 6.9 shows measured power and energy savings in an experimental logic test chip in a 65nm process technology. Nominal supply voltage is 1.2V, and as V_{dd} is scaled down, operating frequency and total power consumption reduce as shown in Figure 6.9(a). Figure 6.9(b) shows energy efficiency as you reduce the supply voltage, where **energy efficiency** here is defined as the number of clock cycles (in billions) of useful work per watt (GOPS/Watt). Notice that the energy efficiency continues to increase with reduction in supply voltage, peaks just above the threshold voltage of the transistor (at about 320 mV in this case), and then starts declining. When the supply voltage nears the threshold voltage, reduction in operating frequency is more severe, compared to reduction in power, and therefore energy efficiency drops.

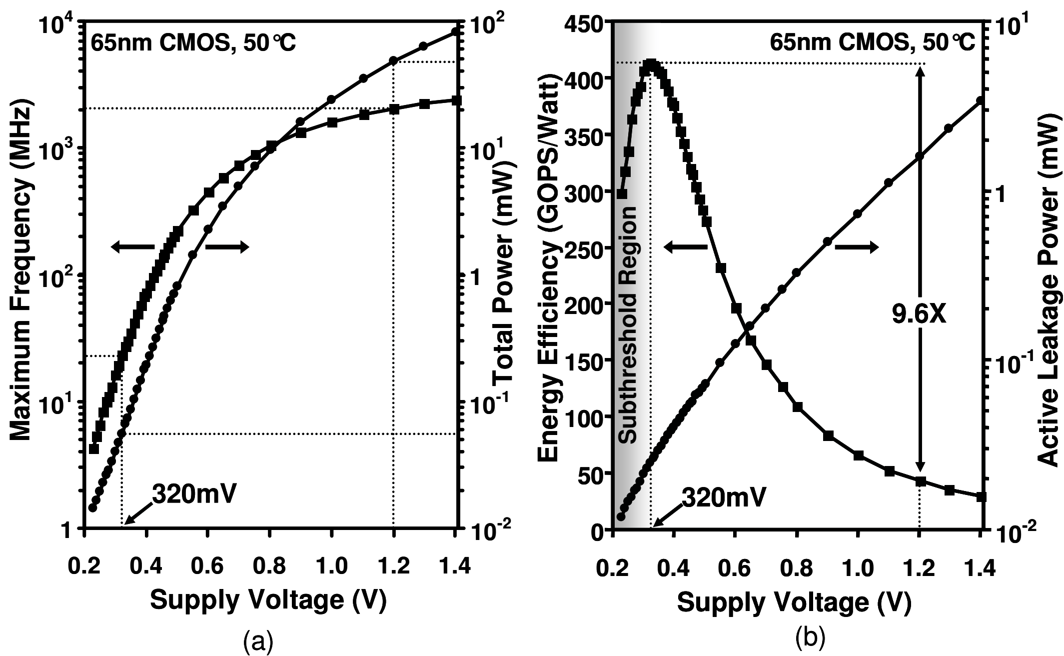


Figure 6.9: Sensitivities to changing V_{dd}.

The downside to this increase in efficiency, at least for Exascale, is that if performance approximating that achieved by the circuit at nominal V_{dd} is desired by reduced voltage implementations, then a significant increase in circuit parallelism is needed. For example, in Figure 6.9, we see that there is approximately a 100X reduction in clock from 1.2V to 320 mV. This would require at least 100 copies of the same circuit to achieve the same performance, with a 100X area cost, and even though the overall savings in power is about 10X.

Figure 6.10 shows how supply voltage scaling benefit will continue even with technology scaling. It plots simulated energy efficiency for the above logic test chip in 65nm, 45nm, and 32nm technologies, with variation in threshold voltage (V_t) of 0 and +/-50mV. Notice that the energy efficiency continues to improve with supply voltage scaling, peaks around the threshold voltage, and has weak dependence on threshold voltage and variations.

In summary, supply voltage scaling has potential to reduce power by more than two orders of magnitude, and increase energy efficiency by an order of magnitude, but at some significant area penalty.

6.2.2.4 Interaction with Key Circuits

Although aggressive supply voltage scaling benefits energy efficiency and power, it also warrants different design practices. That is why most past and present designs do not support supply voltage scaling beyond about 30% lower than nominal. The biggest culprits in terms of circuit types that are difficult are:

- small signal arrays such as memory, caches, register files,
- dynamic logic such as Domino circuits,
- and large fan-in static gates.

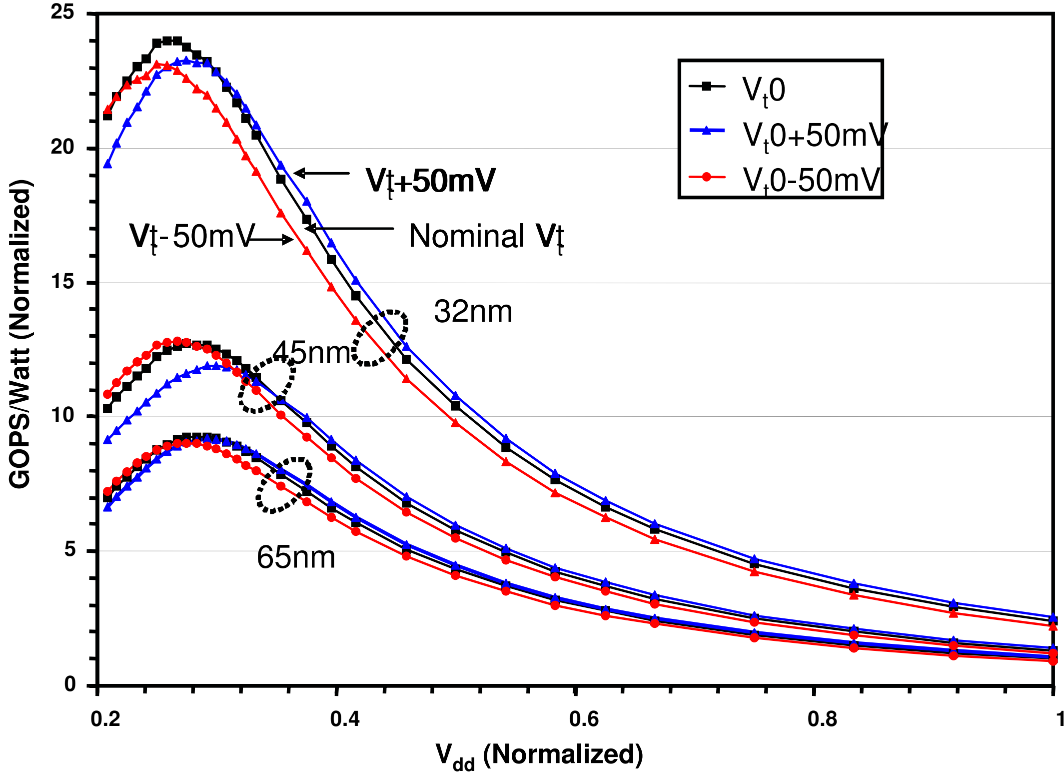


Figure 6.10: Technology scaling, V_t variations, and energy efficiency.

Static RAM (**SRAM**) circuits make up the backbone of a logic chip's need for information storage at almost all but the main memory level. SRAM cells today become unstable at lower supply voltage because they are designed with small transistors for higher performance and lower area. These cells can be designed to work at lower supply voltages by increasing transistor sizes and for full supply voltage (rail to rail) operation. This sacrifices some performance and area, but at a lower voltage, the logic operates at lower frequency anyway, hence performance loss is not a major issue, but the area penalty is, which could be around 10% or so.

Similarly, **Register Files** can be redesigned to operate over a wide range of voltages by replacing Jam-Latches by clocked latches. Again the cost is area and performance.

Domino logic is employed in many of today's chips to increase frequency of operation at the expense of very high power, and thus should be avoided in future designs. Large fan-in gates tend to lose disproportionate amount of performance at lower supply voltages due to transistor body effect. Therefore, designs should avoid using large fan-in gates in the critical paths of the circuits. Overall, designing for low supply voltage operation will be different from what we do today, but not difficult.

6.2.3 Hybrid Logic

Although there are several interesting proposals for performing logic operations using quantum cellular automata (QCA), DNA, diverse types of molecules, optical components, various quantum systems implementing qubits, and other exotic physical processes, none of these have a possibility for contributing to an enterprize-scale computing system within the next ten years. However, there are a few hybrid technologies that involve integrating scaled silicon CMOS with nano-scale

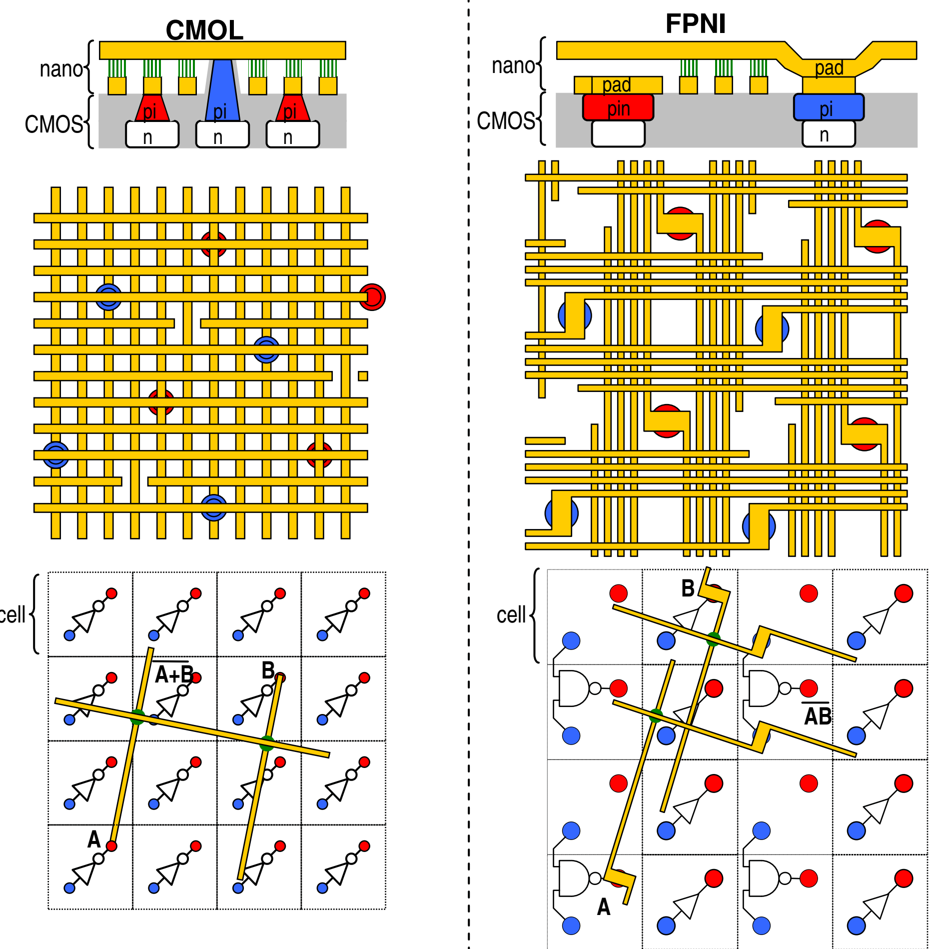
Circuit	Area tm^2				Critical Path Delay ns			Dynamic Power mW	
	CMOS	CMOL 9 nm	FPNI 30 nm	FPNI 9 nm	CMOS	FPNI 30 nm	FPNI 9 nm	FPNI 30 nm	FPNI 9 nm
alu4	137700	1004	17513	5026	5.1	6.53	28.7	0.48	0.061
apex2	166050	914	18983	5448	6	7.10	32.5	0.47	0.059
apex4	414619	672	13457	3862	5.5	5.98	27.1	0.44	0.054
clma	623194	9308	78020	22391	13.1	19.70	85.5	0.78	0.103
diffeq	100238	1194	18983	5448	6	6.86	30.6	0.33	0.044
elliptic	213638	4581	43493	12482	8.6	12.48	56.1	0.50	0.066
ex1010	391331	3486	41252	11839	9	10.03	44.4	0.84	0.106
ex5p	100238	829	11050	3171	5.1	5.42	23.8	0.37	0.047
frisc	230850	4199	43493	12482	11.3	14.02	61.8	0.52	0.068
misex3	124538	1004	14750	4233	5.3	5.52	25.7	0.50	0.061
pdc	369056	4979	48153	13819	9.6	12.74	58.0	0.90	0.110
s298	166050	829	20513	5887	10.7	12.74	58.5	0.25	0.032
s38417	462713	9308	84220	24170	7.3	12.94	63.1	0.93	0.114
s38584.1	438413	9872	66329	19036	4.8	7.80	39.4	1.29	0.153
seq	151369	1296	17513	5448	5.4	6.55	28.9	0.51	0.066
spla	326025	2994	43493	12482	7.3	10.92	48.6	0.84	0.108
tseng	78469	1194	17513	5026	6.3	7.10	29.0	0.25	0.037
total	4494491	57663	598728	172250	126.4	164.45	741.6	10.18	1.29
relative	1.0	0.013	0.133	0.038	1.0	1.30	5.87	1.0	0.13

Data under CMOS and CMOL columns from [142] and FPNI columns from [137]

Table 6.1: Some performance comparisons with silicon.

switching components presently under investigation that could provide higher effective logic density, enhanced performance per power input and/or improved resilience in the time frame of interest.

The first such hybrid technology is to use nonvolatile switches in a nano-scale crossbar array to act as logic devices or configuration bits and switches for a field-programmable gate array (FPGA), as pictured in Figure 6.11. Likharev and Strukov[94] originally proposed this type of architecture, which they named “**CMOL**” to denote a hybrid integrated system that combines molecular switches at the junctions of a crossbar to implement wired-OR functions fabricated on top of a fairly standard CMOS circuit that contains inverters. These researchers executed a series of rigorous simulations of their architecture for a set of 20 benchmark algorithms, and observed performance enhancements of approximately two orders of magnitude in area required to implement an algorithm compared to a standard FPGA architecture that used the same level of CMOS. However, given the aggressive nature of their assumptions (e.g. 4.5 nm wide nanowires in the crossbar), their approach appears to be more than 10 years out. By relaxing some of the assumptions (e.g. using 15 nm wires in the crossbar array, which have already been demonstrated experimentally), designing a more manufacturable process for connecting the nanowires to the CMOS plane, and using the nonvolatile nanoswitch junctions only for configuration bits and connections, Snider and Williams[137] provided a detailed design study for a related architecture they dubbed **field-programmable nanowire interconnect** (**FPNI**). In computer simulations of the same set of benchmarks for their structure, they observed an order of magnitude increase in performance (e.g. area of a chip required for a computation) over a CMOS-only FPGA even in the presence of up to 50% defective switches in the



Schematic diagrams of hybrid-logic circuits. The CMOL design by Likharev and Strukov (left column) places a nanowire crossbar on top of a sea of CMOS inverters. The crossbar is slightly rotated so that each nanowire is electrically connected to one pin extending up from the CMOS layer. Electrically-configured, nonlinear antifuses (green, bottom panel) allow wired-OR logic to be implemented, with CMOS supplying gain and inversion. This is a very high-density design that would not likely be implementable before 2020. FPNI (right column) places a sparser crossbar on top of CMOS gates and buffers. Nanowires are also rotated so that each one connects to only one pin, but configured junctions (green, bottom panel) are used only for programmable interconnect, with all logic done in CMOS. This version was designed to be implementable with today's technology, and is currently in development for space-based applications.

Figure 6.11: Hybrid logic circuits

crossbar, thus realizing both a significant performance improvement and resiliency in a technology that is feasible by 2017. Table 6.1 summarizes these projections.

Other studies of this class of nano circuits include [139], [89], [134], [122], [36], [37], [154], [91], [96], [148], [121], [64], [67], [68], [116], [135], [133], [136], [101], and [49].

A second class of hybrid structures that have been studied are **Programmable Logic Arrays** (**PLA**)[52][53][35][153][164][34]. These structure are basically AND/OR arrays that can implement any n-input m-output Boolean function by "programming" various points in the arrays.

Although these studies relate specifically to FPGA-type architectures, there are potential applications of this technology to Exascale computing. At the present time, there is an effort in place to build a "FPNI" chip for use primarily in space applications where size, flexibility and resilience are at a premium (especially radiation-damage tolerance). By employing different engineering trade-offs, the issues of power and speed could be optimized. A multi-core chip could have some FPGA-like cores to perform specialized functions for the chip, much as many high performance machines today have FPGAs as components. Another possibility is that the main processors cores could incorporate some favorable aspects of the FPNI technology to create hybrid architectures that are more efficient and resilient than today's processors. Adapting such an approach to build specialty cores or introduce some of this technology into "regular" processor cores is a research opportunity for Exascale systems.

6.2.4 Superconducting Logic

Perhaps the most extensively studied non-silicon logic technology uses extremely fast magnetic flux interactions within super-cooled (around 4 °K) superconducting **Josephson Junction** (**JJ**) devices. This technology, in the form of **Rapid Single Flux Quantum** (RSFQ) devices, was the starting point for the HTMT[47] petaflops system project in the late 1990s, and has seen a series of prototype developments since then.

Most recently, a major report written in 2005[3] provided a summary and potential roadmap for the technology. The primary technology conclusion was that if an investment of between $372M and $437M had been made, then by 2010 the technology would have matured to the point where a peta scale design could be initiated. This investment would have targeted a cell library and CAD tool set for RSFQ circuits, a single MCM 1 million gate equivalent processor running at 50 GHz and including 128KB of RAM, and a viable fab facility. The major technical issues that were surfaced in the report included (each discussed in detail below):

- providing memory that is dense and fast enough to support the enhanced logic speeds of the devices,
- developing architectures that were very latency tolerant,
- and providing very high bandwidth communications into and out of the cryogenic cooler.

Although the logic speeds are impressive, as discussed below the density, system level power, and difficulty in transferring data in and out of the required cryostat are all limiters to the technology's general usefulness in across-the-board Exascale systems, except in specialized applications.

The report also draws a possible roadmap for this logic if the investment had been made in 2006, summarized in Table 6.2. As a reference point, a full 64 bit floating point unit in CMOS may take upwards of 50K gates, without much in the way of supporting register files or control logic. Thus the technology marked as "2010" might support 20 such FPUs per 1 cm^2 die, for about 1000

Time Frame	Device Density	Clock Rate	nanoWatts/GHz/Gate
2005	600K JJs/cm^2	20GHz	16
2010	1M JJs/cm^2	50 GHz	8
post 2010 90 nm	250M JJs/cm^2	250GHz	0.4

Table 6.2: 2005 projection of potential RSFQ logic roadmap.

GFlops potential. The "90 nm" technology, if achieved, might pack 500 FPUs for a potential of 125 Tflops per cm^2.

Density and its scaling into future feature sizes is discussed in [23]. The major factor controlling density in JJs is the current densities in the superconducting wires that generate the magnetic fields needed for the junction switches – doubling the current density allows matched stripline widths to narrow by the square root of 2. These wires must now carry current densities of significant magnitude, which limits how small their cross sections can get, and are subject to the Meissner effect for proper operation. Together with the lack of multiple levels of interconnect such as found in silicon, this limits the ultimate sizes to which such devices can be reduced.

6.2.4.1 Logic Power and Density Comparison

The report also summarized several test devices that had been fabricated as of that time. One was an 8 bit serial microprocessor CORE-1 prototype[102][62] demonstrated at 21 GHz local and 1 GHz system, that dissipated about 2.3 mW in the cryocooler. This is equivalent to between 109 and 2300 nanoWatts per MHz, depending on the actual basis of overall performance.

A second demonstration was of an 8 bit parallel microprocessor FLUX-1[24] designed to run at 20 GHz, with a power of 9.2mW, and utilizing 63,107 Josephson junctions on a 10.3 mm by 10.6 mm chip in 1 1.75μm junction feature size. This design corresponded to a power of about 460 nanoWatts per MHz.

As a point of comparison, a modern and complete 32 bit core in a 90nm technology[1] consumes about 40μW per MHz, in an area of as little as 0.12mm^2 when implemented as a synthesized, not custom, design. This includes a multiplier array. In a 2010 technology this might translate into an area of about 0.03mm^2 and a power of about 12μW per MHz. In a 2014 28 nm technology, the design might translate into an area of about 12μm^2 and a power of about 3 μW per MHz.

In terms of area, the report predicted that with the proposed effort a 2010 density of perhaps 1 million JJs per mm^2 would be achievable, which would reduce the FLUX-1's 63K JJs to about 6.3mm^2. Throwing in a factor of 4 for the 8 bit to 32 bit difference, and another factor of 2 for the multiplier array, this implies that 2010 silicon might be functionally around 200 times denser, with 2014 era silicon raising this to over 500 times denser. Even assuming that the projected ultimate potential of 250M JJs per cm^2 was achievable, 2014 silicon would still hold a very significant density lead, and that would improve by another significant factor before silicon runs out.

In terms of power, if we again inflate the power of the FLUX-1 by a factor to allow for a more fair comparison to a full 32 bit core, then we get a number of about 3.7 μW per MHz - about the same as that for silicon in 2014. Using the 2X reduction listed in Table 6.2 from [3], and adding another 2X to approximate the jump from the FLUX-1 technology to the 2005 technology, gives perhaps a 3-4X advantage to the RSFQ.

[1] Based on a MIPS core as described at http://www.mips.com/products/cores/32-bit-cores/mips32-m4k

6.2.4.1.1 Cooling Costs The above comparison is for the logic alone, not the communication with memory, and perhaps more important, the losses in the cooler needed to keep a RSFQ circuit at 4°K. The second law of thermodynamics specifies that the minimum power required (Carnot) to absorb a watt of heat at 4 °K if everything is perfect is given by $(T_h - T_c)/T_c$ where T_h is the hot (ambient) temperature and T_c is the cold temperature. For 300°K and 4°K this Carnot specific power becomes 74 W/W, meaning that 74 W must be expended to keep a 1 W source at 4°K.

In real life, the current state of the art is much worse than this, with typical numbers in the range of 6.5 kW to cool 1.5 W.[2] If this scales to the larger powers one might need for a full-blown Exascale system, it represents a 4300 to 1 multiplier over the logic power.

6.2.4.2 The Memory Challenge

Architecturally, the first major challenge is getting enough memory close enough to such logic to support such computational rates. As of 2005, densities of RSFQ RAMs had been demonstrated at about 16kb per cm^2. As reference, DRAM of that generation ran about 1.4 gigabits per cm^2 - about 1 million times denser. CMOS SRAM, at about 1/40 of DRAM density, was still 25,000X denser. The roadmap called for a 1 Mbit per cm^2 RSFQ memory chip in the 2010 timeframe - about 6000X less dense than DRAM at the time, and 125X less dense than SRAM. Alternative memory technologies, especially MRAM, were also proposed to help out, but required placing them in a "warmer" (40-70°K vs 4°K for the logic). In either case, this still requires a huge amount of silicon (with non-trivial power demands) to be placed in a cryostat if memory densities comparable to those discussed in Sections 7.2.1 or 7.3 are needed.

6.2.4.3 The Latency Challenge

A very related challenge is the latency within such systems. RSFQ is a logic that is inherently pipelined at virtually the device level, meaning that even functional pipelines can grow to the hundreds of cycles, and off chip references even further. Keeping a pipeline with hundreds of stages requires hundreds of independent sets of data to initiate through them, which in turn requires architectures that are much more vector-oriented than current generations of microprocessors. Further, at 250 GHz, a memory chip that is say 100 ns away in the warmer part of the cryo would be 25,000 cycles away from the core. For a typical "byte per flop" of memory data, this would mean that the processing cores might have to generate and track on the order of up to 3,000 concurrent memory references per FPU; even the relaxed taper of the strawman of Section 7.3 still translates into perhaps 500 concurrent memory references. Both of these are far in excess of current practice, and would require very significant amounts of buffering and comparison logic to track them.

6.2.4.4 The Cross-Cryo Bandwidth Challenge

The final challenge is in getting enough bandwidth in and out of each cryostat. This is particularly relevant for systems that may want to grow to multiple cryo systems, such as in a data center scale Exasystem. The problem is in leaving the very cold areas of the cryostat. Doing so with metal wires represents a huge cooling problem; thus optical techniques were proposed in the report. This, however, has the problem of going directly from RSFQ to optical at 4°K. Potential solutions might involve wire from the 4°K to the 70°K region where the memory might live. As is discussed elsewhere in this report (such as in Section 7.5), however, there are no known interconnect technologies that would not require massive amounts of power to be dissipated within the cryo.

[2] See for example the Janis CSW-71D, http://www.janis.com/p-a4k14.html

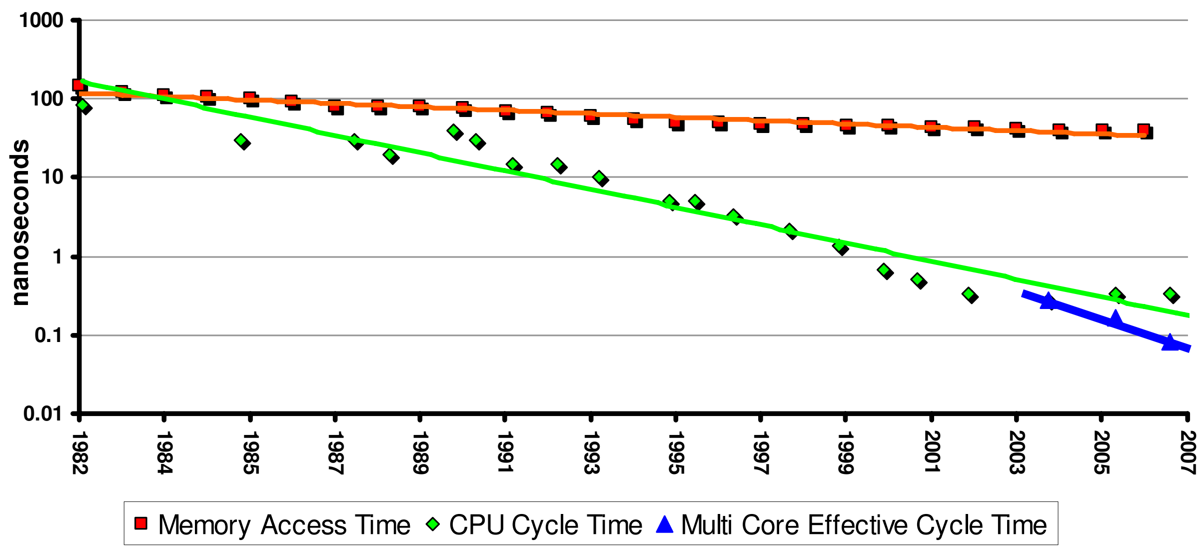


Figure 6.12: CPU and memory cycle time trends.

6.3 Main Memory Today

This section discusses technologies appropriate for use for main memory in computing systems. While the main focus is on today's SRAM, DRAM, and various forms of flash, also covered are some emerging technologies where there is enough of basis to project potential significant commercial offerings in a reasonable time frame.

6.3.1 The Memory/Storage Hierarchy

Today's computers are characterized by having central processors, connected to a hierarchy of different memory types. It has been an evolutionary process getting the semiconductor component industry to this place. Indeed, if one looks at the architecture of the earliest computers the storage hierarchy was essentially flat, yet the hints of hierarchy were there. The Hollerith card decks sitting in file racks of the machine room might have been the first mass storage technology.

Today the situation is more complex. This increased complexity is the result of technological evolution bounded by economics. The technological evolution is brought about by the steady progression of semiconductor scaling and is often expressed in terms of Moore's Law, which states that the density of transistors on a chip doubles every 18 months or so, and these transistors are inherently faster. Economics govern the application of this density increase. The economics of, say, a CPU vendor has in the past often driven that vendor to apply those transistors towards increased performance per cycle, and an increasing clock rate. On the other hand, the economics of a memory vendor drives that vendor to increase the density of the memory devices produced, without significant increases in performance. Similarly, economics govern the evolution of rotating magnetic storage so that once again it is density that is increased, with little improvement in performance.

The result of this evolution is that gaps in performance grow between the CPU and memory, and between memory and rotating storage. Figure 6.12 shows how CPU cycle times have diverged from main memory cycle times over the past 25 years. This difference is now approaching a 1000X, and is colloquially called the "memory wall."

Cell Size (u^2)	Tech Node (nm)	Cell Size(F^2)
IBM/Infineon MRAM		
1.42	180	44
Freescale 6T-SRAM		
1.15	90	142
0.69	65	163
Intel 65nm process 6T-SRAM		
0.57	65	135
Freescale eDRAM		
0.12	65	28
Freescale TFS: Nanocrystaline		
0.13	90	16
Micron 30-series DRAM		
0.054	95	6
Samsung 512Mbit PRAM Device		
0.05	95	5.5
Micron 50-series NAND		
0.013	53	4.5

Table 6.3: Area comparisons of various memory technologies.

In order to maintain the performance of the CPU in the face of this growing gap, the processor vendors have evolved increasingly complex level 1 and level 2 caches, using large portions of their Moore's Law provided transistors to build larger SRAM caches.

6.3.2 Memory Types

In evaluating the suitability of a memory technology for a given application it is helpful to have several metrics by which said memories can be compared. For the purposes of this section, the metrics we will look at are;

1. The speed of the memory;
2. The silicon area required to make the memory;
3. The fault-tolerance of the memory;
4. The power characteristics of the memory.

Because area has been the predominant metric, Table 6.3 lists for relevant technologies and their relative areas per bit, where the term "F" is the semiconductor feature size. Figure 6.13 then graphs the equivalent numbers as projected by the ITRS roadmap on a relative basis, where 1.0 is the relative density of DRAM at that time.

6.3.2.1 SRAM Attributes

Static RAM (**SRAM**) cells are typically constructed from 6 transistors and are most commonly used for today's fastest memory circuits. These cells make up the L1 and L2 cache memories of today's CPUs and can account for over 65% of the die area of some CPUs. The 6 transistors

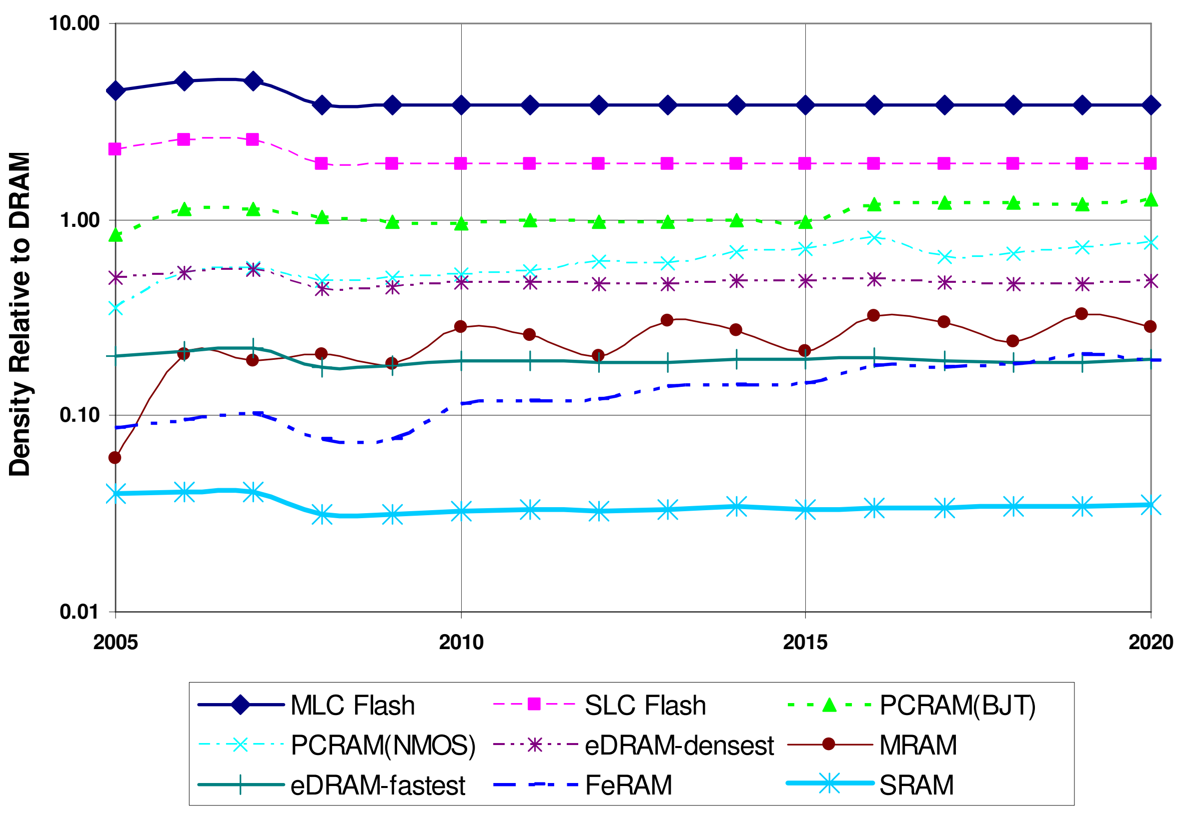


Figure 6.13: ITRS roadmap memory density projections.

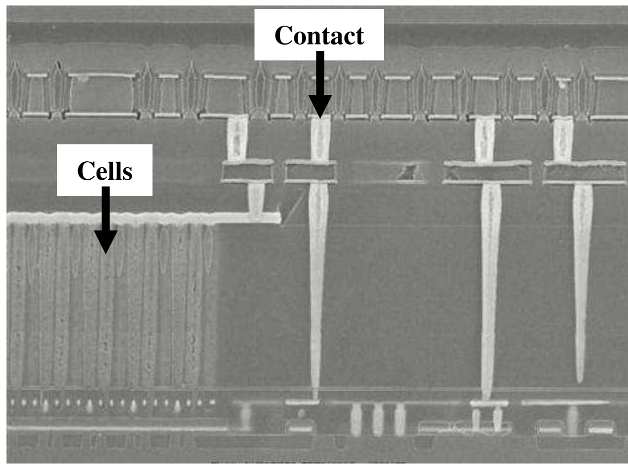


Figure 6.14: DRAM cross section.

and their associated routing makes the SRAM cell one of the largest memory cells. Most SRAM cells are in the range of 140-150F^2. The SRAM cell is a bi-stable latch and requires power to be maintained in order for the cell contents to remain valid. In addition, SRAM cells are subject to radiation-induced failures that affect their **soft error rate** (**SER**), and must be carefully designed with additional ECC bits and a layout that ensures that an SER event does not affect multiple bits in the same data word. SRAM cells may be designed for low power or for high performance. The memories used in CPU caches obviously take the later approach and are thus substantial consumers of power. Approaches such as segmenting the power and reducing voltages to the portions of the array not being addressed to help mitigate SRAM power consumption.

6.3.2.2 DRAM Attributes and Operation

DRAM cells use a capacitor as a storage element and a transistor as an isolation device. In a read operation, the transistor allows the charge on a cell to be placed onto the bit-line, which is sensed by the sense amp and converted to a one or zero. The sense amplifier also boosts the bit-line to a higher voltage, which thus restores the charge on the cell. This read operation is therefore a destructive operation. In a write operation the cell is either drained of its charge or supplied with a charge through the access device and bit-line[79].

There are two types of DRAM cell structures used in commodity DRAM devices. The **stacked cell DRAM** uses a capacitor built above the silicon. The **trench cell DRAM** uses a capacitor built into the silicon. Each has its advantages and its disadvantages, but both face some similar challenges in the next few years. The size of these DRAM capacitors is in the range of 20-30 femto-Farads and it is the ratio of this capacitance to the capacitance of the bit-line that determines the reliability with which these cells can be read. To maintain a reasonable ratio that can be sensed rapidly and accurately the value of this cell capacitance must be maintained. As DRAM cells are typically 6F^2 or 8F^2, there is very little room to build the cell. This is requiring cell aspect ratio to increase at an accelerating rate. The SEM image in Figure 6.14 shows a typical stacked memory cell and the high aspect ratio of that cell.

As the aspect ratio increases it becomes more difficult to etch and fill the trench capacitors. Stacked cell capacitors suffer from toppling and fill difficulties as the aspect ratio increases. Advances in dielectrics and reductions in bit-line capacitances are needed to continue advanced DRAM

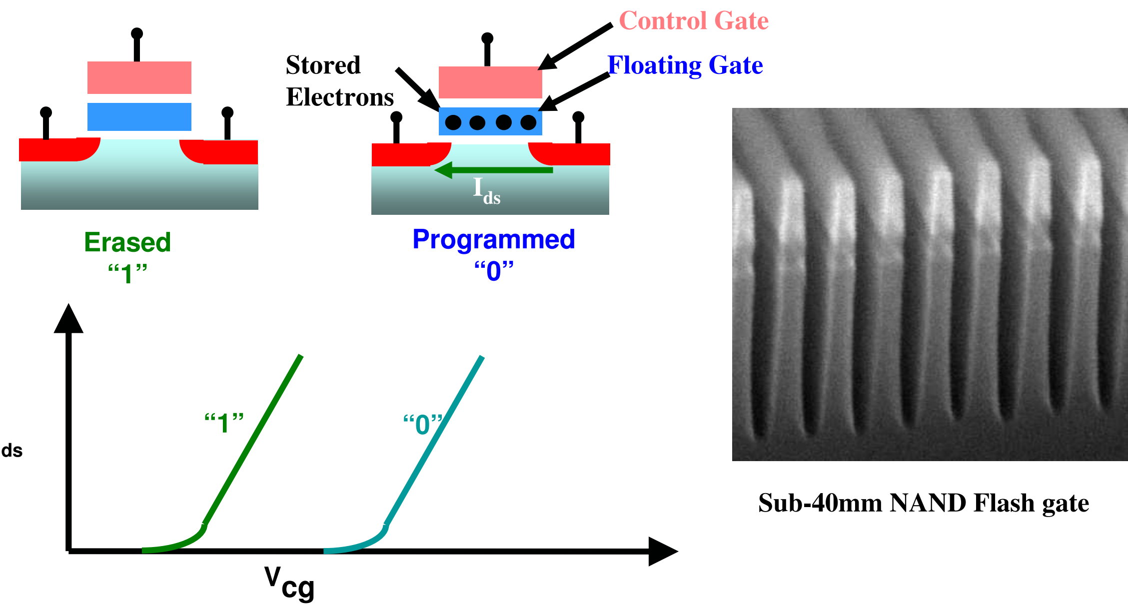


Figure 6.15: Programmed and un-programmed NAND cells.

scaling.

As shown in Figure 6.12, commodity DRAM access times have not kept pace with processors. This is not to say that fast DRAM cannot be built. **Embedded DRAM**, **Fast Cycle DRAM** and **Reduced Latency DRAM** all demonstrate fast DRAM devices. However, all of these carry a substantial die size penalty. In order to achieve the performance, the arrays become less efficient. Each of these fast DRAMs has enjoyed some success, however, although all three have been available for years not one computer maker has elected to use one of them as a main memory technology. Each, however, has merits when used as a cache memory.

6.3.2.3 NAND Attributes and Operation

NAND Flash memory is a non-volatile memory and operates by trapping electrons on a secondary gate structure of a MOS transistor (see Figure 6.15). This secondary gate is a floating gate and receives its charge when the gate voltage is elevated enough for electrons to tunnel onto the floating gate. Here, the charge is trapped and biases the junction when the cell is read.

The gate dielectric and structure have been carefully engineered to retain the trapped electrons for 10 or more years. Still, NAND systems must be carefully designed so that writing nearby bits do not affect previously written data and such that reading the cells does not allow trapped charge to leak from the floating gate[92].

NAND devices have been highly optimized for mass storage applications and are block oriented devices. A typical 16Gb NAND device will be organized as 4096 4Mbit blocks. Individual cells cannot be erased or written, but rather, entire blocks must be erased at a time.

In addition to its non-volatility, NAND has the advantage of having small cells. Typical NAND cells are $4F^2$ and with today's **Multi-Level-Cell** (**MLC**) technology these cells are capable of storing two bits per cell. Future roadmaps show 3 bits/cell and even 4 bits/cell as being possible for certain applications.

NAND memory must be used with error correction. Additional bits are provided within each memory block for the storage of error correction codes. Typically a BCH or Reed-Solomon code

Memory Type	Cell Size	Endurance
SRAM	142F^2	$> 1E15$
DRAM	6F^2	$> 1E15$
NAND	4F^2-2F^2*	10K-100K
PCRAM	4F^2	100K-1E6
MRAM	40F^2	$> 1E15$
* Multi-Level Cells		

Table 6.4: Memory types and characteristics.

will be applied to NAND blocks, allowing several errors to be present without risk of data loss.

NAND Flash memory is not capable of infinite read-write cycles (termed **endurance**). Most **Single-Level-Cell** (**SLC**) NAND is rated for 100K cycles with ECC. MLC NAND is usually rated for fewer cycles, even under 10K cycles.

The slow writes, block-oriented erase and programming, and the limited endurance or today's designs all make NAND unsuitable as a main memory replacement. As a replacement for, or alongside rotating magnetic media, NAND has great potential. Challenges do exist in scaling NAND gate and dielectric structures[113], but the industry is putting significant research efforts into the problems.

6.3.2.4 Alternative Memory Types

There are many types of memory that are being investigated in an effort to find replacements for SRAM, DRAM, and NAND Flash memories. The "Holy Grail" of memories would be one that is faster than SRAM, of unlimited endurance like DRAM, and both dense and non-volatile like NAND Flash. In addition, the cells should be compatible with existing CMOS logic, with a low-cost manufacturing process and consume near zero power. Of those memory types that have demonstrated some degree of commercial viability, few have shown that they are as economical or robust as today's memory leaders. Table 6.4 summarizes the cell sizes and endurance of some of the top memory contenders.

6.3.2.4.1 Phase Change Memory One of the most promising memory types on the near term horizon is **Phase Change Memory** (**PCRAM**. Phase change memories operate by melting a small region of material through resistive heating, and then cooling under controlled conditions to either an amorphous or crystalline solid, which then exhibits two resistance states. The material used in the cell is a chalcogenide glass, and while the thought of melting glass may seem an unlikely memory technology, it may be viable. One of the factors in favor of phase change memory is that it actually improves as it gets smaller: As the melted regions are shrunk, the current required to melt the region is reduced.

Given the characteristics of phase change memories, it is most likely they will find use first as a NOR Flash replacement then perhaps as a NAND Flash replacement. Their currently projected endurance characteristics will keep them from main memory applications.

6.3.2.4.2 SONOS Memory **Semiconductor Oxide Nitride Oxide Semiconductor** (**SONOS**) memory cells are generally considered to be a natural extension of Flash memory technology. Rather than utilize a floating gate for the storage of charge an Oxy-Nitride-Oxide floating trap layer is used. It is anticipated that SONOS memory cells will be constructed that are equal to floating gate

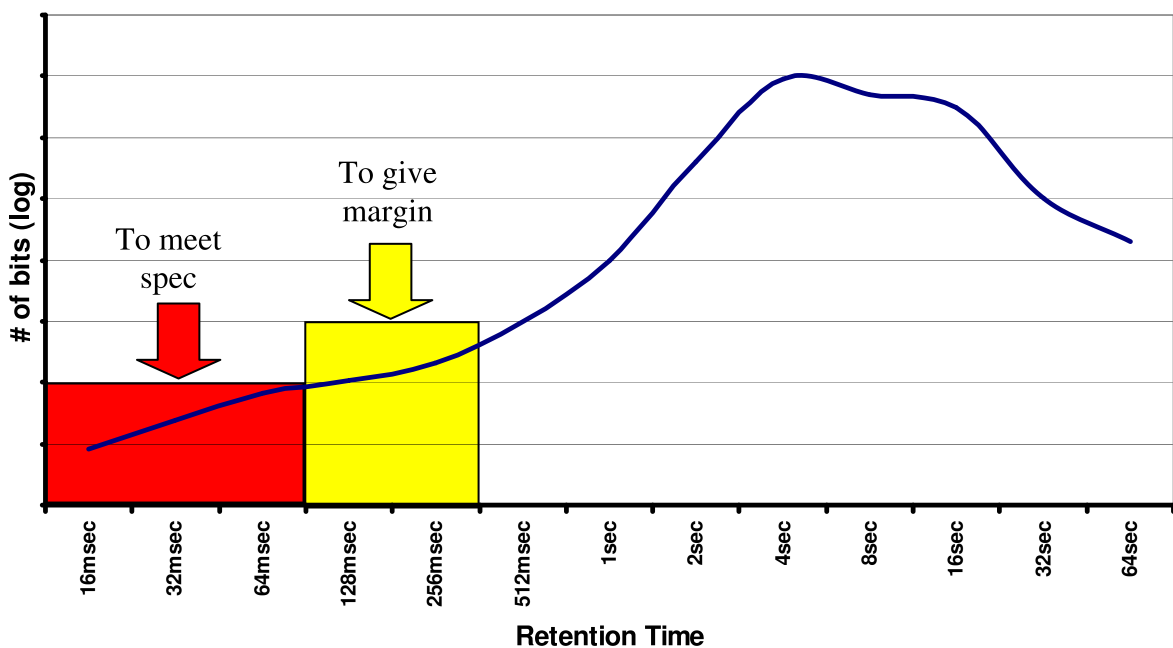


Figure 6.16: DRAM retention time distribution.

(FG) NAND cells. Endurance of SONOS cells has been observed to 10^7 cycles, which is on par with FG NAND. One additional feature of SONOS cells is that the programming voltage is lower, which is a definite advantage as the cells continue to shrink. SONOS cells also exhibit radiation hardness[156].

One area of research for SONOS and other memory technologies is in 3-dimensional stacking of memory cells. As scaling in X and Y become more challenging, integration in the Z-direction by stacking multiple layers of memory cells is showing some promise[70].

6.3.2.4.3 MRAM **Magnetic Random Access Memory** (**MRAM**) is based on the use of **magnetic tunnel junctions** (**MTJs**) as memory elements. MRAM is a potentially fast non-volatile memory technology with very high write endurance. One of the issues with MRAM is that it requires high currents to switch the MTJ bits. These write currents may be in excess of 5mA per bit and present a host of design challenges. This reason alone likely limits MRAM to use in smaller arrays in specific applications, such as rad-hard systems for space applications.

6.3.3 Main Memory Reliability - Good News

The DRAM used in today's computing devices has proven to be one of the industry's reliability success stories. In part, this reliability is due to the on-chip redundancy and how this redundancy is used to provide each memory chip with a maximum level of operating margin. While DRAM for computing applications generally specifies a 64msec refresh period, the processes used to build DRAM generally results in cells that retain a readable charge for several seconds.

Programmable redundancy was first suggested for DRAM as early as 1969; however, it was only implemented on a production basis with the 64Kbit and 256Kbit density generations in the mid-1980s. The programmability of these redundancy solutions has taken multiple forms, and today is split between **laser-trimmed fuses** and **electrical anti-fuses**. The most notable

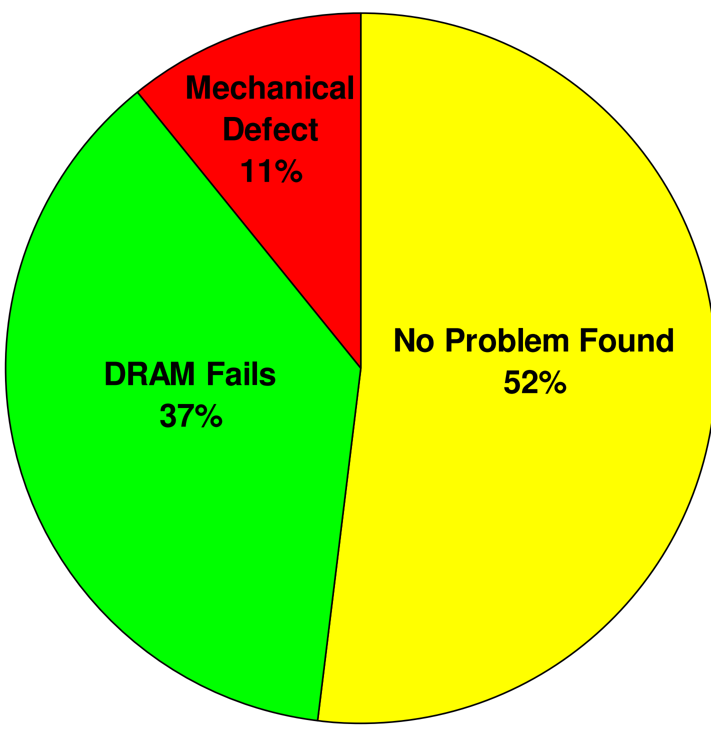


Figure 6.17: Memory module RMA results.

advantage of the electrical anti-fuse is that the arrays can be repaired late in the manufacturing process, even after packaging and burn-in.

Currently, DRAM manufacturers are using all available fuses to maximize the margin in the devices, as illustrated in Figure 6.16. It might be possible in the future to reserve a portion of the DRAM redundancy elements for use within an operational system such that field failures could be repaired in a live system. Clearly this would not be possible with laser-trimmed fuses. For electrical anti-fuses there are still challenges. No standards exist for accessing these fuses and even how these fuses interact with the memory array. Even within one manufacturers product portfolio there are often significant differences in fuse configurations between parts. "Super-voltages" need to be applied to appropriate interface pins which may not be compatible with other elements connected to those pins. Some engineering could resolve these issues although it is unlikely such a system would find its way into mass-market commodity DRAM.

6.3.3.1 Trends in FIT Rates

A **FIT** is defined as the failure rate for one billion operational hours. Existing mature DRAM products have shown tremendous improvements in FIT rates, even into single digits (less than 10 failures per billion hours). However, the sheer volume of memory, and number of potential memory devices, in at least the largest of Exascale machines will ensure that memory failure is an issue that must be dealt with. For example, if the memory FIT rate is 5 per memory device, and an Exascale machine has 0.1 Exabyte of memory made from 1 Gigabyte memory devices, the system could experience a memory failure every two hours. Clearly the Exascale machine must be designed with some resiliency for failures in the memory subsystem.

One common source of failures today is not accounted for in these FIT numbers - the memory DIMM sockets. The data in Figure 6.17 from a major manufacturer of memory devices shows the

distribution of **RMA** (**Reliability, Maintainability, and Availability**) results on analysis on reported failures in that company's memory modules. Of the 52% which indicate "No Problem Found" it must be assumed that issues with module sockets are responsible for a large portion of the returns.

While RMA data is typically indicative of early field failures, the memory socket problem may also have a long term aspect. Anecdotal evidence from a large support organization indicates that simply removing and re-inserting the memory modules cures nearly all memory-related problems.

6.3.3.2 Immunity to SER

Memory soft errors due to ionizing radiation are a common part of the collective experiences of most large system designers. As recently as 2003 at least one supercomputer was built without regard for soft error rates, and demonstrated its usefulness as a cosmic ray detector rather than as a computer. This system was subsequently dismantled and re-built with support for Error Correcting Code (ECC) memory with much better success.

While **soft error rate** (**SER**) is a bad memory (pun intended) for the industry, the good news is that, for properly designed devices and systems, SER for DRAM is but a memory and DRAM memory is largely without SER.

What makes DRAM SER resilient? Ionizing radiation may be viewed as discrete events at the silicon level. Such radiation has finite energy. DRAM cells, word lines, and bit lines are all highly capacitive. These forms of ionizing radiation simply lack the energy needed to affect the cell. Even when a cell would strike one of these nodes, the timing of the strike would need to be at just the right time to even be noticed. Furthermore, DRAM cells are so small that they individually represent a small target, but collectively represent a large capacitance in a unit area. If ionizing radiation strikes the silicon, there is a lot of capacitance in the area to absorb the effect. Certain nodes in the DRAM device could still remain sensitive to ionizing radiation, however the industry has been careful to avoid layouts that are such. This is particularly true of the sense amp area.

SRAM cells are not so fortunate. In order to allow increased operating speeds with reasonable power the SRAM cell uses very low capacitance nodes. As SRAM cells are quite large, $140F^2$, they present a very large target for ionizing radiation. To keep SER at reasonable levels SRAM arrays must employ careful layouts - such as ensuring that neighboring cells are addressed in different words - and must employ external error correction.

6.3.3.3 Possible Issue: Variable Retention Time

There is a seldom-discussed characteristic of DRAM that must be addressed due to the impact it could have on an Exascale system. This issue is the **Variable Retention Time**, or **VRT**, bit. Sometimes called "flying bits" the VRT bit is one that was first publicly acknowledged in 1987 by AT&T Bell Labs[16], and was confirmed by IBM[6], to exist in all known DRAM technologies and process nodes. The VRT bit is a memory bit that shows changes in its retention behavior, typically flipping its retention time between two values due to some external influence. This external influence might be temperature (packaging processes, reflow), stress (mechanical and electrical), x-rays (from inspection), or other influences.

6.3.3.3.1 Causes The causes of VRT bits are uncertain. It is likely that the mechanism has actually changed with different process generations. Today the most likely cause of VRT bits is the possible existence of a trap in or near the gate of the access transistor. The activation of the trap

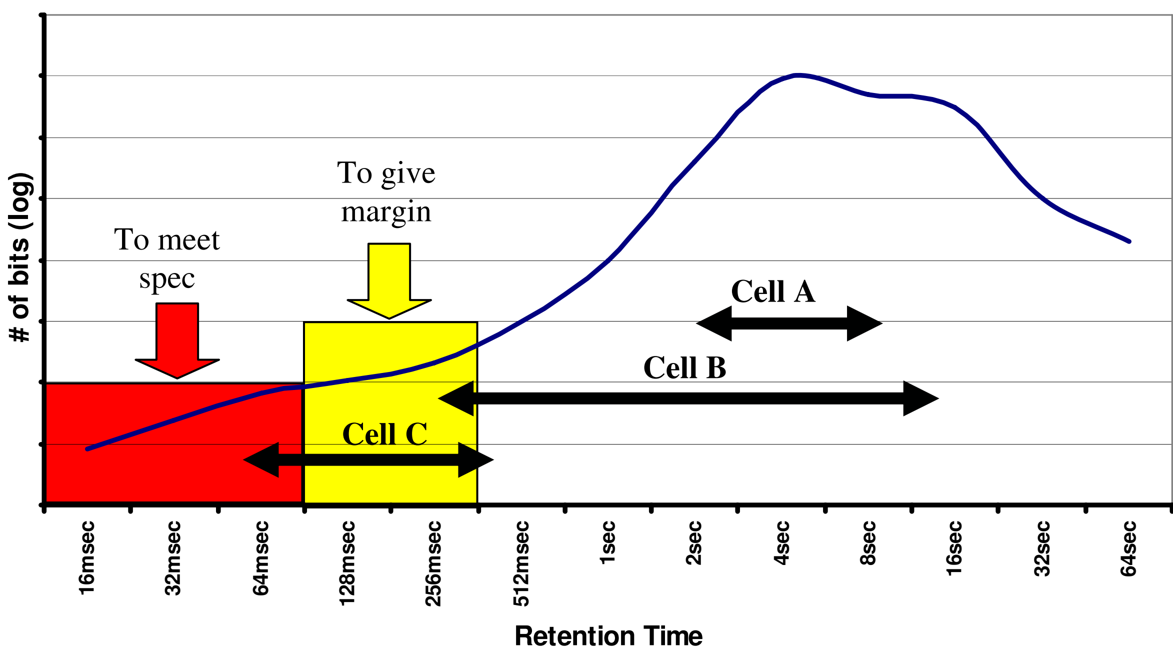


Figure 6.18: Variable retention time as it affects refresh distribution.

possibly affects the leakage of the access device, moving retention from long to short or from short to long.

6.3.3.3.2 Effects Consider the distribution of retention time for the billion-plus bits on a DRAM device. As previously discussed, redundancy in the array and fuses are used to remove short retention time cells from the array of cells seen by the system. Additional redundancy and fuses are used to provide a margin between the specified refresh time for the device and the refresh distribution of cells in the array. Now suppose the DRAM device is subjected to one or more of the VRT trigger events. It might be the DRAM being reflow soldered to a module PCB. It could be the injection molding package operation. It could be PCB inspection. If a cell is affected by this event and experiences a shift in its retention time to a time shorter than the refresh period seem by the device in the system that bit may appear to be bad. Of course another trigger event may cause the cell to return to its long retention time.

In Figure 6.18 a VRT shift in the retention time of Cell A or Cell B does not cause a noticeable change in memory behavior: all bits still look good. However Cell C shifts it's retention time between an acceptable time and a time that is shorter than the system specified 64 msec. This cell will show up bad if it is in the short retention time state, and good in the long retention time state.

6.3.3.3.3 Mitigation Fortunately the frequency of VRT cells is low, and it is statistically unlikely that multiple VRT bits would show up in a single DRAM word. At the system level the addition of ECC is sufficient to remove the risks of VRT cells. As process technologies evolve, the industry continues to search for ways of limiting and reducing the prevalence of VRT cells in devices. VRT should remain as a concern however, as circuits continue to scale and as innovative packaging solutions are incorporated into Exascale systems.

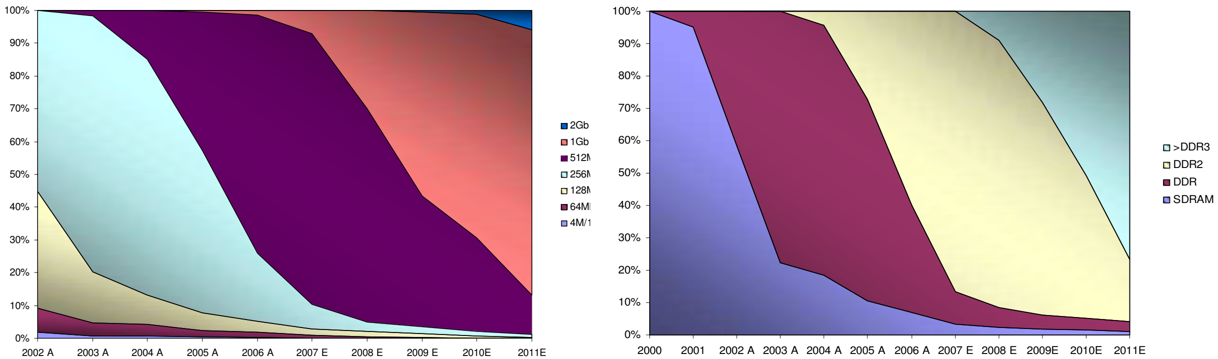


Figure 6.19: Industry memory projections.

6.3.4 The Main Memory Scaling Challenges

As shown in other sections of this report, the Exascale system demands unprecedented volumes of memory and that the memory must have both extremely high bandwidth and low latency. The sheer volume of memory constrains the power of individual memory devices. To achieve desired latency and bandwidth requires new architectures and interfaces between CPU and memory.

While commercial systems would greatly benefit from the types of architectures and interfaces discussed in this document, evolution of commercial memory is driven by committee (**JEDEC**) and may not serve the needs of such Exascale systems.

6.3.4.1 The Performance Challenge

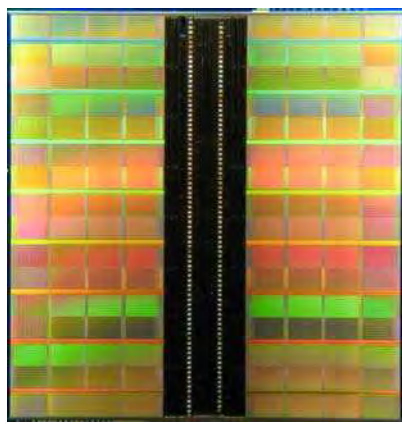
Memory performance may be broken into two components: **bandwidth** and **latency**. Commercial DRAM evolution continues to improve interface bandwidth, while sacrificing memory latency. As the industry has moved from DDR to DDR2 and now to DDR3, the interface speeds have increased while the latency has also increased. The reason for this is straightforward. As the interface speed increases the memory chips must incorporate deeper and more complex pipelining logic due to the near constant access time for the array. Unfortunately the additional logic consumes more silicon area and more power. The challenge for the Exascale system will be to deliver the performance while simultaneously decreasing die size and power.

Commodity DRAM is forecast to continue density scaling with modest improvements in interface speed. The iSupply data[73] in Figure 6.19 show the projected trends for commodity DRAM. The Y axis in both cases is the percent of chips shipped.

6.3.4.1.1 Bandwidth and Latency DRAM can deliver higher bandwidth and lower latency. Figure 6.20 shows a die photograph of a reduced latency DRAM (RLDRAM) device. This device uses architectural changes to the sub-arrays to achieve faster array access times. There is a significant 40-80% die area penalty in making this type of change to the array. Additionally, this part supports a 36-bit wide interface, which has some additional die-size impact (more interface logic in the middle). If an Exascale system is to take advantage of lower DRAM latency and high bandwidth, a new architectural approach must be taken.

6.3.4.1.2 Tradeoffs There are some tradeoffs that can be made between bandwidth and latency. One tradeoff demonstrated in commodity DRAM is the use of additional I/O pipelining

(a) A Conventional DRAM



(a) A Reduced Latency DRAM

Figure 6.20: Reduced latency DRAM.

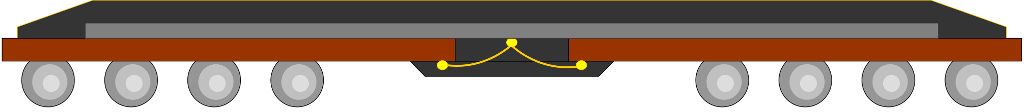


Figure 6.21: Center-bonded DRAM package.

to give bandwidth, at the expense of latency. Some non-standard DRAM architectures have even gone beyond commodity DRAM in this regard. This tradeoff has made sense for DRAM manufacturers as the peripheral transistor performance has been lagging behind that of logic processes. The additional pipeline stages become necessary to achieve the high bandwidth specifications. As in CPU architectures, the logic between pipeline stages is also reduced as the frequency climbs. Of course, an unintended tradeoff made when following this course is that the logic power and logic area of the DRAM increase.

One interesting study could be made of the possible power and performance results of an architecture which minimizes I/O pipelining logic, utilizing wide I/O buses, and instead utilizes some silicon area for improving latency and power through array segmentation.

6.3.4.1.3 Per-pin Limitations DRAM processes have been optimized for low leakage (increased data retention time) and high manufacturability. Low leakage transistors are not on par with high performance logic transistor performance. In addition, DRAM devices are generally optimized towards low pin counts for low cost packaging and low cost test.

Commodity DRAM has traditionally favored full-swing, single-ended signaling, although next generation DRAM may have low-voltage differential signaling. While differential signaling will likely double the signal bandwidth, it takes twice as many pins to deliver the data. Power will likely climb again.

6.3.4.2 The Packaging Challenge

Commodity memory packaging is driven largely by cost. A standard lead frame technology remains the preferred low-cost package for most applications. However, this is changing. In certain markets,

the need for increased density or improved signal integrity justifies additional packaging expense. The micro-FBGA packaging utilized in some server memory modules is one such example. Most commodity DRAM solutions rely on memory die with centrally-located I/O pads. I/O is limited to one or two rows of contacts that run down the center of the die. Figure 6.21 illustrates a cross section of a commodity micro-FBGA solution.

If the Exascale system requires memory with wider I/O, it is unlikely that existing packaging technologies will provide a viable cost-effective solution. Pin count, memory density and signal integrity likely drive the solution towards some sort of 3D die-stacking technology. There are many such technologies, but many are not suitable for DRAM. Wire-bonded solutions allow limited stacking as they typically require die periphery connections.

Existing commodity DRAM is already a highly 3D device. Cell capacitors, whether trench or stacked technology, are high aspect-ratio devices either constructed atop the silicon or etched into it. Capacitor surfaces are intentionally roughened to increase surface area. Access transistors are highly-engineered 3D structures optimized with unique channel profiles to reduce leakage.

In order to achieve densities required in certain applications, DRAM devices have been stacked at the package level for years. Only recently have stacked die-level DRAM devices been in high-volume production. Innovative packaging technology has enabled these solutions, but the industry is advancing towards more integrated stacking solutions that can be achieved at the wafer scale using semiconductor processing techniques.

Through silicon vias compatible with DRAMs are under development at most DRAM manufacturers. Interest in stacking DRAMs also comes from outside the DRAM industry as others look to integrate more power-efficient or high performance multiprocessor solutions.[16][80]

Some of the challenges with efficient stacking of DRAM are related to the existing 3D structure of the cells. To efficiently stack die requires die that are sufficiently thinned so that the through-wafer via etch may be done economically. However, the 3D structure of the cell limits the degree to which the die may be thinned. As previously noted, thinning of the die may also affect the refresh performance of the DRAM cells. Additional research is needed to overcome these challenges to develop a 3D stacking process that is high-yield and economical.

6.3.4.3 The Power Challenge

The Exascale system will face major challenges in the area of memory power consumption. Power in DRAMs come from two major sources: accessing the memory arrays, and providing bits off-chip. Commodity memory devices have only recently begun to address power concerns as low power DRAM devices have become standard for applications in mobile phones and portable electronics. Most often these DRAM devices have achieved power savings through modest process enhancements and architectural advances. In many cases the architectural advances have been towards improving the standby current, with a goal of improving battery life for portable consumer electronics systems. The Exascale system will demand a more aggressive approach to power management. Once again, this is due in large part to the sheer scale of the system. However, one point must be clearly understood: the DRAM cells themselves are very efficient. It is not the cells, but rather the interface to those cells that is the major consumer of memory power.

6.3.4.3.1 Module Power Efficiency Figure 6.22 illustrates the historical trend for power in commodity memory modules as a function of off-bit bandwidth. Each such memory module has multiple DRAM die that are driven together, and together provide system bandwidth. As can be seen, the power efficiency of memory, measured in mW/GB/s is improving at only a modest rate.

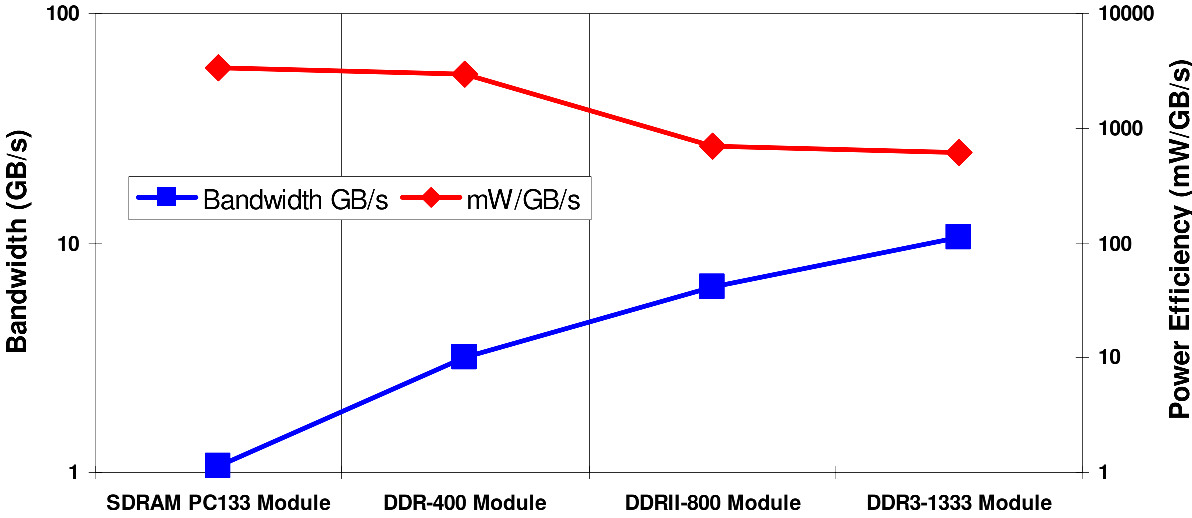


Figure 6.22: Commodity DRAM module power efficiency as a function of bandwidth.

6.3.4.3.2 Cell Power Memory power at the cell level is fairly easy to understand. A DRAM cell may simply be viewed as a capacitor which must either charge a bitline or be charged by that bitline. The DRAM capacitor is currently in the range of 25 femto Farads for most commodity applications. The bitline capacitance is considerably higher, typically several times that of the cell capacitance. If one assumes a total capacitance of 100 fF and a cell voltage of 1.8V then the energy to write or read a cell may be approximated as:

$$Energy = Capacitance * Voltage^2 = 100fF * 1.8V^2 = 81femtoJoules \tag{6.9}$$

This level of energy efficiency is never seen at the periphery of the memory device. A memory cannot exist without the row and column decoders required to access the desired cells. Steering logic is required to direct data from sense amps to I/O. All of these consume additional energy regardless of the type of memory cell technology employed.

Voltage scaling is slowing for DRAM as it is for logic. Figure 6.23 shows the trend in DRAM operating voltage, which is not expected to scale far beyond 1 volt. The possibility of reduced cell capacitance, device variability in the sense amplifiers and noise margin are all contributors to a slowing of voltage scaling.

An area of research worthy of study would be development of the technologies required to scale DRAM cells and periphery circuitry to lower voltage operation. Operation at 0.5V is, in theory, possible, but remains out of reach with current developmental paths. Such research could bear significant commercial dividends as well. By some estimates[87], data centers consume 14% of the country's power production growth. The same study estimates memory power consumption to be 27% of the data center's power load.

6.3.4.4 Major Elements of DRAM Power Consumption

As shown in the previous section, DRAM cells are actually quite efficient by themselves. However, there remains room for improvements to the cell. Voltage scaling must be extended and is not supported by current roadmaps and research. Voltage scaling must be extended to the periphery of the DRAM array as well, a task that also remains out of the reach of current roadmaps and

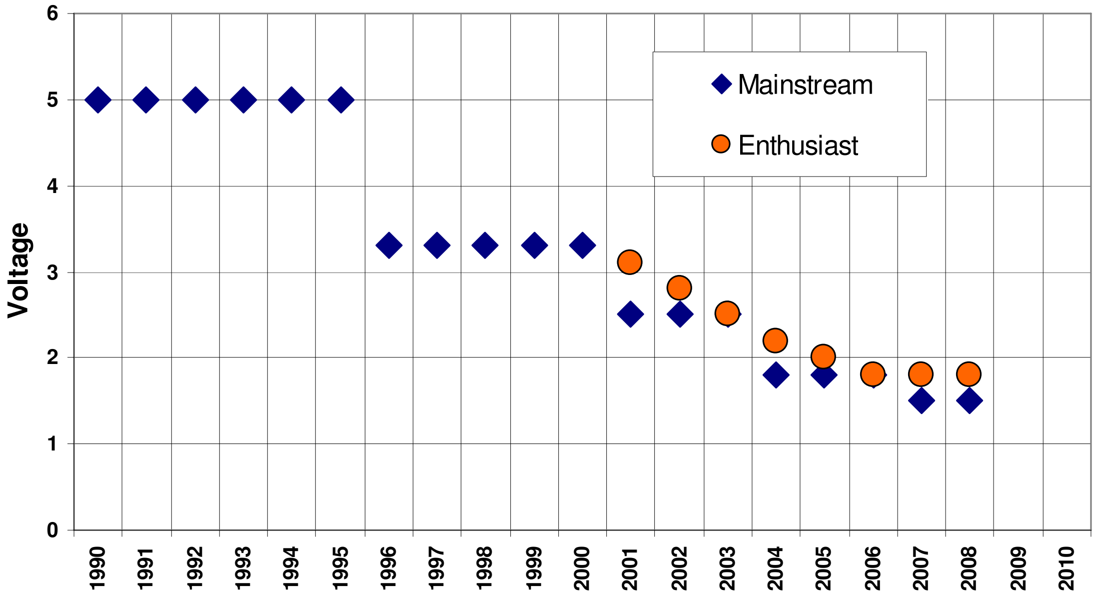


Figure 6.23: Commodity DRAM voltage scaling.

DDR2-400 IDD Specifications		
Operation	Symbol	Current
Idle	IDD2P	7ma
Refresh (Burst)	IDD5	280ma
Precharge	IDD2Q	65ma
Activate	IDD1	110ma
Read	IDD4R	190ma
RWrite	IDD4W	185ma

Table 6.5: Commodity DRAM operating current.

research. In order to understand the impact of DRAM periphery in the overall memory power picture it is helpful to study current devices and trends.

6.3.4.4.1 DRAM Operating Modes DRAM operation may be broken into six operations: **Idle**, **Refresh**, **Precharge**, **Activate**, **Read**, and **Write**. For the purposes of this discussion on Exascale systems we may ignore the power down modes that are often used in notebook and desktop computers in their Standby operation. For a state of the art DDR2-400 memory device, typical operating current for each of these operations is shown in Table 6.5, along with their common notation.

As Idle is of little interest, and Refresh is a small percentage of overall operation we will focus our attention on the Precharge, Activate, Read and Write operations.

Before a DRAM cell can be read, the bitlines associated with that row must be brought to a V/2 level, where V is the operating voltage of the DRAM array. This is the Precharge operation and is really just a preparation for a Read or Refresh operation.

The Activate operation is the opening of a memory row for a read or write operation. During the Activate operation the row decoders select the appropriate wordline and drive that wordline to

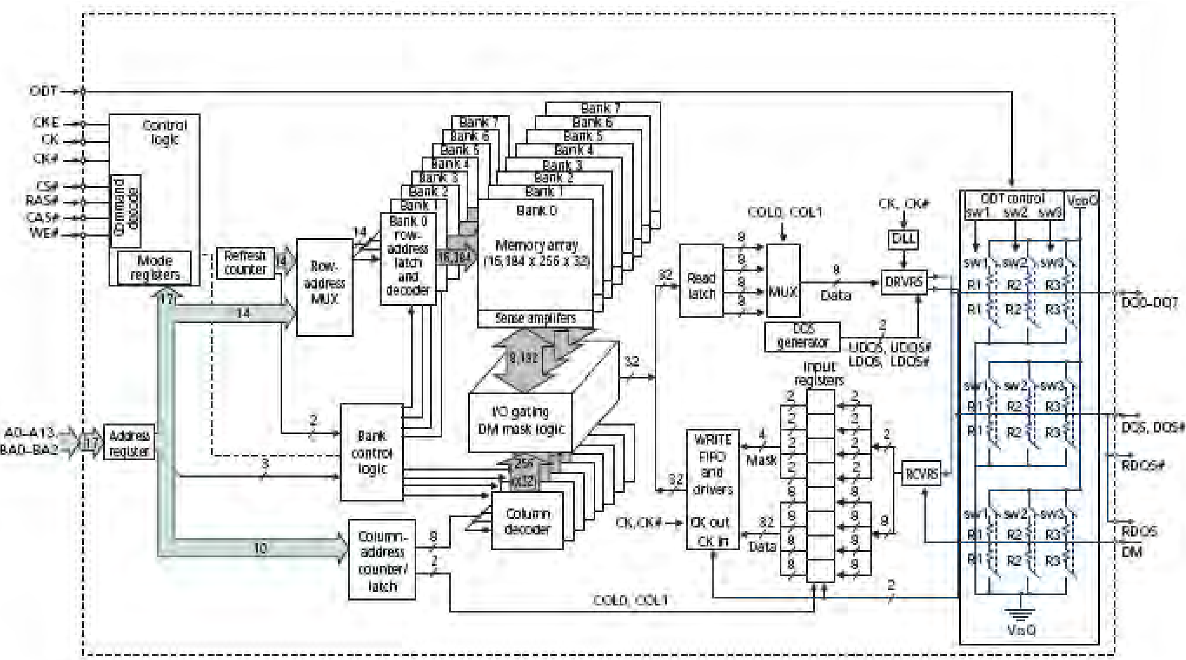


Figure 6.24: Block diagram of 1Gbit, X8 DDR2 device.

a higher pumped voltage, typically approximately twice Vdd. As the cells charge the digit lines the sense amplifiers are activated. As far as the array is concerned this is the highest power operation the array experiences.

Steering the data from the sense amplifiers, through registers and pipelines to the output and driving the data from the part accounts for the Read operation power.

Similarly, a write operation begins with a Precharge and Activate. As with the Read operation, the row decoders must decode and drive the appropriate wordline with the pumped voltage. The written data is driven into the array by the sense amplifiers. The bits in the row that are not being written are refreshed in this same operation.

6.3.4.4.2 DRAM Architecture The block diagram of a commodity DDR2 DRAM device of Figure 6.24 is representative of the architecture of any modern DDR2 or DDR3 DRAM. However, there are several elements not shown in Figure 6.24 that are of import as one considers power and performance of the memory array.

First, the block diagram does not show the charge pumps necessary to drive the wordlines. The optimized low-leakage transistors which comprise the access devices in the memory array require this pumped voltage to enable low $R_{DS_{on}}$.

Second, the block diagram does not show the actual construction of the memory **banks**. Each of these 8 banks is further broken down into multiple **sub-banks**, which are further broken down into multiple **mats**, each of which is further divided into multiple sub-arrays. While the diagram indicates a single 14 bit to 16,384 row decoder per bank, in reality there are many decoders that drive the individual sub-banks, mats, and sub-arrays.

Finally, the "I/O Gating and DM Mask Logic" block is greatly simplified. Helper flip-flops store the output of the sense amplifiers. A critical speed path in the architecture on most designs is the gathering and steering of data to and from the multitude of sub-arrays. It is a complex logic element that incorporates most of the fastest (and therefore most power-hungry) transistors in the

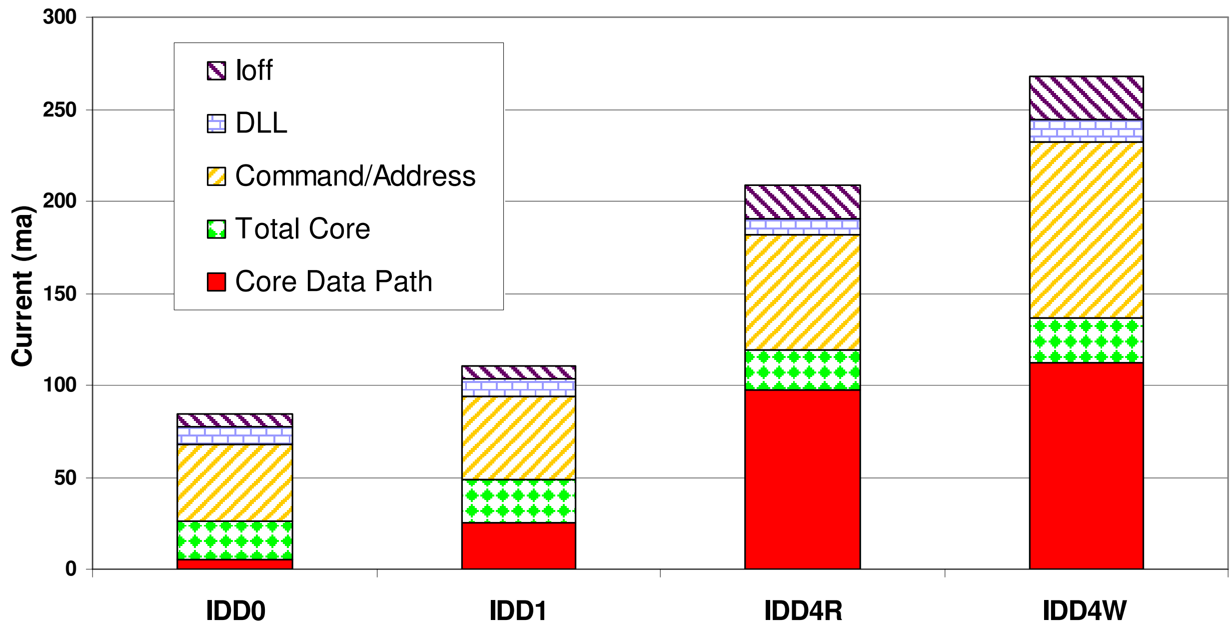


Figure 6.25: DDR3 current breakdown for Idle, Active, Read and Write.

device. Also included in this logic are structures for device testing.

Figure 6.25 tallies the current drawn in each of the major sections of a commodity DDR3 DRAM device as a function of operation mode. Multiplying this by V_{dd} and by the percent of time that the chip is performing that type of operation yields power. The numbers presented are similar to those from a DDR2 device as shown in Figure 6.24.

This figure illustrates what has been said before: The total core power is dwarfed by the power of the surrounding circuitry when the device is operating.

6.3.4.4.3 Power Consumption Calculations An accurate memory power calculator is beyond the scope of this report. Such technology does exist for today's commodity DRAM devices and the reader is invited to experience one[106]. Tools for analyzing SRAM power and allowing architecture optimization have existed for many years. One such tool, CACTI, shows some potential and could be enhanced to support DRAM architecture analysis and the advanced ITRS roadmap[158][98][145].

For the purposes of this report the needs of the Exascale machine must be considered, especially a data center-sized system. It is safe to say that the commodity DRAM roadmap does not fulfill the needs of the Exascale machine. Power consumption is simply too great for the type of memory footprints under consideration. For example, given that a DDR3 chip today consumes just over 600 mW/GB/sec, the power budget for an Exascale data center machine requiring 1 EB/sec of main memory bandwidth would be just over 600 megawatts, which is simply not viable.

But step back for a minute and consider the memory at the cell level. As previously stated, DRAM cells are not inefficient, with cells requiring only about 80 femto Joules for switching. If one considers just a memory array delivering 1 EB/sec bandwidth, these DRAM cells consume only about 200 Kilowatts of power! (based on a future 1 volt, 25fF cell cap). When bitline capacitance is added, the number becomes 800 Kilowatts, based on a cell to bitline cap ratio of 1:3[79].

In current DRAMs, one must consider the power consumed across the entire row. In typical

commodity DRAM, 8K bits are accessed for any read or write operation. If the system is able to use additional data from the 8K row then the row may be left "open" for sequential accesses. However, in practice there is usually little locality in memory references, and it is most efficient to "close the row" to prepare for another random access to the device. What is the power penalty paid here? For a commodity 8-bit wide DRAM device that bursts 4 bytes to fill a cache line, and with an 8K bit row, the power consumption jumps to over 51 Megawatts for 1 EB/sec. (Note that the row access rate is now reduced to 1/4th of the previous due to the 4-byte burst at the interface.) Clearly, over-fetching of unused data by opening an entire DRAM row is the most significant power problem with scaling commodity DRAM architectures to the Exascale machine.

Driving the wordline/indexwordline can also consume substantial power. Here, the voltage is pumped, and the line is long, resulting in significant driving energy. If we assume the wordline capacitance is 3pF for the entire 8K row, and that a word line must be driven to $2 * V_{dd}$ at a rate 1/4th that of the data date (burst transfer length of 4), the power consumption of the aggregate Exascale memory wordline is about 1.8 Megawatts for the same future 1V memory cell. Of course, this does not include any of the pumps, level translators or row decoder power.

It should be clear that the DRAM memory cell power is not the limiting factor for the power consumption of the Exascale memory subsystem. Other technologies may be proposed for the memory cell, but these will result in the same power-hungry memory subsystem if the periphery of the array is required to perform the same types of operations.

Future research directions to enable the Exascale memory subsystem must include analysis of techniques to reduce power consumption around the memory array.

6.3.5 Emerging Memory Technology

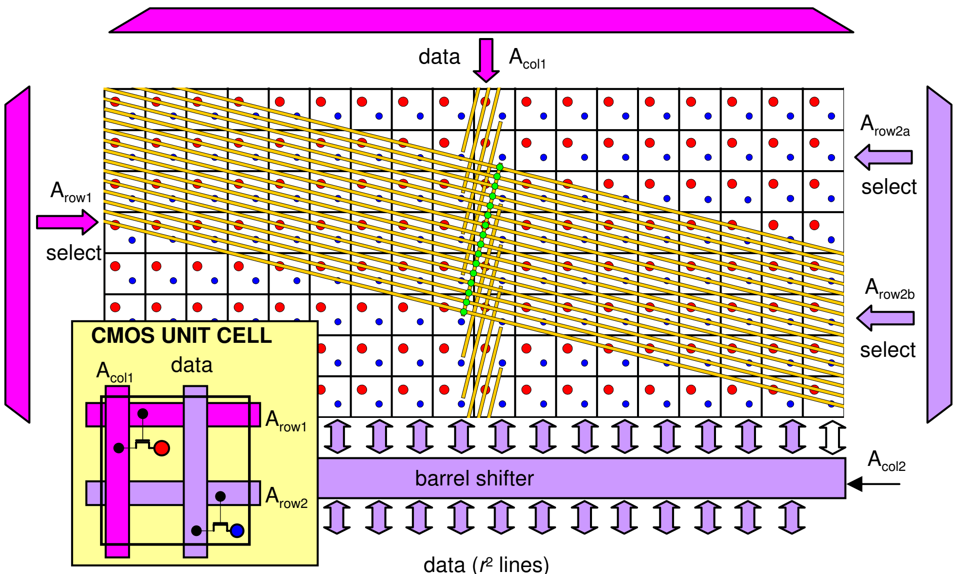
There is currently a significant effort world wide to develop new **non-volatile random access memory** (**NVRAM**) that can complement or replace existing memory and storage technologies. While many different NVRAM contenders are rising to this challenge; at this stage, it does not look like any will replace an existing technology outright. However, there may be applicability with new system architectures that place a layer of NVRAM between DRAM and magnetic disk storage to buffer the latency and bandwidth gaps between present memory and storage technology options. Driving this bandwidth gap wider (and thus making such architectures more valuable) are two trends: (i) off-chip access requires more and more clock cycles and (ii) significant nonvolatile storage is demanded by an increasing number of applications.

Today the highest density emerging memory technologies will meet this need by combining a dense crosspoint memory array with a fast, nonvolatile crosspoint device. A crosspoint memory offers the highest possible $4F^2$ cell density in a single layer, and most variations offer the possibility of significantly improving over this by being able to stack many layers of crossbars on top of a single addressing and control layer. Several device technologies may offer the needed combination of high bandwidth and low-latency nonvolatile electrical switching. These technologies include phase-change PCRAM, ferro-electric Fe-RAM, magnetic MRAM, and resistive RRAM[29][51][14].

Of these, FeRAM and MRAM devices are currently the most mature, with 1-4 Mb chips available for niche applications now, but expansion into large-scale computing environments has been hampered by poor device scaling (these technologies are limited in size by super para-electricity and super para-magnetism, respectively, which are made worse by the inevitable high temperature operating environments of an Exascale system), complex device structures and large fluctuations in corresponding behavior, and high power demands.

Phase change RAM are also power hungry, and show mechanical device instability over time.

Resistive RAM devices based on various metal sulfides or oxides, although less mature at present,



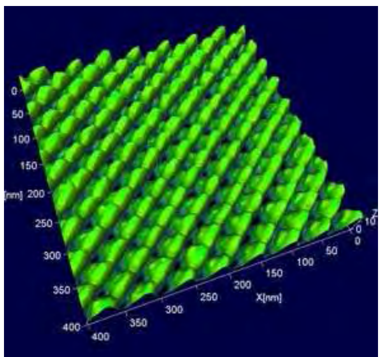
Addressing nanoscale bits in a particular memory segment with four CMOS decoders in 'CMOL' memory. The figure shows only one (selected) column of the segments, the crosspoint nanodevices connected to one (selected) segment, and the top level nanowires connected to these nanodevices. The nanowires of both layers fill the entire array plane, with nanodevices at each crosspoint. The inset shows a simple schematic of the CMOS cell structure for this memory.

Figure 6.26: Nanoscale memory addressing.

offer a compelling combination of relatively high speed/low latency (tens of nanoseconds), non-volatility (months to years) and low energy (2 pJ per bit in present unoptimized devices) switching, which could meet checkpoint demands and might even shift the requirements for DRAM for Exascale computing systems. At present, the metal oxide systems are receiving the most attention, since they are closely related to materials already found in today's integrated circuit fabrication facilities (especially hafnium dioxide) and thus the amount of process work needed to incorporate them onto chips and the worries of incompatibility with existing fabs is minimal.

One of the primary problems with crosspoint NVRAM is the **multiplexer/demultiplexer** (**mux/demux**) needed to read and write information into the memory[28][56]. There have been several schemes introduced to enable the transition from scaled CMOS to nanoscale crossbars, including the use of coding theory to design defect-and fault-tolerant demuxes. The most ingenious scheme was proposed by Strukov and Likharev[94], who showed that it is possible to layer an extremely high density crossbar memory on top of a lower density mux/demux and still achieve complete addressability with defect- and fault-tolerance (see Figure 6.26). They simulated the bandwidth for such systems and determined that 1TB/s for a 10 ns read time with ECC decoding was achievable as long as the array is large enough that all overheads are negligible. The major challenge in this case would be to transmit the data off of or into an NVRAM package at the data rate that it can handle - this may actually require photonic interconnect or close proximity to DRAM to achieve.

To date, the highest density NVRAM structure that has been demonstrated is a 100 Gbit/cm^2 crossbar that was fabricated using imprint lithography, Figure 6.27[76][77]. Since this structure is made from two layers of metal nanowires that sandwich a layer of switchable dielectric material, this technology is well suited to stacking multiple crossbars on top of each other. Since the storage



An Atomic Force Microscope (AFM) topograph of a defect-free region in a 17 nm half-pitch nanowire crossbar fabricated by imprint lithography. This corresponds to a local memory density of ~100 Gbit/cm^2. Using the CMOL memory demultiplexing scheme of Strukov and Likharev, this crossbar could be placed over a two-dimensional CMOS array for reading and writing. Multiple crossbars could be stacked on top of each other to achieve even higher bit densities.

Figure 6.27: Nanoscale memory via imprint lithography

Year	Class	Capacity (GB)	RPM	B/W (Gb/s)	Idle Power(W)	Active Power (W)
2007	Consumer	1000	7200	1.03	9.30	9.40
2010	Consumer	3000	7200	1.80	9.30	9.40
2014	Consumer	12000	7200	4.00	9.30	9.40
2007	Enterprise	300	15000	1.20	13.70	18.80
2010	Enterprise	1200	15000	2.00	13.70	18.80
2014	Enterprise	5000	15000	4.00	13.70	18.80
2007	Handheld	60	3600	0.19	0.50	1.00
2010	Handheld	200	4200	0.38	0.70	1.20
2014	Handheld	800	8400	0.88	1.20	1.70

Table 6.6: Projected disk characteristics.

is non-volatile, there is no static power needed to hold or refresh the memory, and thus the thermal issues of stacking multiple memories are also minimized.

Defect tolerance in such devices is a major concern, with a growing effort to define fault tolerance mechanisms and project potential characteristics[9][33][95][141][142][143][163].

6.4 Storage Memory Today

This section discusses the technologies from which mass store memory used to implement scratch, file, and archival systems. Clearly, the major technology today is spinning disk, although there are several others emerging.

6.4.1 Disk Technology

Rotating magnetic disks have been the major technology for scratch and secondary storage for decades, so it was deemed important in this study to understand whether or not it can continue in that role into the Exascale regime, and discussions were held with leading disk firms, along with compilations of relevant historical data[2].

Disks have several major properties that need to be tracked: capacity, transfer rate, seek time, and power. Table 6.6 lists projections of these properties for three classes of disk drives:

Consumer: high volume disks where capacity and cost is paramount.

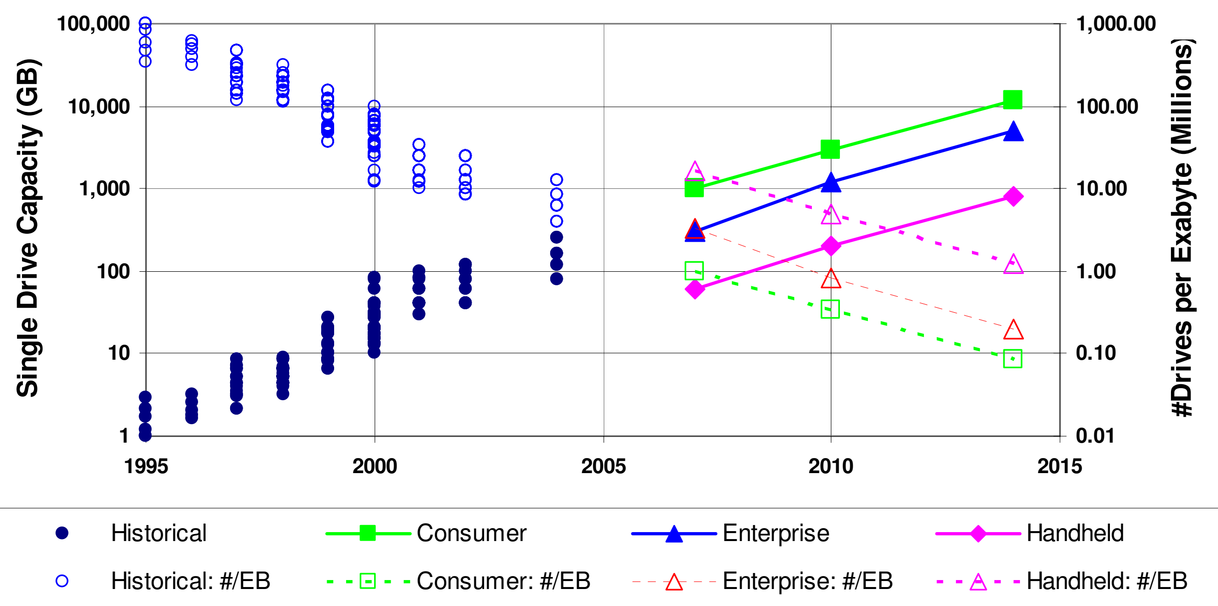


Figure 6.28: Disk capacity properties.

Enterprise: disks where seek and transfer time is paramount.

Handheld: disks where power and small size for embedded applications are paramount.

6.4.1.1 Capacity

To put these projections in perspective, Figure 6.28 graphs both historical and projected capacity. As can be seen, 10X growth over about 6 year periods seems to have been the standard for decades. Assuming that the basic unit of secondary storage for data center class systems is an exabyte, then depending on the type of drive, between 83 thousand and 1.3 million drives of 2014 vintage are needed per exabyte. Consumer drive technology, with its emphasis on capacity, requires the fewest drives, and handhelds the largest.

Any additional drives for ECC or RAID are not counted here, so these numbers are optimistically low.

Also not studied here is the actual physical volume needed for such drives.

6.4.1.2 Power

Another major parameter is power. Figure 6.29 projects the active power in MW for an exabyte of disks of each class. In 2013-2014, the range is between 0.8 and 3.8 MW, with enterprise class taking the most power (bigger drive motors to run drives faster).

We note that the power numbers here are for the drives only; any electronics associated with drive controllers needs to be counted separately.

Again ECC or RAID is not considered, so real numbers would be higher.

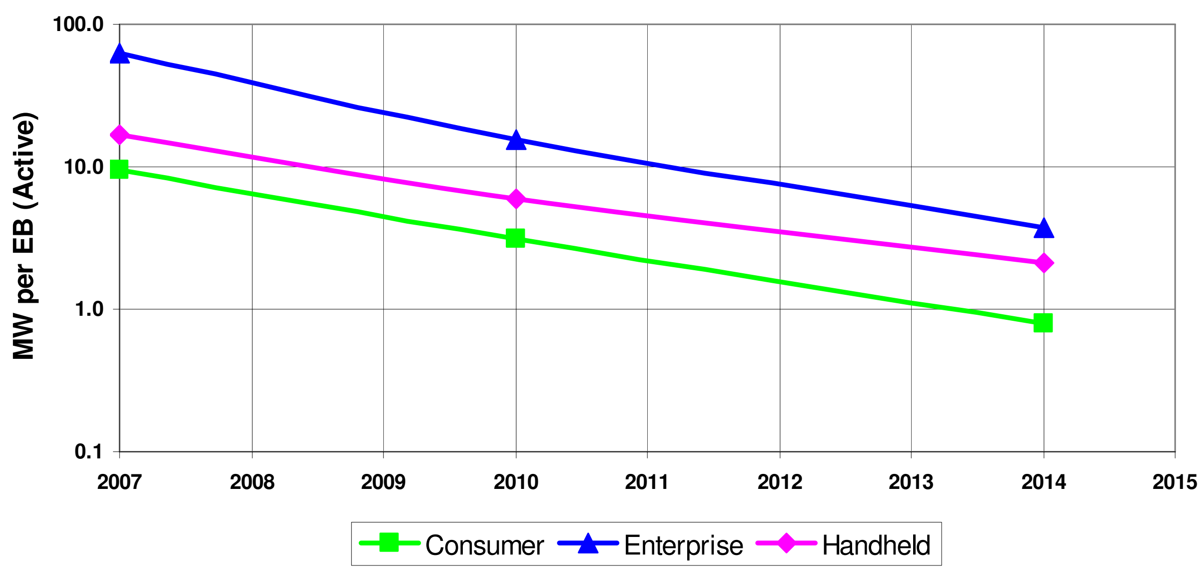


Figure 6.29: Disk power per Exabyte.

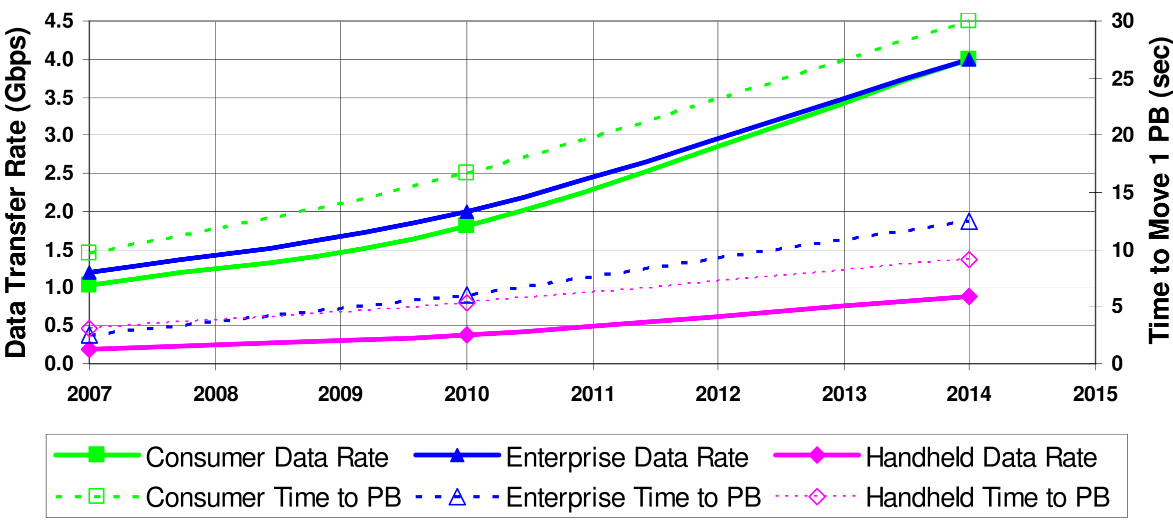


Figure 6.30: Disk transfer rate properties.

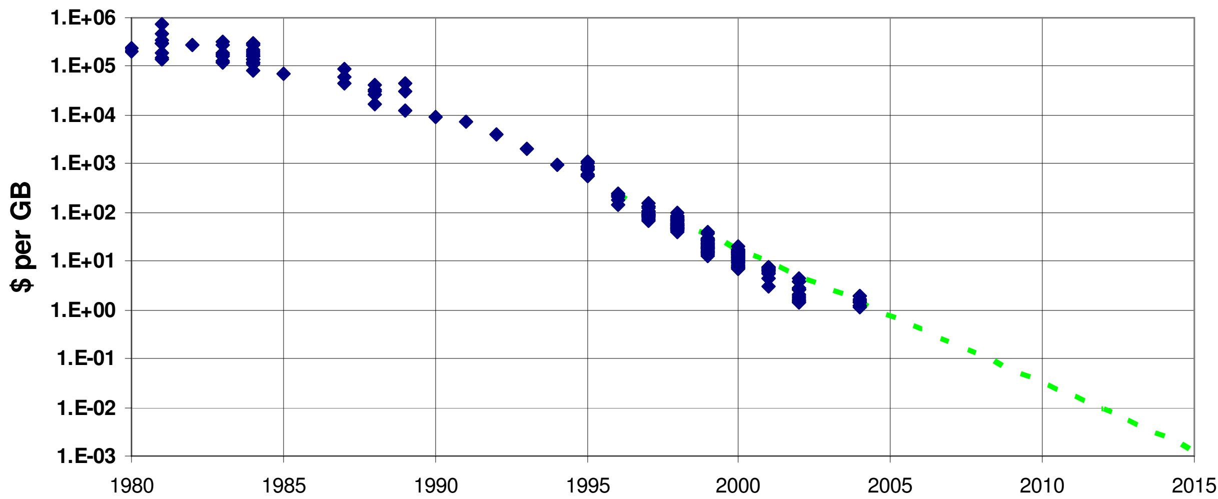


Figure 6.31: Disk price per GB.

6.4.1.3 Transfer Rate and Seek Time

The transfer rate of a drive provides one part of the overall bandwidth capabilities of a single drive. This transfer rate is defined as the maximum data rate that data can be streamed from a drive, given little or no head seeks. Achieving such rates requires data to be transferred in large (multi-MB) segments, and to be physically recorded in optimal places on a drive so that when one block is read, the next block to hold the next relevant chunk of data is directly under the disk heads.

For each of the three classes of drives, Figure 6.30 provides projections of the maximum transfer rate under the above conditions.

Of course, in real systems the number of seeks and the average seek time become a dominating consideration, especially for file systems with many small files. In general, the average seek time is proportional to the rotational rate. Unfortunately, as shown in Table 6.6 this rate seems essentially flat for all classes of machines, meaning that future drives of any class will not respond to seek requests any faster than they do today. This will become a huge problem, especially for searching directories and accessing many small files.

6.4.1.4 Time to Move a Petabyte

Also included on Figure 6.30 is an estimate of how much time it would be required to move one PB of data, assuming that we were able to perfectly stripe the data across the number of drives needed to contain an exabyte (from Figure 6.28), and that each such drive transferred data at the maximum possible rate. This transfer time increases even as the transfer rate of an individual drive increases because the number of drives over which the assumed exabyte of data is striped is decreasing. This is a very relevant number in discussions related to scratch disks and time to checkpoint data from memory to disk.

This is also again an optimistic minimal number, with no consideration for problems with drives out of sync, skew between the data arrival from the individual drives, or seek times.

Also again ECC or RAID is not considered, so one would expect that additional time would be needed to read out and perform any needed error detection or correction.

6.4.1.5 Cost

Although not a direct metric for this study, cost is still a consideration. Figure 6.31 gives some historical data on price per GB of capacity. To get to an exabyte, we would change $1 from this figure to $1 billion. The reduction rate in something in excess of 10X per 5 years, leading to a predication of a few $10 millions for an exabyte in 2015.

Again neither RAID, controllers, nor interconnect cables are included in these estimates.

6.4.2 Holographic Memory Technology

Holographic memory refers to the use of optical holograms to store multiple "pages" of information within some storage medium. Recording is done by splitting a light source into two coherent beams, passing one through an image, and then recombining them on some photosensitive storage material, typically a photopolymer of some sort[39]. The recombination causes interference patterns, and if the intensity of the light source is high enough, the interference pattern is stored in the material. Readout is achieved by shining just the original reference light source through the material, and then detecting the resultant image.

For many materials, changing the angle of the light beams permits multiple holograms to be stored in the same material.

Given that the original data is an "image," such holographic storage systems are "page"-oriented memories - reading and writing are in units of "images." If an image is a "bit-pattern," then in terms of a digital storage medium, a holographic memory is said to be a **page-oriented memory**. Typical sizes of such pages seem to be around 1 Mbit of data.

Two forms of such memories have been demonstrated to date: one based on 3D cubes of storage material, and one based on disks. In 1999, a breadboard of a 10 GB non-volatile cube was demonstrated which when combined with the required optical bench required 154in^3[27]. More recently, a commercial product[3] has demonstrated a drive with removable optical disks with 300 GB capacity with an overall form factor of about 700in^3, a seek time of 250 ms, a transfer rate of 20 MB/s, 1.48Mb pages, and page write times of about 1 ms. The storage material in the latter demonstrated 500 Gb per in^2[10], which is perhaps 10X the current density of hard disks, and about 60 times the density of today's DVDs. This density is what makes the technology of interest, especially for archival storage.

In comparison, current DRAM chip density is around 84 Gb per in^2, and flash is about 2-4X that. Of course, this is density and not dollars per Gbyte, which today favors the spinning medium by a significant factor.

If bits per cubic (not square) inch are the metric, then disk drive technology seems to hold a significant advantage – a current commercial drive[4] fits 1 TB in about 24in^3, or about 100X denser. Even silicon has an advantage, especially as we find better 3D chip packing. The current holographic memories take a real hit on the volume needed for the optics, and unless techniques that reduce this by several factors (perhaps by sharing lasers), this is unlikely to change.

In terms of seek times (a key parameter when doing random reads), current disks are in the 8-9 ms range, significantly faster than the early commercial offering. Again, this is an first-of technology, and significant advances are probable, but there is still significant ground to make up.

[3] InPhase Tapestry 300r; http://www.inphase-technologies.com/downloads/2007-11PopSciAward.pdf

[4] Seagate Barracuda ES.2 http://www.seagate.com/docs/pdf/datasheet/disc/ds_barracuda_es_2.pdf

6.4.3 Archival Storage Technology

As discussed in Section 5.6.3.3, archival storage is experiencing both huge capacities, rapid growth, and problems with metadata for huge numbers of files. Today, storage silos and tape farms of various sorts are keeping up with the 1.7-1.9 CAGR of current installations, but it is unclear whether they will be able to make the jump to Exascale, especially for the data center class of systems which is liable to start off with 1000X of Petascale.

Besides the capacity issue, the metrics raised in [55], especially for "scanning" an archive as part of advanced data mining applications, focus attention on the need for dense enough and fast enough storage to hold not the data but the metadata that defines and controls the actual data files, especially when the potential for millions of concurrent accesses is present. Such data is read-mostly, at high speed and low power. It may be that some of the emerging solid-state storage mechanisms, such as a rearchitected flash, may very well become an important player in designing such systems.

6.5 Interconnect Technologies

The term **interconnect** revolves around the implementation of a path through which either energy or information may flow from one part of a circuit to another. Metrics involve both those that represent "performance" and are to be maximized, as in:

- **current flow**: as in when implementing power and ground delivery systems.
- **signalling rate**: as in the maximum rate that changes at the input of an interconnect can be made, and still be detected at the other side.
- **peak** and **sustainable data bandwidth**: as when digital information is to be transferred reliably via signalling.

and metrics that are to be minimized, as in

- **energy loss**: either during the transport process (resistive loss in the interconnect), or in the conversion of a bit stream at the input into a signal over the interconnect and the conversion back to a bit stream at the other end.
- **time loss**: where there is a latency involved between the launch of the information (or energy) at one end and its reception and conversion back into a useable for at the other end.

The challenges for Exascale systems lie almost wholly on those metrics associated with information transfer, namely latency, data bandwidth, and energy loss, and our emphasis here will be on the metrics of data bandwidth (measured in gigabits per second) and on energy loss (measured in pico Joules per data bit transferred).

Further, these challenges exist at multiple levels:

- on chip: over a distance of a few mm,
- chip-to-chip: where the chips are in very close coupling as in a stack,
- chip-to-chip: where the chips are further separated on some sort of substrate such as a PC board.
- board-to-board within a single rack,
- and rack-to-rack.

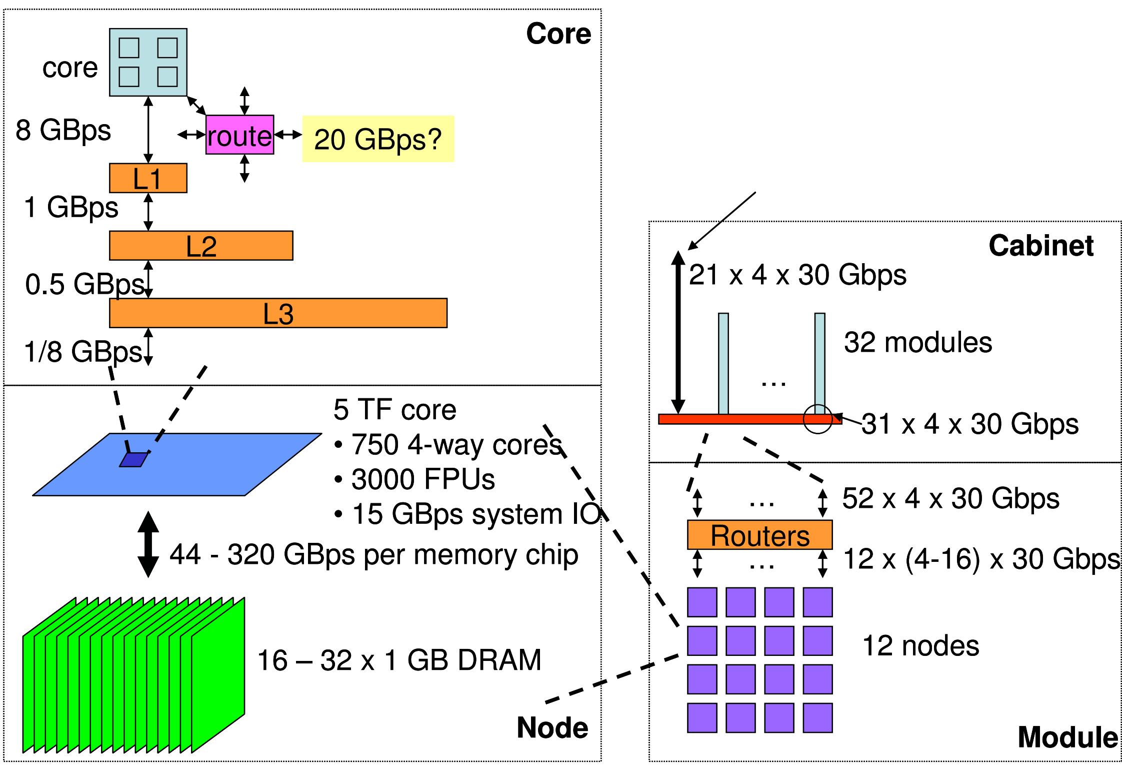


Figure 6.32: Interconnect bandwidth requirements for an Exascale system.

6.5.1 Strawman Interconnect

The first three types of interconnect are clearly relevant to all three classes of Exascale systems, "board-to-board" is relevant primarily to departmental and data center classes, and "rack-to-rack" primarily to data center scales. For discussion, Figure 6.32 summarizes possible interconnect bandwidth requirements for an Exascale data center system, as taken from the aggressive strawman presented in Chapter 7.3. The following paragraphs then discuss what was assumed for the aggressive strawman design in each of the above categories of interconnect, and set the stage for the discussion of alternatives in the following sections:

6.5.1.1 Local Core-level On-chip Interconnect

These include intra-core and local memory access. These lines are at most a few mm long and are point to point buses. At full swing with today's technologies, these consume 110 fJ/bit/mm, and at reduced swing, 18 fJ/bit/mm. Reduced swing circuits have the disadvantage of requiring a amplifier receiver so the signal can be returned to full swing. A lower power receiver solution is to use a clocked receiver, for example a sense amp style circuit, but this adds latency depending on the clock phase used. Using low-swing signaling, L1-L3 memory bandwidth only consumes 3 W and has negligible area impact.

6.5.1.2 Switched Long-range On-chip Interconnect

The strawman does not discuss on-chip routing. However, it is plausible that an Exascale computing chip would benefit from a low-latency on-chip router. For example, the Intel Teraflop research

vehicle emphasizes on-chip routing [150]. Alternatives including using shared memory, and/or supplementing routed interconnect with switched interconnect. The implications of switchable routing will be discussed below.

6.5.1.3 Supporting DRAM and CPU Bandwidth

Large amounts of DRAM bandwidth are required. The suggested sustained bandwidth of 320 GBps per GB chip is about two orders of magnitude more than DRAMs provide today. At 30 Gbps per wire pair, 170 data pins per memory would be required. At 10 Gbps, 512 pins would be required. While the lower line rate would reduce the complexity of the SerDes, it would, in contrast, increase the packaging requirements. Assuming 16 GB of total memory and 30 Gbps per pair, the CPU would require 2,720 memory data IO pins. Assuming a 32-bit address space, 64 differentially connected address pins would be needed per memory. Add in 16 pins for control, gives a total of 250 pins per memory chip, or 4,000 pins for the CPU. Add in 4-16 pairs (12-40 pins including control) for the assumed data IO, and 2,000 power and ground pins for power delivery and integrity, gives a total of 6,000 pins on the surface.

6.5.1.4 Intramodule Bandwidth

In the strawman, each module is assumed to contain 12 nodes and 12 router chips (drawn here as one router). At 12-40 pins per node, the router has to handle up to 480 I/O connections to the nodes. It also has to handle ∼500 I/Os to the board.

6.5.1.5 Intermodule Bandwidth

The equivalent of the system backplane or midplane has to be able to manage 32 blades, each with ∼500 30 Gbps I/O in a point-to-point configuration.

6.5.1.6 Rack to Rack Bandwidth

In the strawman of Section 7.3, each rack is connected to every other rack by an unswitched 4x30 Gbps bundle (e.g. 8 wires carrying differential traffic or perhaps optical fiber). The strawman has a router at each module level so that there are no more than two router chips between any two CPU chips (neglecting any on-chip routing). Packet routing is assumed, with packet sizes starting at 64 bits.

In general it is accepted that fully routable packet-style routing is needed for a large scale computer that will be suitable for a wide range of applications. However, several large scale applications do display regular inter-process interconnect patterns. Thus, it is possible, but not verified, that additional performance (i.e. more efficient use of bandwidth) might be gained from the addition of a circuit switch function. For example, some applications map well onto a grid architecture, while others map well onto an architecture that reflects how operations are mainly formed on the main diagonal(s) of a matrix. However, given the limited total power budget, adding a circuit switch function could only be done by reducing CPU or memory, which would be an overall win only if the remaining resources are more efficiently used with circuit-switched communication. A performance analysis would have to be done across a broad range of applications before this could be deemed worthwhile.

6.5.2 Signaling on Wire

Classically, virtually all signalling within computing systems has been done by current transfer through wires. Within this category, there are two major circuit variants: **point-to-point** and switched. Each is discussed in a separate section below.

Note that **busses**, where there are multiple sources and sinks electrically tied to the same wire at the same time, are not addressed here - the aggregate capacitance of such interconnect makes them rather power inefficient, at least for longer range signalling.

6.5.2.1 Point-to-Point Links

Point-to-point interconnect occurs when there is a well-defined source transmitter for the data and a well-defined receiver, and the two circuits do not switch roles. Within this category, there are three variants relevant to Exascale: on-chip, off-chip, and switched. Each is discussed below.

6.5.2.1.1 On-Chip Wired Interconnect At the 32 nm node, we estimate a line capacitance of 300 fF/mm. With a 0.6 V power supply, **full swing signaling** gives a signaling energy of 110 fJ/bit-mm. Many schemes have been proposed to reduce interconnect power through voltage scaling. These schemes require an amplifier at the receiver to amplify the swing back, with some additional power needed. Alternatively a clocked sense can be used for a lower power receiver, with the implication that the latency includes a crossing of a clock phase boundary.

When using **lower swing interconnect**, two benefits arise. The first is reduced power consumption. A swing of 0.1 V reduces the power down to 18 fJ/bit-mm. A number of schemes have been demonstrated, and it is not necessary to distribute a 0.1 V supply. The second benefit is increased range without repeaters. Simple equalization schemes can be used instead. Repeater-less ranges of excess of 10 mm have been demonstrated in 0.18μm technology [162], and it is reasonable that even an interconnect to L3 cache might be possible with minimal or no repeater requirements.

It is important to remember that longer range interconnect are routed in the relatively coarse wiring near the top of the chip, and that adding layers of this scale of wiring is not very expensive. Thus given, the relatively modest amounts of on-chip interconnect anticipated in Figure 6.32 we do not see providing this wiring, at reasonable equalized RC delays, to present any problems within the scope of anticipated technologies.

6.5.2.1.2 Off-chip Wired Interconnect We will discuss two types of off-chip interconnect - extensions of current wired schemes and chip-to-chip schemes assuming some style of 3D interconnect.

Point-to-point electrical links are limited by the frequency-dependent attenuation of traveling waves by skin-effect resistance and dielectric absorption. As these properties of wires do not change as semiconductor technology scales, we do not expect substantial changes in the performance of off-chip electrical links over time.

Today, relatively little attention is given to the power consumed in chip-to-chip communications and thus there is tremendous potential for scaling. For example, a commercial standard 3 Gbps SerDes link consumes approximately 100 mW, and can communicate up to 100 cm in distance. This works out to 30 pJ per bit. There are multiple reasons for this high power consumption. First, the link itself is operating at around 400 mV swings using "always-on" current-mode drivers and receivers. Second, the overhead is significant - for clock and data recovery using a **Phased Locked Loop** (**PLL**) or **Delay Locked Loop** (**DLL**) and for the muxes, demuxes, and flip-flops require to interface the slow on-chip data rates to this boosted rate. Overall, today's commercially

Technology		Pitch	Power
	Through Silicon Vias	< 5 µm	1 -11 fJ/bit
	Capacitive face-to-face	36 µm	2 pJ/bit
	Inductive face-up	30 µm	0.14 pJ/bit

Figure 6.33: Comparison of 3D chip stacking communications schemes.

deployed links have been designed with little attention to power consumption; the power can be reduced by an order of magnitude or more.

The key to power reduced is to work out how power can be saved while giving a reliable link. A key constraint is the signal to noise ratio (**SNR**) at the receiver. Achieving a **Bit Error Rate** (**BER**) of 10^{-18} requires a signal to noise ratio of better than 9. (With today's error correction schemes it is generally accepted that power is better spent on increasing signal swing, to reduce error rate, rather than on error correction.) Thus if the noise at the receiver is well controlled, then the driver signal swing can be reduced, even down to 40 mV. Combined with voltage mode circuits and careful management of overhead, a power level as low as 14 mW at 6 Gbps can be demonstrated, or 2 pJ per bit [117]. Note that the bulk of this energy is used, not to transmit or receive the bit, but rather on clock generation and recovery — to generate a transmit clock and recover a receive clock. By 2010 we expect this signaling energy to be reduced to 0.5pJ per bit as more efficient clocking circuits are developed and the entire link is operated from lower supply voltages.

Signaling rates will continue to increase, but as these rates increase, maximum signaling distances will drop. Links operating at 6-10Gb/s are commonplace today. By 2015 we expect that links operating at 30-40Gb/s will be available. These links will operate at reasonable bit error rates up to about 30dB of attenuation. At 30Gb/s this corresponds to about 10m of 24AWG cable or about 1m of PCB stripguide.

6.5.2.1.3 Direct Chip-Chip Interconnect This study anticipates that some form of 3D assembly will be beneficial in an Exascale system. This leads to the possibility of large-scale deployment of direct chip to chip interconnect schemes. A summary of direct chip-chip interconnect schemes, and their state of the art, are summarized in Figure 6.33. **Through-silicon-vias** (**TSV**) can be used to vertically stack and interconnect wafers and chips. At the time of writing, the state of the art is a 3 wafer stack with less than a $5\mu m$ via pitch. The extra power of communicating

through a vertical via is about that of communicating through the same length of wire. Assuming a $100\mu m$ via, that works out to 2 - 11 fJ, depending on the signal swing. However, in bulk CMOS the TSV must be passivated, giving an effective dielectric thickness to ground much less than that in the SOI case. Today, the state of the art is a 1 μm passivation, giving a capacitance for a 100 μm via of around 44 fJ (equivalent to about 150 μm of wiring). By 2015 this parasitic should be two to three times better.

By 2015, 3D chip stacking with TSV's will be far enough advanced that the limitations on chip stacks, and via size will be controlled by thermal, power delivery and cost considerations, not basic technology. Sub-micron TSVs have been demonstrated in the laboratory. However, the requirements to get current in and power out will limit practical chip stacks to 4-6 die in any application requiring high performance, high current, logic. 3D technology will be discussed more in Section 6.6.

However, through-silicon via assembly requires extensive integration in one fab facility. A simpler way to vertically integrate chips is to fabricate them separately and communicate through matched capacitors or inductors, forming a series capacitor or transformer respectively. The size must be large enough to get sufficient signal swing at the receiver - roughly $30\mu m$ is the minimum feasible pitch. The power is higher as the signal swing at the receiver is reduced (100 mV) and a bias network is needed to compensate for the lack of DC information. Power can be reduced by using coding so that no receiver bias network is needed and by using clocked sense-amp style receivers. The power numbers given in Figure 6.33 are all from published sources [107] [69]. There are some limitations to these technologies not often discussed. In particular, the parasitics must be well controlled, and dense grids can not be placed beneath these structures in a metal stack.

6.5.2.2 Switches and Routers

The pin bandwidth of routers for interconnection networks has been increasing at Moore's law rates [81]. We expect this exponential scaling to continue until routers become power limited. The YARC router used in the Cray BlackWidow, for example, has a bidirectional pin bandwidth of 1.2Tb/s. If this router used efficient signaling circuitry with a signaling energy of 2pJ/bit, the power used by the I/O bandwidth of the chip would be just 2.4W. Clearly we are a long way from being power limited in our router designs. By 2015 we expect pin bandwidth to by scale another order of magnitude. In our straw man we assume a modest 7.7Tb/s.

As the pin bandwidth of switches and routers increases, it is more efficient to use this increase by making the radix or degree of the routers higher than simply to make the channels higher bandwidth. In the 2015 time frame, routers with a radix of 128 to 256 channels each with a bandwidth of 60-120Gb/s will be feasible. Such high-radix routers lead to networks with very low diameter and hence low latency and cost[81]. High-radix networks can use a Clos topology or the more efficient flattened butterfly or dragonfly topologies. The latter two topologies require globally adaptive load balancing to achieve good performance on challenging traffic patterns.

While circuit switching consumes less power than packet switching, it provides less utility as discussed above. However, it is commonly used on the local scale. For example, the Sun Niagara includes a 200 GB/s crossbar in its eight-core 90 nm chip, at a power cost of 3.7 W (6% of a total power of 63 W), or 2.35 pJ/bit. Here the crossbar was used to enable sharing of L2 cache. The energy/bit should scale below 1 pJ/bit by 2015. The energy/bit consumed in a crossbar is dominated by interconnect energy.

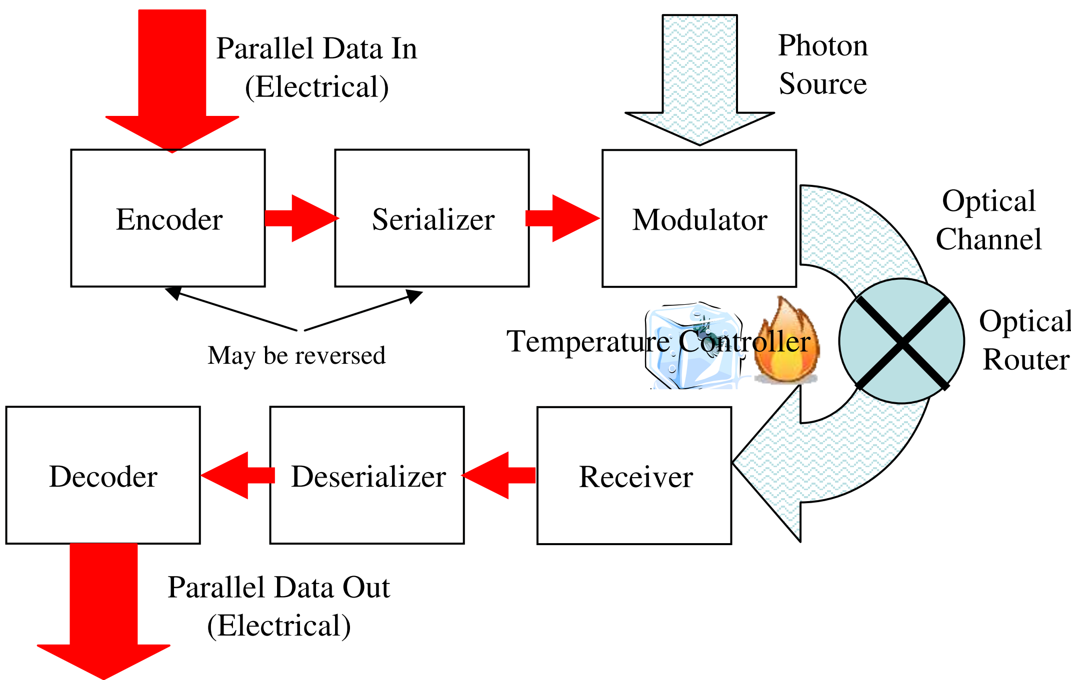


Figure 6.34: Entire optical communication path.

6.5.3 Optical Interconnects

Optical interconnect uses photons instead of electrons for signalling. As such, all communication is point to point, although in recent years optical routing that does not require conversion back to electrical has been introduced experimentally. Both are discussed below.

Also, in these discussions it is important to make fair comparisons with wire-based signalling, especially when considering power. In particular, this means accounting for all possible sources of energy loss in a system, including at least the following (see Figure 6.34):

- The serializers typically needed to go from bit parallel data at the electrical end to very high rate serial electrical bit streams.
- The encoders needed to add whatever error correcting information is needed to permit reliable communication.
- The convertors needed to convert from the high rate serial electrical streams to photonic streams.
  - If photons are generated directly from the electrical signal, as with a laser diode, the conversion inefficiency of the process must be included.
  - When modulators are assumed for the conversion process, the power needed for the original photon source needs to be included, and amortized over all channels it sources.
- If some form of optical routing is needed, the power that needs to expended to general the routing signal to the router.
- The detection at the receiving end of the photons and conversion back into an electrical form. This usually includes not only the conversion back into electrical form, but also clock recovery.

- The deserialization of the received high speed electrical output from the receiver into a data parallel form (this usually also requires clock recovery).
- Decoders to process the bit stream and correct for errors.
- In addition, many of the emerging electro-optical and optical-optical devices are very temperature sensitive, and heaters or coolers, along with temperature sensors and a feedback controller, may be needed to maintain a constant temperature that permits a stable wavelength.

6.5.3.1 Optical Point to Point Communications

Today, optical links are regularly used for longer range (> 10 m) rack to rack communications. The existence of low loss fiber, means that less energy and volume is used to communicate over these distances using optics, rather than electronics. The open question is to what extent optical communications can displace electrical communications, including at the rack, board, and chip levels? It is commonly agreed that optics offers certain fundamental advantages for interconnect, including low loss, scaling without adding energy, and potential for high wiring density. However, practical issues have always prevented its employment in these shorter range applications. Prime amongst these is power consumption. Electro-optical and optical-electrical conversion has traditionally consumed more power than that gained by the low link loss. Another limiter is the lack of a packet routing technology. While all optical circuit switching is possible, no technology has yet shown real promise of enabling packet routing. Other issues include the lack of highly integrated sub-systems, the relatively low reliability of the III-V devices required (when compared with CMOS), and (often) the need for tight temperature control.

Though there is an active community exploring these issues, and possible directions have been identified, significant R&D investment is required to make short range optical interconnect more useful. The potential for power reduction is being explored, as is the technology for integration. These are being explored in anticipation of bringing optics into mainstream computing. Unfortunately, the current markets for optical components are relatively small, so there is a bit of a "chicken and egg" problem in justifying large-scale commercial investment.

Today, a typical short-range optical link consists of a **vertical-cavity surface-emitting laser** (**VCSEL**) connected by a multi-mode fiber to an optical receiver. The advantages of this approach include the low-cost VCSEL, and the ease of coupling to a (low-cost) multi-mode plastic fiber. IBM has demonstrated a similar architecture for board-level optical interconnect, replacing the multi-mode fiber with a multi-mode embedded optical waveguide. They have demonstrated this capability at 5 pJ/bit NOT including serial/deserializing multiplexing (SerDes) and **Clock and Data Recovery** (**CDR**), at a data rate of 10 Gbps. $250\mu m \times 350\mu m$ pitch. Several other groups have demonstrated similar capabilities, at least in part.

However, with the recent emergence of integration into an SOI substrate, a potentially better link architecture is to follow the approach shown in Fig. 6.35. Instead of using a directly modulated laser, a DC laser is used, and a digital modulator employed to provide a signal. A DC laser is more reliable than a directly modulated laser, and less temperature control is needed. Modulation is then done by modulating a fraction of this laser energy. Interference-based Mach-Zender or ring modulators can be used. The receiver can be highly integrated by using a silicon process modified with a Germanium step. As well as improved reliability, this approach offers great potential for improved integration and power consumption. Everything beside the InP laser can be built in silicon. SOI waveguides can be built using a $0.5\mu m\pm$ wide trace on top of glass. However, to function correctly, the modulators must be connected using single mode waveguides. The modulators rely on subtly

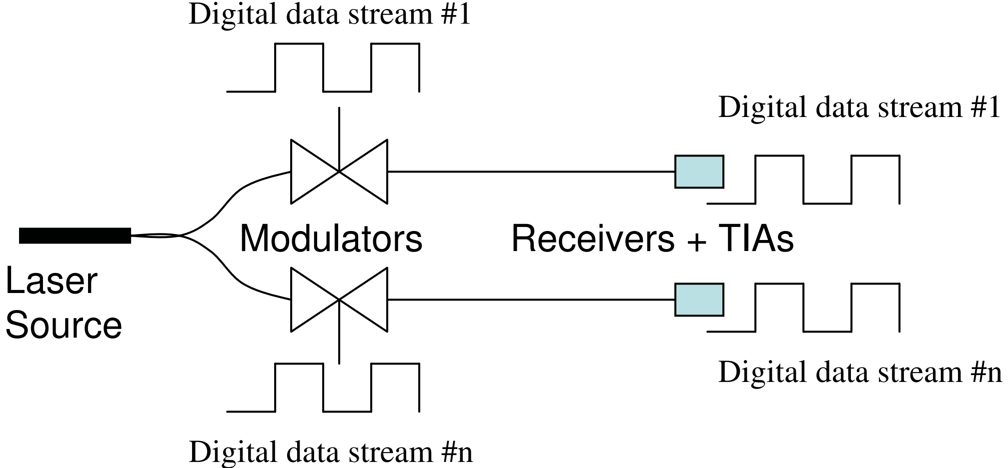


Figure 6.35: Modulator approach to integrated optics.

changing the delay of one light path so as to enable constructive and destructive interference. Thus they only work at one wavelength. The disadvantage of requiring single mode waveguides is that the any external connection, via a single mode fiber, requires sub-micron alignment for acceptable coupling losses (in contrast a multi-mode connection can withstand a multi-micron misalignment). Though pick-and-place machines can be modified to align single fibers, no technology is currently available to align multiple optical waveguides in parallel. Any modified printed circuit board style process would not have the required large-scale registration. Large PCB panels that are available today typically provide $10\mu m$ registration at best. Sub-micron registration is required to simultaneous align multiple single mode fibers. Step-and repeat lithographic processes provide the required alignment but not over a large area. Nanopositioners could be modified to provide the required precision but this has been little explored.

For example, Luxterra has demonstrated this approach with a 40 Gbps link, connected via single fibers, with a total energetics of 55 pJ/bit, not including SerDes and CDR. However, this speed is not going to be energy efficient, and, at slower rates, it is generally agreed that the energy could be significantly better.

A potential power budget for a near future implementation, based in part on [78] is shown in Table 6.7. The near term column could be considered part of a technology roadmap. Achieving the numbers given in the long term column would require significant investment, and should not be considered as part of the roadmap. Consider the "near term" column first. At 2 Gbps, the receive power needs to be better than 0.2 mW to achieve a BER of better than 10^{-15}. Assuming minimal modulator and connector losses, a source of 0.3 mW is (aggressively) possible. However, this is the DC power consumption. A typical interconnect only sends useful bits for around 10% of the time. This activity factor has to be included in any calculation for energy per bit, based on DC powers, and is accounted for in the Table. A modulator would have an input capacitance of around 100 fF, giving 0.1 pJ/bit at 1 V supply.

The commercial state of the art for optical receivers is 10 mW at 10 Gbps, or 1-10 pJ/bit depending on the assumed activity factor [58]. However, there is potential for scaling. A low capacitance MODFET (modulated-doping field effect transistor) optical receiver has potential for operating at 1 mW total power consumption at 2 Gbps, at acceptable BERs, giving the receive energies listed in Table 6.7. Again, these amplifiers consume DC power. The total link power of a

Component	Near Term	Long Term
Laser @ 2 Gbps 10% activity factor (AF)	0.3 mW per channel 0.15 pJ/bit 1.5 pJ/bit	 same? same
Modulator	0.1 pJ/bit	0.01 pJ/bit
RX + TIA 10% activity factor	0.5 pJ/bit 5 pJ/bit	0.05 pJ/bit? 0.05 pJ/bit?
Sub-total 100% AF 10% AF	0.75 pF/bit 7.5 pJ/bit	0.21 pJ/bit 1.5 pJ/bit
Temperature Control	?	?

Table 6.7: Energy budget for optical modulator.

short-term scaled link is thus 0.75 pJ/bit for 100% activity factor and 7.5 pJ/bit for a 10% activity factor. While the power achieved is not low enough to replace chip to chip links at the board and backplane level, this level of power consumption has potential to favor optical interconnect at distances over 50 - 100 cm.

The right hand column in Table 6.7 assumes aggressive long term scaling of the key technologies. These technologies should not really be considered part of the "roadmap" as considerable R&D investment would be required to achieve the listed potential. Quantum Well Modulators have potential to scale to 10 fF input capacitance, or smaller, permitting a ten-fold reduction in modulator power consumption [30]. New classes of optical MOSFETs have the potential to allow optical receivers to operate in voltage, rather than current, mode and reduce the power down to levels comparable with a large CMOS gate [114]. These have potential to operate at power levels of 0.05 pJ/bit and even lower.

One power consumption that is not included in Table 6.7 is that required for cooling. Optical modulators require tight temperature control for correct operation, typically to $\pm 10°C$. Typically, active temperature control is used, often **Thermal Electric Coolers** (**TEC**). The additional power that would be consumed operating an active cooler will depend on the total heat load at the spot being cooled. Thus it would be a required overhead but difficult to estimate at this stage.

6.5.3.2 Optical Routed Communications

By adding additional modulators, the approach outlined in Figure 6.35 can be extended to build an all-optical circuit switched architecture. Since each additional modulator consumes only 0.1 pJ/bit, the additional power over that shown in Table 6.7, would be minimal. Of course the (probably) electrical network that must route the switching commands to the modulators must also be taken into account. Thus, if modulator based optical interconnects were employed in an Exascale computer, interesting circuit-switched functions might be added for minimal extra power and cost. For example, Bergman proposes a low-latency crossbar architecture for future multi-core computers[127].

However, optics still lacks the capability to enable the switching function most useful to the interconnect network of an Exascale computer - a packet router. Packet routing requires that routing header information accompany the data, so that the path taken by the packet can be set up and taken down as the packet passes through the switch. This reflects the usually unpredictable nature of intra-computer communications. To date, optical switches require a separate electrical network for circuit set-up and re-route. This introduces an overhead of 10s' of ns every time the circuit is reconfigured. Thus they are best suited for computational tasks that benefit from node-

to-node bandwidth that changes substantially with each task, or for tasks that benefit from very large bursts of data being routed between CPUs. At this stage, there is little evidence that the potential throughout benefit of such switching would justify reducing another resource in order to account for the required power, except for those links where optics is already justified over electrical interconnect, i.e. longer inter-node links.

6.5.4 Other Interconnect

Other possible interconnect technologies that could impact HPC in the future include the following:

- **Carbon Nanotubes** (CNT). Due to their excellent conductivity, carbon nanotubes have potential to permit an increase in wire density while also improving latency and power consumption [59]. They can be routed tightly at low resistance, with a pitch better than $0.8\mu m$. Cho et.al. [59] predict a power consumption of 50 fJ/bit/mm at full swing. Thus a power consumption of 6 fJ/bit/mm at reduced swing would be reasonable. However, given the currently limited state of demonstration and the open technical issues involving patterning, contacts, etc., it is unlikely that CNTs will be mature enough to be part of a 2105 Exascale computer chip.

- **Nano-enabled Programmable Crosspoints**. The emerging resistive memories discussed earlier in Section 6.3.5 also have potential to serve as the base technology for high-density switch-boxes. These could reduce the area and power impact of SRAM-based programmable switch boxes by almost an order of magnitude, enabling new ideas in configurable computing to be investigated. So far, this concept has been mainly explored in specific applications, so its potential impact on HPC is largely unknown. At the least, it would outperform the crossbar switches discussed above, in at least power per operation. Given that a roadmap exists for such memories to be commercially deployed next decade, their incorporation into logic CMOS devices is plausible and should be considered.

6.5.5 Implications

The summary roadmap for interconnect is presented in Table 6.8. The three metrics evaluated are wire density, power/bit and technology readiness. The energy/bit numbers come from the discussion above. Wire density is for a single layer of routing only. On-chip long-distance wires are assumed to have $2\mu m$ width and space. Chip-to-chip routing is assumed to have 1 mil ($25\mu m$) width and 4 mil space (to control crosstalk). Note these numbers are per-wire. Differential routing halves these numbers. PCB-embedded multi-mode wires have been demonstrated at $100\mu m$ width and $1000\mu m$ pitch. Single-mode waveguides can be much narrower but (as yet) can not be fabricated on a large panels.

Overall, it is fairly clear that copper interconnect is here to stay, at least for the time frame of an Exascale computer. Today the cross-over point between optics and electronics, in terms of energy/bit and $/Gbps is at long distances - several meters. However, optical interconnect is highly unoptimized. Unfortunately, this is largely due to the lack of a volume market for optimized optical interconnect. With new, better optimized devices and circuits, together with greater integration, the cross-over point is likely to shrink to something better than 100 cm, perhaps 50, perhaps shorter.

One significant advantage of optics is that, if desired, circuit switching can be added to point-to-point optical interconnect with relatively little overhead. However, these new technologies assume single-mode fiber, requiring sub-micron alignment. While the technology for single fiber alignment

Technology	Density (wires/mm)	Power (pJ/bit)	Technology Readiness
Long-range on-chip copper	250	18 fJ/bit-mm	Demonstrated
Chip-to-chip copper	8	2 pJ/bit. Includes CDR	Demonstrated. Potential for scaling to 1 pJ/bit
Routed interconnect in 2015	n/a	2 pJ/bit router or non-blocking circuit switch 1 pJ/bit	roughly the same for packet
Optical State of Art (multi-mode)	10	9 pJ/bit. NOT including CDR	Demonstrated.
Optical (Single mode) in 2010	300	7.5 pJ/bit SOI waveguides PCB-embedded waveguide does not exist	Assumes lithographed
Optical (Single mode) in 2015	300	1.5 pJ/bit	At early research stage
Optical Routing		Add 0.1 pJ/bit (2010) for each switch	
Optical - temperature control			TEC cooler demonstrated
CNT bundles	1250	6 fJ/bit-mm	Undemonstrated

Table 6.8: Summary interconnect technology roadmap.

Approach	Wires/mm or sq.cm e.g. 2 wires/mm 1 mm	Bandwidth/mm Routable signal pairs per mm per layer * # layers * bit-rate per signal pair	Comments
Laminate (Ball Grid Array)	20 wires/mm/layer (2-4 signal layers) ~2,000 max total pin count	6 pairs/mm @ 30 Gbps = 180 – 720 Gbps/mm (1-4 signal layers) Package I/O: 500 pairs = 15 Tbps	1 mil line/trace presents practical limit. 1 mm BGA ball pitch
Silicon Carrier	50 wires/mm/layer (2 signal layers) Has to be packaged for I/O	12 pair/mm @ 30 Gbps = 360 – 720 Gbps/mm (1-2 signal layers)	2 signal layers is practical limit.
3DIC Stack	~10-40 wires/mm vertically around edge	Total: 100 – 200 pair @ 10 Gbps ➔ 0.5 - 2 Tbps (assumes memory)	Limited interconnect performance
3D IC with Through Silicon Vias	In excess of 10,000 vias per sq.mm.	In excess of 100,000 Tbps/sq.cm. Really determined by floorplan issues	Chip stack limited to 4-8 chips, depending on thermal and other issues
Stacked Packages	1/mm on periphery	25 pairs total @ 10 Gbps ➔ 250 Gbps	Not very applicable to high performance systems
Stacked Silicon Carriers	Vertical connections @ 20 um pitch ➔ 250,000 / sq.cm	62500 pairs @ 30 Gbps ➔ 1900 Tbps / sq.cm	Limited by thermal and coplanarity issues.
Stacked Silicon Carriers	Vertical connections @ 100 um pitch ➔ 10,000 / sq.cm	2500 pairs @ 30 Gbps ➔ 75 Tbps / sq.cm	Early demonstration only. Air cooled to < 117 W total.

Figure 6.36: Representative current and future high-end level 1 packaging.

is mature, there is no equivalent of the Printed Circuit Board in large scale single mode interconnect. A significant investment would be needed for this technology to be available.

6.6 Packaging and Cooling

6.6.1 Packaging

For convenience, packaging is generally considered within the scope of a hierarchy, as follows:

- **Level 1 Packaging** connects, contains, and cools one or more silicon die and passives, such as capacitors and resistors. A typical level 1 package would be an organic (plastic) surface mount.
- **Level 2 Packaging** connects a set of level 1 packaged components on a common substrate, such as a printed circuit board (PCB).
- **Level 3 Packaging** interconnects a set of level 2 packages. A typical structure would be a backplane or midplane, into which a number of PCBs are plugged using separable connectors. Commonly called a "rack," a level 3 package might span a meter or more in distance and connect several tens of PCBs.

In an Exascale computer, Level 3 packaging and rack to rack interconnectivity is not trivial, but is not driven as much by technology as the other two, and is thus not discussed further. The one key issue of connectivity is typically provided today by cables, electrical for shorter ranges and optical for longer ranges.

6.6.1.1 Level 1 Packaging

An extrapolation of future capabilities based on current level 1 packaging technologies is given in Figure 6.36. The most ubiquitous package technology today is based on organic (i.e. plastic) **Ball Grid Arrays** (**BGA**) onto which chips are flip-bumped. The leading edge state of the art is typified by the Endicott Interconnect Technology HyperBGA product [71]. This BGA supports 1 mil features and uses laser drilled vias to achieve high densities. Nonetheless, it is limited to a maximum pin-out of around 2,000 pins, on a 0.5 - 1 mm grid. It represents the current limit for manufacturing for laminate technologies. Exceeding these limits would require a level of precision beyond that of today's equipment, i.e. would require a sizeable R&D investment. Assuming half the pins are used for power distribution, a 2,000 pin package can only support 500 signal pairs. Assuming a future 30 Gbps capability on these pairs, that amounts to 15 Tbps, or 2 TBps total in and out of the package. A previous section suggested a CPU I/O requirement of 0.7 - 10 TBps. A HyperBGA package could support the low-end, but not the high-end of this range. Even at the low-end, breakout to the next level of packaging with good noise control (see below) might be difficult.

One way to increase the wire density beyond current laminate technologies would be to use lithography instead of screen-printing. Lithography can be used on a silicon carrier to create very high wire densities, down to $10\mu m$ line and space, though coarser features are often used so as to improve yield. However, most current silicon carrier technologies are limited to 4 layers - power, ground and 2 signal. Thus, the total wire density per unit of cross-section, for the entire wire stack, ends up being about the same as that for a high-end laminate, due to the latter's higher signal layer count! This is due to planarity limitations introduced by the requirements for thick dielectrics so that transmission line structures can be built. New concepts would be needed to make high layer count silicon carriers. Note that the silicon carrier itself must be packaged, for example, on top of a laminate.

The next row in Figure 6.36 shows commercial state-of-the-art 3D chip stacks, built using wire-bonds or some form of stacking and edge processing [1]. These are often used for memories, so as to improve their form factor and physical density. However, their I/O is relatively limited and won't support a stack of high bandwidth DRAMs as anticipated in this study. (Note, 10 Gbps I/O rate is used here, rather than the more aggressive 30 Gbps to reflect the reduced capability of logic built in a DRAM process.)

Through-Silicon Vias (TSV) can enable a 3D chip stack with very high internal connectivity. This is a technology that is likely to be mature by 2015, partly because of current DARPA funded efforts. The available vertical bandwidth is very high, and is limited by practical considerations such as silicon area tradeoffs, etc. Several vendors are designing 3D integratable memories. However, it is difficult to envision a vertical chip stack consisting of more than four to eight chips. There are many practical reasons for this. First current has to be brought in and heat removed. Doing either through a large number of chips is highly impractical except for very low-power circuits. Improving power delivery or heat removal capability requires that more silicon vias be added to all layers, as more chips are stacked. Second, test and yield issues works against high layer counts. Each chip in the stack must either be pretested before integration, or methods to cope with the accumulated yield loss introduced. Unfortunately, even an eight-chip stack (which is unlikely to have acceptable

heat removal) does not integrate sufficient memories with a CPU to meet the anticipated memory needs.

Two other currently emerging technologies use silicon carriers to create 3D packages. One approach involves adding additional interconnect layers and chips to a single carrier. Yet to be demonstrated, there would be concerns about maintaining planarity and cooling. Since the chips are essentially encased in polymer, heat removal would be as difficult as for the 3D case.

Another approach, currently being demonstrated by Irvine Sensors, involve stacking separate silicon carriers using interconnect studs. Stress issues limit the width of the structure to two to three cm. However, one significant advantage of this approach over the previous discussed one is the ability to include heat spreaders in the 3D package. Irvine plans to demonstrate an air cooled package, capable of dissipating a total of 117 W. With additional innovation, greater thermal capacities would be possible.

6.6.1.2 Level 2 Packaging

Level 2 packaging almost invariably consists of a **Printed Circuit Board** (**PCB**). The state of the art supports high capacity in one of two ways. The first way is to use fine line technologies (1 mil traces) and laser drilled vias. However, the layer count is limited in this approach. The second is to go to high layer counts and use conventional line widths: e.g. use a 0.2" thick PCB, and support 4 mil ($100mum$) wide traces. Such a board might support 10 X-direction routed signal layers, 10 Y-direction routed, and 20 power/ground layers. Using normal routing rules for differential pairs would give a density of one pair every 28 mil (crosstalk forces the pairs apart). The maximum cross-section bandwidth would be 28 pairs per signal layer per cm, or 280 pairs per cm (10 layers), or 2800 Gbps (at 10 Gbps per pair for memory). This translates into a potential cross-section bandwidth of 350 GBps per cm at 10 Gbps or 1.05 TBps at 30 Gbps.

A combination of a high end **Single Chip Package** (SCP) with a high end, thick, PCB would enable such a packaged CPU to have a total bandwidth of 2 TBps with 30 Gbps signaling, and 666 GBps with 10 Gbps signaling. The limit is set by the package technology, not the board technology. Today, there are no investments going on to improve the capacity of conventional packaging. This would work if the required memory bandwidth was at the low end of the scale (16 memories $\times$ 44 GBps = 704 GBps) but not if larger memories or memory bandwidths were required.

Moving to a silicon carrier, single tier or stacked, does not immediately alleviate the situation. A 1 sq. cm die could provide a total peripheral off-chip bandwidth of 600 GBps at 10 Gbps and 1.8 TBps at 30 Gbps. However, with careful yield management (using finer lines for short distances) and planning, the I/O density could be increased locally just at the chip edge, to support possibly two to three times this bandwidth. An example is given in [103]. However, this solution would still not support the higher end of the possible bandwidth and capacity requirements.

Possible 3D solutions to providing sufficient memory bandwidth and memory capacity are discussed in Chapter 7.

6.6.2 Cooling

An Exascale computer presents a number of novel thermal design challenges, including the following:

- At the module level, any potential 3D chip assembly must be cooled sufficiently to ensure reliable operation, to limit leakage power, ensure timing budgets are predictably met, and to guarantee DRAM refresh times are accurate. Heat fluxes of up to 200 W/sq.cm. are anticipated.

Approach	Thermal Performance	Comments
Copper Heat Spreader	Thermal conductivity = 400 W/(m.K)	
Diamond	Thermal conductivity = 1000 - 2000 W/(m.K)	Expensive
Heat Pipe	Effective conductivity = 1400 W/(m.K)	Very effective
Thermal Grease	Thermal conductivity = 0.7 - 3 W/(m.K)	
Thermal vias with 10% fill factor	Effective Conductivity = 17 W/(m.K)	
Thermal Electric Coolers	Limited to less than 10 W/cm^2 and Consumes Power	
Carbon Nanotubes	Excellent	Early work only

Table 6.9: Internal heat removal approaches.

- At the rack level, the total thermal load is likely to be in the range of 10-200 KW. A solution is required that supports this level of cooling while maintaining required temperatures and permitting provisioning of high levels of system interconnect.
- At the system level, the total thermal load of an Exascale computer is anticipated to be 10s of MWs, and this heat load must be properly managed.

6.6.2.1 Module Level Cooling

The main objective of module level cooling is to control the chip junction temperature in order to minimize potential failure mechanisms such as electro-migration, and to minimize leakage currents in memories and logic transistors. Typical objectives are in the 85 C to 100 C range. Generally, chip level cooling is evaluated in terms of a thermal resistance:

$$R_\theta = \Delta T / Q \tag{6.10}$$

where ΔT is the temperature drop and Q the total heat being transferred. For convenience, thermal resistance is typically broken into two components, internal and external resistance. The internal resistance is determined by the conductivity of the materials and structures between the chip junction and the circulating coolant. The external resistance is determined by the ability of the circulating coolant to remove this heat away from the chip vicinity.

Some of the available structures that are used for the internal portion of the heat flow are summarized in Table 6.9. The last entry in this table is for an array of tungsten thermal vias that might be used in a 3D chip stack. Assuming that 10% of the chip area is given over to thermal vias, then the effective thermal conductivity is 17 W/(m.K). This calculation is only included to illustrate the relative difficult of cooling chips internal to a 3DIC. **Thermal Electric Cooling** (**TEC**) is included as a reference. TECs can remove heat without any temperature drop. However, the supportable heat flux is limited and they consume almost as much electrical power as they conduct heat power. Thus they are unlikely to be used in an Exascale computer except for points requiring critical temperature control, such as optoelectronic modulators.

Available external cooling mechanisms are summarized in Table 6.10. The most common cooling mechanism is forced air cooling using fans and heat sinks. There have been several demonstrations

Approach	Thermal Performance	Comments
Air (Finned Heat Sink)	$R = 0.6 - 1.0K/W$	Individual Heat Sink & Fan for 12 mm die Can dissipate up to 100 W for 60 K rise
Water (Channel Heat Sink)	$R = 0.3 - 0.6K/W$	Individual Heat Sink & Fan for 12 mm die Can dissipate up to 170 W for 60 K rise Requires 0.1 bar pump
Immersion	$R = 0.4K/W$	
Microchannel and Cooling capacity of $300 - 800W/cm^2$	pump 0.3 - 8 bar	
Two-phase	$R = 0.1K/W$	
Spray Cooling and Cooling capacity of $300 + W/cm^2$		
Refrigeration and $R = 0.05K/W$	Consumes more power than heat removed	

Table 6.10: External cooling mechanisms.

in which 100 W have been air cooled satisfactorily [161]. Air cooling capacity can be increased a little by using larger heatsinks and louder fans.

Water cooling using a microchannel cooling plate built as part of the package can handle up to 170 W based on existing technology [161]. The water is pumped through the cooler and then circulated to an external air cooler. With cooling channels of around 1 mm width, a pump pressure of around 0.1 bar is needed. Narrower channels improve thermal performance at the cost of bigger pumps. Another concern with water cooling are the prevention and management of leaks.

Direct immersion in a non-water coolant, such as a FC-72, can be used to simplify the overall cooling design. However, since these coolants do not offer the thermal performance of water, the overall cooling solution tends to provide the same as direct water cooling. For example, [82] recently a demonstrated a cooling performance equivalent to 0.4 K/W.

Other techniques have been long investigated but have never been implemented. Reducing the channel size and integrating the channels into the silicon chip has been demonstrated many times, and can improve thermal performance further. However, these introduce further mechanical complexity, increase the potential for leaks and require stronger and more power hungry pumps. Two-phase cooling (i.e. boiling) improves heat removal capacity (e.g. [74]) but at the expense of complexity and the need to carefully manage the boiling interface. In this work, using $100\mu m$ channels, 2W of pump power was needed to dissipate 3W from the chip, illustrating the power cost of the pumps required for microchannel cooling. Spray cooling can support heat fluxes in excess of $300W/cm^2$ with a better than 60 C temperature drop, but, again, at the expense of complexity. Through refrigeration can be very effective it is unlikely to be used here due to the added power consumption it implies. Cooling to liquid nitrogen temperatures takes twice as much work as the heat being removed. Even cooling from an ambient of (say) 40°C to 10°C, takes 11% more work than the heat being rejected.

Another issue that often rises in cooling design is the complexity introduced by the requirement to get the same power, electrically, that is being extracted thermally. With a DC limitation of

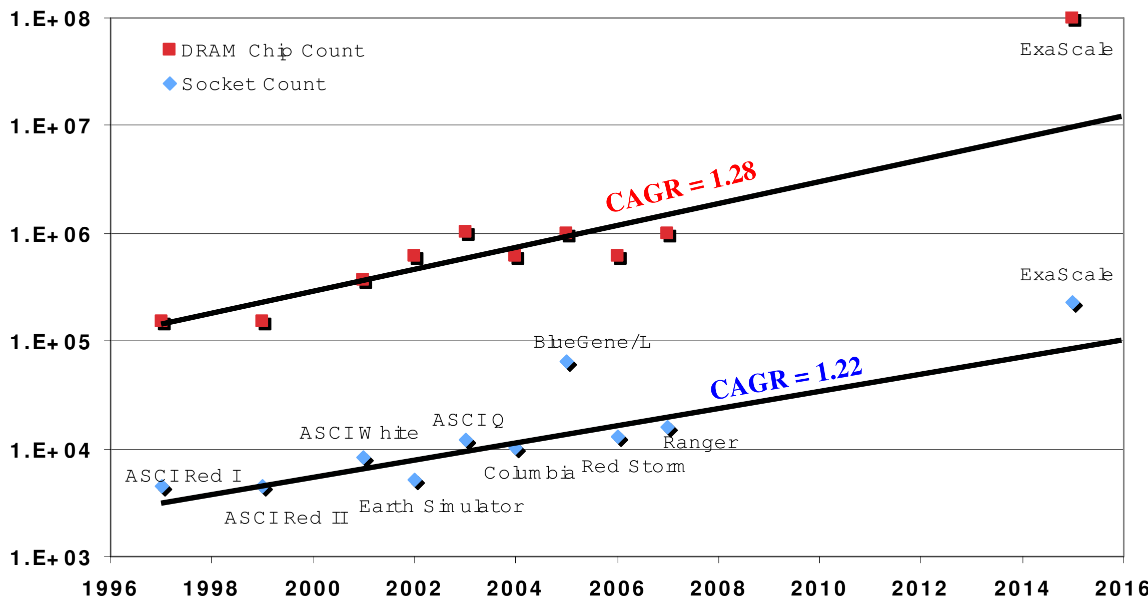


Figure 6.37: Estimated chip counts in recent HPC systems.

about 1 A per solder bump, 200 bumps are required just to distribute DC to CPU chip. Since bumps will have to be added to reduce power/ground inductance so as to control simultaneous switching noise, over 1,000 bumps might be needed unless alternative power delivery technologies arise. Unfortunately, the bumped side of the chip is not terribly effective at removing heat.

A typical solution is to use the front side of the chip to get DC power in, and the backside to get heat out. This approach is almost universally used. However, it complicates potential 3D designs. This will be discussed further in Chapter 7.

6.6.2.2 Cooling at Higher Levels

No matter whether air or water cooling is used at the module level, it is very likely that the machine as a whole will rely on air cooling above the module level, particularly as we go into either the departmental and especially the data center class Exascale systems. This will require careful design of the plenums etc., so as to efficiently manage air flow without excess noise.

For a data center system as a whole, at very best the target 20 MW of power has to be dissipated, even if simply vented externally. If the external ambient is higher than desired, then air conditioned cooling is required at least on the incoming air. To bound the problem, assume that 20 MW of cooling is required. That is equivalent to around 70 M BTU/hour. An air conditioner with a (future) SEER rating of 20 would consume 350 KW to provide this amount of cooling. Overall up to 5% of the power budget is consumed in system cooling.

6.7 System Resiliency

While the failure rate of any particular component may be relatively small, the resiliency of a computing system depends strongly on the number of components that it contains. This is particularly true of the data center class systems, and as such is the focus of this section.

	Hardware	Software	Network	Environment	Human	Unknown
% Breakdowns	62%	18%	2%	1%	1%	16%
% Downtime	60%	19%	1%	2%	0%	18%

Table 6.11: Root causes of failures in Terascale systems.

As shown in Figure 6.37 the number of components in recent supercomputing systems is increasing exponentially, the compound annual growth rate of memory (at 1.28X per year) exceeding slightly that of sockets (at 1.22X per year). Assuming continuation of past system growth, a 2015 system will consist of more than 100,000 processor chips ("sockets") and more than 10 million DRAM chips. While an Exascale system may require more aggressive system scaling, even past scaling trends will present substantial resiliency challenges.

6.7.1 Resiliency in Large Scale Systems

The supercomputing community has gained experience with resiliency through Terascale and emerging Petascale machines. Terascale systems were typically designed from standard commercial components without particular resiliency features beyond replication of components such as power supplies. Schroeder and Gibson [124] and Gibson[50] analyzed failure logs for these machines and reported that Terascale systems with thousands of sockets experienced failures every 8–12 hours, corresponding to 125–83 million FIT (failures in time, which is failures per 10^9 hours). The study also showed that the greatest indicator of failure rate was socket count; across all of the systems they examined, the average failure rate was 0.1–0.5 fails per year per socket (11–57 KFIT per socket). Schroeder and Gibson also analyzed the root causes of failures, with the averages summarized in Table 6.11. Their results show that hardware is the most dominant source of failures, including both intermittent and hard failures.

More recent systems, such as IBM's BlueGene Supercomputer, a 64K socket system with a Top500 performance of 280 TFlops, achieve better resiliency than reported by Schroeder [5]. The BlueGene chips and systems were designed with a resiliency (FIT) budget that is much more aggressive than the capabilities of Terascale systems. The FIT budget summarized in Table 6.12 shows that power supplies are most prone to failure and that while expected failures per DRAM chip is small, the sheer number of chips make DRAM the largest contributing factor to failures. Nonetheless, the overall FIT budget for the entire system is only 5 million (76 FIT per socket or 0.001 failures per year per socket), corresponding to a hardware failure rate of once every 7.9 days. Assuming that hardware accounts for only half of the failures, the aggregate mean time to interrupt (MTTI) is 3.9 days. The source of improved failure rates stems from more robust packaging and enhanced hardware error detection and correction.

Additional studies have measured failure rates due to specific causes in the system. Schroeder and Gibson examined disk drive reliability and report that disk drives are the most frequently replaced components in large scale systems [125]. However, disk drives have not traditionally dominated the cause of node outages because they are often replaced pro-actively when they begin to show early warning signs of failure.

In a different study, Michalak et al. examined the susceptibility of high performance systems to uncorrectable soft errors by monitoring a particular memory structure protected only by parity [104]. They report one uncorrectable single event upset every 6 hours (167 million FIT), reinforcing the need to design for **Single Event Upset** (**SEU**) tolerance.

Figure 6.38 shows the expected error rates as a function of socket count and three different per-socket failure rates ranging from 0.1 (representing the best observed failure rate from Schroeder)

Component	FIT per Component	Components per 64K System	FIT per System
DRAM	5	608,256	3,041K
Compute + I/O ASIC	20	66,560	1,331K
ETH Complex	160	3,024	484K
Non-redundant power supply	500	384	384K
Link ASIC	25	3,072	77K
Clock chip	6.5	1,200	8K
Total FITs			5,315K

Table 6.12: BlueGene FIT budget.

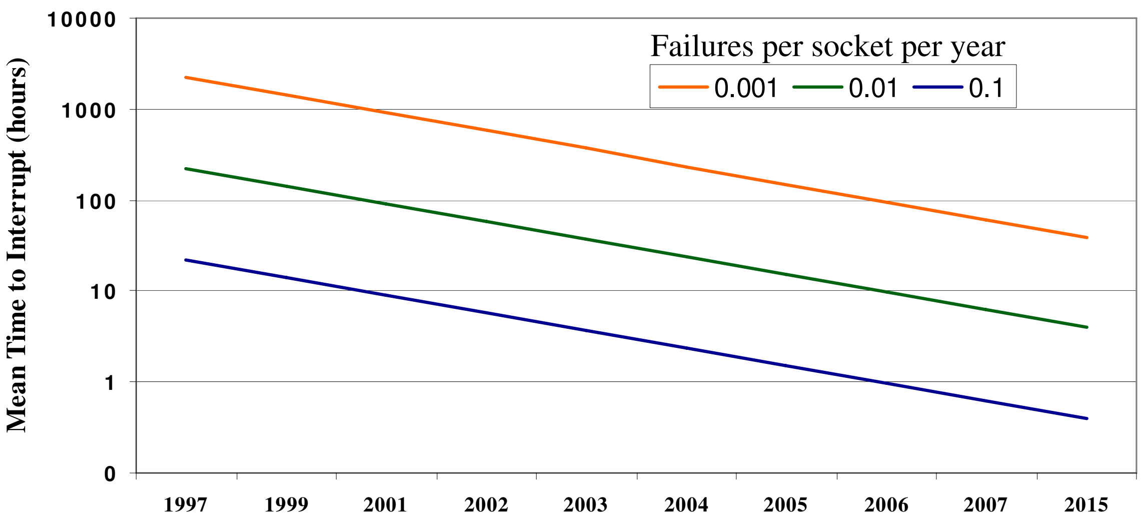


Figure 6.38: Scaling trends for environmental factors that affect resiliency.

to 0.001 (representing a system with aggressive resiliency). The number of sockets is assumed to increase at 25%/year to match a performance demand of 2x system performance per year and 2x socket performance every 18 months, reaching 220K sockets in 2015. Schroeder's per-socket resiliency assumptions results in a MTTI of 24 minutes, while a factor of 10 improvement in resiliency results in a failure every 4 hours.

6.7.2 Device Resiliency Scaling

While the above analysis assumed constant device failure rates, in fact technology scaling will make sustaining chip-level reliability substantially more difficult, with three major causes:

- **Hard Failures**: Shrinking feature size is a root cause of increased susceptibility to hard failures and device wearout. The ITRS identifies more than 20 difficult short-term semiconductor reliability challenges, including dielectric breakdown, thermal stresses, and electromigration[13]. Because device dimensions will continue to shrink while the power supply voltage will level out, electric fields will increase and contribute to several different breakdown modes. Temperature is also a substantial contributor to device failure rates, increasing the importance of thermal management in future systems. The ITRS sets as a goal 10-100 FITs per chip in the coming generations but recognizes that there are no known solutions for 32nm and beyond.
- **Single-Event Upsets**: Single event upsets (SEU) are influenced primarily by node capacitance in integrated circuits. DRAM cells are becoming less susceptible to SEUs as bit size is shrinking, providing a smaller profile to impinging charged particles, while node capacitance is nearly constant across technology generations. Logic devices, latches, and SRAMs are all becoming more susceptible due to the drop in node capacitance need to scale to higher circuit speeds and lower power [128]. Researchers predict an 8% increase in SEU rate per bit in each technology generation [63]. The ITRS roadmap sets as a goal a steady 1000 FIT per chip due to SEUs and indicates that potential solutions may exist [13]. However, enhanced microarchitectural techniques will be required to mask a large fraction of the chip-level FITs and achieve a satisfactory level of resiliency.
- **Variability**: As transistors and wires shrink, the spatial and temporal variation of their electrical characteristics will increase, leading to an increase in speed-related intermittent or permanent faults in which a critical path unexpectedly fails to meet timing. Researchers predict that the threshold voltage for transistors on the same die could easily vary by 30% [19]. Managing variability will be a major challenge for process, device, circuit and system designers.

Figure 6.39 summarizes several of these effects as a function of time, with a projection into the Exascale time frame.

6.7.3 Resiliency Techniques

As resiliency has become more important in the high-performance commercial marketplace, microprocessor designers have invested more heavily into means of detecting and correcting errors in hardware. These techniques typically fall into the following categories:

- **Encoding**: Parity and SECDED codes are commonly used to detect or correct errors in memory structures and in busses. Encoding provides low-overhead protection and on-line

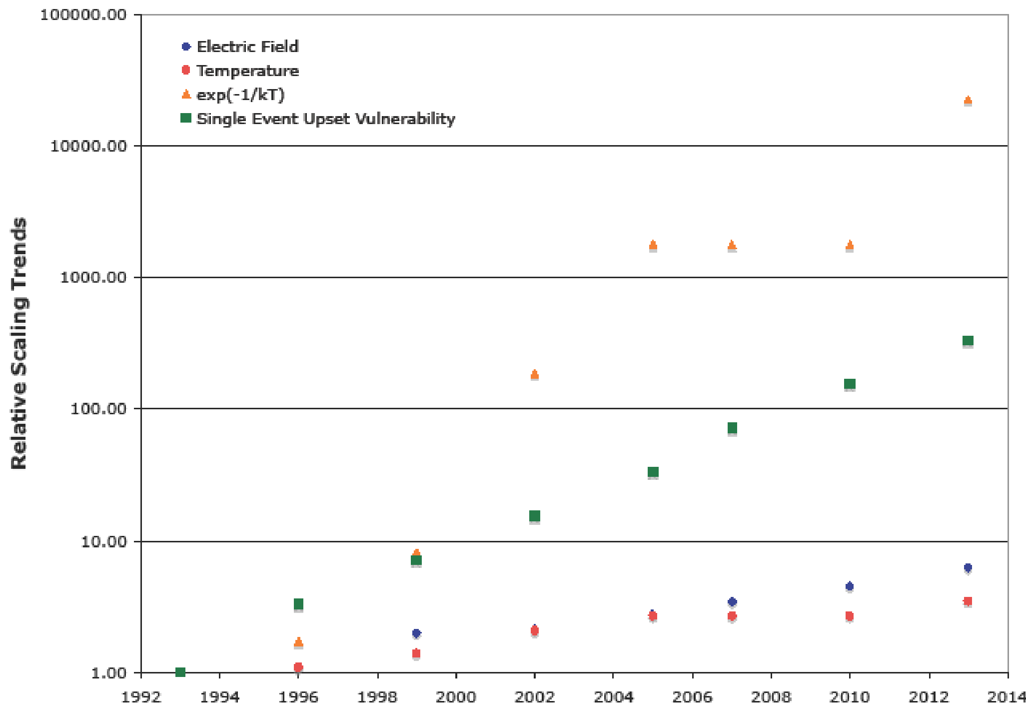


Figure 6.39: Increase in vulnerability as a function of per-socket failure rates.

correction of corrupted bits and can be used to protect against intermittent single upset events as some forms of permanent hard failures. Encoding overheads are typically in the 5-10% range for those structures to which encoding can be applied.

- **Scrubbing**: Memory scrubbing involves frequently reading, correcting, and immediately rewriting regions of memory, and eliminates latent errors in stored data before it is used by flushing unused data items from the storage and fixing any correctable data elements. Scrubbing can be applied to memory structures, caches, and register files and is typically a means of reducing the likelihood that a single-event upset will cause a program failure. Scrubbing typically incurs very little area and time overhead.

- **Property Checking**: Many operations in a processor or system can be verified by checking properties during execution. While a simple example is a bus protocol in which no two clients should be granted the bus at the same time, this technique can be widely applied to different parts of a processor and system. Property checking can be implemented with relatively little overhead.

- **Sparing**: Hard component failures can be tolerated, often without system failure, by providing a spare component that can be swapped in. This technique can be applied at the chip level (spare processing elements or rows/columns in a DRAM) or at the system level in the form of spare power supplies or disk drives. Sparing relies upon some other mechanism for detecting errors. Sparing is typically considered as 1 of N in which a spare can be swapped in for one of several components. A larger value of N results in less overhead.

- **Replication**: Replication can be used for both detection and correction. Detection typi-

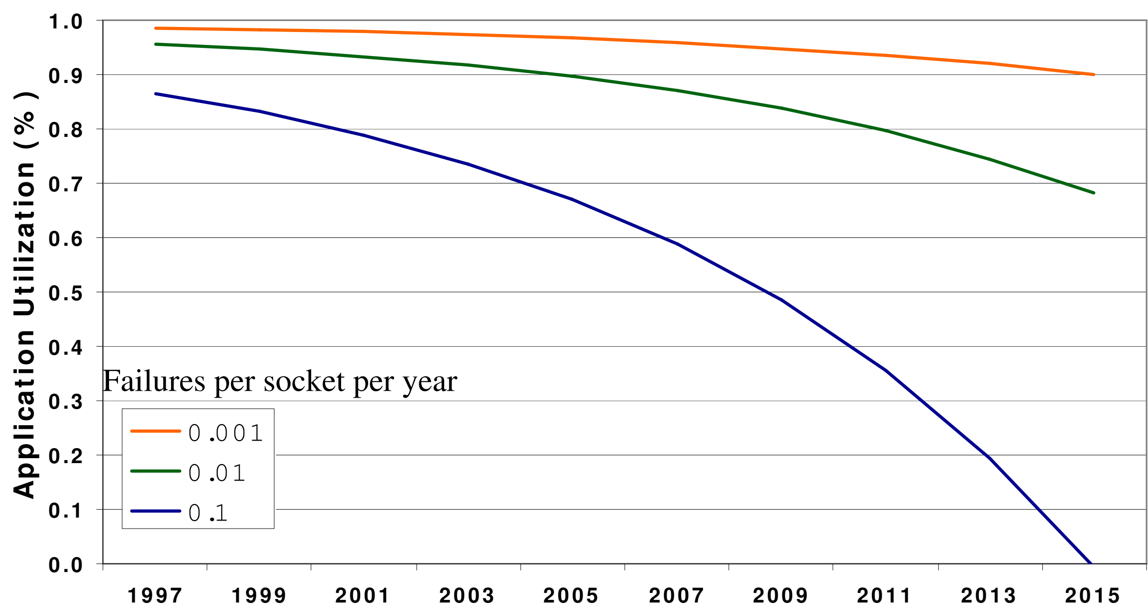


Figure 6.40: Projected application utilization when accounting for checkpoint overheads.

cally requires two parallel versions of the component to operate simultaneously and compare outputs. An error is detected when the comparisons mismatch, and requires some alternate form of recovery. **Triple modular redundancy** (**TMR**) provides three copies of the component and votes among the three with the majority providing the defined correct output. Replication is much more expensive, requiring 100-200% overhead.

The IBM Power6 microprocessor incorporates encoding, scrubbing, property checking, and sparing as a means to increase resiliency and reduce the FIT rate [120]. Processors such as the IBM G5 replicate significant pieces of the pipeline in a self checking mode [138]. Highly available systems, such as the Compaq Nonstop Himalaya, employed TMR to minimize the down-time in mission critical systems. Modern supercomputers supply fail-over spare power supplies within a chassis (tens of nodes) as well as spare power circuits in the power distribution network.

6.7.4 Checkpoint/Rollback

At the system level, recovery is often implemented using a checkpoint-rollback scheme, where a **checkpoint** is the copying of enough of application memory to an alternative storage medium in a fashion that allows for the application to be stopped and then at some arbitrary time to be restarted by moving the copied data back from this medium (**rollback**). Today, most such schemes are "application-agnostic," that is do not try to minimize the amount of data copied by selection. Thus, for sizing purposes, a checkpoint operation requires copying essentially all of data memory.

Typically, the application's execution must be suspended during this checkpointing period so that the data being checkpointed is consistent. This dead time represents an overhead that reduces system utilization, and thus effective system application-level performance.

The cost of a checkpointing scheme depends on the time to take a checkpoint, the checkpointing interval, time to recover, and the rate at which recoveries are necessary (typically correlated with the MTTI). As an example, BlueGene/L aims for a single-checkpoint cost of 12 minutes for application-

initiated checkpointing rollback/recovery, and employs a several techniques to reduce the overhead, including incremental checkpointing and memory hashing [115]. At this cost, a checkpointing interval of 2.5 hours will result in an overall 8% overhead when faults do not occur (8 minutes/150 minutes), meaning that the system can execute user applications at most 92% of the time (if no faults occur requiring rollback).

Knowing the time to perform the checkpoint (t), the period between checkpoints (p), and the mean time between interrupts (MTTI) allows this utilization to be computed as[50]:

$$(1 - AppUtilization) = (t/p) + p/(2 * MTTI) \tag{6.11}$$

with a minimal overhead found when:

$$p = sqrt(2 * t * MTTI) \tag{6.12}$$

This actual utilization percentage must then be applied to the performance of the system as a whole to come up with a "sustained performance." Figure 6.40 shows the projected application utilization rates of large scale systems as a function of machine size and MTTI resulting from 0.1–0.001 failures per socket per year. The time to take a checkpoint is held constant at 12 minutes, and the checkpointing is assumed to occur at optimal intervals. Rollback/recovery overhead quickly dominates the application, rendering the machine useless using observed Terascale failure rates.

Extrapolating this to Exascale systems, if the checkpoint time is similar, but the equivalent failure rate per socket does not improve from its current 0.1 value, then MTTI will remain in the few hours time scale. Also, if we assume that Exascale memory will be orders of magnitude greater than the Terascale machines used for this study, it is highly likely that, unless extraordinary increases in bandwidth the checkpointing memory is made, that this 12 minute checkpoint time must itself escalate to the point where the machine is useless. Figure 6.30 in Section 6.4.1.3 gives an absolute minimum time per petabyte of about 15 seconds for a system in 2015 with 1 Exabyte in about 150,000 drives running at 100% of max bandwidth. For a 10 PB disk system running at perhaps 30% of peak bandwidth and supporting the low-memory 3.6PB configuration of the aggressive strawman of Section 7.3, this time might balloon out to nearly 5 hours - clearly a ridiculous number! Thus, in the absence of significantly new storage technologies, per socket and machine MTTI **must** be driven down in order for applications to make good use of large scale machines.

6.8 Evolution of Operating Environments

Operating environments represent the collection of system software that manage the totality of computing resources and apply them to the stream of user tasks. As described in section 4.2, conventional operating environments comprise node operating systems, core compilers, programming languages and libraries, and middleware for system level resource allocation and scheduling. It also supports external interfaces to wide area networks and persistent mass storage but these are dealt with elsewhere in this report.

Near term evolution of operating environments is required to address the challenges confronting high performance computing due to rapid technology trends described earlier in this report. Among these are the reliance on multi-core processor components to sustain continued growth in device performance, the increasing application of heterogeneous structures such as GP GPUs for acceleration, the increase in total system scale measured in terms of number of concurrent threads, and the emergence of a new generation of pGAS (Partitioned Global Address Space) programming languages.

A major effort is underway through a number of industry and academic projects to develop extensions to Unix and Linux operating systems to manage the multi-core resources of the next generation systems. The initial starting point is a kernel on every core. This requires cross operating system transactions through the I/O name space for even the simplest of parallel processing. Operating system kernels that can manage all the cores on a single socket or even multiple sockets are being developed. This is similar to earlier work on SMPs and enterprize servers of previous generations. But they must now become ubiquitous as essentially all computing systems down to the single user laptop is or will shortly become of this form. One of the key challenges is the implementation of light weight threads. Conventional Unix P-threads are relatively heavy weight providing strong protection between threads but requiring a lot of overhead work to create and manage them. User multi-threaded applications will require threads with a minimum of overhead for maximum scalability.

A second evolutionary trend is in node or core **virtualization**. As a wider range of microarchitecture structures and instruction sets are being applied with systems taking on a diversity of organizations as different mixes of processor types and accelerators are being structured. Virtualization provides a convenient and standardized interface for portability with the assumption that the virtualization layer is built to optimally employ the underlaying system hardware. Where this is not feasible, separate interface libraries are being developed to make such resources at least accessible to users should they choose to take advantage of them.

The community as a whole is converging in a set of functionalities at the middleware level. These are derived from a number of usually separately developed software packages that have been integrated by more than one team in to distributions that are ever more widely used, especially across the cluster community. However, the individual pieces are also being improved for greater performance, scalability, and reliability, as well as advanced services. This is particularly true in the area of schedulers at the system level and microarchitecture level.

6.9 Programming Models and Languages

To support the extreme parallelism that this report predicts will be required to achieve Exascale computing, it is necessary to consider the use of programming models beyond those in current use today as described in section 4.3. The road map for programming models and languages is driven by two primary factors: the adoption rate in the application community of application developers and the investment available to develop and deploy the underlying compiler and run times required by new languages and models. Both pose significant challenges which must be overcome. Although the uncertainties posed by these factors makes a precise road map difficult, we describe the most likely path in this section.

6.9.1 The Evolution of Languages and Models

In the past decades, there have been numerous attempts to introduce new and improved programming languages and compilers targeted both at mainstream and high-end computing. While several of these attempts have succeeded, most have failed. To understand what leads to the success and failure of new approaches in general, it is instructive to examine their lifespan.

Programming languages and models are generally initiated by one of two paths: **innovation** and **standardization**. In the innovation path, there is generally a single organization which is proposing some new concept, produces a tool chain, attempts to attract users (usually themselves first), and aims to get a significant user base. This is important because the costs of implementing infrastructure are high and must be justified over a large (at least potential) user community.

The standardization path is quite different. It involves a group of organizations coming together either to combine related language and programming model concepts or amend existing standards. Here, the innovation introduced is far less than the other approach, but the community of users is almost guaranteed and often a commitment exists to the required tools at the outset.

Regardless of the path taken, these efforts have many common elements: they start with a period of rapid change in which experimentation is performed but in which the user community is small; then there is a crucial period where the rate of change slows and the user community is either attracted to the ideas and a critical mass is established, or the project fades away; and finally, for successful projects, there is a life-cycle support in which only very minor changes are made due to pressure from the user community.

In addition to these factors, a number of others influence success or failure. For example, the ideas represented in Java had been proposed numerous times before, but until they were popularized in C++ object-oriented programming, these new ideas did not achieve critical mass. Finally it should be noted that the path from innovation to adoption is quite long, usually about a decade.

6.9.2 Road map

Here we list and describe the path that specific language and model efforts will most likely take in the next 5-7 years without a significant uptick in funded research and development efforts.

The major incumbent models and languages in the high-end world will continue to remain the dominant mechanism for high-end computing. MPI will continue to dominate the message passing models and will certainly be widely deployed. It will most likely evolve to include some new features and optimizations. For a class of applications and users, this will be viewed as sufficient. In the various non-message passing realms, the dominant influence on language will be the arrival of significant multi-core microprocessor architectures. It is clear that the non-high-end community needs to develop models here if the architecture is to be successful. It appears somewhat likely that some current language efforts which involve either threaded approaches (Cilk, pthreads etc.) or pGAS (Chapel, Co-Array Fortran, Titanium, UPC, X10) will influence the course.

There is also a trend towards better language and model support for accelerator hardware. For example, Cuda (NVidia) and BTBB (Intel) are competing to support the graphics processor accelerators. SIMD extensions remain popular. And support for memory consistency models, atomic memory operations and transactional memory systems are being investigated to support better shared memory program synchronization. We also see a trend to hybridization of many of the above approaches. While some of these are laudable for either local node performance or synchronization, in their current form most of these mechanisms seem too specialized for widespread adoption.

Chapter 7

Strawmen: Where Evolution Is and Is Not Enough

This chapter attempts to take the best of known currently available technologies, and project ahead what such technologies would get us in the 2013-2014 time-frame, where technology decisions would be made for 2015 Exascale systems. The results of these projections will provide the basis for the challenges that await such developments.

7.1 Subsystem Projections

To set the stage in terms of general feasibility and potential challenges, a series of short projections were made of different aspects of Exascale systems, and are documented in the following sections. While in most cases the focus was on the data center class of Exascale systems, the results are for the most part directly translatable to the other two classes.

The order of presentation of these results are more or less in a "bottom-up" correspondence to parts of the generic architectures presented in Chapter 4.

Since the goal of this study was not to design Exascale systems, many of these brief studies do not focus on point designs but on generic numbers that can be scaled as appropriate to match desired application needs. Thus memory capacity is discussed in terms of "per petabyte," and power dissipation expressed in terms of system energy expended per some unit of performance and some baseline implementation.

7.1.1 Measurement Units

Much of the baseline numerics below were chosen to be reasonable but rounded a bit to make calculations easy to follow. Thus for example, we use as a baseline of performance an **exa instruction processed to completion** (1 **EIP**), that corresponds to executing and retiring 10^{18} more or less conventional instructions. When we wish to denote the completion of 1 EIP in 1 second, we will use (in analogy to MIPs) the term "EIPs" (Exa Instructions Processed per second) Again, the term EFlops refers to the completion of 10^{18} floating point operations in a second, and is a different measure than EIPs.

In most typical programs perhaps 40% of such instructions reference memory; we will round that up to 50%. Thus for each EIP executed per second, perhaps $0.5*10^{18}$ distinct references to memory are made from the core itself into the memory hierarchy.

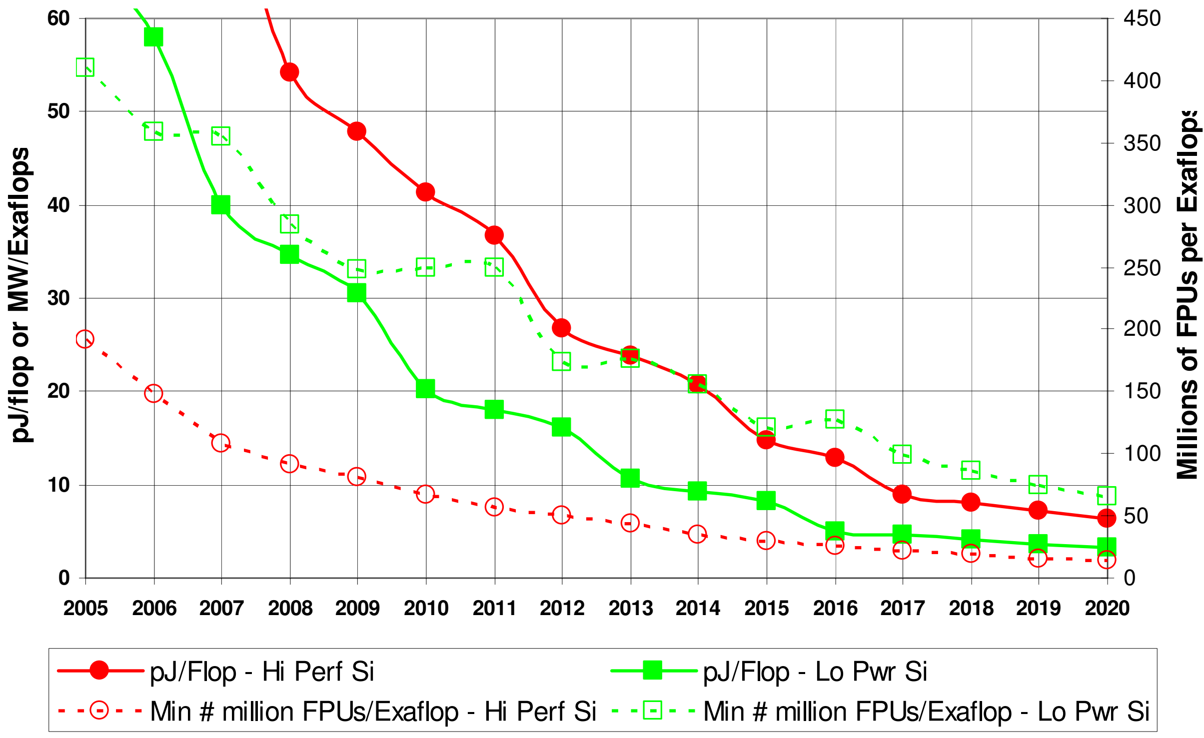


Figure 7.1: Projections to reach an Exaflop per second.

Further, since in most cached hierarchies only some percentage of such memory references actually go off to real memory, and this percentage varies both on the size of the caches and the application, we will baseline a unit of cache miss at 2%. Thus for each EIP executed per second, and each 2% of miss that the application actually experiences, there are about $0.5*10^{18} * 0.02 = 10^{16}$ distinct references that must reach main memory.

Finally, assuming a classical computer ISA where individual floating point instructions trigger one floating point operation, and that the percentage of such instructions in a program is on the order of 10% (again to make the math simple to follow), then achieving a sustained 1 EFLOP/s (10^{18} flops) requires 10 EIP/s, which in turn requires 10^{17} real memory accesses for each 2% of overall cache miss exhibited by the application. Thus a 20% miss rate for a sustained 1 EFLOP/s application with a 10% floating point mix would see on the order of 10^{18} distinct memory accesses per second (or 1 exa accesses per second).

For reference Murphy[109] analyzes a range of high end applications where floating point mixes range from 6 to 40%, and miss rates out of even very large caches can exceed 60%.

In terms of the relationship between energy and power, we note that numerically, if computing some operation takes X pJ of energy, then computing 10^{18} of them in one second consumes X MW of power.

7.1.2 FPU Power Alone

While this study does not equate exaflops to Exascale computing, understanding whether or not, and when, conventional silicon can provide an exaflop per second within the desired power limits is a useful exercise to bound the rest of the discussion. Using the ITRS data from Section 6.2.1, Figure 7.1 projects the energy expended for a single flop as a function of time for both high performance

Core	L1:I+D (KB)	FPU	Power	Tech	Area	Vdd	Clock (GHz)	Native pJ/Cycle	90nm pJ/Cycle	90nm Area
Niagara-I	24	No	2.06	90	11.9	1.2	1.2	1719	1444	11.9
Niagara-II	24	yes	3.31	65	12.4	1.1	1.4	2364	3274	23.9
MIPS64	40	No	0.45	130		1.2	0.6	750	436	0.0

Figure 7.2: Energy per cycle for several cores.

silicon and low power variants. The baseline for this estimate is a 100 pJ per flop FPU built in 90 nm technology, as discussed in [43] .

As noted before, numerically an X pJ per operation is the same as X MW for an exa operation of the same type per second. Thus, it isn't until 2014 that the power for flops alone drops below 20MW for conventional high performance CMOS. If fact, it isn't until 2020 that the power is low enough to even conceive of enough margin to account for other factors such as memory or interconnect.

The story for low power CMOS is better. In the 2013-2014 timeframe FPUs alone would dissipate in the order of 6-7 MW, enough to at least leave some space for the other functions.

A second key observation to make from this data is the number of FPUs that are needed at a minimum to reach the exaflop per second limit. The dotted two curves in Figure 7.1 give an estimate of this for both types of logic, assuming that the implementations are run at the maximum possible clock rate (where power density becomes a significant chip problem). High power silicon needs only about 50 million copies to get to an exaflop per second, but this is assuming a 20+GHz clock and a power dissipation per unit area of almost an order of magnitude greater than today. The low power silicon's maximum rate is only in the 5-6GHz range, but now requires at a minimum 200 million copies, and still has a power dissipation density exceeding today.

The clear conclusion is that if we are to rely on silicon for floating point functions in 2013-1014, then it has to be using the low power form, and even then there are problems with power density (requiring lowering the clock) and in the huge concurrency that results (which gets even either worse at lower clock rates). The complete strawmen discussed later in this chapter assume both.

7.1.3 Core Energy

Clearly FPU energy alone is not the only processing-related logic that will probably be present in an Exascale system. To get a handle on how large such energies might be, Figure 7.2 takes some data on existing 64 bit cores[93][159][110], and computes an energy per clock cycle, both for the real base technology and normalized to 90 nm (the 2005 base in the prior charts). The results reveal both a large variation in energy per cycle, and a significant multiple over what the FPU assumed above would be.

The Niagara rows are especially revealing. Both are multi-threaded, which implies that there is a bigger register file, and thus a larger energy component for reading registers, than for classical single-threaded cores. Whether or not this accounts for the approximately 3X difference from Niagara I to the MIPS64 line is unclear, especially considering the MIPS64 is a dual issue microarchitecture, while that for the Niagara is only single issue.

In addition, the Niagara I does not have any integrated FPU, while that for the Niagara II has not only an FPU but a whole separate graphics pipeline. When normalized, the difference in energy of about 1300 pJ is significantly more than that for our assumed FPU, even if we subtract out about 30% of the measured power to account for static leakage.

The bottom line from this discussion is that it is not just the FPUs that need to be considered,

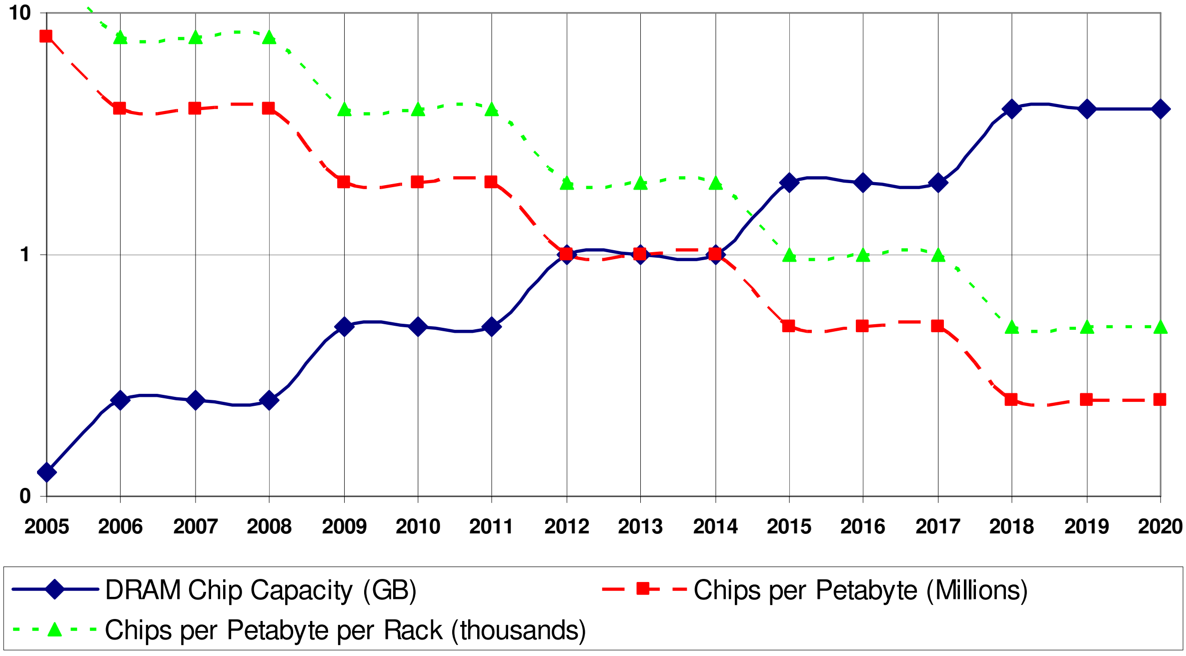


Figure 7.3: DRAM as main memory for data center class systems.

even for flop-intensive applications, and that if we are to minimize overall energy, it is essential to devise microarchitectures where the overhead energy for fetching instructions, decoding them, fetching operands, and managing their retirement is both as low as possible, and spread out over as many flops as possible.

7.1.4 Main Memory from DRAM

The needed capacity of the "main memory" level of the memory hierarchy of Chapter 4 has proven to be one of the most volatile requirements of Exascale applications, with "proof of concept" systems possible with just a few petabytes of storage, but more demanding and perhaps significant applications (that is those that would justify such a machine) needing in the hundreds of petabytes. The following sections walk through various aspects of implementing such main memory using today's DRAM technology as it is projected to mature through time.

7.1.4.1 Number of Chips

Figure 7.3 includes a projection for the capacity of high volume commodity DRAM memory chips through time. In the 2013-2014 time frame this is about 1 GB per chip.

The second line in this figure is the number of chips needed to reach a petabyte. In 2013-2014 this is about a million chips per PB. Thus, if the actual application requirement is for 100PB of main memory, this would take 100 million commodity DRAM chips. This implies a real need for a denser packaging of DRAM than what is used today.

The third line on the graph is the number of chips per rack, assuming 500 racks in the system. Again, this is per PB, so if the actual need was for 100PB, then in 2013-2014 we would need on the order of 200,000 commodity DRAM chips to be packaged in each rack. For reference, today's supercomputers may house at most a few thousand such chips per rack.

Assuming a FIT rate of 10 per billion hours per chip, such volumes of chips quickly translate into a mean time between memory chip failures of between 1 and 100 hours, depending on the overall memory capacity.

Note that no ECC or other redundancy in included in these numbers, so some growth in chip count for a realistic system should be expected (perhaps 12% for a standard SECDED code on each 64 bit word). While increasing the overall system FIT rate from a component perspective, such density increase would improve the MTBF of the memory part of the system.

7.1.4.2 Off-chip Bandwidth

As discussed in Section 7.1.1, a total number of distinct accesses to main memory of between 10^{16} (10 peta accesses) and 10^{18} (1 exa access) per second as an application-driven goal is not unreasonable. Assuming a conventional DRAM-like interface to these chips, each access results in the transfer of between 8 and 32 bytes of data (a word to a cache line) across some set of chip boundaries - not counting ECC. Rounding up a bit, this translates into an aggregate transfer rate off of the memory chips of between 0.1 and 32 exabytes per second.

We note that these aggregate rates are independent of the memory capacity. Thus on average the data rate per chip is this number divided by the number of chips. For 1 PB (a million chips), this translates to 100 to 32,000 GB/s - per chip! These numbers are far in excess of any projected commodity memory chip signalling protocol. For 100 PB (100 million chips), this reduces to a mere 1 to 320 GB/s per chip.

We also note that today's memory parts are typically perhaps a byte wide, and to access larger entities, multiple chips are ganged and accessed in parallel. This means that the address and control bits that must be transferred per access must be repeated per chip. If this address and control information averages 32 bits per access, then if it must be broadcast identically to 8-10 chips in parallel, there are upwards of 300+ extra bits that must be transferred per access across chip boundaries. This has the potential to almost double the overall bit transfer rate to and from a main memory made of DRAM.

Finally, we note that going to a denser memory technology makes the off-chip bandwidth requirements even worse, since there are fewer chips for any particular memory capacity.

7.1.4.3 On-chip Concurrency

For this section an **independent memory bank** is that part of a memory circuit that can respond to an address and deliver a row of data as a result. This includes row decoders, memory mats, and sense amplifiers, but need not include anything else. Thus an **active bank** is one that is responding to a separate memory request by activating a memory mat, and reading out the data.

For reference, today's memory chip architectures support at best about 8 such independent banks.

Understanding how many such banks need to be capable of independent access on a typical memory chip is thus important to determine how much "overhead logic" is needed to manage the multiple banks, and how much power is dissipated in the memory mats where the data actually resides.

An average for the number of accesses per second that must be handled per chip can be estimated by taking the total number of independent memory references per second that must be handled, and dividing by the number of memory chips. The first is a property of the application and the caching hierarchy of the processing logic; the second is related directly to the capacity of the memory system. Figure 7.4(a) lists this for a variety of memory reference rates (where 10^{16} to 10^{18} is what

		Main Memory Capacity (in PB)			
		1	10	100	1000
Accesses/sec	1.E+15	1	0.10	0.01	0.001
	1.E+16	10	1	0.10	0.01
	1.E+17	100	10	1.00	0.10
	1.E+18	1000	100	10	1
	1.E+19	10000	1000	100	10

(a) References per chip (in billions)

		Main Memory Capacity (in PB)			
		1	10	100	1000
Accesses/sec	1.E+15	10	1	0.1	0.01
	1.E+16	100	10	1	0.1
	1.E+17	1000	100	10	1
	1.E+18	10000	1000	100	10
	1.E+19	100000	10000	1000	100

(b) Active Banks per chip

Figure 7.4: Memory access rates in DRAM main memory.

was discussed previously) and the spectrum of main memory capacities discussed for applications of interest.

To convert this into a number of independent banks requires estimating the throughput of a single bank, that is, the maximum rate that a single bank can respond to memory requests per second. For this estimate, we assume a bank can handle a new request every 10 ns, for a throughput of 10^8 references per second per chip. (This is a bit of an optimistic number, given current technology, but not too far off.) Figure 7.4(b) then uses this number to compute the number of active banks for each of the prior configurations. As can be seen, if main memory access rates exceed 10^{16} per second (on the low side of what was discussed above), then it isn't until memory capacities exceed 100 PB (100 million chips) that the average number of active banks drops down to what is close to today.

7.1.5 Packaging and Cooling

The degree of difficulty posed by the packaging challenge depends on the memory bandwidth requirements of the eventual system. In this section we use as a basis the strawman machine as discussed in Section 7.3. The lower end of the requirement, 44 GBps from the CPU to each of 16 DRAM die can most likely be satisfied with conventional high-end packaging. The high-end requirement of 320 GBps to each of 32 DRAMs is well beyond the means of any currently available packaging technology. Some embedded applications were investigated to obtain another perspective on the possible size requirement. As discussed elsewhere in this document, the usual requirement of 1 Byte per second and 1 Byte of memory for every FLOPS, would require larger scales of memory system but would have significant power issues. A system anywhere near this scale would require significant advances in interconnect and packaging. In particular, advances in 3D system geometries would be required.

An advance in 3D packaging also presents an opportunity to use geometry to reduce power consumption. With a conventional packaging approach, aggressive scaling of interconnect power

would permit memory-CPU communications at an energy cost of around 2 pJ/bit. On the other hand, some 3D integration technologies, would permit power levels to approach 1-20 fJ/bit, depending on the length of run of on-chip interconnect required. Even at the low end bandwidth of 16 x 44 GBps, this represents a power savings of around 10 W per module, which could be more productively used for FPUs or additional memory capacity.

As well as provisioning interconnect, packaging also plays roles in power and ground current distribution, noise control (through embedded capacitors), heat removal and mechanical support, so as to ensure high reliability. The simultaneous roles of current delivery and heat removal create a geometric conundrum as the high power areas of the chip need lots of both at the same time. This requirement leads to the usual solution in single-chip packaging of using the front-side of the chip for current delivery and the backside for cooling. Such arrangements are not easily scaled to 3D integration as one side of the chip has to be used for the 3D mating. As a result, the typical state of the art expected for circa 2015 would be a 3D chip stack mating a small number (3-4) of memory die to the face side of a CPU. While the back-side of the CPU chip is used for cooling, extra through-silicon vias are built into the memory stack in order to deliver current to the CPU. Stacks beyond 4-5 die are not expected, due to the combined problem of sufficiently cooling the interior die (through the hot CPU), and providing current delivery through the memory die stack.

Some options that could be pursued are summarized in Figures 7.5 to 7.7. The first suggestion (distributed 3D stacks) simply avoids this problem by portioning the CPU amongst a number of smaller memory stacks, and integrating these 3DICs using a silicon carrier. However, the cost of this is a dramatic reduction in inter-core bandwidth, once they cross chip boundaries. This is unlikely to be acceptable in many applications.

The second suggested option is referred to as an "advanced 3D IC," that is a 3D IC with package layers incorporated into it, in the 3D stack. These package layers are used for current distribution and heat removal from die within the stack. The result would be a heterogeneous chip stack, tens of units high. Considerable challenges exist in this option. The technology to build such a stack with high yields does not exist today.

A variant of the "advanced 3D IC" solution is a more package-centric solution that allows 3D integration of multiple chip stacks. This could be done using a combination of chip-on-wafer and wafer-on-wafer technologies. However, it would still require considerable technology investment.

A variant of Sun's proximity connection could also be adapted to solve this problem. However, a combination of technologies could be used for interconnect, rather than just relying on capacitive coupling. As discussed in Section 6.6, Sun's original scheme implicitly requires current distribution and heat removal through **both** sides of the 3D package, which of course is difficult. However, supplementing their approach with through-silicon-vias could relieve this situation and provide an interesting solution.

Another approach would be to use something like Notre Dame's "quilt packaging"[15] to provide high density edge connections, on a tight sub-20μm pitch. In their solution, all the chips are face-up, so the two-sided power delivery and removal problem is easily solved. However, one challenge with both of these solutions resolves around the fact that they are both edge I/O approaches. Numerous, long power-hungry on-chip wires would be required to join peripheral memories to interior cores.

Several groups have investigated edge-mounting of die on the planar surface of another die. This leads to a memory edge mounting solution, such as shown in Figure 7.7. There are several complications with this solution. It does not directly solve the two-side power delivery/removal problem. Also, the memories would require long, power-hungry on-chip traces.

Finally, it should be realized that advances in areas outside of packaging could simplify the creation of an Exascale 3D solution. In particular, if efficient voltage conversion could be incorporated within the chip stack, then the "two-sided problem" is greatly simplified. Delivering 100 A at 1 V

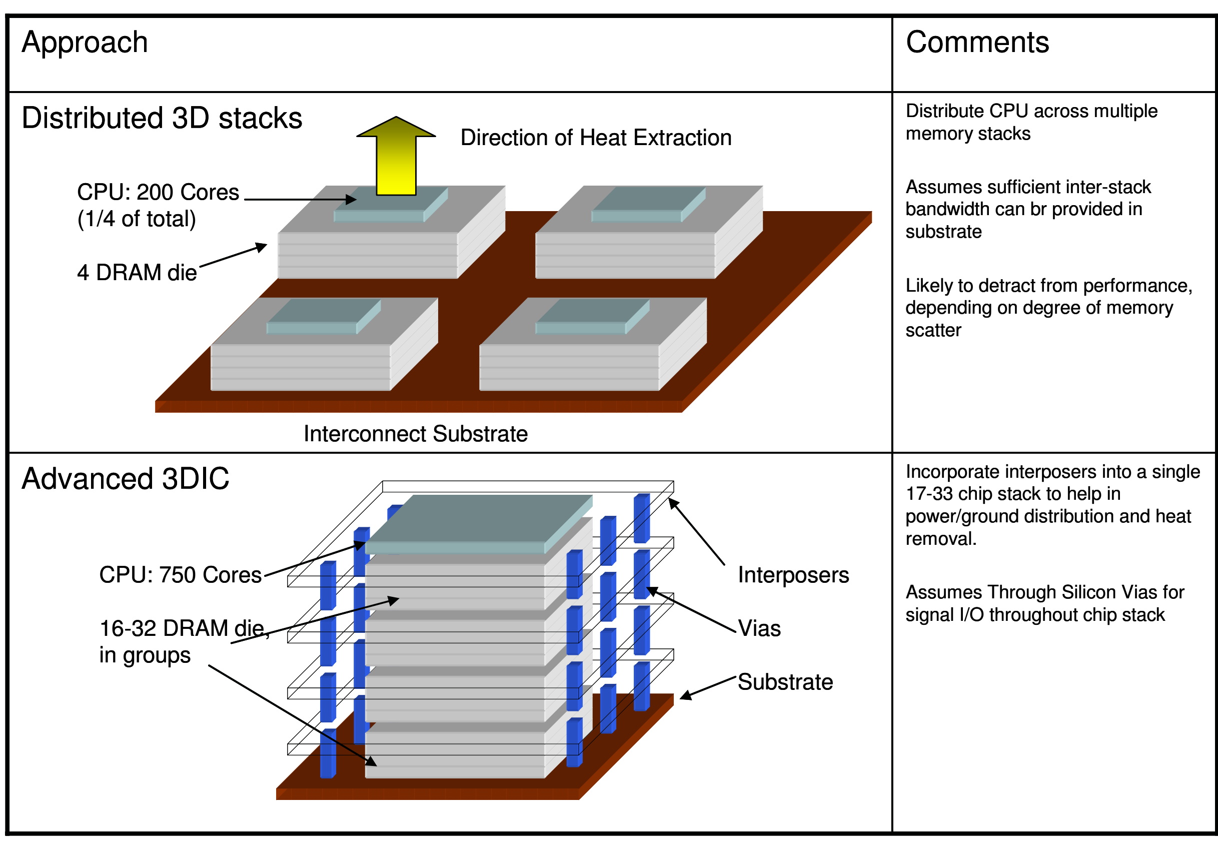
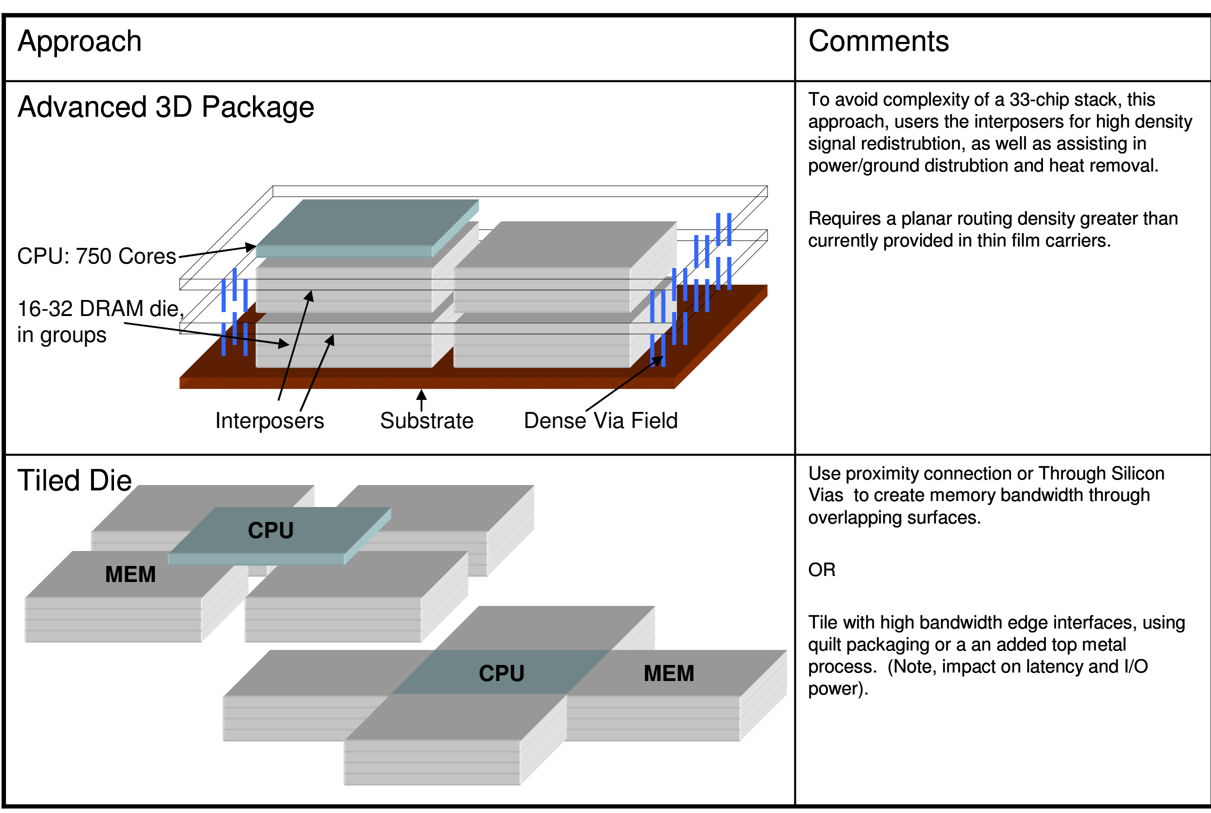


Figure 7.5: Potential directions for 3D packaging (A).



Approach	Comments
Advanced 3D Package	To avoid complexity of a 33-chip stack, this approach, users the interposers for high density signal redistrubtion, as well as assisting in power/ground distrubtion and heat removal. Requires a planar routing density greater than currently provided in thin film carriers.
Tiled Die	Use proximity connection or Through Silicon Vias to create memory bandwidth through overlapping surfaces. OR Tile with high bandwidth edge interfaces, using quilt packaging or a an added top metal process. (Note, impact on latency and I/O power).

Figure 7.6: Potential directions for 3D packaging (B).

Approach	Comments
Edge Mounting CPU	Stack memory die on top of CPU using a 3D quilt process or functional equivalent. Requires considerable technological innovation
Orthogonal Solutions On-die or in-stack voltage conversion Embedded liquid cooling mm-thick high capacity heat pipes	These solutions provide ways to alleviate some of the difficulties in the solutions above.

Figure 7.7: Potential directions for 3D packaging (C).

	PB of Main Memory				
	0.006	0.5	3.6	50	300
Scratch Storage					
Capacity (EB)	1.2E-04	0.01	0.15	2	18
Drive Count	1.0E+01	8.3E+02	1.3E+04	1.7E+05	1.5E+06
Power (KW)	9.4E-02	7.8E+00	1.2E+02	1.6E+03	1.4E+04
Checkpoint Time (sec)	1.2E+03	1.2E+03	5.8E+02	6.0E+02	4.0E+02
Checkpoint BW (TB/s)	5.0E-03	4.2E-01	6.3E+00	8.3E+01	7.5E+02
Archival Storage					
Capacity (EB)	0.0012	0.1	7.2	100	600
Drive Count	1.0E+02	8.3E+03	6.0E+05	8.3E+06	5.0E+07
Power (KW)	9.4E-01	7.8E+01	5.6E+03	7.8E+04	4.7E+05

Table 7.1: Non-memory storage projections for Exascale systems.

is a lot harder than delivering 1 A at 100 V. Similarly, releasing one side from a cooling function provides a similar improvement. For example, incorporating silicon micro-channel cooling into the chip stack removes the need for using one side for heat removal.

On the topic of cooling, it was presumed for this study that at the system level, the computing stack would be air-cooled. Large scale deployment of water chillers and circulating coolants does not lend itself to embedded solutions in war fighting craft and vehicles. However, this global issues does not prevent the local solution from using liquid phase solutions locally. As long as the liquid does not require replenishment, then such local solutions might be reasonable. There are severe challenges in such solutions though. Leaks are still a problem in liquid cooling, and a leak-free technology would be a worthwhile investment. Also, techniques to circulate and pump the coolant are needed on the physical scale of a small 3D package. Similarly, local heat exchangers would be needed if local liquid cooling solutions are to be used. Heat pipes provide an excellent way to use liquid phase cooling locally without mechanical complexity. Advances in the capacity of thin heat-pipe like solutions would be welcome in an Exascale computer.

7.1.6 Non-Main Memory Storage

Using the numbers from Section 5.6.3, especially as articulated in Table 5.1, and the projections from Section 6.4.1, Table 7.1 summarizes numbers for scratch and archival storage systems using disk technology from 2014. Consumer-grade disks are assumed because of the need for sheer density. As before, these numbers do not include either additional space for RAID or for controllers and interconnect, and as such represent a lower bound.

The Checkpointing time and bandwidth assume that *all* of the drives in the Scratch system are accepting data concurrently, and at their maximum projected data rates. This is not reasonable, especially for the larger systems.

The numbers given here are given as a function of main memory. As such, the options were chosen to match different sizes of systems:

- The 6TB column corresponds to the main memory capacity of a single rack of the aggressive strawman of Section 7.3.
- The 0.5PB column corresponds to the "sweet spot" for a departmental system as suggested in Table 5.1.

- The 3.6PB column corresponds to the main memory capacity of the complete exaflops aggressive strawman of Section 7.3.
- The 50PB column corresponds to the "sweet spot" for a data center class system as suggested in Table 5.1.
- The 300PB column corresponds to a data center class system that has the same memory to flops ratio as today's supercomputers.

As can be seen, the scratch disk counts for both classes of systems are not unreasonable until the main memory capacities approach ratios typical of today, such as 1.4M drives and 14 MW for a 0.3 to 1 byte to flop ratio at the data center scale.

Checkpointing time across all systems is in the realm of 10 to 20 minutes (under the optimistic assumptions of all drives storing at max rate). When stretched to more realistic times, this implies that checkpointing times may be in the region of the MTBF of the system. As discussed in Section 6.7.4, this may render the data center class systems effectively useless, indicating that there may be a real need in finding a memory technology with higher bandwidth potential (especially write rates) than disk. Variations on Flash memory may offer such a capability at competitive densities and power.

The archival numbers of course are higher, and probably reach the limits of rational design at 3.6PB main memory for the data center class strawman of Section 7.3, where 600,000 drive at close to 6 MW are needed.

7.1.7 Summary Observations

The above short projections lead inescapably to the following conclusions:

1. If floating point is key to a system's performance, and if CMOS silicon is to be used to construct the FPUs, then, for the data center class, to have any hope of fitting within a 20 MW window, the low operating power variant must be chosen over today's high performance design. This is also true of the other two classes, as the use of high performance logic results in power densities that exceed their form factor limits.
2. Energy for the processor core must be very carefully watched, lest power exceed that for the basic operations by a large factor. This implies that microarchitectures must be designed and selected for low energy per issued operation.
3. Unless at best a very few PB of main memory is all that is needed, then DRAM technology by itself is inadequate in both power and capacity.
4. To reduce the bit rates and associated energy of transfer to manage the address and control functions of an access, each DRAM chip needs to be rearchitected with a wide data path on and off chip, low energy per bit interfaces, and many potentially active banks.
5. Unless memory capacity gets very large, then using DRAM for main memory requires that the architecture of the DRAM provide for far more concurrently active mat access (regardless of word widths) than present today.

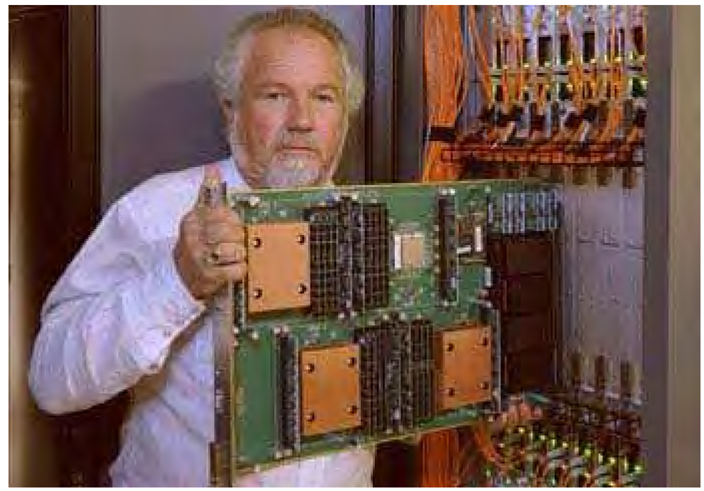


Figure 7.8: A typical heavy node reference board.

7.2 Evolutionary Data Center Class Strawmen

To baseline where "business-as-usual" might take us, this section develops two strawmen projections of what evolution of today's leading edge HPC data center class systems might lead to. This will stand in contrast to both the trend lines from Section 4.5 (that assumes technology will advance uniformly at the rate that it has for the last 20 years, but with the same architectural and packaging approach) and from the aggressive strawman of Section 7.3 (that assumes we can choose the best of current technologies in a relatively "clean-sheet" approach).

The architectures used as the two departure points for this study are "heavy node" Red Storm-class machines that use commodity leading edge microprocessors at their maximum clock (and power) limits, and "light node" Blue Gene/L class machines where special processor chips are run at less than max power considerations so that very high physical packaging densities can be achieved.

7.2.1 Heavy Node Strawmen

Machines such as the Red Storm and its commercial follow-ons in the Cray XT series assume relatively large boards such as pictured in Figure 7.8[90]. A few dozen of these boards go into an air-cooled rack, and some number of racks make up a system.

7.2.1.1 A Baseline

A "baseline" board today contains

- multiple (4) leading edge microprocessors such as from Intel or AMD, each of which is the heart of a "node." A substantial heat sink is needed for each microprocessor.
- for each node a set of relatively commodity daughter memory cards holding commodity DRAM chips (FB-DIMMs as a baseline).
- multiple specialized router chips that provide interconnection between the on-board nodes and other boards in the same and other racks.

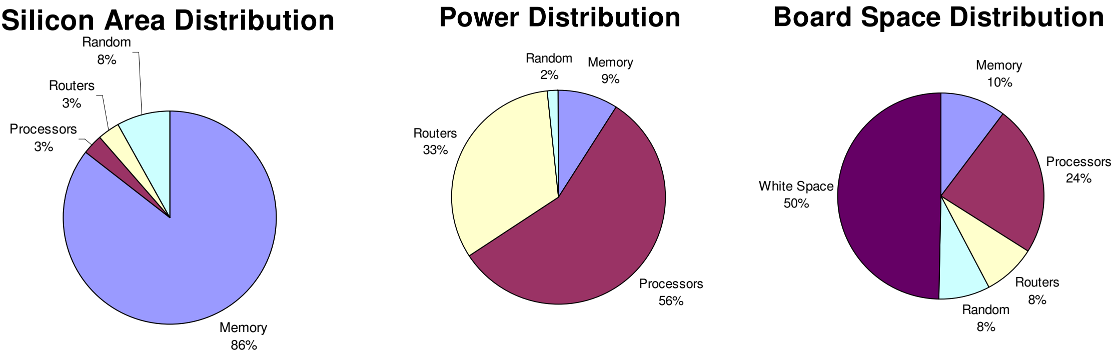


Figure 7.9: Characteristics of a typical board today.

- an assortment of support circuitry.

Very often the chip set associated with each node is referred to as a **socket**, since the single biggest physical entity on such a board is the microprocessor and its heatsink as pinned in its socket.

Figure 7.9 then summarizes the approximate characteristics of such a board as to where the silicon is used, which chips dissipate what part of the power, and how the board space is occupied.

In terms of a system baseline, we assume 4 sockets per board and nominally 24 boards per rack. A 2.5 MW system consisting of 155 racks would contain 12960 sockets for compute and 640 sockets for system, or one system socket for approximately every 20 compute sockets.

At 40 TB of storage and an R_{peak} of 127 Tflops, the baseline has a ratio of about 0.31 bytes per peak flop.

7.2.1.2 Scaling Assumptions

Extrapolating the same architecture forward into the future, what kind of parts are on such a physical board would not change much: a relatively constant amount of real estate is needed for each of the major subassemblies, especially the heat-sinked parts. To a first order what might change is the number of nodes/sockets on a board (fill the white space), the complexity of the microprocessor chips (succeeding generations would include more cores), the density of memory chips on the memory daughter cards will increase, and perhaps the number of memory cards (if performance per node increases faster than memory density increases). Thus our assumptions about future versions of such a system are as follows:

- The die size for the microprocessor chip will remain approximately constant, meaning that the number of cores on each microprocessor chip may grow roughly as the transistor density grows, as pictured in Figure 4.3. We do not account for relatively larger L3 caches to reduce pressure on off-chip contacts.

- V_{dd} for these microprocessors will flatten (Figure 4.7 and 6.2), as will maximum possible power dissipation (Figure 4.10), which means that the maximum clock rate for these chips will approximately flatten as discussed in Section 6.2.1.5.

- On a per core basis, the microarchitecture will improve from a peak of 2 flops per cycle per core in 2004 to a peak of 4 flops per cycle in 2008, and perhaps 8 flops per cycle in 2015.

- The system will want to maintain the same ratio of bytes of main memory to peak flops as today, and to do this we will use whatever natural increase in density comes about from commodity DRAM, but coupled with additional memory cards or stacks of memory chips as necessary if that intrinsic growth is insufficient.

- The maximum number of sockets (i.e. nodes) per board may double a few times. This is assumed possible because of a possible move to liquid cooling, for example, where more power can be dissipated and allowing the white space to be used and/or the size of the heat sinks to be reduced. For this projection we will assume this may happen at roughly five year intervals.

- The maximum number of boards per rack may increase by perhaps 33% again because of assumed improvements in physical packaging and cooling, and reduction in volume for support systems. For this projection we will assume this may happen once in 2010.

- The maximum power per rack may increase by at best a power of 16, to somewhere around 200-300KW. We assume this is a doubling every 3 years.

- We ignore for now any secondary storage (or growth in that storage) for either scratch, file, or archival purposes, although that must also undergo significant increases.

7.2.1.3 Power Models

We assume two power models for these extrapolations. The **Simplistic Power Scaled Model** assumes that the power per microprocessor chip grows as the ITRS roadmap has predicted, and that the power for the memory associated with each socket grows only linearly with the number of memory chips needed (i.e. the power per memory chip is "constant"). We also assume that the power associated with both the routers and the common logic remains constant.

This is clearly optimistic, since increasing the flops rate of the microprocessor most probably requires a higher number of references to memory and a higher traffic through the routers. In a real sense, we are assuming here that both memory access energy and the energy cost of moving a bit, either across a chip boundary or between racks, will improve at least as fast as the increase in flops.

In contrast, for the **Fully Scaled Power Model** we assume the microprocessor power grows as before (as the ITRS roadmap), but that both the memory and router power scale linearly with the peak flops potential of the multi-core microprocessors. This naively assumes that the total energy expended in these two subsystems is used in accessing and moving data, and that the energy to handle one bit (at whatever the rate) is constant through time (i.e. no improvement in I/O energy protocol). This is clearly an over-estimate.

Neither model takes into account the effect of only a finite number of signal I/Os available from a microprocessor die, and the power effects of trying to run the available pins at a rate consistent with the I/O demands of the multiple cores on the die.

7.2.1.4 Projections

There are a variety of ways that a future system projection could be done. First is to assume that power is totally unconstrained, but that it will take time to be able to afford a 600 rack system. The second is that we assume that we cannot exceed 20 MW. (In either case, however, we do assume a peak power per rack as discussed earlier).

	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020
Chip Level Predictions															
Relative Max Power per Microprocessor	1.00	1.05	1.10	1.10	1.10	1.10	1.10	1.10	1.10	1.10	1.10	1.10	1.10	1.10	1.10
Cores per Microprocesor	2.00	2.52	4.00	5.04	6.36	8.01	10.09	12.71	16.02	20.18	25.43	32.04	40.37	50.85	64.07
Flops per cycle per Core	2.00	2.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00	8.00	8.00	8.00	8.00	8.00	8.00
Flops per cycle per Microprocessor	4.00	5.05	16.00	20.17	25.44	32.04	40.37	50.85	64.07	161.43	203.40	256.29	322.92	406.81	512.57
Power Constrained Clock Rate	1.00	0.94	1.10	0.99	0.90	0.81	0.88	0.79	0.71	0.80	0.72	0.83	0.73	0.65	0.59
Relative Rpeak per Microprocessor	1.00	1.19	4.39	4.98	5.76	6.48	8.88	9.99	11.42	32.38	36.79	52.86	58.73	66.08	75.51
Actual Rpeak per Microprocessor	9.60	11.44	42.15	47.82	55.26	62.16	85.27	95.93	109.64	310.82	353.20	507.46	563.85	634.33	724.94
ITRS Commodity Memory Capacity Growth	1.00	1.00	1.00	2.00	2.00	2.00	4.00	4.00	4.00	8.00	8.00	8.00	16.00	16.00	16.00
Required Memory Chip Count Growth	1.00	1.19	4.39	2.49	2.88	3.24	2.22	2.50	2.86	4.05	4.60	6.61	3.67	4.13	4.72
Relative Growth in BW per Memory Chip	1.00	1.00	1.00	2.00	2.00	2.00	4.00	4.00	4.00	8.00	8.00	8.00	16.00	16.00	16.00
BW Scaled Relative Memory System Power	1.00	1.19	4.39	4.98	5.76	6.48	8.88	9.99	11.42	32.38	36.79	52.86	58.73	66.08	75.51
Socket Level Predictions ("Socket" = Processor + Memory + Router)															
BW Scaled Relative per Socket Router Power	1.00	1.19	4.39	4.98	5.76	6.48	8.88	9.99	11.42	32.38	36.79	52.86	58.73	66.08	75.51
Simplistically Scaled per Socket Power	1.00	1.05	1.34	1.18	1.21	1.24	1.16	1.18	1.21	1.31	1.35	1.52	1.28	1.31	1.36
Fully Scaled Relative per Socket Power	1.00	1.12	3.53	3.15	3.71	4.28	4.88	5.63	6.67	20.77	25.15	44.44	35.32	42.11	51.64
Simplistically Scaled Relative Rpeak/Watt	1.00	1.14	3.29	4.21	4.74	5.21	7.66	8.45	9.43	24.75	27.20	34.88	45.98	50.26	55.42
Fully Scaled Relative Rpeak/Watt	1.00	1.06	1.24	1.58	1.55	1.51	1.82	1.77	1.71	1.56	1.46	1.19	1.66	1.57	1.46
Simplistically Scaled Rpeak/Watt	0.04	0.05	0.13	0.17	0.19	0.21	0.31	0.34	0.38	1.00	1.10	1.41	1.86	2.04	2.25
Fully Scaled Rpeak/Watt	0.04	0.04	0.05	0.06	0.06	0.06	0.07	0.07	0.07	0.06	0.06	0.05	0.07	0.06	0.06
Board and Rack Level Concurrency Predictions															
Maximum Sockets per Board	4	4	4	4	8	8	8	8	8	16	16	16	16	16	16
Maximum Boards per Rack	24	24	24	24	32	32	32	32	32	32	32	32	32	32	32
Maximum Sockets per Rack	96	96	96	96	256	256	256	256	256	512	512	512	512	512	512
Maximum Cores per Board	8	10	16	20	51	64	81	102	128	323	407	513	646	814	1025
Maximum Cores per Rack	192	242	384	484	1628	2050	2584	3254	4101	10331	13018	16402	20667	26036	32804
Maximum Flops per cycle per Board	16	20	64	81	204	256	323	407	513	2583	3254	4101	5167	6509	8201
Maximum Flops per cycle per Rack	384	484	1536	1936	6513	8201	10336	13018	16402	82650	104142	131218	165336	208285	262436
Board and Rack Level Power Predictions															
Max Relative Power per Rack	1	1	1	2	2	2	4	4	4	8	8	8	16	16	16
Simplistic Power-Limited Sockets/Rack	96	92	72	96	158	155	256	256	256	512	512	507	512	512	512
Fully Scaled Power-Limited Sockets/Rack	96	86	27	61	52	45	79	68	58	37	31	17	43	36	30
Simplistically Scaled Relative Rpeak per Rack	96	109	316	478	911	1001	2274	2558	2924	16577	18838	26788	30072	33831	38664
Fully Scaled Relative Rpeak per Rack	96	102	119	304	298	291	699	681	658	1197	1123	914	2554	2410	2246
System Predictions: Power Unconstrained, Gradual Increase in Affordable Racks to Max of 600															
Max Affordable Racks per System	155	200	250	300	350	400	450	500	550	600	600	600	600	600	600
Max Cores per System	29760	48441	9.6E+04	1.5E+05	5.7E+05	8.2E+05	1.2E+06	1.6E+06	2.3E+06	6.2E+06	7.8E+06	9.8E+06	1.2E+07	1.6E+07	2.0E+07
Max Flops per cycle per System	59520	96882	3.8E+05	5.8E+05	2.3E+06	3.3E+06	4.7E+06	6.5E+06	9.0E+06	5.0E+07	6.2E+07	7.9E+07	9.9E+07	1.2E+08	1.6E+08
Simplistically Scaled System Rpeak (GF)	1.0E+05	1.5E+05	5.4E+05	9.8E+05	2.2E+06	2.7E+06	7.0E+06	8.7E+06	1.1E+07	6.8E+07	7.7E+07	1.1E+08	1.2E+08	1.4E+08	1.6E+08
Fully Scaled System Rpeak (GF)	1.0E+05	1.4E+05	2.0E+05	6.2E+05	7.1E+05	7.9E+05	2.1E+06	2.3E+06	2.5E+06	4.9E+06	4.6E+06	3.7E+06	1.0E+07	9.9E+06	9.2E+06
System Power (MW)	2.5	3.2	4.0	9.7	11.3	12.9	29.0	32.3	35.5	77.4	77.4	77.4	154.8	154.8	154.8
System Predictions: Power Constrained to 20 MW															
Maximum Racks	155	200	250	300	350	400	310	310	310	155	155	155	78	78	78
Simplistically Scaled System Rpeak (GF)	1.E+05	1.E+05	5.E+05	1.E+06	2.E+06	3.E+06	5.E+06	5.E+06	6.E+06	2.E+07	2.E+07	3.E+07	2.E+07	2.E+07	2.E+07
Fully Scaled System Rpeak (GF)	1.E+05	1.E+05	2.E+05	6.E+05	7.E+05	8.E+05	1.E+06	1.E+06	1.E+06	1.E+06	1.E+06	1.E+06	1.E+06	1.E+06	1.E+06

Figure 7.10: Heavy node strawman projections.

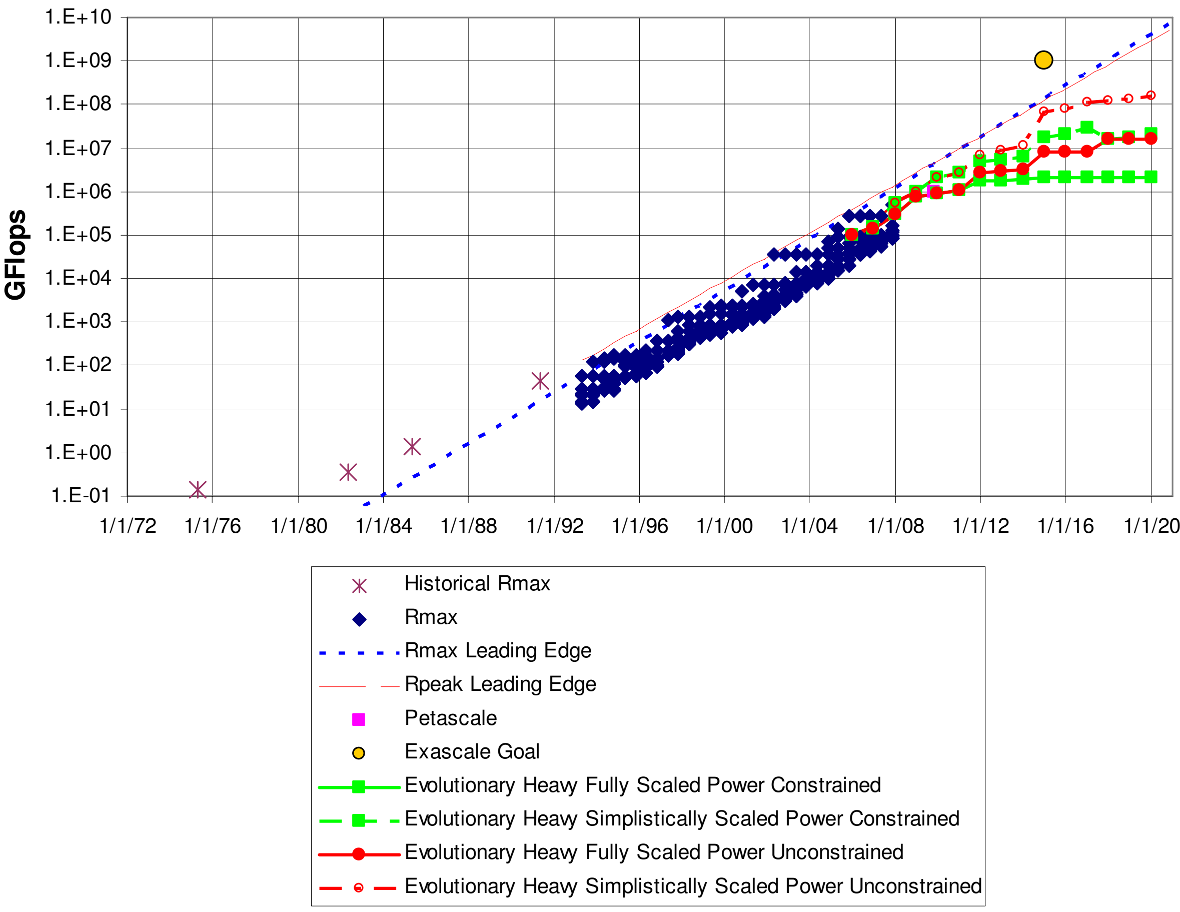


Figure 7.11: Heavy node performance projections.

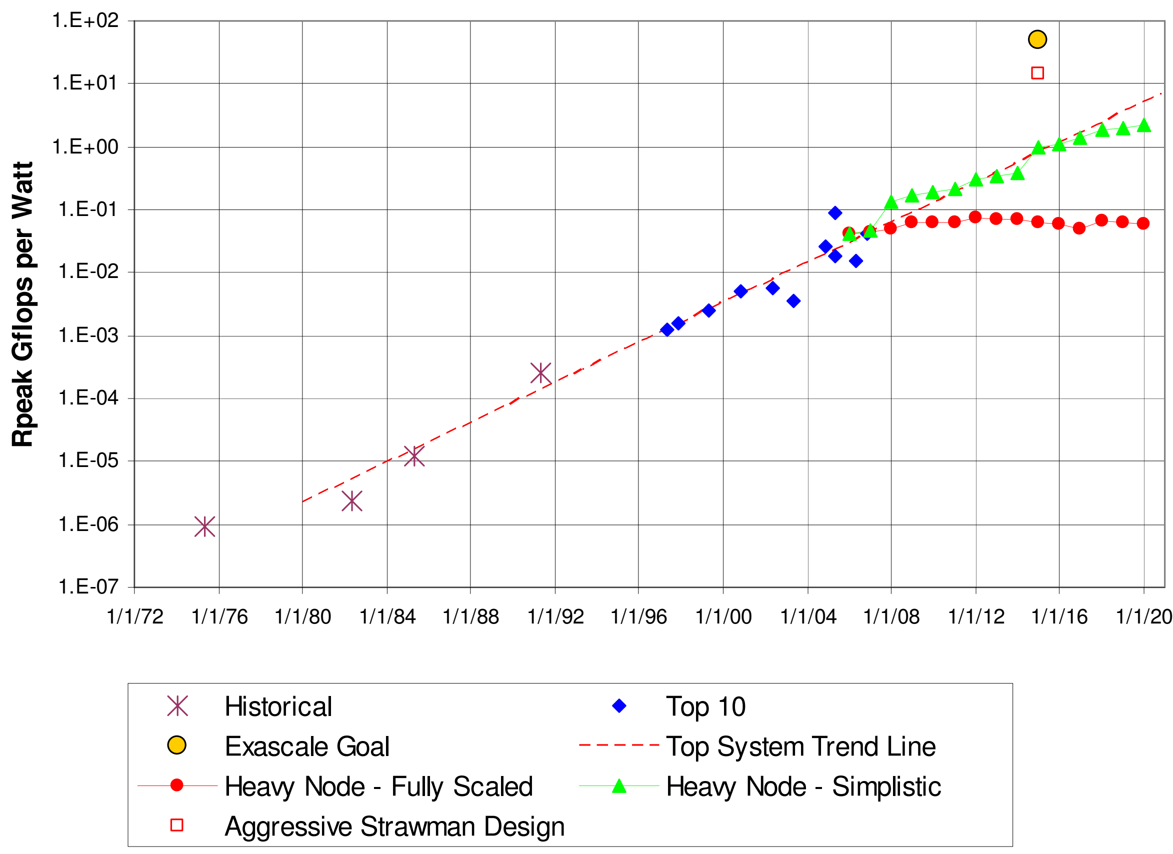


Figure 7.12: Heavy node GFlops per Watt.

Figure 7.10 summarizes some projections based on the above assumptions, all based on numbers relative to 2006, with a graphical representation of peak Linpack in Figure 7.11. In this figure, the lines marked "Power Constrained" represent a capping of power at the 20MW limit.

Figure 7.12 then also converts these numbers into gigaflops per watt, which is directly relevant to all three classes of Exascale systems.

Some specific observations include:

- Because this model used year-to-year numbers on each component, and real systems are not updated every year, there is some year-to-year "noise."
- The clock limitations due to the power limits are significant, and greatly hamper the potential performance growth per microprocessor chip.
- The "gigaflops per watt" graph indicates that whereas the "simplistic" model seems to follow a trend line, the "fully scaled" model becomes flat, largely because the power becomes dominated not by the microprocessors but by the transport of data - to memory and to the routers. Clearly this implies a need for better off-chip protocols in terms of energy per bit, regardless of the system architecture.
- None of the overall estimates come any closer than within an order of magnitude of an exaflops, with the power constrained models running between two and three orders of magnitude too low.
- In general there is a considerable spread between the "simplistic" predictions and the "fully scaled." Significant and more detailed projections would be needed to more accurately decide where in this range the real numbers might lie. However, both are so far off of an "exa" goal that it is clear that in any case there is a real problem.

7.2.2 Light Node Strawmen

The prior section addressed the possible evolution of systems based on leading edge high performance microprocessors where single thread performance is important. In contrast, this section projects what might be possible when "lighter weight" customized processors are used in an architecture that was designed from the ground up for dense packaging, massive replication, and explicit parallelism.

7.2.2.1 A Baseline

The basis for this extrapolation is the Blue Gene series of supercomputers, both the "/L"[48] and the "P"[147] series, with general characteristics summarized in Table 7.2. Here the basic unit of packaging is not a large board with multiple, high powered, chips needing large heat sinks, but small "DIMM-like" **compute cards** that include both memory and small, low-power, multi-core **compute chips**. The key to the high density possible by stacking a lot of such cards on a board is keeping the processing core power dissipation down to the point where only small heat sinks and simple cooling are needed.

Keeping the heat down is achieved by keeping the architecture of the individual cores simple and keeping the clock rate down. The latter has the side-effect of reducing the cycle level latency penalty of cache misses, meaning that simpler (and thus lower power) memory systems can be used. For the original Blue Gene/L core, the former is achieved by:

	Blue Gene/L[48]	Blue Gene/P[147]
Technology Node	130 nm	90 nm
FPUs/Core	2 fused mpy-add	2 fused mpy-add
Cores per Compute Chip	2	4
Clock Rate	700MHz	850MHz
Flops per Compute Chip	5.6 Gflops	13.6 Gflops
Shared L3 per Compute Chip	4 MB embedded	8 MB embedded
Compute Chips per Node	1	1
DRAM capacity per Node	0.5-1 GB	2 GB
DRAM Chips per Node	9-18	20-40
Nodes per Card	2	1
Cards per Board	16	32
Boards per Rack	32	32
Nodes per Rack	1024	1024
Cores per Rack	2048	4096
R_{peak} per Rack	5.73 Tflops	13.9 Tflops
R_{max} per Rack	4.71 Tflops	
Memory per rack	1 TB	2 TB
Memory per Flop	0.17B/Flop	0.14B/Flop
Power per Compute Chip	12.9 W	
Power per rack	27.6KW	40KW
Gflops per KW	212.4	348.16
Power per Pflops	4.7 MW	2.9 MW

Table 7.2: Light node baseline based on Blue Gene.

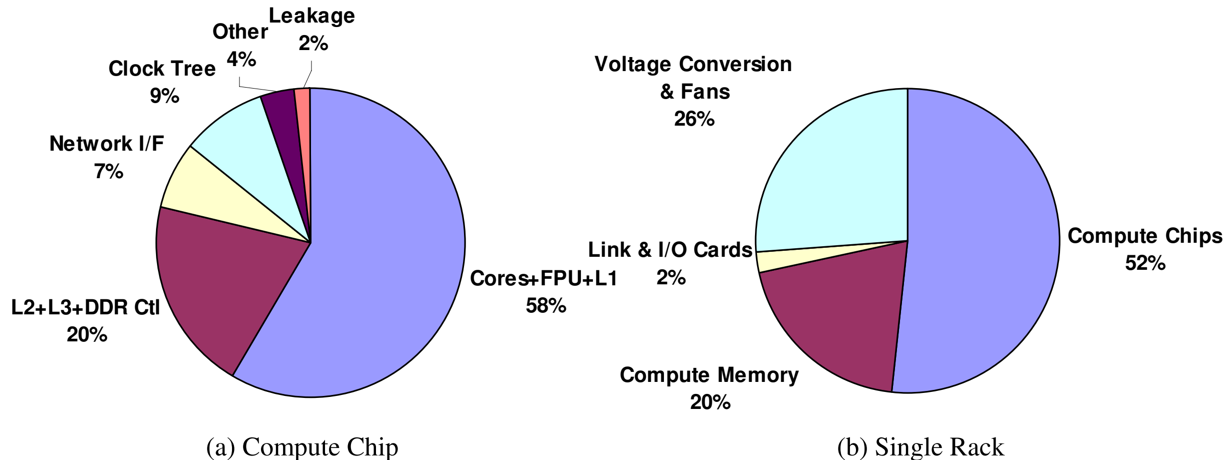


Figure 7.13: Power distribution in the light node strawman.

- a simple 7 stage pipeline,
- simple two instruction issue with 3 pipelines (load/store, integer, and other),
- a floating point unit with 2 fused multiply-adders and an extended ISA that supports simple SIMD operations at up to the equivalent of 4 flops per cycle,
- an integrated memory controller.

Several of these cores are implemented on each compute chip that also includes complete memory controllers and interface and routing functions. Thus, a complete single **compute node** consists of a compute chip and some small number of commodity DRAM chips.

One or more of these compute nodes are placed on a small circuit card akin in size to a DIMM card. Multiple such cards are then stacked on a larger board, which in turn plugs into a **midplane** holding many such boards, and several such midplanes packaged in a rack.

The current system interconnect topology is a 3D torus, with an 8x8x8 sub cube on each midplane and 4 link cards that glue the surfaces of this cube to other cubes on other midplanes. From an application perspective, this network supports a message-passing protocol, rather than a pGAS-like protocol as assumed for the heavy weight strawman of the previous section. A second interface supports a **collective network**, which is a tree-like topology that permits synchronization, barriers, and broadcasts.

In addition, on a per rack basis there are 32 compute cards and 2 I/O cards used for system access and I/O.

In terms of power, Figure 7.13 gives an approximate distribution of power by subsystem for both the compute chip and a single rack, using numbers from Blue Gene/L[20]. These numbers assume only 9 DRAM chips (512 MB) per compute node.

7.2.2.2 Scaling Assumptions

For consistency, the scaling assumptions used in Section 7.2.1.2 for the heavyweight strawman is modified only slightly:

- The die size for the compute chip will remain approximately constant, meaning that the number of cores on each chip may grow roughly as the transistor density grows, as pictured

in Figure 4.3. We do not account L3 cache sizes that grow more than linearly with core count to reduce pressure on off-chip contacts.

- V_{dd} for these microprocessors will flatten (Figure 4.7 and 6.2).
- The power dissipation per chip will be allowed to increase by 25% with every 4X gain in transistor density (this mirrors roughly the increase from Blue Gene/L to Blue Gene/P). Even at the end of the ITRS roadmap, this will still remain far under the per chip limits from the ITRS.
- On a per core basis, the microarchitecture will improve from a peak of 4 flops per cycle per core today to a peak of 8 flops per cycle in 2010.
- The system will want to maintain the same ratio of bytes of main memory to peak flops as the most recent Blue Gene/P (2 GB per 13.9 Gflops, or 0.14 to 1), and to do this we will use whatever natural increase in density comes about from commodity DRAM, but coupled with additional memory chips as necessary if that intrinsic growth is insufficient.
- The maximum number of nodes per board may double a few times. This is assumed possible because of a possible move to liquid cooling, for example, where more power can be dissipated, allowing the white space to be used, memory chips to be stacked (freeing up white space), and/or the size of the heat sinks to be reduced. For this projection we will assume this may happen at roughly five year intervals.
- The maximum number of boards per rack may increase by perhaps 33% again because of assumed improvements in physical packaging and cooling, and reduction in volume for support systems. For this projection we will assume this may happen once in 2010.
- The maximum power will be the same as projected for the heavy-weight strawman.
- The overhead for the rack for power and cooling will be the same percentage as in the Blue Gene/L.
- We ignore for now any secondary storage (or growth in that storage) for either scratch, file, or archival purposes, although that must also undergo significant increases.
- The rack overhead for I/O and power is the same percentage of compute power as the baseline, namely 33%.

For this sizing, we will ignore what all this means in terms of system topology.

7.2.2.3 Power Models

We assume the same two power models as for the heavy weight strawman (Section 7.2.1.3). The **Simplistic Power Scaled Model** assumes that the number of cores grows in proportion to the technology density increase, the power per core changes as the ITRS roadmap has predicted, and that the power for the memory associated with each node grows only linearly with the number of memory chips needed (i.e. the power per memory chip is "constant"). We also assume that the clock rate of the cores may change as governed by the maximum power dissipation allowed per compute chip. Finally, we assume that the power associated with compute chip memory and network interfaces remains constant, i.e. the energy per bit transferred or accessed improves at the same rate as the bandwidth needs of the on-chip cores increase.

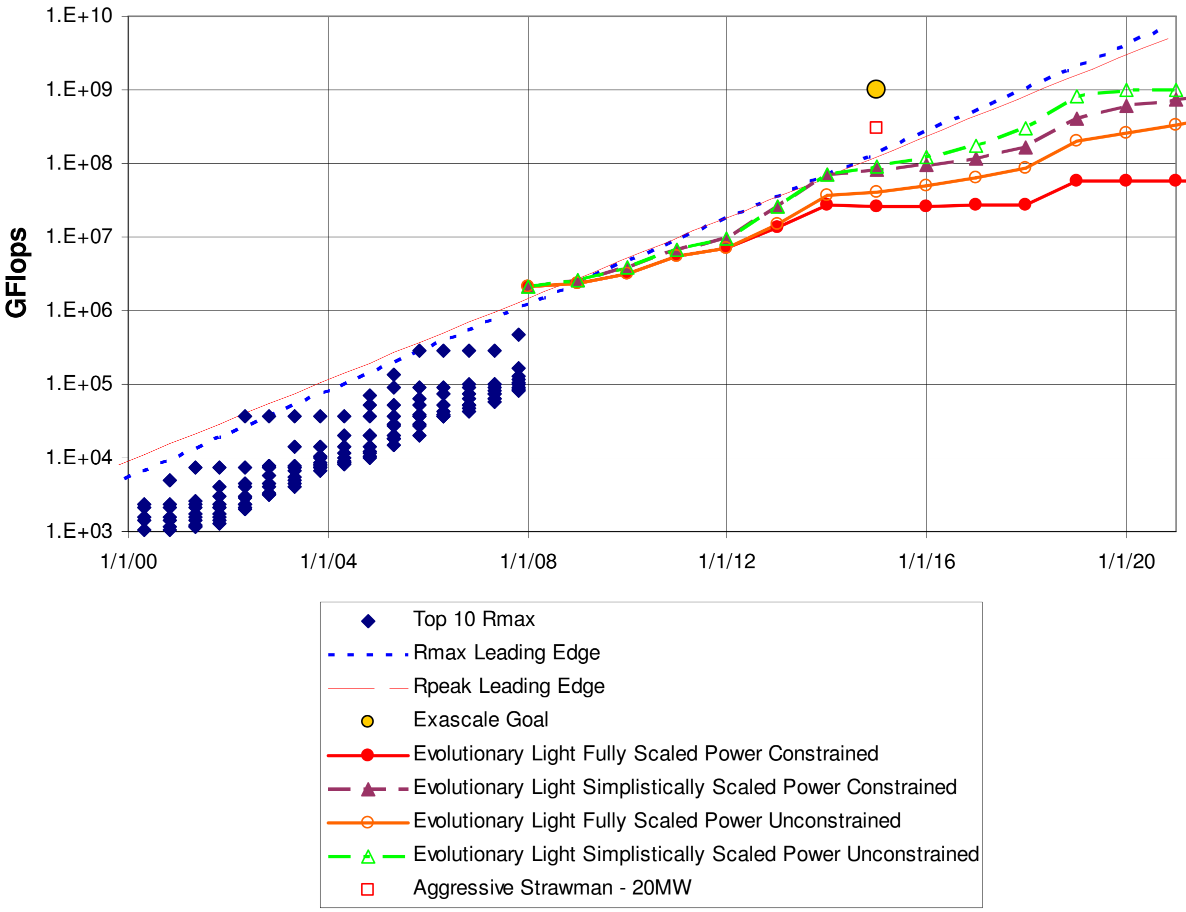


Figure 7.14: Light node strawman performance projections.

In contrast, for the **Fully Scaled Power Model** we assume the core power grows as above, but that both the memory and network interface power scales linearly with the peak flops potential of the multi-core microprocessors. This naively assumes that the total energy expended in these two subsystems is used in moving data, and that the energy to move one bit (at whatever the rate) is constant through time (i.e. no improvement in I/O energy protocol). This is clearly an over-estimate.

As with the heavyweight strawman, neither model takes into account the effect of only a finite number of signal I/Os available from a microprocessor die, and the power effects of trying to run the available pins at a rate consistent with the I/O demands of the multiple cores on the die.

7.2.2.4 Projections

Figure 7.14 presents the results from the extrapolation in the same format as Figure 7.11. One indication of the potential validity of these projections is that even though they were based on the Blue Gene/L chip, for which a great deal of data is available, the first projections seem to line up well with what is known about the Blue Gene/P chip and system.

As can be seen, this approach does get closer to the Exaflops goal, but not until after 2020. It also, however, has a narrower window between the optimistic and pessimistic power models.

The performance per watt, Figure 7.15, is within one to two orders of magnitude by 2015, a

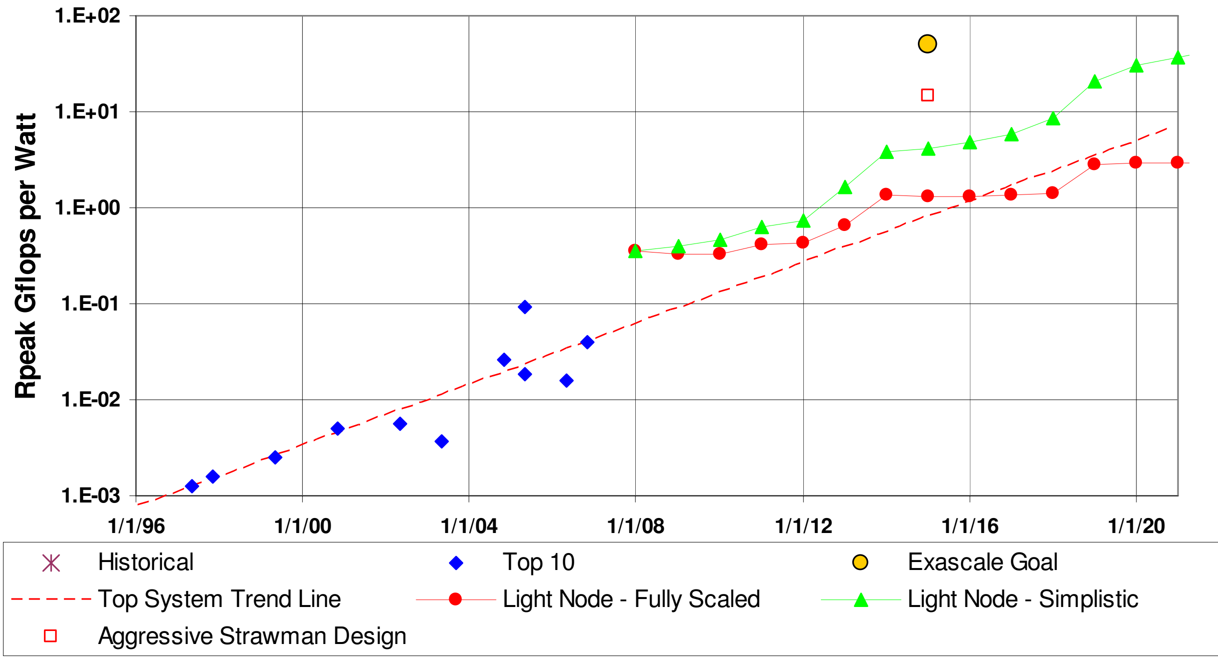


Figure 7.15: Light node strawman Gflops per watt.

substantial improvement over the heavy node strawman.

As much of an improvement this is, it still does not reach the Exascale goals. Looking closer into the details of the power model, a key observation that jumps out is that the power consumed in off-chip communication, and in memory references, is still a major contributor, and needs to be worked seriously.

7.3 Aggressive Silicon System Strawman

In this section we develop a strawman architecture for an Exascale computer system based on silicon, but with a clean sheet of paper and an aggressive but balanced approach to design options. While the design target is a data-center class system, the design will be done in a way that yields significant insight (discussed at the end) into what is feasible for both the departmental and embedded classes. The exercise indicates the scale of such a machine and exposes a number of design, programming, and technology challenges.

Our development proceeds in a bottom-up manner, starting with the floating-point units (**FPUs**) and working upward until we reach the level of a complete system.

Table 7.3 summarizes our bottom-up composition of a system. We assume a 2013 technology node of 32 nm as a baseline for the projection - this represents a reasonable technology out of which a 2015 machine might be fabricated. We start with an FPU (along with its register files and amortized instruction memory). Four FPUs along with instruction fetch and decode logic and an L1 data memory forms a **Core**. We combine 742 such cores on a 4.5Tflops, 150W active power (215W total) **processor chip**. This chip along with 16 DRAMs forms a **Node** with 16GB of memory capacity. The final three groupings correspond to the three levels of our interconnection network. 12 nodes plus routing support forms a **Group**, 32 Groups are packaged in a **rack**, and 583 **racks** are required to achieve a peak of 1 exaflops.

Level	What	Perf	Power	RAM
FPU	FPU, regs,. Instruction-memory	1.5 Gflops	30mW	
Core	4FPUs, L1	6 Gflops	141mW	
Processor Chip	742 Cores, L2/L3, Interconnect	4.5 Tflops	214W	
Node	Processor Chip, DRAM	4.5Tflops	230W	16GB
Group	12 Processor Chips, routers	54Tflops	3.5KW	192GB
rack	32 Groups	1.7 Pflops	116KW	6.1 TB
System	583 racks	1 Eflops	67.7MW	3.6PB

Table 7.3: Summary characteristics of aggressively designed strawman architecture.

Year	Tech (nm)	V	Area (mm^2)	E/Op (pJ)	f (GHz)	Watts/Exaflops	Watts/FPU
2004	90	1.10	0.50	100	1.00	1.0E+08	0.10
2007	65	1.10	0.26	72	1.38	7.2E+07	0.10
2010	45	1.00	0.13	45	2.00	4.5E+07	0.09
2013	32	0.90	0.06	29	2.81	2.9E+07	0.08
2016	22	0.80	0.03	18	4.09	1.8E+07	0.07
2019	16	0.70	0.02	11	5.63	1.1E+07	0.06

Table 7.4: Expected area, power, and performance of FPUs with technology scaling.

Figure 7.16 diagrams the structure of one such group.

The strawman system is balanced by cost and our best estimate of requirements. The ratios of floating point performance to memory capacity in Table 7.3 are far from the conventional 1 byte per flops, or even from the "traditional" ratios found in the historical ratios from the Top 500 (Section 4.5.4) or in the strawmen of Sections 7.2.1 and 7.2.2. However it is roughly cost balanced, and with globally addressable memory across the system, should be adequate to hold at least some Exascale problems. It is, however, about 10X what 2008-2010 Petascale machines will have, implying that it is sufficient capacity for at least classes II and III of Section 5.6.1.

Similarly the bandwidth ratios of Table 7.7 do not provide a word of global bandwidth per flops, but rather budgets power across the levels of the storage hierarchy in a manner that returns the most performance per Watt. This is how implementable Exascale systems must be balanced.

In contrast to this fixed allocation, a system in which power allocation is adaptively balanced across levels of the bandwidth hierarchy is considered in Section 7.3.7.

7.3.1 FPUs

If we define an Exascale system as one capable of performing 10^{18} 64-bit floating-point operations, then at a minimum, our system needs a number of FPUs capable of this aggregate arithmetic rate. Table 7.4 shows how the area and power of FPUs is expected to scale with semiconductor technology. The first three columns of the table reflect the technology nodes projected by the International Technology Roadmap for Semiconductors (ITRS)[13]. The next three columns show how the area, energy per operation, and performance of a single floating point unit scale as line width (column 2) and voltage (column 3) are reduced. A **floating point operation** (**flop**) here is defined as a 64-bit floating-point fused multiply-add. The final two columns give the power to achieve an Exaflops (10^{18} 64-bit floating-point operations) with just FPUs (i.e. no overhead, memory, or transport), and the resulting power per FPU.

The baseline area, energy, and frequency for 90nm is taken from [42] and represents the power-

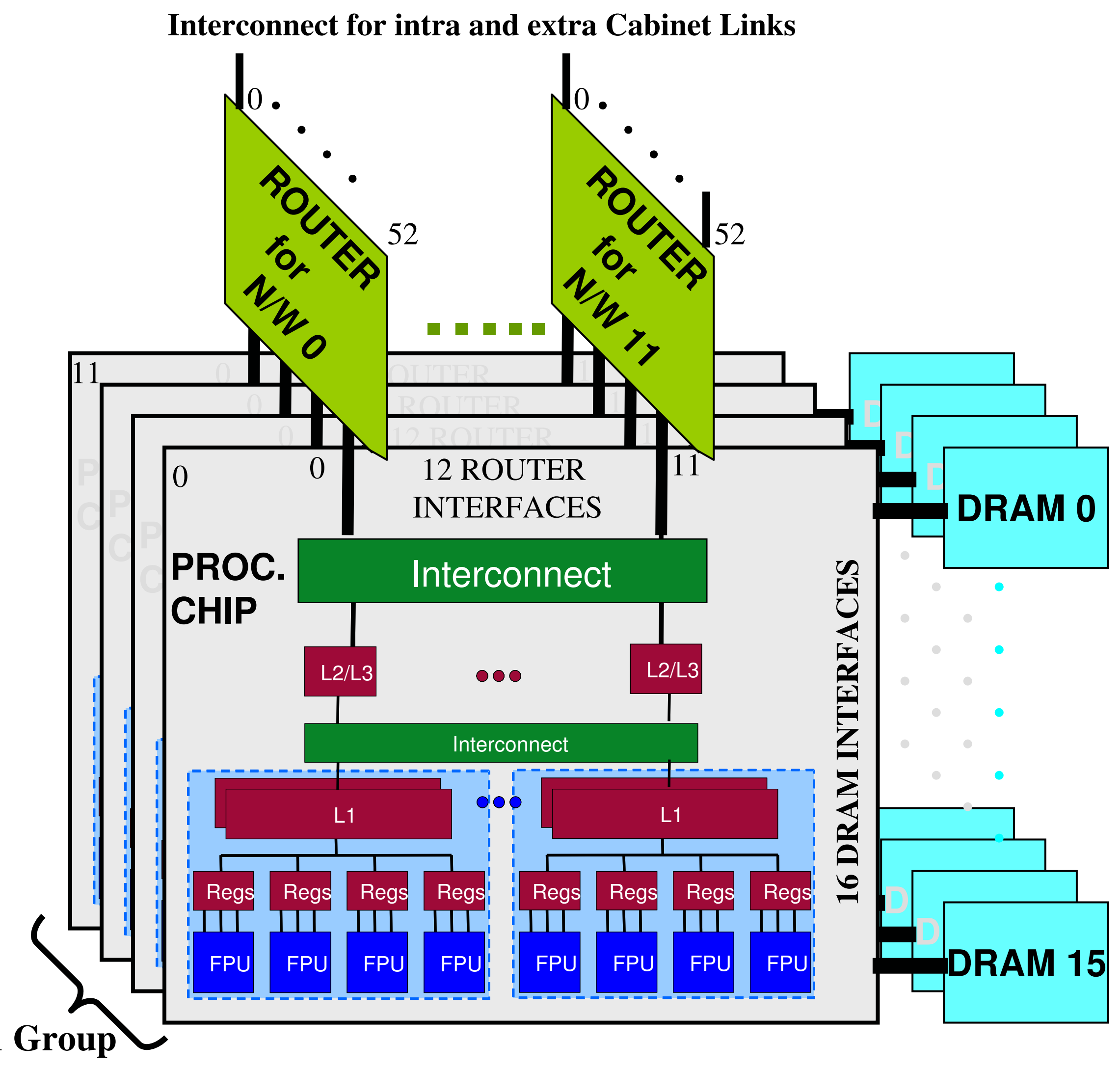


Figure 7.16: Aggressive strawman architecture.

Year	Tech (nm)	V	Area (mm^2	E/Op (pJ)	f (GHz)	W/Exaflops	W/FPU
2004	90	0.8	0.50	52.9	0.6	5.3E+07	0.03
2007	65	0.8	0.26	38.2	0.9	3.8E+07	0.03
2010	45	0.8	0.26	38.2	0.9	3.8E+07	0.03
2013	32	0.6	0.06	10.6	1.5	1.1E+07	0.02
2016	22	0.5	0.03	5.1	1.9	5.1E+06	0.01
2019	16	0.5	0.02	3.7	3.1	3.7E+06	0.01

Table 7.5: Expected area, power, and performance of FPUs with more aggressive voltage scaling.

Unit	Energy per op (pJ)	Comment
FPU	10.6	Just the arithmetic operation
Register	5.5	Two reads, one write, 128 regs
Instruction RAM	3.6	32KB each access amortized across 4 FPUs
L1 Data RAM	3.6	64KB each access amortized across 4 FPUs
Core (per FPU)	23.3	Total core energy per FPU per op
Core (Total)	93.4	

Table 7.6: Energy breakdown for a four FPU processor core.

optimized FPU designed for the Merrimac Supercomputer. FPUs that are optimized for reduced latency or for higher operating frequencies may dissipate considerably more energy per operation. Energy scaling with process improvement is computed by assuming that energy per operation (CV^2) decreases linearly with line width (C) and quadratically with voltage (V^2). Maximum operating frequency is assumed to increase linearly as line width decreases.

The table shows that just the floating point units required to reach an Exaflops of performance will require 29MW in the 32nm technology node that will be available for a 2015 machine. Because this power number is prohibitively high, we also consider scaling supply voltage in a more aggressive manner than projected by the ITRS roadmap. The area and power of FPUs under this more aggressive voltage scaling is shown in Table 7.5. As above, energy is assumed to scale quadratically with voltage. Operating frequency is assumed to scale linearly with overdrive (V_{DD} - V_T) where we assume the threshold voltage V_T is 250mV.

Power lost to static leakage is not considered here but will be in the final numbers.

Table 7.5 shows that by aggressive voltage scaling we are able to significantly reduce the energy per operation (by nearly a factor of three at the 32nm node) at the expense of operating frequency. With this approach, the power required by the FPUs to achieve an Exaflops at the 32nm node is reduced to 11MW - just over 50% of the overall desired system power target.

Table 7.5 represents a fairly straightforward, if aggressive, voltage scaling of existing FPU designs. Further power savings may be possible by innovations in circuit design. Also, supply and threshold voltage can be optimized further — leading to even greater power savings at the expense of performance.

7.3.2 Single Processor Core

As a first step in building up to a processor core, we consider combining our FPU from the previous section with a set of local memory structures to keep it supplied with data, as summarized in Table 7.6. As we shall see, aside from the FPUs, the bulk of energy in our strawman machine will be consumed by data and instruction supply. At the 2013 32nm node, we estimate that a three-port

register file with 128 registers consumes 1.8pJ for each access and thus 5.5pJ for the three accesses (2 read and 1 write) required to support each floating-point operation. (Note a fused multiply-add actually requires a third read for a total of 7.3pJ). Adding in the FPU energy from Table 7.4 gives a total energy of 16.1pJ per operation for arithmetic and the first-level data supply.

Instruction supply and L1 data supply are dominated by memory energy. Reading a word from a 32KB memory array in this 32nm process requires about 15pJ. Hence, if we read an instruction from an I-cache for each operation, we would nearly double operation energy to 31.1pJ. However, by employing some degree of SIMD and/or VLIW control, we can amortize the cost of instruction supply across several FPUs. For our aggressive strawman design, we assume each instruction is amortized across four FPUs for a cost of 3.6pJ per FPU.

Similarly we assume that a single L1 Data Memory is shared by four FPUs for an energy cost of 3.6pJ per FPU. The energy of an L1 access is that to access a single 32KB L1 bank. However, the L1 memory may be composed of many banks to increase capacity. This can be done without increasing energy as long as only one bank is accessed at a time. For our aggressive strawman design, we assume that the L1 data memory is 2 32KB banks for a total of 64KB. For the purpose of the strawman we do not specify whether the L1 data memory is managed explicitly (like a scratch-pad memory that is simply directly addressed) or implicitly (like a cache which would require additional energy for tag access and comparisons). Most effective organizations will allow a hybrid of both approaches.

The energy breakdown of a 4-wide SIMD or VLIW processor core with registers, instruction RAM, and data RAM as described above is shown in Table 7.6. Note that even with sharing of the instruction and data memories across four FPUs the minimal functionality of a processor core more than doubles the energy required per operation to 23.3pJ. This raises the total power needed for an Exaflops machine to 23MW - again just considering the cores and their L1, and nothing above that.

This estimate has assumed an extremely power-efficient architecture. With a more conventional approach the energy per operation could be many times larger. Fetching one instruction per operation, employing complex control, permitting out-of-order issue, adding tag logic and storage, or increasing L1 data bandwidth, for example, could each increase the energy significantly. However, with careful design, the number suggested here should be approachable.

7.3.3 On-Chip Accesses

At the next level of our strawman architecture, we consider combining a number of our simple 4-way cores on a single chip. The number of cores we put on a single chip is limited by power, not area. In this power-limited regime the architecture of our multi-core chip is directly determined by its power budget. We assume a limit of 150W for the active logic on chip and consider the budget shown in Table 7.7. Given a power allocation of 70% (105W) for the cores (59% FPUs + 11% L1) the chip can support 742 4-way cores (2968 FPUs) for a peak aggregate performance of 4.5 teraflops. As shown in the table, the amount of power allocated to each level of the storage hierarchy determines the bandwidth (BW in GWords/s) at that level. The column labeled "Taper" is the number of FPU operations per single word access at each level. The numbers in the DRAM and Network rows reflect only the energy consumed in the multi-core processor die. Accesses at these levels consume additional energy in memory chips, router chips, and transceivers.

Note that to improve the yield of these ∼3K FPU chips some number of additional cores and memory arrays would be fabricated on the chip as spares that can be substituted for bad cores and memory arrays during the test process via configuration. Alternatively one can simply configure all "good" cores on each chip for use and allow the number of cores per chip to vary from chip to

Item	Percent	Watts	Units	BW (GW/s)	Taper	Comments
FPUs	59.0%	88.5	2968	4495	1	Includes 3-port reg and I-mem
L1 Data	10.9%	16.4	742	1124	4	64KB per 4 FPUs
L2	6.9%	10.4	371	562	8	256KB per 2 L1s
L3	7.5%	11.3	189	286	16	Global access to L2s
DRAM	10.0%	15.0	59	89	50	Attached to this chip
Network	5.6%	8.4	13	27	164	Global access
Taper = number of flops executed per access						

Table 7.7: Power budget for strawman multi-core chip.

chip.

At some distance, the energy required to access on-chip memory becomes dominated by the energy required to communicate to and from the memory. The distance at which this occurs depends on the signaling technology used. At the 32nm process node we estimate on-chip line capacitance at 300fF/mm. With an 0.6V supply and full-swing signaling this gives signaling energy of 110fJ/bit-mm, or 6.9pJ/word-mm for a 64-bit word. With a signaling protocol that uses a reduced signal swing of 0.1V, these numbers can be reduced to 18fJ/bit-mm and 1.2pJ/word-mm. With array access energy of 14.6pJ, the point at which access via full-swing signaling becomes dominated by signaling energy is at a distance of 2.1mm. With more efficient 0.1V signaling, this crossover distance is increased to 12.7mm. For the rest of this analysis we assume the use of the more efficient signaling system with 0.1V signal swings.

For our strawman design, as pictured in Figure 7.16 global on-chip memory consists of a 256KB RAM (composed of 32KB subarrays) for every two 4-way cores, for a total of 371 arrays for 92.8MB on-chip storage (again we ignore the distinction between cache and directly addressable memory). These arrays are connected to the processor cores using an on-chip interconnection network that uses efficient signaling. To account for locality we divide all of this on-chip memory into “L2” and ”L3” levels. Both of these levels share the same on-chip memory arrays. L2 accesses reference memory arrays associated with a local group of 16 4-way cores. L3 accesses are to any location on the chip. Each L2 access requires traversing an average of 3.4mm of network channels while an L3 access requires an average of distance of 21.3mm. Note that the memory arrays can support much higher bandwidth than is provisioned at all three levels. However the power budget (and possibly the on-chip network bandwidth) limits the bandwidth to the level shown in the table. The numbers in the table consider only the energy needed to transport payload data over the on-chip interconnect. This is slightly optimistic; control overhead will add to the energy at these levels - the amount of overhead depends on the granularity of transfers. Our data-only estimates are accurate enough for the purposes of this strawman.

The DRAM and Network rows of Table 7.7 assume an energy of 2pJ/bit to cross between two chips. The DRAM number includes one global communication on chip (21.3mm) and one chip crossing. The network number includes one global on-chip communication and two chip crossings - one to get to the router, and a second to get to the DRAM on the destination chip. Additional energy consumed in the DRAM chips and the network are discussed below.

Table 7.8 shows the area breakdown for the multi-core processor chip. Power limits FPUs (which includes register files and instruction memories) to occupy less than half the chip. The capacity of the L1-L3 arrays is then adjusted to fill the remaining area on the die. Note that the capacity of the memory arrays sets their area while the bandwidth provided at each level sets their power allowing these two parameters to be adjusted independently.

Item	Units	Each (mm^2)	Total (mm^2)
FPUs	2968	0.06	188
L1	742	0.09	66
L2/L3	371	0.36	132
			386

Table 7.8: Area breakdown of processor chip.

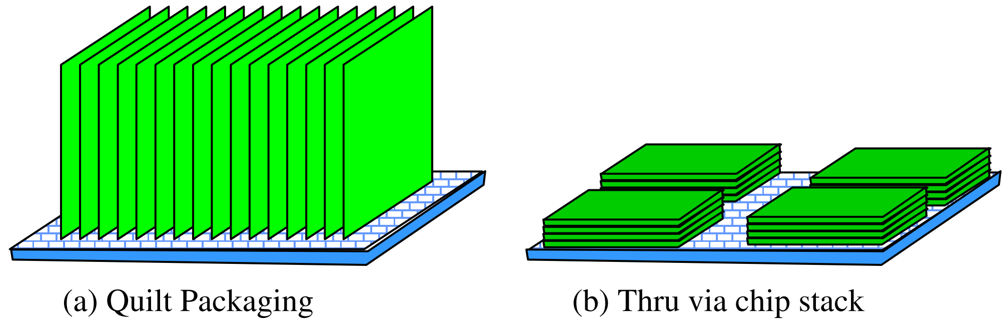


Figure 7.17: Possible aggressive strawman packaging of a single node.

Finally, as discussed in Section 6.2.2.1, leakage currents will still be significant, and account for roughly 30% of total chip power. This would raise the actual processor chip power to about 215W.

7.3.4 Processing Node

As pictured in Figure 7.16, a **processing node** consists of the multi-core processor chip described above and 16 1GB DRAM chips for an aggregate memory of 16GB/node. Each DRAM chip is assumed to have a 32-bit wide interface that operates at 11Gb/s per signal for a per-chip bandwidth of 5.5GWords/s. The 16 chips provide an aggregate bandwidth of 88GWords/s - slightly less than the 89GWords/s specified in Table 7.7. This DRAM interface consumes 512 differential signals (32 per DRAM chip) on the processor chip for the data path. We assume the control/address path consumes an additional 128 differential signals (8 dedicated to each DRAM), for a total of 640 signals.

Figure 7.17 diagrams two ways in which such a node may be packaged so that chip to chip interconnect is as direct as possible.

A contemporary commodity DRAM chip has an access energy of about 60pJ/bit. However, analysis suggests that by 2013 a DRAM can be built that can access an internal array with an energy of 0.5pJ/bit, transport a bit from the sense amp to the interface circuitry with an energy of 0.5pJ/bit, and drive a bit off chip with an energy of 2pJ/bit for a total access energy of 3pJ/bit. We use this more aggressive DRAM energy estimate here to assume design of a specialized DRAM optimized for high end systems. The internal access energy includes charging a 100fF bit line to the full power supply (0.1pJ) and 5× overhead. The transport includes driving 12mm of wire through a low swing and associated drivers, receivers, and repeaters.

A total energy of 3pJ/bit (or 192pJ/word) and a frequency of 5.5GW/s gives a power of 1.1W per chip or 17W for the aggregate node memory. This brings the total node power to 230W. A breakdown of how this power is distributed is shown in Figure 7.18.

The node memory could be stacked horizontally or vertically on the multi-core processor die.

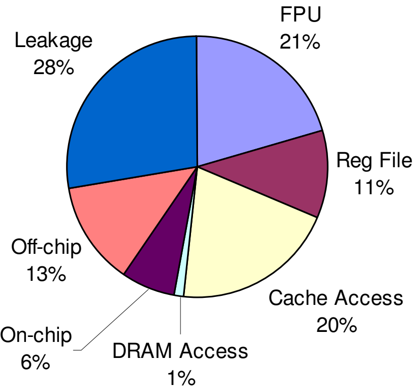


Figure 7.18: Power distribution within a node.

However this would only be useful to the extent that it reduces the energy per unit bandwidth for accessing the DRAM or the cost of providing the high pin count otherwise required. The numbers described in this strawman can be realized using existing signaling technology packaged on conventional printed circuit boards.

By conventional measures having 16GB of memory associated with a 4 teraflops processor seems small. However, this ratio is roughly balanced by cost and by interface bandwidth. Modern DRAMs are connected point-to-point and creating larger memory systems by aggregating more memory chips requires interface chips to terminate the signal pins on these memories. In this strawman design, the processor chips act as these interface chips, but perform arithmetic operations as well.

Also, within this estimate we do not assume any leakage power on the DRAMs - in actuality there may be some due to the enhanced logic circuits assumed, but they would be dwarfed by the processor chip.

Note that an Exaflops system requires 223,000 processing nodes and provides 3.4PB total memory.

7.3.5 Rack and System

Like the processor die, the contents of a rack are power limited. We assume a single rack can support 120kW. The amortized power of interconnection network components is 62.7W/node, bringing the total node power to 295W/node. Hence we can support up to 406 processing nodes in one rack. To make the network a bit simpler we will use $12 \times 32 = 384$ nodes (283,444 cores and 1,133,776 FPUs) per rack. The aggregate floating point performance of the rack is 1.7 petaflops. A total of 583 racks is required to provide an exaflops of performance.

7.3.5.1 System Interconnect Topology

A simple interconnection network, based on a **dragonfly topology** is used to connect the processors in a rack. This network is seamlessly connected to the on-chip network and provides a global memory address space across the entire system. To provide the aggregate global node bandwidth of 20GWords/s (1260Gb/s) in Table 7.7 we provide each processor chip with 12 full duplex 4-bit wide **channels** that operate with a 30Gb/s signaling rate. Each of these 12 channels connects to a separate parallel network and traffic originating at the node is load balanced across them. Global bandwidth can be smoothly varied by provisioning fewer than all 12 parallel networks. Figure 7.19 diagrams these connections.

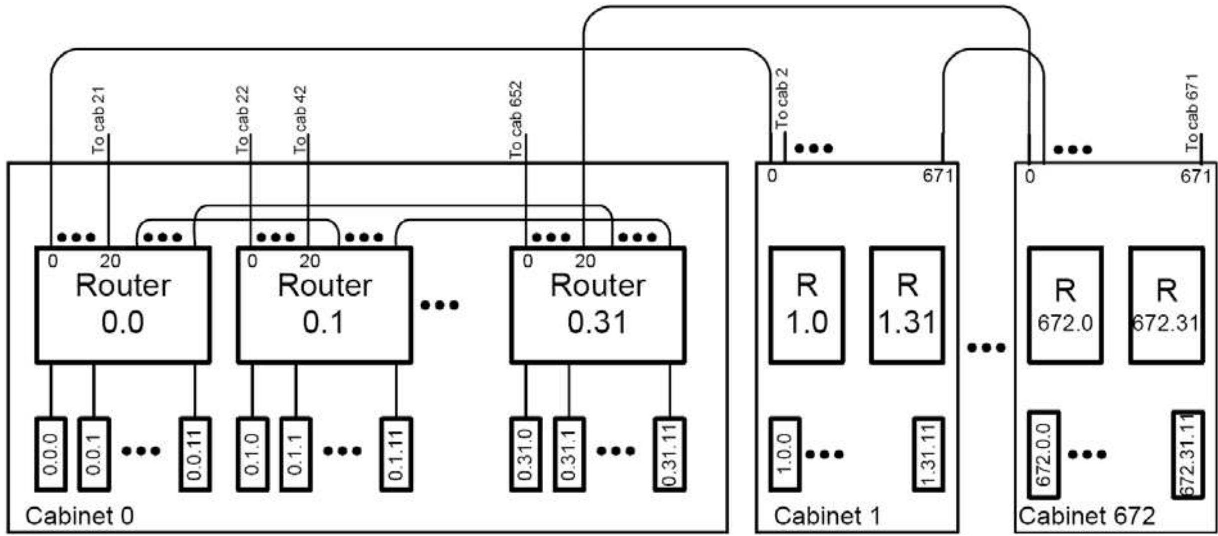


Figure 7.19: The top level of a dragonfly system interconnect.

Each of the 12 parallel networks is constructed by dividing the 384 nodes in each rack into 32 **groups** of 12 processors each. A radix-64 **router chip** is associated with each of the groups giving 32 routers for each of the 12 parallel networks, or a total of 384 router chips (the same number as the processor chips). The 61 channels on each router chip are divided into 12 **processor channels** (one for each processor in the group), 31 **local channels** (one for each other group), and 21 **global channels** (each of which goes to a different rack). Each group's router connects to a different set of 21 racks so that the 21 x 32 = 672 global channels out of each rack reach 672 different racks enabling machines of up to 673 racks. For smaller configurations multiple connections are made to the same distant rack in a modulo manner.

The network can be routed minimally or non-minimally. For a **minimal route** to a distant rack, a packet traverses between two and four routers:

1. It is forwarded to any of the 12 routers connected to its source processor (the source router). This picks one of the 12 parallel networks.

2. It is forwarded over a local link to a router within the current parallel network with a connection to the destination rack, unless it is already at that router or is already in the correct rack. This selects the group within the source rack attached to the correct destination rack.

3. It is forwarded over a global link to the destination rack unless it is already at the correct rack. This selects the destination rack.

4. It is forwarded over a local link to a router connected to the destination processor unless it is already there. This selects the destination group.

5. Finally, the packet is forwarded to the destination processor for further forwarding over the on-chip interconnection network within this processor chip.

Note that each minimal packet traverses two local links and one global link. Hence with our ratio of 12:32:21 processor:local:global links the processor links will be the bandwidth bottleneck as is desired to deliver the specified node bandwidth.

Like all butterfly derivatives, the dragonfly can become unbalanced using minimal routing on adversarial traffic patterns. In such cases **non-minimal routing** must be employed - for example by using global adaptive load-balanced routing. This is accomplished by forwarding the packet to a different intermediate rack (randomly or adaptively selected) before forwarding it to the destination. In this case, a route traverses up to three local and two global channels. For such non-minimal routes the global channels become the bottleneck. However the network is very close to balanced.

7.3.5.2 Router Chips

Our radix-64 router chip has 64 full duplex channels, with each path consisting of a 4-signal high-bandwidth differential link, all operating at 30Gb/s. For purposes of estimating energy, we assume that transmitting each bit across a router takes 4pJ - 2pJ for the energy to drive the outgoing link, and 2pJ to traverse the router internals. This gives an overall router active power of 30.7W. Applying the leakage tax discussed earlier to the on-chip portion raises this to about 37.5W. We also assume that the 21 global channels attached to each router require an additional 10pJ/bit for external transceivers or buffers, so the 2520Gb/s of global channel bandwidth out of each router requires an additional 25.2W giving a total interconnect power of 62.7W per node. Note that global bandwidth can be varied independently of local bandwidth by omitting global channels from some of the parallel networks. This allows a system to be configured with a desired amount of local bandwidth - by provisioning N (between 1 and 12) parallel networks, and separately setting the amount of global bandwidth by provisioning G (between 1 and N) of these networks with global channels.

7.3.5.3 Packaging within a rack

While there are many options as to how to package all the logic that goes into one rack, we assume here that each group of 12 processor chips, their 192 DRAM chips, and the associated 12 router chips are all packaged on a single board, with edge connections for both the local and global channels. There are up to 52x12=624 such channels that must be connected to each such board. 32 of these boards make up a rack, along with the disks, power supplies, and cooling subsystems.

We may also assume that each of these 32 boards plug into a single backpanel, eliminating the need for cabling for the channels connecting groups within a rack. This accounts for all but 21 of the channels of each of the 12 routers per group. The remaining 12x21=252 channels per board (8064 overall) are distributed to the other racks as pictured in Figure 7.19. Since there are 12 parallel networks, it makes sense to assume that the 12 channels that interconnect each rack are "bundled" into a single cable, meaning at least 582 cables leave each rack - one to each of the other racks.

7.3.6 Secondary Storage

To provide checkpoint storage, scratch space, and archival file storage up to 16 disk drives are associated with each group of 12 processors. These drives are located in the rack and attached via high-speed serial channels (the evolution of SATA) to the routers in the group. They are accessible via the network from any processor in the system. Using projections for 2014 disk drives (Section 6.4.1), the 16 drives will provide an aggregate of 192TB of storage, 64GB/s of bandwidth, and dissipate about 150W. The 64GB/s of bandwidth is sufficient to checkpoint the 192GB of DRAM storage in the group in 3 seconds, if they can all be run in parallel at full bandwidth.

Assuming a node MTBF of 10^6 hours we have a system MTBF of about 3 hours. With a checkpoint time of 3 seconds (8.3×10^{-4} hours), a checkpointing interval of about 3 minutes is near

Item	Percentage	Watts	Units	Bandwidth	Taper
FPUs	84.3%	126.5	4240	6421	1
L1	62.5%	93.7	4240	6421	1
L2	79.1%	118.6	4240	6421	1
L3	99.7%	149.6	2523	3821	1.7
DRAM	100.0%	150.0	592	897	7
Network	100.0%	150.0	234	354	18

Table 7.9: Power allocation for adaptive node.

optimal which gives a checkpoint overhead of about 1.7% (see Section 6.7.4).

For scratch and archival storage, the disks provide an aggregate of 6.1PB of storage for each rack and 3.6EB of storage in an Exaflops system. Larger amounts of storage if needed can be provided external to the main system. Given that the configuration supports a total of 3.4PB of DRAM memory, this 3.6EB of disk storage should be more than sufficient for both checkpointing, and scratch usage, even with RAID overheads for redundancy. In fact, it may very well offer at least rudimentary archival support.

An intermediate level of non-volatile memory - with latency, bandwidth, and cost/bit between DRAM and disk would be valuable to reduce the latency of the checkpoint operation and speed access to frequently accessed files. For example, suppose a technology existed for memory chips with 16GB density (16x the density of the 1GB DRAMs expected in 2014) and a

bandwidth of 5.5GB/s (1/8 the bandwidth of the DRAM chips). An array of 8 of these chips associated with each processing node would provide ample storage for checkpointing and would provide an aggregate bandwidth of 44GB/s - reducing the time to take a checkpoint to 0.36s which is nearly an order of magnitude faster than checkpointing with disks.

7.3.7 An Adaptively Balanced Node

In Sections 7.3.2-7.3.5 we architected a strawman system with a fixed allocation of power to the various levels of the hierarchy. In this section, we revisit this design and sketch a system in which *each* level of the hierarchy is provisioned so that it can consume *all* of the power allocation by itself. The result is a node organization with higher bandwidth at each level of the hierarchy than shown in Table 7.7. To prevent a chip from oversubscribing the total power limit, a throttling mechanism is used to monitor overall power use (e.g. by counting the number of accesses at each level of the hierarchy) and throttle instruction issue or other performance mechanisms when the short-term average power exceeds a threshold.

With this *power-adaptive* organization all levels of the hierarchy cannot operate a maximum capacity simultaneously — or the overall power limit would be exceeded by many fold. However, this arrangement enables each node to *adapt* its power use to the application at hand. An application segment that demands very high DRAM bandwidth and relatively little floating-point can use all of its power on DRAM bandwidth. A different application segment that can operate entirely out of registers may use all of its available power on floating point, with little for memory or communication. Together, this may allow a single design to match more readily to a wider class of applications, even though for any one class some part of the machine's resources are "under-utilized."

Table 7.9 shows the power allocation for the adaptive node. We provision 4240 FPUs in 1060 cores. At our 1.5GHz clock rate, this provides a peak performance of 6.4Tflops when operating out of registers. When operating at peak rate from registers, 84.3% of the power is consumed

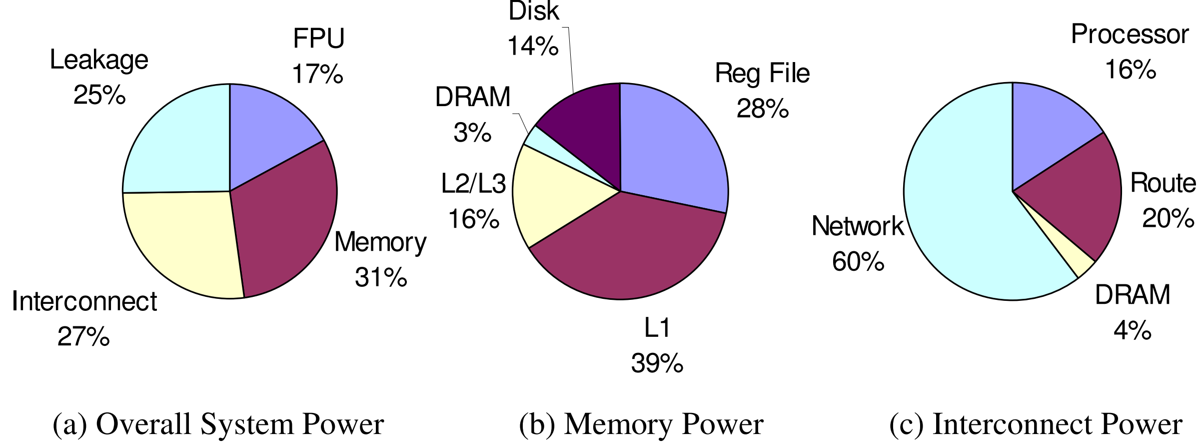


Figure 7.20: Power distribution in aggressive strawman system.

by the FPUs with the balance consumed by accessing registers, fetching instructions, and control overhead.

For on-chip access, we provision the L1 and L2 so they can each support one access for every floating-point unit each cycle. This requires 62.5% and 70.1% of the total power respectively. Thus, we cannot maintain the peak floating-point rate at the same time that these memories are being accessed at full bandwidth. To stay within the power envelope, we restrict L3 accesses to one access for each 1.68 floating point units per cycle. This consumes all available power.

For off-chip access, both the DRAM interface and the network interface are also configured so that they can each consume all available power. This widens the DRAM interface by an order of magnitude to a bandwidth of nearly 900GWords/s. The network bandwidth is half of this rate. With this adaptive power provisioning, the off-chip data rates required now pose packaging challenges where with the fixed budget shown above they were well within the capabilities of existing technologies.

7.3.8 Overall Analysis

Figure 7.20(a) breaks down the power usage within the aggressive strawman design. The categories include that for the FPUs alone, for accessing any memory structure in the system, for interconnect (both on and off-chip), and for leakage. As can be seen, power is fairly evenly distributed, with FPU power the lowest and memory power the highest.

Figure 7.20(b) then breaks down just the power associated with accessing any memory structure in the system. This distribution is much less evenly divided, with about 2/3 of the power spent closest to the FPUs in the register files and in the L1 instruction and data caches. The L1 energy is evenly split between instruction and data accesses.

Figure 7.20(c) diagrams the distribution in power spent in transferring data, both on-chip and between chips. Here the clear bulk of the power is spent in transferring data between racks.

7.3.9 Other Considerations

The power estimates above are a clear "best-case" scenario - it is very highly unlikely that any real implementation is likely to come in at any lower power levels. The estimates above do consider the effects of leakage current - at the system level it raises total power from 49.8MW to 67.7MW, or

an overall tax of about 26%. However, there are several other factors that may in fact increase the total power of a real system, but are not in the above estimate:

- The ratio of flops (1 exaflops) to memory (3.6PB) is 0.0036 - about two orders of magnitude less than what is seen in many of today's supercomputers, such as described in the heavy evolutionary strawman of Section 7.2.1.
- There is no ECC on the memory - this might apply about a 12% upside to both the memory and the transfers between memory and processor chips.
- There is no energy expenditure for matching tag arrays in caches - we treat the on-processor arrays as directly addressed "memories" in order to get a minimum energy measure.
- Energy for data references from any memory is for the exact 8-byte words desired - no extra transfers for line prefetch is accounted for.
- Energy for address and command transfers on the processor-memory interfaces is not considered.
- Likewise, ECC and protocol overhead needs to be considered on the node-node and group-group links.
- There is no overhead for clock distribution on the processor chips - in many real chips today this is a significant percentage of the total die power.
- There is no overhead for redundancy management in the processors, such as duplicate cores, spare FPUs, comparators, etc.
- There is no overhead for redundancy and RAID controllers in the scratch disk estimates.
- Other than the register files, there is no energy expenditures accounting for core instruction decode logic, or for non-FPU processing such as address computations.

7.3.10 Summary and Translation to Other Exascale System Classes

Although the Exascale system described above targeted the data center class system, pieces of it can be used to get a handle on the characteristics of the other two classes: departmental and embedded. To drive such a discussion, Table 7.10 summarizes estimates for five different variations:

1. **Exaflops Data Center**: represents the aggressive design as discussed earlier - a system that has a peak capability of 1 exaflops regardless of power or other limits.
2. **20 MW Data Center**: the same system derated to fit a 20 MW power limit. This yields perhaps 30% of an exaflops, and thus represents the best we can do with silicon in a 20 MW envelop.
3. **Departmental**: A single rack from the first column.
4. **Embedded A**: A single processor die and its 16 DRAM chips. Neither a router chip nor disk drives are counted.
5. **Embedded B**: the same as Embedded A, but where the number of cores (and thus power) on the processor die has been derated to a peak of one teraflops.

	Exascale System Class				
Characteristic	Exaflops Data Center	20 MW Data Center	Department	Embedded A	Embedded B
Top-Level Attributes					
Peak Flops (PF)	9.97E+02	303	1.71E+00	4.45E-03	1.08E-03
Cache Storage (GB)	3.72E+04	11,297	6.38E+01	1.66E-01	4.03E-02
DRAM Storage (PB)	3.58E+00	1	6.14E-03	1.60E-05	1.60E-05
Disk Storage (PB)	3.58E+03	1,087	6.14E+00	0.00E+00	0.00E+00
Total Power (KW)	6.77E+04	20,079	116.06	0.290	0.153
Normalized Attributes					
GFlops/watt	14.73	14.73	14.73	15.37	7.07
Bytes/Flop	3.59E-03	3.59E-03	3.59E-03	3.59E-03	1.48E-02
Disk Bytes/DRAM Bytes	1.00E+03	1.00E+03	1.00E+03	0	0
Total Concurrency (Ops/Cycle)	6.64E+08	2.02E+08	1.14E+06	2968	720
Component Count					
Cores	1.66E+08	50,432,256	2.85E+05	742	180
Microprocessor Chips	223,872	67,968	384	1	1
Router Chips	223,872	67,968	384	0	0
DRAM Chips	3,581,952	1,087,488	6,144	16	16
Total Chips	4,029,696	1,223,424	6,912	17	17
Total Disk Drives	298,496	90,624	512	0	0
Total Nodes	223,872	67,968	384	1	1
Total Groups	18,656	5,664	32	0	0
Total racks	583	177	1	0	0
Connections					
Chip Signal Contacts	8.45E+08	2.57E+08	1.45E+06	2,752	2,752
Board connections	1.86E+08	5.65E+07	3.19E+05	0	0
Inter-rack Channels	2.35E+06	7.14E+05	8,064	0	0

Table 7.10: Exascale class system characteristics derived from aggressive design.

For each of these system classes there are four sets of summary characteristics:

1. Some key absolute functional attributes from Section 2.1, namely peak performance, total storage, and power.
2. Some normalized attributes.
3. Component counts.
4. Counts of connections of various sorts: chip, board edge, and channels between boards. The chip contacts are individual contacts such as C4 pads that support individual signals - power, ground, clocking, and control contacts are not counted. The board edge counts assume a board with 12 nodes (processor + router + DRAMs) on it, and again represents individual connector "pins" that carry individual electrical signals (two needed for a differential link). The inter-rack channels represent the number of distinct router compatible channel paths that couple racks.

The last two sets of metrics were included to give some sense of how reliability issues may come into play, especially for larger systems.

7.3.10.1 Summary: Embedded

The two embedded columns indicated that even though a processor chip plus DRAMs make a rather potent assembly of about the performance desired for an embedded class implementation, the power of at least 153 W is still excessive by some non-trivial factor, even considering the low 0.014 bytes per flop present in the 1 teraflops configuration. Boosting this memory capacity to something more reasonable will increase power in two ways: per DRAM chip power and increased power per processor-memory link. The latter is likely because there are insufficient contacts available in silicon at that time to allow for many more memory interfaces, meaning that each interface must now support multiple DRAM chips and the added capacitance they introduce.

In terms of memory bandwidth, the Embedded B column does support more bandwidth per flop, mainly because we assume the number of cores drops to 180 from 742. However, no additional bandwidth is gained by adding more DRAMs.

Also, there is still considerable concurrency - at least 720 flops per cycle must be extracted from the application in order to achieve peak efficiency. This is orders of magnitude above any "embedded class" processing system of today.

7.3.10.2 Summary: Departmental

The Departmental system consists of one rack from the exaflops system. This is again in the right performance range of 1.7 petaflops (if we can run at 100% efficiency) and is the right physical size (1 rack), but is again significantly off the mark in power and memory capacity. While the active power of the electronics for a single rack is 116 KW, the overhead for power supply inefficiencies and cooling may raise the wall-plug power to upwards of 200KW - probably too much for a small machine room. Also 0.0036 bytes per flop is 10-100X below the expected Petascale machines of 2010, of either the heavy or light node types of Section 4.5. With the percentage of power due to memory at 29% (Figure 7.20), getting to even 0.1PB of DRAM would add on the order of 1/2 MW to the total.

Concurrency is also considerably higher - on the order of 1 million operations must be independently specified by the program in each and every machine cycle.

Component	FITs/Component	Exascale 1		Exascale 2	
		# Components	FITs	# Components	FITs
Processor chip	1000	224K	224M	224K	224M
DRAM chip	5	3,582K	18M	14,330K	72M
Flash chip	5	1,791K	9M	7,164K	36M
Router chip	1000	224K	224M	224K	224M
Disk Drive	1000	229K	299M	299K	299M
Power Supply	100	37K	4M	37K	4M
HW FITs			777M		857M
Other FITs			777M		857M
Total FITs			1,554M		1,715M
MTTI (minutes)			39		35

Table 7.11: Failure rates for the strawman Exascale system.

Also, from the packaging perspective, the number of individual pins carrying individual signals between each board and the backplane is on the order of 5,000 pairs (52x12x4x2), which is not practical with any technology of today.

On the bright side, for the amount of DRAM capacity in this system the disk capacity is 1000X - more than ample for scratch or secondary, and bordering on useful for a more general file system with some archival properties.

7.3.10.3 Summary: Data Center

The same comments spoken for the departmental system can be applied to the Data Center class. As discussed in Chapter 5, different applications may scale from today to an exaflops in several different ways, with a weather model, for example, taking anywhere between 10 and 100 PB of main memory (Section 5.7.2). Even the lower of these numbers is almost 3X the assumed capacity, meaning that the 67MW number would be low by an additional large factor. Again we have not factored in environmental power considerations.

Concurrency in the full exaflops system nears one billion operations in each cycle.

Some additional significant reliability concerns also surface in this class. There are over 4 million chips, three hundred thousand drives, almost a billion signal-carrying chip contacts, and about 170,000 cable bundles between racks that are each carrying about a dozen channels. Tripling the memory to a mere 10PB adds another 10 million chips and 1 billion chip contacts.

7.4 Exascale Resiliency

We can now analyze the resiliency of the aggressive silicon strawman described in Section 7.3. Table 7.11 shows an overview of two sample Exascale Systems: Exascale 1 with 16GB of DRAM per processor chip and Exascale 2 with 64GB of DRAM per processor chip. According to the strawman, we assume 16 disk drives per 12-node cluster and 8–32 flash chips per processor chip, for data storage and checkpointing. We assume the same failure rate for Flash and DRAM (5 FITs per chip) as was budgeted for BlueGene/L assuming that the decrease in single event upset rates (SEUs or soft errors) will be offset by a slow increase in hard failures. The processor and router chips are custom low-power circuits and we assume 1000 FIT per chip based on SIA projections, which accounts for improvements in chip-level resiliency as well as more severe chip failure modes.

		Exascale 1	Exascale 2
Disk Checkpoints (sustained 270 MB/sec)	Checkpoint Latency (Seconds) Availability	60.0 77%	240.0 52%
Disk Checkpoints (sustained 2.7 GB/sec)	Checkpoint Latency (Seconds) Availability	6.0 93%	24.0 85%
Flash Checkpoints (sustained 22 GB/sec)	Checkpoint Latency (Seconds) Availability	0.7 97%	2.9 95%

Table 7.12: Checkpointing overheads.

We assume a 100,000 hour reliability on disk drives (1000 FIT) and a lower 100 FIT rate on power supplies.

These assumptions produce a hardware failure rate of 777–857 million FIT for an Exascale system. Based on experience that hardware accounts for about half of the overall system failures, an Exascale system could be expected to have a failure rate of 1.5–1.7 billion FITs. These failure rates corresponds to a failure every 35–39 minutes.

Table 7.12 provides an analysis of the effect of failure rate on the availability of the system. For this analysis, availability is defined to be the fraction of time that the machine operates at full capacity, assuming that repair time is zero. While aggressive sparing with automatic fail-over can help a system approach ideal zero-repair time, the system will still experience some degradation. The table shows checkpoint latency for three different scenarios: (1) local disk checkpointing at 270 MB/second (1/20 the raw disk bandwidth) to account for system overheads, (2) local disk checkpointing at 2.7 GB/second (optimistically 1/2 the raw disk bandwidth), and (3) local flash checkpointing at 22 GB/second (1/16 of the raw DRAM bandwidth). The total checkpointing latency is a function of both the checkpointing bandwidth and the memory capacity, assuming checkpointing of the entire contents of DRAM. The slowest checkpointing rates are 1–4 minutes, while the fastest are 1-3 seconds. The machine utilization is computed using an optimal checkpointing interval that maximizes the availability, accounting for the overhead to take a checkpoint as well as the work lost when a failure occurs. The utilization ranges from 52% to 95% depending on the checkpointing bandwidth, failure rate, and memory capacity.

This analysis emphasizes the importance of design and technology developments on the capabilities of the machine, as utilization degradation requires a larger machine to sustain a given level of performance. Even 90% utilization requires a machine to have 10% more components and consume 10% more power to achieve a particular Exascale metric. This analysis also indicates that fast rollback and recovery schemes, coupled with automatic fail-over, can reduce the effect of significant failure rates. If these mechanisms are sufficiently effective, even higher failure rates can be tolerated, which gives designers the opportunity to choose less reliable components or technologies as a part of the cost/power/performance/utility optimization.

7.5 Optical Interconnection Networks for Exascale Systems

We develop here an exercise to explore a simple model for the insertion of photonic interconnect technologies within the context of the strawman Exascale system design in 7.3. The goal of this analysis is to expose key system metrics which may be enabled by optical interconnects, particularly potential gains in the available bandwidth under an equivalent power budget to the strawman design. The unique challenges associated with optical technologies (i.e. the lack of equivalent optical RAM) will of course require new architectural approaches for the interconnection networks.

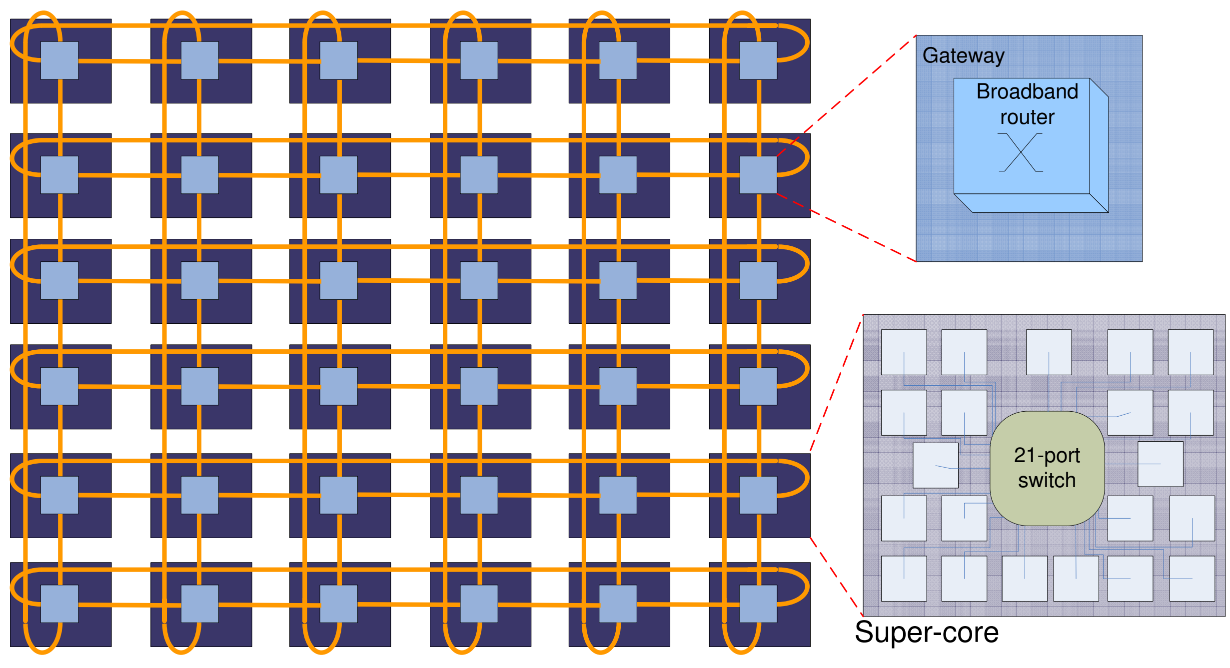


Figure 7.21: Chip super-core organization and photonic interconnect.

The analysis here however makes no attempt to design any aspects of the interconnection network and makes only broad assumptions regarding the topology, flow control, etc.

7.5.1 On-Chip Optical Interconnect

We begin as in the silicon system strawman design (7.3) in a bottom-up fashion with the processor chip. We organize the processor chip into groups of cores, or **super-cores**. For the 742-core chip, we have 36 super-cores each containing 21 cores (see Figure 7.21). An optical Gateway which contains the electronic/photonic plane interface as well as a routing switch is associated with each super-core. The super-cores form a regular 6x6 grid on-chip which is connected via a photonic on-chip network. The gateways provide the electro-optic (E/O) and opto-electronic (O/E) interfacing between the processor super-cores and the photonic interconnect. The optical Network on-chip (NoC) network topology can be assumed to be a mesh, or some derivative thereof, such as a torus. In addition to the E/O and O/E interface, the gateways include a broadband router switch which can direct a multi-wavelength optical message either onto the on-chip network or to an off-chip fiber port. The gateway router switch can be configured to receive multi-wavelength messages from off-chip. The on-chip electronic network as described in the strawman is assumed to remain and its full power budget of 8.4W will be included in the calculations. However as the design is further optimized and the optical on-chip network is used for an increasing fraction of the traffic (particularly for large, high-bandwidth message transfers) the power budget allocated to the on-chip electronic interconnect may be reduced.

7.5.2 Off-chip Optical Interconnect

The strawman design chip is rated at 4.5 Tflops. We assume here that the optical NoC can provide 1 B/s for each flop. This is equivalent to 36 Tb/s available to the chip or 1Tb/s per super-core. In addition, the 36 broadband router switches at each gateway can direct 1Tb/s to an off-chip fiber

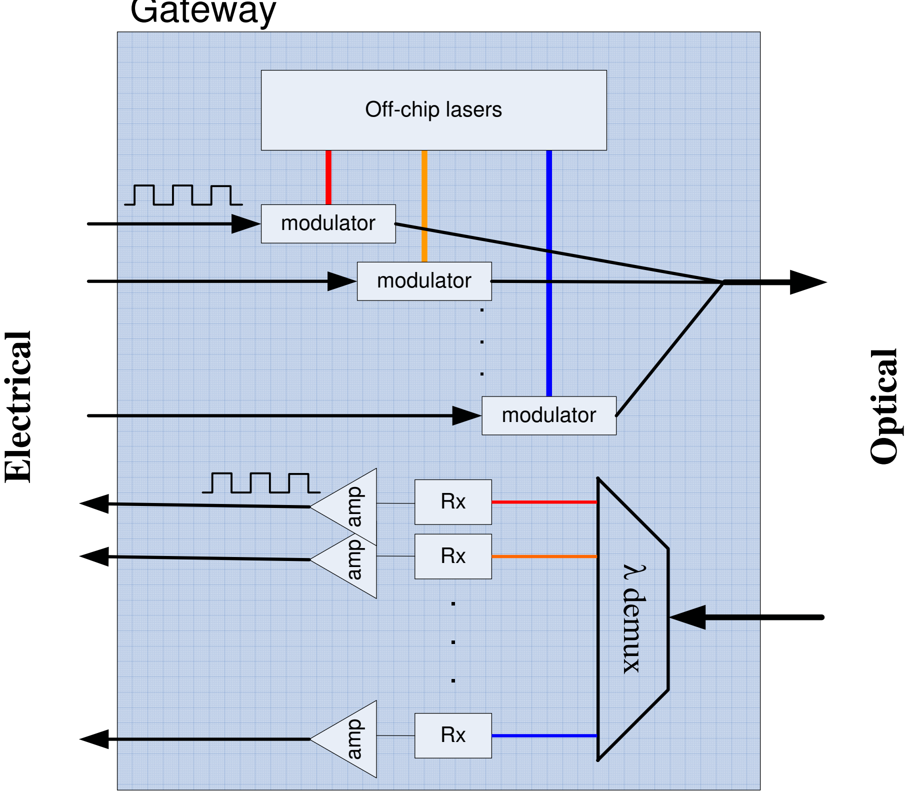


Figure 7.22: Gateway functional block design.

Modulator power	0.1 pJ/bit
Receiver power	0.1 pJ/bit
On-chip BroadBand Router Power	0.5 mW
Power per laser	10 mW
Number of wavelength channels	250

Table 7.13: Optical interconnect power parameters.

port and are configured to receive 1Tb/s from an off-chip port (these broadband ingress/egress switches are not explicitly shown in the figure).

The gateway (see Figure 7.22) E/O interface consists of a bank of electro-optic modulators fed by multiple off-chip lasers (one for each of the wavelengths) and the electronic signaling from the processor plane. The O/E interface consists of passive wavelength demultiplexing and a bank of receivers (one for each of the wavelengths).

Based on current measurements and projections for silicon photonic ring-resonator modulators and receivers in the 2013-2014 time frame we employ 0.1 pJ/bit for the energy consumed at each of the E/O and O/E interfaces. In addition, the laser power consumption which is continuous is assumed to be 10mW per wavelength channel. Optical links are generally more power efficient at higher frequencies and typically operate today at 10 GHz or 40 GHz. However to simplify this calculation and to avoid additional computation of the serialization and de-serialization energies, we assume here that the optical signals run at the electronic tributary rate of 4 GHz. To achieve our designed bandwidth per super-core of 1 Tb/s, 250 wavelength channels are required. Although

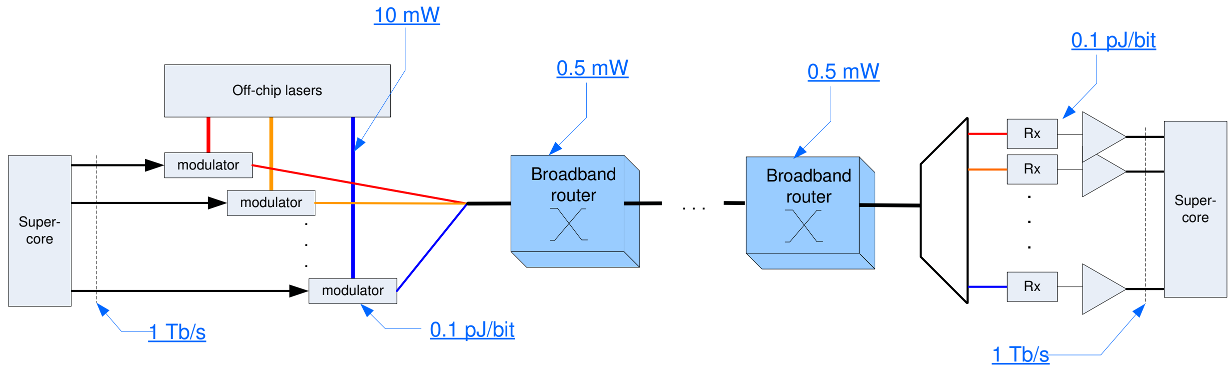


Figure 7.23: Super-core to super-core optical on-chip link.

Parameter	Value
Cores per supercore	21
Supercores per chip	36
broadband switch routers	36
Number of lasers	250
Bandwidth per supercore	1 Tb/s
Optical Tx + Rx Power (36 Tb/s)	7.2 W
On-chip Broadband routers (36)	18 mW
External lasers (250)	2.5 W
Total power for photonic interconnect	9.7 W

Table 7.14: Optical on-chip interconnect power consumption.

aggressive, this is not an unrealistic number, as it has been demonstrated within the context of commercial telecom fiber optic transmission. These components power parameters are summarized in Table 7.13. Each chip would require 250 external lasers each consuming 10mW for a total of 2.5W per chip.

Broadband silicon photonic router switches are used at each gateway to route optical messages within the on-chip network and to off-chip optical ports, as pictured in Figure 7.23. Based on today's silicon photonic switch technology, each router switch is assumed to consume approximately 0.5 mW. Importantly, this is independent of the number of wavelengths being routed simultaneously or the bit rate in each channel.

We now have all the components necessary to estimate the power consumed by the chip itself. Table 7.14 summarizes the calculations of the power consumed by on-chip photonic interconnection network.

There are several key observations from this simplified analysis. The total power of the on-chip photonic interconnect, estimated at 9.7 W is equivalent to the on-chip electronic interconnect in the strawman design of 8.4 W (Table 7.7). The photonic interconnect provides about a factor of 28 higher bandwidth (36 Tb/s versus 1.28 Tb/s) to the chip in comparison with the electronic on-chip interconnect. Secondly, the broadband router switches consume practically negligible power. As expected, the dominant contribution to the power dissipation comes from the E/O and O/E interfaces. Once these are performed however, the optical signal does not need to be regenerated

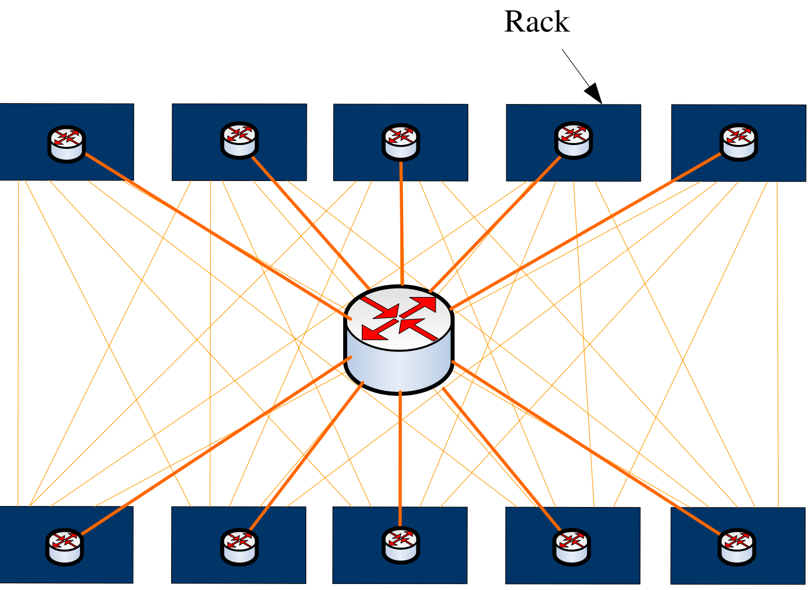


Figure 7.24: Optical system interconnect.

as it propagates off-chip and across the systems. This is the critical advantage that photonic interconnects can provide and will become apparent as we continue to compute the total Exascale system interconnect power dissipation.

From the strawman design, each processing node contains 16 chips of 1GB of DRAM each at 5.5 GWords/s. The total bandwidth of memory for each node is therefore 88 GWords/s, or 5.63 Tb/s. In this analysis we assume that the same optical Tx/Rx interface technologies are employed for the memory gateways.

Approximately 40 wavelength channels (or lasers) each operating at 4 Gb/s are needed to achieve this bandwidth across the 36 super-cores. This puts the power for the memory gateways per node: 40 * 10 mW + 5.63 Tb/s * 0.2 pJ/bit = 1.526 W. It is clear that if we were to use a much higher bandwidth at the memory interface, such as 4.5 TB/s to match the bandwidth at the core, the memory interface power consumption would be similar to the on-chip interconnect or approximately 9.7 W.

The power to access DRAM was estimated at 10 pJ/bit for the strawman design. This includes the access and transmission of the data. It is unclear how much of this is contributed by each, so it is difficult to estimate the fraction of power contributed by the DRAM itself and the fraction that would fall under the optical interconnect budget.

7.5.3 Rack to Rack Optical Interconnect

At the system level, Figure 7.24, there are 583 racks each containing 384 processing nodes. We assume here that the racks are connected by a transparent optical network using a double layer hierarchical network topology with 2 optical MEMS 384x384-port routers per rack. These routers consume 100 W of power each, regardless of bit rate per port. We note here that this estimate is based on current optical MEMS cross-connects designed for telecom applications and is expected to be significantly less (by a factor of 10) when customized for computing systems. Racks are organized into approximately 60 groups of 10 with a central router, as shown in the figure below. The racks are also fully connected point-to-point to avoid the central router becoming a bottleneck

Parameter	Value
Bandwidth per chip	36 Tb/s
Bandwidth to/from memory per node	5.6 Tb/s
Number of MEMS routers	1226
Total Power for chip interconnect	9.7 W
Total Power for external network	122.6 KW
Total Power for node memory interface	1.53 W
Total power consumed by optics	2.6 MW

Table 7.15: Optical system interconnect power consumption.

for close-proximity transfers, which should be exploited in the mapping of the application to the system architecture. Note that because both the NoC and external optical network are integrated and transparent, data generated from one super-core on one chip in one rack traveling to any other super-core in the system will consume power per bit only at the gateway interfaces, set at 0.2 pJ/bit.

The only remaining power calculations are to add the power from the 1226 (583*2 + 60) routers to the total power from the racks. Table 7.15 summarizes these power calculations.

7.5.4 Alternative Optically-connected Memory and Storage System

Exascale memory and storage systems must mitigate the disparity in bandwidth among levels of the hierarchy, achieve adequate storage performance, minimize energy expenditure, and provide adequate resilience. As discussed above, familiar DRAM main memory, disk, and tape hierarchy will probably not suffice. Memories in the form of commodity-based single-die silicon substrates will not deliver the requisite bandwidth and latency at acceptable power levels. Copper-based interconnect up and down the hierarchy may be inadequate to deliver the performance needed within the power budgeted. A truly serious approach to build an Exascale system within a decade may very well require alternative structures such as photonic communication between logic and 3D optically connected memory modules (OCM). Such a strawman memory system is discussed here.

As a baseline for a strawman data center scale memory system we assume a 32 petabyte main memory. Depending on the types of memory available, one may be able to afford more than that, in both cost and power terms, through the use of an OCM technology. data center projections shows storage file system to main memory ratios fall typically in the 100:1 range, and hence 4 exabytes of usable file storage is a reasonable estimate (adding 20 percent for ECC and other metadata). A substantial amount of storage needs to be much higher bandwidth and lower latency than current spinning disk technology, which is not improving for either of these performance measures.

Application checkpoints through defensive I/O to disk files play an essential role in ensuring resilience in today's Petascale systems. The growing disparity between disk bandwidth and integrated devices argues for more research into solid-state distributed storage solutions that are much faster and more energy efficient than mechanical systems. Several nonvolatile RAM (NVRAM) technologies look promising, as discussed previously. Some combination of DRAM and NVRAM in the same stack will significantly improve bandwidth and save energy. Depending on the cost and capability of the NVRAM, it may be able to supply most of the storage capabilities for an Exascale system, or at least act as a high capacity buffer to enable a better match between memory and

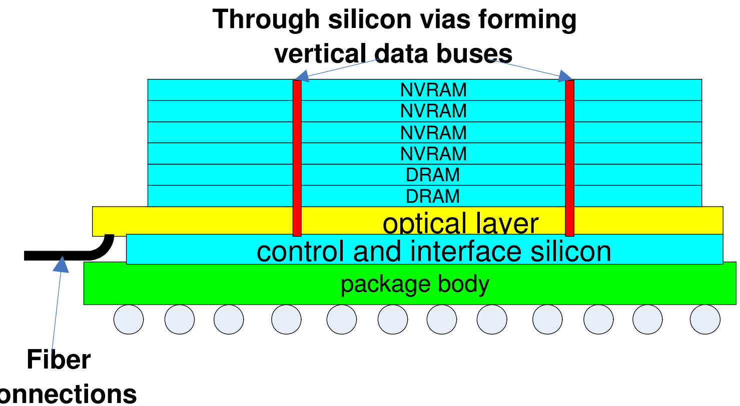


Figure 7.25: A possible optically connected memory stack.

storage.

The ITRS projects that each high performance DRAM die in a 2017 (20 nm) process will cost .03 microcents/bit with a capacity of 1-2 GB per die. A 32 PB commodity memory would therefore require 16.7-33 million die, each with 81-270 pins. The cost for just the memory die would be $86.5M. The cost in power and DIMM packaging would increase this number significantly given the need for somewhere between 1.3 and 4.5 million wires. An OCM package such as that shown in the figure below that combines two layers of DRAM and four layers of high density NVRAM can reduce the number of components by at least 6, since the wires to each OCM module can be replaced by shorter in-module vias, and the external connection will only need 4 waveguides per OCM. Interconnect power alone for an Exascale system can be reduced from 15 MW (for all-electrical) to less than 1 MW for an optically interconnected system. A simple analysis for the DRAM portion of the OCM shows that at peak access rates, an 8 gigabyte OCM will consume 2.0 watts in 2017 and a 4 gigabyte OCM will consume 1.4 watts. This puts the total memory power requirement between 8.5 MW and 12 MW.

The OCM module, with a capacity of 4 or 8 gigabytes is constructed from a 3D stack of DRAM and NVRAM dies as shown in Figure 7.25. Each processor socket can connect optically to 64 of these modules. Using a photonic scheme enables each socket to be connected to more memory than would be possible by stacking the memory directly on top of a processor chip, and also avoids the problem of heating of memory by the processor when memory is stacked directly on top of a processor chip. More OCMs can be attached to the same memory controller channel for larger configurations. The channel consists of four waveguides, each of which carries 64 wavelengths modulated at 10 GHz for a total per channel memory bandwidth in excess of 160 gigabytes/sec per OCM, and 10 terabytes/sec per processor socket. Power and performance issues motivate an OCM organization where each access is a complete 128 byte cache line.

NVRAM capacity should be at least twice that of main memory, since a failure may occur during a checkpoint operation. More should be provided if affordable. It makes sense to place the NVRAM in the OCM stack: this allows fast in-stack DRAM to NVRAM copy, which saves energy since data will not have to be transported over a global interconnect.

There are several technologies currently competing for ultra-dense NVRAM, as discussed in Section 6.3.5. Simple crossbar circuits on top of functional CMOS have been demonstrated with a local density of 10 GB/cm^2 for a single layer. Further, the crossbar technology itself is stackable, so that effective densities as high as 100 GB/cm^2 on a single die are conceivable. There is a significant amount of activity in examining the switch material for nanometer-scalable resistive-RAM (RRAM)

junctions incorporating metal oxide layers (NiO and TiO2), that in principle can be written in less than 10 ns for less than 1 pJ/b energy dissipation. The resiliency of these systems is being designed to handle 1-10% defects in the crosspoints.

7.6 Aggressive Operating Environments

By the end of the next decade, expected technology trends and possible architectural responses to exploiting them will impose extreme demands on software operating environments which will subsume the current responsibilities of run-time systems, operating systems, virtual machine layers, and system level schedulers and resource managers. As a consequence, they may look very different from today's operating environments. This section summarizes the extreme demands that will be placed on the operating environment software across the trans-Exaflops performance regime and discusses some elements of one possible approach to addressing them. Through this perspective, a set of general research questions is derived that may help.

7.6.1 Summary of Requirements

From this study, potential properties for Exascale computing systems has been explored and estimated. Here we highlight those that will have the largest impact on operating environments:

- **Parallelism** - concurrency of action will have to be elevated to unprecedented levels to reduce execution time, achieve target performance levels, and maintain adequate efficiency. It appears that sustained parallelism of a minimum of 100 million-way will be required to meet this requirement, regardless of technology. But time distances (measured in processor core cycles) to almost every resource whether local memory, remote nodes, or system services must also rely on additional parallelism to hide such latencies. While many factors contribute to determining exactly what the level of additional concurrency required for this purpose is, two orders of magnitude may prove a conservative estimate. The result is that future environments should be assumed to support parallelism of ten billion-way or more.

- **Latency** - As touched on previously, latencies measured as delay time in processor core cycles to intra-node and remote resources can be a source of extreme performance degradation. Multi-ten thousand-way latencies will demand locality management and latency hiding methodologies far more aggressive than today's systems. The apparent latency will be aggravated by contention for shared resources such as memory bank access and global system interconnect.

- **Overheads** - While conventionally a combination of static scheduling, global barrier synchronization, and large point to point data block transfers, future computations on Exascale systems may be very different. Application algorithms of future large scale problems will be multi-discipline, sparse, dynamic, and potentially irregular in data structure and operations on them. A major consequence is that the run-time resource management and application task synchronization overhead will become a significant part of the total execution.

- **Reliability** - As the number of devices grows and their size shrinks to nano-scale features, single point failures may have devastating effects on the operational viability of the system. Conventional practices of checkpoint/restart will be infeasible as the time to dump core to external storage will exceed the MTBF of the system. In the future, no system will be static in structure but will be continually modifying its form as hard faults dictate isolation of failed

components and transfer of data and tasks to functionally elements. An aggressive operating environment will out of necessity have to contend with this continuously changing physical platform while providing a global virtual interface to basic functional capabilities.

7.6.2 Phase Change in Operating Environments

To meet these unprecedented challenges, a change in execution model and operating environments is probable. While it is unclear at this time what form such a revolutionary environment may take on, it is possible to consider one such strategy as an exemplar for a class of possible alternatives. Such a strategy would mark a phase change in operating environments with respect to conventional practices as it directly addresses the challenges and requirements described above.

The levels of physical and abstract parallelism combined with the unprecedented potential efficiency losses due to overhead and latency will drive strategic changes. It is imperative that our objective function, our metric of success, be changed to reflect the realities of the costs and tradeoffs. Even today, we use as our principal measure of efficiency the ratio of sustained to peak floating point performance. In fact, the floating point ALU is among the least expensive resources in terms of die area, cost, or power consumption. Instead, the precious resources and therefore the overall efficiency are memory capacity, memory access bandwidth, and system bandwidth and latency. In addition, there should be high availability of flow control to handle over-subscription with respect to peak capability of these bottlenecks.

Dynamic scheduling of resources is required both for load balancing and reallocation in the presence of faults, as well as exigencies of system demand outside the user application domain. For this reason, the conventional practice of the user code controlling the dynamic scheduling is infeasible (especially when billions of threads of control may be present), and must be performed by a system sensitive set of policies and mechanisms while minimizing overhead impact on the application execution. In turn, this forces an important change in the relationship between operating and run-time systems, forcing a merger where there was once clear separation of roles and responsibilities. It also changes the relationship between the programmer and the resources. Where conventionally the programmer is largely responsible for explicit management of locality and resources, in the future the system operating environment will have to take on this responsibility, with declarative directives from the user indicating effects to be achieved rather than imperative statements of explicitly how to achieve them.

7.6.3 An Aggressive Strategy

An alternative operating environment to the incremental derivative described in Section 6.8 is suggested that may achieve superior scalability and robustness for dynamically adaptive systems while providing a relatively simple implementation strategy. The approach proposed employs lightweight kernel elements in each local collection of physical threads. Unlike typical lightweight kernel methods, the virtualization is not limited to each "node" but across the entire system through an intermediate set of protocols between kernel element instances. For simplicity, the brief description here is considered for three primary functionalities, although in a realistic implementation other support mechanisms would be provided as well. The three basic system level functions are: distributed memory, threads, and communication fabric.

The expected physical memory will likely comprise multiple distributed sets of local memory banks with a minimal manager for each fixed sized bank. The local memory kernel serves this resource in an object oriented manner through a defined protocol of functions. However, this protocol extends beyond the local memory bank to include interaction and possible negotiation

with other like local memory kernels serving separate memory banks. This synergistic operation permits collective functionality across local memory resources to satisfy relatively global memory demands. Such functionality includes dynamic data distribution, global address translation, fault management, and copy semantics.

The other two global virtual functions, distributed threads and communication fabric, are achieved through a similar global combination of local lightweight kernel elements. Within the operational framework under consideration, tasks in the form of lightweight threads are scheduled to be executed where their primary operand data are located and when there are available execution resources. Thus, an aggressive operating environment will support a global distributed thread execution environment where the work may move to the data when appropriate rather than the data always moving to the work, a potentially costly operation.

The communication fabric is also supported by the synergy of like lightweight kernel elements managing the flow of information, routing, and address translation. An important property of all three of these distributed functionalities is that they adapt to changing structures due to diagnosed faults and other drivers of reconfiguration. As a physical module is deemed faulty, its local kernel elements are turned off and the neighboring elements book-keep the fact that they no longer represent available resources, providing a degree of system adaptive reconfigurability.

The operating environment supports a message-driven work queue execution model in addition to the more conventional process oriented message-passing model of computation. This advanced strategy for controlling distributed processing exposes a high degree of parallelism through lightweight synchronization mechanisms and is intrinsically latency hiding. This is because the work queue model, assuming sufficient parallelism, does not wait on remote actions but processes a stream of incoming task requests on local data. This method also reduces overhead by localizing synchronization within the flow control or within the data itself, thus eliminating such unscalable techniques like global barrier synchronization. The strategy outlined here is highly scalable if the operation of the local lightweight kernel elements are primarily dependent in the state of their neighboring domains of local resources. The design of these kernel elements is simple. Complexity of operation is derived, not through complexity of design, but rather through the emergent global behavior of the synergistically interacting collections of simple local kernel elements. This greatly bounds difficulty of software design, debugging, and scalability.

7.6.4 Open Questions

An aggressive strategy to billion-way parallelism (or more) processing systems' operating environments presents many challenges and questions to be resolved before realization of the promise implied. The sketch of an aggressive operating environment above exposes a number of issues that must be resolved prior to realizing the promise of such future systems. Here we briefly discuss some of these key research questions.

- **Model of Computation** - The key issue driving all research for future Exascale computing system architectures is the principal model of computation that will provide the governing principles of parallel computation and the interrelationships among the physical and abstract computing elements. If the model is an immediate extension of current conventional practices based on a mix of communicating sequential processes globally and local shared memory operation, then locality management becomes paramount to avoid performance degradation due to latency and overhead. If, as has happened many times in the past, the field transitions to a new model better capable of exploiting the emergent technologies and enabling new classes of dynamic applications, then this enabling paradigm or perhaps a multiplicity of

such paradigms needs to be devised against a set of specified requirements. It should be noted that although a significant shift in methodology is suggested, this does not assume a disruptive discontinuity with respect to legacy application codes and techniques. Rather, any such innovation should subsume as a semantic subset the typical practices of the past.

- **Synergy Algorithm** - In most simple terms, the choice space for managing large scale resources is either a centralized control system or a fully distributed control system. Most systems today are either the former or a cluster of such systems; the extreme being Grid based widely distributed systems. There has been little exploration of the latter, fully distributed control systems. Yet, it is possible that the degree of scaling expected by the nano-scale era will make any centralized methodology infeasible for both efficiency and reliability reasons. A class of algorithms must be derived that enables and supports the synergistic interaction of simple local rule sets to achieve the desired emergent behavior of each of the critical global functionalities of the system yielding a global virtual machine that is dynamically adaptive to resource demands in the presence of faults. This resilient strategy is scalable to the regimes of parallelism demanded and should be the subject of future research. In this context, we will see a merger of operating system and run-time system roles and responsibilities which today are largely mutually isolated.

- **Global Name Space Management** - Whether merely at the user application level or as an intrinsic of the hardware architecture, every program exhibits a global or hierarchical name space; the set of referents to which the algorithm is applied. Conventional practices assume a physically fragmented name space on most scalable systems (DSM is an exception) at least to the level of the node. Load balancing is handled if at all by user transformation of effective addresses from one node to another; a low level and costly effort with many restrictions and highly error prone. Research that finds a superior mode of operation somewhere between full DSM and PGAS is required to guide future system design in support of user name spaces in dynamic adaptive resource management systems, in part due to the needs of resilience.

- **Fault Management Methodology** - New methods are required to provide effective functionality in systems of the scale being considered. It is recognized that conventional practices of checkpoint/restart will not scale to Exascale system structures, in part because the MTBF with single point failure modes will cross the threshold for which it will be shorter than the time to dump core. Research will be required to address this challenge by building in micro-checkpointing semantics directly in to the execution model and to provide low overhead and robust mechanisms to support it. Key is the management of commits of global side-effects to ensure that any such are performed only after verification of correctness. A second challenge is the dynamic remapping of virtual to physical addresses and its efficient implementation in system interconnect routing. This may require a second level of hardware address translation to complement conventional TLBs (translation lookaside buffers).

- **Programming Models** - Programming languages have served poorly in recent years as the programmer has been responsible for hands-on management of the low level resources as they are applied to the user application. Research is required to provide an API that serves the new class of operating and run-time system models developed in the context of dynamic adaptive resource management and application execution. While it is often thought that the user needs access to the lowest level to achieve adequate performance, an alternative strategy will have to be devised that supports hardware/software co-design so that the architecture, new run-time, and programming interface are devised to be mutually complementing. Multiple levels

of programming may be a resulting set of solutions depending on computational domain with source to source translation proving the norm to lower level APIs. Compilation strategies will rely more heavily on run-time resource management and thus become more simple. However, they will also have to contend with heterogeneous hardware structures and therefore require a more advanced protocol between compiler and run-time elements.

7.7 Programming Model

While that programming model for the strawman proposed in this section cannot be specified completely at this time, many of its properties may be considered with some confidence. Among these are:

- Expose and exploit diversity of parallelism in form and granularity;
- Lightweight (low overhead) local synchronization;
- Intrinsic latency hiding and locality management;
- Local/incremental fault management and graceful degradation;
- Global name space and virtual to physical address translation;
- Dynamic resource management and load balancing;
- Energy minimization per operation;

Other properties may be assigned as well in response to in depth consideration of the needs of device technology and system scale envisioned for the next decade.

In each of these areas, the development of new technologies is required, but in many cases quite feasible. For example, many of the features of the HPCS programming models are designed to expose parallelism in many forms. These features, and especially the tools behind them, can be enhanced for the even greater parallelism required for Exascale. Energy minimization models already exist in the embedded computing market and it is quite likely that their features could be applied here as well. The major challenge will be to produce a model which combines all of the features in a coherent and productive whole.

7.8 Exascale Applications

In 2007, Scientists and engineers from around the country have attended three town hall meetings hosted by DOE's **Simulation and Modeling at the Exascale for Energy, Ecological Sustainability, and Global Security** (E3) initiative. At these meetings, participants discussed the future research possibilities offered by Exascale supercomputers capable of a million trillion calculations per second and more. A primary finding was that computer scientists will need to push the boundaries of computer architecture, software algorithms, and data management to make way for these revolutionary new systems.

Typical exercises at this series of workshops included extrapolations of science problems of today to Exascale. Many of these exercises involved conceptually increasing the size and scope of the input data, adding new physics and chemistry to the calculations, increasing resolution, and coupling disparate system models.

The following subsections describes how several applications could plausibly be mapped to the kinds of systems extrapolated in Section 7.3. In all cases, the match indicates a reasonable to good match to the capabilities of the machine, and thus a reasonable claim of achieving "Exascale performance."

7.8.1 WRF

Consider a futuristic calculation that might be carried out using WRF. The largest run of WRF ever carried out to date is described in [105]. The performance characteristics and a related performance model were presented in Section 5.7.2. The record calculation is 2 billion cells, 5km square resolution, 101 vertical levels on a half-hemisphere of the earth. Using rounded-off numbers for ease of projecting to Exascale, this calculation achieved about 10 Tflops on 10,000 5 GFlops nodes (about 20% of theoretical peak) on a system with 1 GB of memory per node and sustained memory bandwidth of about 1 byte per flop.

The strawman architecture of Section 7.3 would confer about a 10x increase in total memory capacity over the machine of today (this is the strawman's least dimension of improvement). One could still increase the number of cells under simulation with WRF to about 20 billion, going down to about 1km square resolution on such a global calculation, thereby to capture such effects as cloud dynamics in the atmosphere and interaction with topography on the ground – a calculation of great scientific interest.

The strawman architecture would represent about an order-of-magnitude increase in the number of nodes over the number used in the record run. However given WRF's inherent parallelism and the relative robustness of the strawman network, this should pose little challenge to WRF (indeed in unpublished work WRF has been run at 100K nodes on BG/L - the same order-of-magnitude in number of nodes as the strawman.)

Although the strawman represents about a 1000x increase in peak flops per node, it delivers only about a 100x increase in memory bandwidth per node. WRF is memory bandwidth limited (see Section 5.7.2). Efficiency (percentage of peak) could then fall an order-of-magnitude (to 2% from 20%).

Reading the performance prediction for WRF to improvements in flop issue rate and memory bandwidth (more exactly 1 / memory latency) off of Figure 5.11 one should then be able to run the 10x larger problem 100x faster than on todays machines (if todays machine had the memory capacity). This is a perfectly plausible definition of Exascale as a 10x larger problem 100x faster (a 1000-fold increase of todays capability).

This above projection is the optimistic one based on the notion communications overheads will not grow as log(n) of node count (not implausibly optimistic given WRF's locality, and likely increased computation to communication ratio for the more highly resolved problem). A more pessimistic projection could be read off of Figure 5.14 (25x faster on the 10x bigger problem), but that still represents a 250x capability boost for WRF.

7.8.2 AVUS

Consider a futuristic calculation that might be carried out using AVUS[26]. The performance characteristics and a related performance model were presented in Section 5.7.2.

A future calculation might model the entire structure of a full aircraft interacting with atmosphere and sound waves hypersonically (next generation fighter) under maneuvers. This problem maps plausibly to an order-of-magnitude increase in memory footprint (current calculations focus

typically on a part of the aircraft i.e. wing, tail, or fuselage) so a 10X memory capacity boost allows full aircraft model to be held in memory with useful resolution.

Like WRF, AVUS is not highly communication bound, and is quite scalable by weak scaling (making the problem larger). It is however even more memory-bandwidth-bound than WRF. Reading the performance prediction for AVUS to improvements in flop issue rate and memory bandwidth (more exactly the reciprocal of memory latency) off of Figure 5.12 one should then be able to run the 10x larger problem 70x faster than on todays machines (if todays machine had the memory capacity).

This above projection is the optimistic one based on the notion communications overheads will not grow as log(n) of node count (not implausibly optimistic given AVUS's locality, and likely increased computation to communication ratio for the more highly resolved problem). A more pessimistic projection could be read off of Figure 5.14 (50x faster on the 10x bigger problem - AVUS is even less communications dependent than WRF) still a 500x capability boost for AVUS.

7.8.3 HPL

Recall from Section 5.7.2 that even HPL has some memory references and some slight dependency on memory bandwidth. Reading off projected performances from Figure 5.13 it is predicted that a machine such as the strawman could run an HPL problem 10x larger than one that can be solved today 125x faster than today, a greater than 1000x boost in capability.

7.9 Strawman Assessments

This chapter has provided a series of insights that seem directly relevant to achieving Exascale computing technology, including:

1. Silicon technology has reached the point where power dissipation represents the major design constraint on advanced chips, due to a flattening of both chip power maximums and of V_{dd}.

2. In terms of chip microarchitecture, the above constraints have driven the silicon design community towards explicit parallelism in the form of multi-core processor chips, with flat or even declining clock rates.

3. For Exascale systems, power is perhaps the major concern, across the board. Real progress will be made when technical explorations focus not on power, but on "energy per operation" in regimes where there is still enough upside performance (clock rate) to moderate the explosive growth in parallelism. For silicon, this appears today to lie in low voltage, but not sub-threshold, logic running around 1-2 GHz in clock rate.

4. From an overall systems perspective, the real energy (and thus power) challenges lie not so much in efficient FPUs as in low energy data transport (intra-chip, inter-chip, board to board, and rack to rack), and in the accessing of data from dense memory arrays.

5. DRAM memory density growth is slowing, because of both a flattening in the basic bit cell architecture and because of growing concerns about the increasingly fine features needed within the memory arrays. In fact, while in the past, DRAM has led the industry in improving feature sizes, it is now flash that will drive DRAM.

6. The voltage used with the DRAM memory structures has long since flattened at a much higher level than that for logic of any kind, meaning that the energy per bit accessed inside commercial products will see little decline in the future.

7. Modern commodity DRAM chip architectures have major energy inefficiencies that are built in because of the protocols that have developed to communicate with them. With current memory system architectures, the transmission of addresses and commands is replicated to multiple chips, with each chip then internally accessing and temporarily storing much more data than is passed to the outside world. While there appears to be nothing technologically that stands in the way of rearchitecting DRAMs into a much more energy efficient form, the cost-driven nature of the commodity DRAM business will preclude that from happening on its own accord.

8. Flash memories hold significant density advantages over DRAM, but their currently relatively high write power and relatively limited rewrite lifetimes preclude their serious use in Exascale systems. However, it does appear that both problems may be solvable by relaxing retention times. This, however, requires rearchitecting both the devices and the chip architectures, something that is difficult to do in the cost-driven commercial market.

9. A variety of novel non-silicon devices and chip architectures have been proposed, with perhaps the greatest potential coming from those that can implement dense non-volatile memory structures, particularly ones that can be built in multiple layers, especially above conventional logic. As with the optical devices, however, there is still significant development work needed to bring them to commercialization, and significant architectural work to determine whether, and how best, to marry them with conventional technology.

10. A variety of novel on-chip optical devices have been prototyped, but before definitive statements can be made about their real potential, complete end-to-end energy per bit cost estimates must be made and compared to advanced all electrical protocols in a full system context. This includes the electrical costs of serializing parallel data from a conventional core to a high speed serial stream (and then back again at the other end), the fixed overhead costs of the light sources, the costs of switching the photonic routers (especially those distant from the source and to which routing information must be sent), and in providing appropriate temperature control for the optical devices, especially as large numbers of wavelengths are employed. In addition, these devices are just now being prototyped at low levels of integration, and there is still significant work that needs to be done to complete their characterization and optimize their designs and architectures for commercialization, especially by 2015.

11. Conventional spinning magnetic disk drives continue to advance in density, although latency for random accesses has been flat for years, and data rates, while growing, are still no match for solid state memories of any kind. However, at least through 2015 they seem to continue to hold an edge in overall physical densities over alternative emerging mass storage technologies.

12. A variety of alternative chip packaging and cooling technologies are emerging that may prove useful in moving memory and logic closer together, particularly in ways that lower the energy per bit transported, and thus result in significantly lower system power. Leveraging such technologies, however, requires rearchitecting the underlying chips.

13. Both fault mechanisms and fault rates will degrade as we go forward. Silicon below 45 nm will begin to exhibit more instabilities and wearout mechanisms that will be exacerbated

by lower voltages with less margin. Many of these effects will be temperature and/or time-dependent, such as the variable retention bit problem being observed today in DRAMs. All these, especially when coupled with the massive numbers of components in the data center scale systems, will introduce more complex failure patterns and higher FIT rates into system designs.

14. From the three strawmen system architectures explored, the heavy weight strawman based on leading edge commercial microprocessors, the light weight strawman based on lower power and simpler multi-core scalable compute chips, and the aggressive design based on low voltage logic and minimal overhead, the following observations are made:

    - Designs based from the beginning on massive replication of small chip sets with tailored and balanced design characteristics are by far more energy (and thus power) efficient. This was obvious from the light weight and the aggressive strawmen projections.
    - Reducing energy overhead costs in CPU microarchitecture is a necessity, and results in very simple cores that are then massively replicatable on a processor die.
    - Reducing the on and off-chip data transport costs is crucial, with low swing signalling a necessity.
    - A combination of memory chip architectures, off-chip contacts, and chip-to-chip data transport energy costs will tend to keep the number of DRAM die that can be supported by a single processor die in a dense configuration to a relative handful, meaning that the intrinsic bytes to flops ratio of future Exascale systems is liable to be significantly lower than that seen on classical computers.
    - The aggressive strawman design is not possible without a rearchitecture of the DRAM chips to solve the above problems, and a packaging scheme that allows lower energy chip-to-chip transport to occur.
    - Integrating significant routing capabilities into each processing die seems to pay significant dividends in reducing overall system power, especially if high bandwidth pGAS systems are desired.
    - Regardless of architecture, massive concurrency that is largely visible to, and must be expressed by, the program seems inevitable. When overheads for latency management are included, the total number of threads that an application may have to express for execution on data center scale problems will reach into the billions.
    - This explosion in concurrency will also exhibit itself at both the departmental and embedded scale. The numbers present at the departmental scale will rival those expected in the near-term Petascale systems, meaning that such systems will be unusable unless the heroic programming efforts needed for today's supercomputers can be greatly simplified. While not as severe at the embedded scale, there still will be the emergence of the need to express embedded applications in a several hundred-way parallel fashion, something that is not common today.
    - There are at least a few applications such as WRF, AVUS, and HPL, that appear scalable to an "exa" level at reasonable efficiencies, even with the relatively low bytes to flops ratios that are liable to be present.

Chapter 8

Exascale Challenges and Key Research Areas

The goals of this study were two-fold: determine what are the major technological barriers to enabling Exascale systems by 2015, and suggest key research directions that should help accelerate the reduction or elimination of these barriers.

As reference, Figure 8.1 (a variant of Figure 7.11) places one possible specific goal of an exaflops in a data center class system in the context of the current Top 500 projections from Section 4.5, and the evolutionary strawmen projection of Section 7.2. Assuming that Linpack performance will continue to be of at least passing significance to real Exascale applications, and that technology advances in fact proceed as they did in the last decade (both of which have been shown here to be of dubious validity), then while an Exaflop per second system is possible (at around 67MW), one that is under 20MW is not. Projections from today's supercomputers ("heavy weight" nodes from Section 7.2.1 and "lightweight" nodes from Section 7.2.2) are off by up to three orders of magnitude. Even the very aggressive strawman design of Section 7.3, with processor, interface techniques, and DRAM chip designs we have never tried commercially at scale, is off by a factor of greater than 3 when an upper limit of 20MW is applied, and this is *without* some major considerations discussed in Section 7.3.9 such as a very low memory to flops ratio. This gap will get even worse when we consider more stressing applications.

While this result is disappointing at the data center scale, the aggressive strawman design by itself does indicate at least a 3X better potential than the best of the extrapolations from current architectures – a significant advance in its own right. Further, a 3X improvement in terms of power at the departmental scale is of potentially tremendous commercial value, since it would mean that today's data center class Petascale system will fit (aggressively) in 2015 into a very small number of racks. This in turn has the breakthrough potential for growing the Petascale user community many-fold.

Likewise at the embedded level, the techniques that make the aggressive strawman possible seem to offer the potential for an order of magnitude increase in power efficiency over today. This alone will enable a huge increase in embedded computational potential.

The conclusion from this is that regardless of the outcome of the data center, the results of this study indicate that if the challenges described here can be met, then there is a very significant payoff potential across the board. In particular, the study group's conclusion was that there are four such very specific major consensus **challenges** for which there is no obvious technological bridge with development as usual, and/or that will need particular attention to ensure that they do not rise to the level of a showstopper. These challenges focus on energy, memory, concurrency,

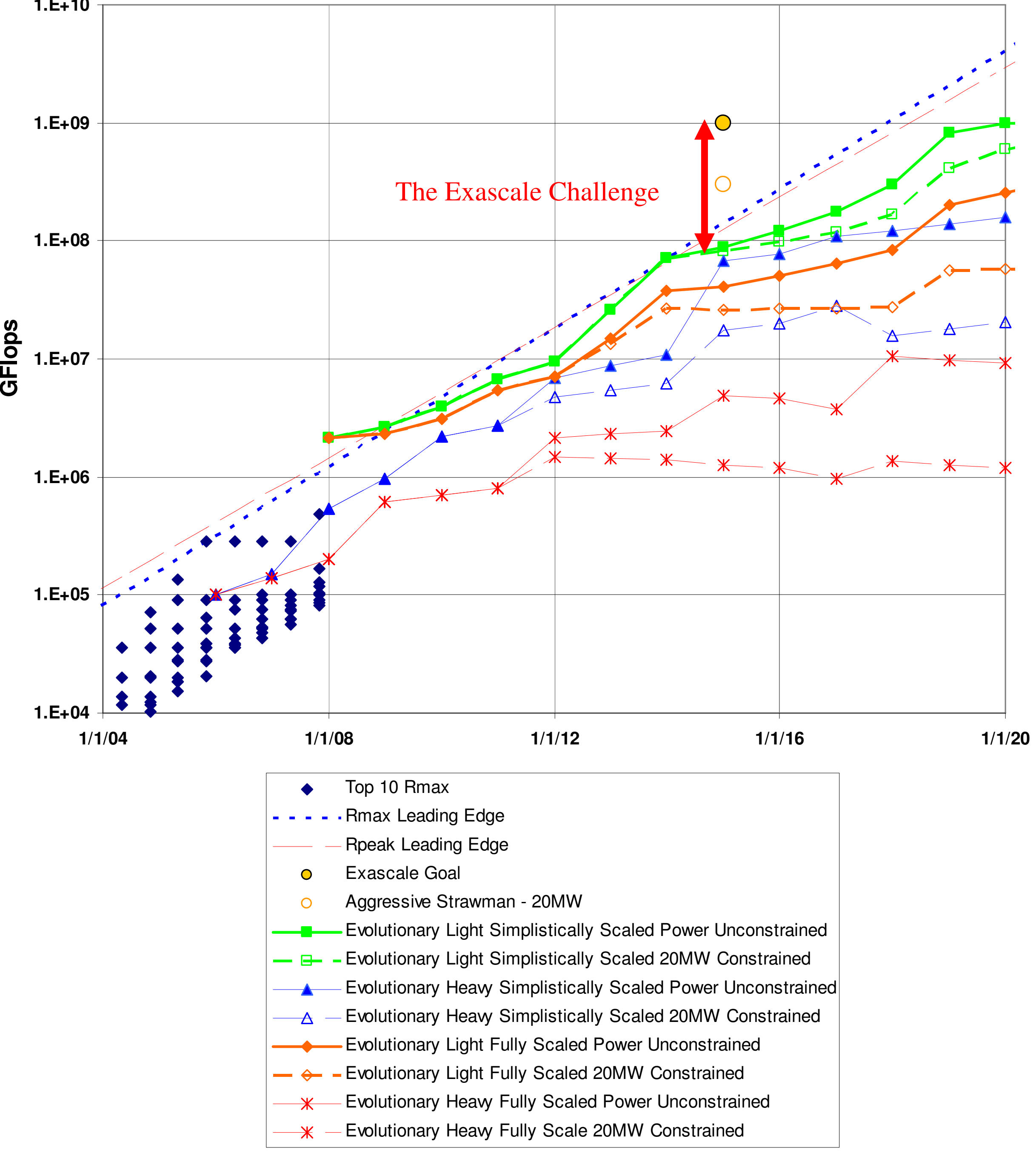


Figure 8.1: Exascale goals - Linpack.

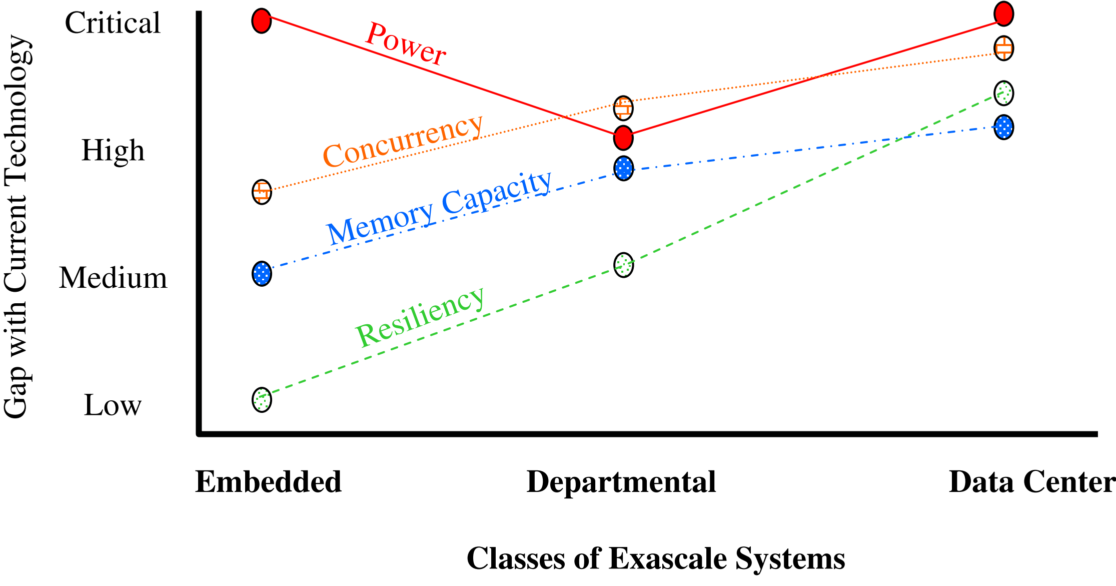


Figure 8.2: Critically of each challenge to each Exascale system class.

and overall system resiliency. Each of these challenges are discussed in greater detail in Section 8.1. However, Figure 8.2 diagrams notionally how important each of these challenges is to each class of Exascale system. As can be seen two of them, power and concurrency, are or real concern across the board, while the other two become of increasing importance as the size of the system increases.

The study group also developed a list of research thrust areas (Section 8.2) where significant progress in the next few years could go a long way towards reducing the barriers caused by the challenges, and thus enabling Exascale systems. Table 8.1 summarizes the relationship between these barriers and the proposed research directions. Its columns (challenges) are discussed in more detail in Section 8.1, and the rows (thrusts) in Section 8.2.

8.1 Major Challenges

As was demonstrated in Chapter 7, there are multiple areas where the natural progression of technology appears inadequate to enable the implementation of systems that come anywhere achieving Exascale attributes in the desired timeframe. Each of these thus represent a major **Challenge**, and is discussed individually below.

8.1.1 The Energy and Power Challenge

The single most difficult and pervasive challenge perceived by the study group dealt with **energy**, namely finding technologies that allow complete systems to be built that consume low enough total **energy per operation** so that when operated at the desired computational rates, exhibit an overall **power dissipation** (energy per operation times operations per second) that is low enough to satisfy the identified system parameters. This challenge is across the board in terms of energy per computation, energy per data transport, energy per memory access, or energy per secondary storage unit. While there has been a recognition of this challenge before, the focus has been predominately on the energy of computation; the real impact of this study is that the problem is much broader

Exascale Research Thrust	Challenges: Power & Energy Crit	Power & Energy Gap	Memory & Storage Crit	Memory & Storage Gap	Concurrency & Locality Crit	Concurrency & Locality Gap	Resiliency Crit	Resiliency Gap
Technology & Architectures	High	High	High	Med			High	Med
Architectures & Programming Models	Med	Med			High	High	High	Med
Algorithms & Applications Development	Low	Med	Med	Med	High	High	Low	High
Resilient Systems	Med	Med	Med	Med			High	High

Crit. = criticality of thrust area to the Challenge for widespread solutions.
Gap = the gap between the maturity of existing research and the needed solution.
A "Med." in the Hardware row reflects existence of lab prototype devices.
Blanks for any entry imply that the interaction is indirect.

Table 8.1: The relationship between research thrusts and challenges.

than just "low-power logic" - it truly is in the entire system. In fact, in many cases it has be become clear to the panel that the non-computational aspects of the energy problem, especially the energy in data transport, will dwarf the traditional computational component in future Exascale systems.

While the resulting power dissipation is a challenge for all three classes of Exascale systems, it is particularly so for the largest data center class. The design target of 20MW for the electronics alone was chosen to have some flexibility above that of today's largest systems, but still not be so high as to preclude it from deployment in anything other than specialized strategic national defense applications. Figure 8.3 presents some historical data along with a "trend line" and the Exascale goal assuming a Linpack reference. The reference metric in this case is "Gflops per Watt," where the power is taken over the entire system, not just the floating point units. As can be seen, even if the positive trends of the last two decades were capable of being maintained, in 2015 power would still be off by between a factor of 10 and 100.

As discussed in Section 7.2.1, for at least a "heavy node" system architecture, although the most optimistic of possible projections barely makes the current trend line, more realistic estimates seem to be barely more than flat from today. The "light node" system architecture is better, but still is a factor of 10 off. Even the very aggressive strawman design of Section 7.3, with processor and interface techniques and DRAM chip designs we have never tried commercially at scale, is off by a factor of greater than 3, and this is *without* some major considerations discussed in Section 7.3.9 such as a very low memory to flops ratio.

As discussed in the roadmaps of Chapter 6, a variety of factors are responsible for this effect, especially the flattening of V_{dd} and the concurrent flattening of clock rates, that in turn force performance growth by physical parallelism alone and increasing power-consuming area. Further, as the point studies and strawmen of Chapter 7 indicated, even aggressive circuit and architecture approaches to lowering energy do not close the gap in a timely fashion.

The following subsections discuss individual components of this power and energy challenge in detail.

8.1.1.1 Functional Power

First, if a *peak exaflop* per second was the metric (i.e. a Linpack-based extrapolation of today's Top 500 rankings), then silicon based floating point units (FPUs), by themselves, exceed 20 MW by

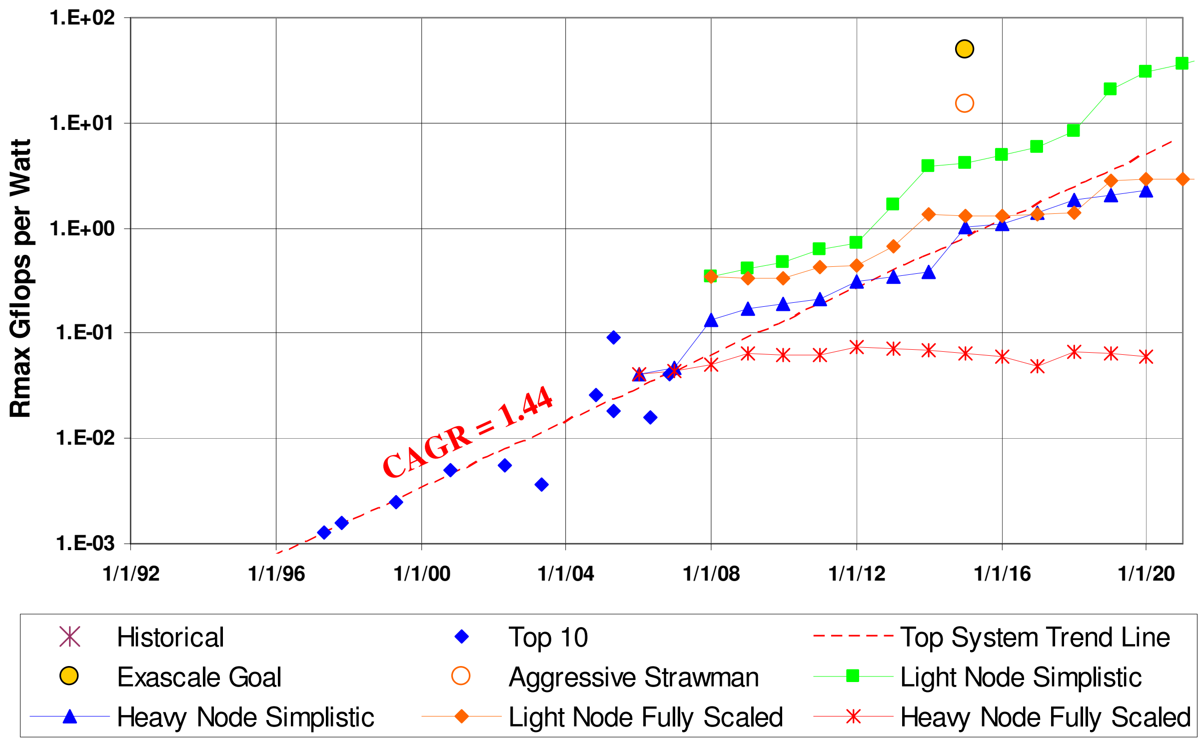


Figure 8.3: The power challenge for an Exaflops Linpack.

themselves using today's high performance logic (Figure 7.1). We are still around 10 MW even if aggressive circuit designs and lowered clock rates were used that reduced V_{dd} to 2X the threshold, and then somehow are used at 100% efficiency - regardless of the resultant explosive growth in required parallelism to literally hundreds of millions of simultaneous operations.

Again, as discussed in prior chapters, both the probability of 100% efficient use of numbers of FPUs far in excess of anything we've managed to date, and the applicability of floating point operations alone as metrics for successfully deployable Exascale applications are indicators that this problem is in reality even more severe than depicted.

8.1.1.2 DRAM Main Memory Power

Next, if DRAM is used for main memory, then its power is a function of:

- Total memory capacity: this includes the continuous energy needed for refresh.
- Total number of independent accesses that must be maintained per second: each memory access that escapes any cache hierarchy must activate some memory bank on some memory chip to retrieve the necessary data.
- The number of bits read per access versus the number of bits actually transferred.
- Data bandwidth needed to move the accessed data from the on-chip memory banks to the off-chip contacts of the DRAM (regardless of how such DRAM are packaged).

As discussed in Section 7.1.4, such numbers may be reasonable for very small memory capacities and non-memory-intensive applications, but rapidly run out of control when either memory

capacities exceed around 5PB or applications such as GUPS are the driving factors.

8.1.1.3 Interconnect Power

Interconnect power comes in several forms: on-chip, chip to nearby chip, board to board, and rack to rack. Even with a variety of circuit techniques (such as low-swing on-chip interconnect), emerging chip-chip techniques (such as through via), and optical signalling techniques, the energy to move one bit through any one level was on the order of 1-3 pJ (Section 6.5). For any one level of the transport path that requires on the order of an exabyte per second, the power thus balloons to 10-30MW, and multiplies as multiple types of interfaces must be crossed by the same data.

8.1.1.4 Secondary Storage Power

Today the only media for scratch and persistent storage is disk, and projections for the highest density per watt in the 2014 timeframe are for up to 5MW for each exabyte (Section 6.4.1.2). Further, the actual amounts of such storage needed is a function of the main memory capacity, which if system implementations move into the larger regimes to support emerging and stressing applications, may result in upwards of 100EB of such storage, as discussed in Section 7.1.6.

8.1.2 The Memory and Storage Challenge

The second major challenge is also pervasive, and concerns the lack of currently available technology to retain at high enough capacities, and access information at high enough rates, to support the desired application suites at the desired computational rate, and still fit within an acceptable power envelope. This information storage challenge lies in both **main memory** and in **secondary storage**. By main memory we mean the memory characterized by the ability of a processor to randomly access any part of it in relatively small increments with individual instructions. By secondary storage we mean the memory where data is typically accessed in "blocks" with access handled by subroutines, not individual instructions. This includes both **scratch storage** for checkpoints and intermediate data files, **file storage** for more persistent data sets, and to some extent **archival storage** for the long term preservation of data sets.

While this challenge is felt by all Exascale classes, it is particularly severe at the data center scale, where affordable capacities are probably at least an order of magnitude less than what is needed for the projected application suite.

8.1.2.1 Main Memory

DRAM density today is driven by both the architecture of an individual bit cell (and how the capacitor that stores the information is merged into the access logic) and the basic metal-to-metal feature size of the underlying level of lithography (how close can individual row and column access wires be placed). Today, and for the foreseeable future, the architecture of a basic cell will be stuck at $6F^2$, where F is the technology feature size (1/2 pitch of M1). Second, this feature size is driven today not by DRAM but by flash memory technology, and there are serious concerns as to how this can be achieved below 45 nm. Together, this makes the ITRS projections[13] of 1GB per commodity DRAM chip in the desired timeframe rather aggressive. However, even at 1GB per chip, each PB of main memory translates into 1 million chips, and with realistic capacity needs for data center class systems in the 10s to 100s of PB, the number of such chips grows excessively, resulting in multiple power and resiliency issues, not to mention cost.

Additionally, there is a significant challenge in bandwidth, that is how to get enough data off of each memory die to match the desired rates. While as shown in Section 7.1.4.2, it is possible with projected silicon technology to provide enough such bandwidth, the resulting chips look nothing like the commercial high-volume parts of today, or the next decade. Thus even if DRAM capacities were sufficient, there is a significant challenge in how such chips should be organized and interfaced with other system components, and in how such chips could be brought to market in a way that is economically competitive with today's commodity DRAM.

The challenge thus is to find a way to increasing memory densities and bandwidths by orders of magnitude over that which is projected for 2014, without running into other problems. The study considered flash memory in various forms as an existing technology that might be employed somehow, since it has both significant density and cost advantages over DRAM. However, its slow access times, and limited rewrite lifetimes made it unsuitable for at least the fast random access part of the main memory role. The study did, however, encounter several other emerging non-silicon memory technologies, as described in Section 6.3.5, that have the potential for such density gains, but not on a commercialization path that today will result in useable devices in the appropriate time frame. Further, it is unclear how best to architect memory systems out of such devices, as replacements for DRAM, or perhaps as a new level of memory within the overall information storage hierarchy.

8.1.2.2 Secondary Storage

As discussed previously (Section 6.4.1), current scratch and file systems have been implemented with the same disk technology that has Moore's law-like growth in density, and some increase in bandwidth, but has been essentially stagnant in seek time for decades. Also, as described in Section 5.6.3, growth in storage requirements above the main memory level has been continuous, with scratch needs growing at a rate of 1.7X to 1.9X per year, overall projections for scratch in the 20-40X main memory, and that for file systems in the 100X range.

For data center class systems, projected disk storage density growth using these factors is acceptable as long as the implemented main memory is in the low petabyte range. However, there are significant Exascale applications with needs way beyond a few petabytes that would in turn make achieving sufficient secondary storage a real difficulty, particularly in complexity and in power.

While flash memory may have a role to play here, flash as currently designed does not have a sufficient level of rewrites to make it acceptable as is. The alternative memory technologies mentioned as possibilities for main memory are also possibilities, but again there are currently significant challenges to making them viable enough, with the right systems architectures, to take on such replacement roles.

A second consideration deals again with bandwidth. For scratch storage the need to do check-pointing requires copying the bulk of main memory to disk, usually with the application suspended in the process. Today, for memory-rich systems this process often takes up to 50% of the execution time, and with the stagnation of disk bandwidth, this fraction could grow even higher, leaving no time for computation to advance before another checkpoint is required. Thus, while extrapolation from the strawman indicates that with the sheer number of disks needed to provide such backup for the 3-4PB main memory range may provide sufficient bandwidth, this may not hold true if main memory needs on a per operation basis grow to ratios commensurate with today's systems.

Another major concern with storage is with the growing application-level need to perform mall unaligned I/O. Because of the flat seek times foreseen for future disks, achieving the peak bandwidths assumed above can only be done by performing all transfers in large megabyte or bigger sized blocks, and aligned with the basic disk block size. Unfortunately, many of today's

more critical applications perform file I/O that does not have such characteristics. As the degree of concurrency grows as discussed above, such random unaligned I/O will become even more prevalent, since it will become infeasible to require huge numbers of independent threads to synchronize at a point where data can be buffered into bigger segments than what each thread processes.

Finally, the managing of **metadata** (file descriptors, i-nodes, file control blocks, etc.) associated with data structures on disks is beginning to severely hamper Petascale systems. Before any I/O requests can actually touch a disk to perform the data transfers (regardless of the transfer length), the run time must determine on which disk a particular block of data in a certain file is located, whether or not some other threads are already accessing any overlapping data, and how to schedule the transfer to minimize seek time. With today's high end applications often opening literally millions of files during a single run, and with the potential for hundreds of thousands to millions of physical disks to maintain just catalogs, the amount of such metadata that must be accessed and sorted through is becoming enormous, and a severe impediment to maintaining high levels of efficiency of processing. Many of the emerging Exascale applications, especially ones maintaining massive persistent databases, will make this process even worse.

8.1.3 The Concurrency and Locality Challenge

As discussed earlier, concurrency can be measured in three ways:

- The total number of operations (such as floating point operations) that must be instantiated in each and even cycle to run the applications at Exascale speeds.
- The minimum number of threads that must be run concurrently to provide enough instructions to generate the desired operation-level concurrency.
- The overall thread-level concurrency that is needed to allow some percentage of threads to stall while performing high-latency operations, and still keep the desired dynamic thread concurrency.

8.1.3.1 Extraordinary Concurrency as the Only Game in Town

The end of increasing single compute node performance by increasing ILP (Instruction Level Parallelism) and/or higher clock rates has left explicit parallelism as the only mechanism in silicon to increase performance of a system. Thus in the embedded class, what was a single core processor will rapidly become a 1,000 core device. In the departmental scale, downsizing a Petascale machine with perhaps a large fraction of a million function units to a few racks will still require a million function units. Further, at the data center class, scaling up in absolute performance will require scaling up the number of function units required accordingly (Section 7.1.2) into the billion range.

Efficiently exploiting this level of concurrency, particularly in terms of applications programs, is a challenge for which there currently are no good solutions. Solving it requires that

- the simplicity of programming an application for a medium sized cluster of today's computers becomes as easy as programming an application today for a single core,
- the heroics needed to produce applications for today's supercomputer Petascale systems be reduced to the point where widespread departmental systems, each with different mixes of applications, are feasible,
- and that some way is found to describe efficient programs for systems where a billion separate operations must be managed at each and every clock cycle.

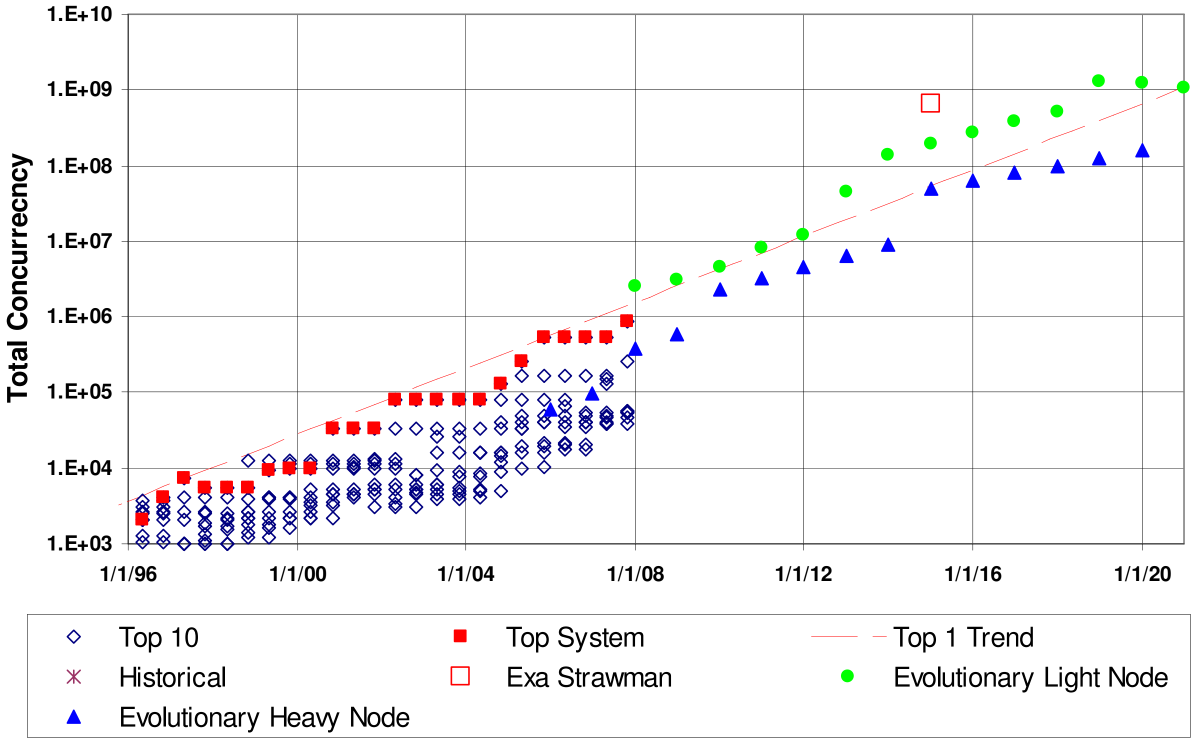


Figure 8.4: The overall concurrency challenge.

Each of these stress the state of the art beyond today's limits; especially for the data center class of systems there is little idea of even how to create "heroic" programs that actually work efficiently.

Figure 8.4 attempts to place the concurrency challenge in perspective, where concurrency is defined in Section 4.5.3 as the total number of operations (flops in this case) that must be initiated on each and every cycle. This figure is drawn by taking the trend line for concurrency from Section 4.5.3.4, and including the heavy and light node systems projection from Sections 7.2.1 and 7.2.2, the light node system projection from Section 7.2.2, and the estimate from the strawman of Section 7.3. As can be seen, even if the current trends are maintainable, billion-way concurrency will be needed for exaflops systems, and the 2015 strawman simply requires it about 5 years earlier. Further, and equally important, this level of concurrency is three orders of magnitude larger than what we have today, or expect to see near term.

Figure 8.5 graphs a similar analysis, but assuming an architecture like today's where the major unit of logic is a single-threaded processor. As discussed in Section 4.5.3.1, here there is no clean trend line that fits the top system, albeit there is a super-exponential trend in the median of the Top 10. In any case, the strawman estimate three orders of magnitude higher than any system today. Given that as the chart shows it took a decade to be able to efficiently utilize a 10X increase in processor parallelism, to expect that 1000X can be handled in less than that is a long stretch.

Making these issues of concurrency even harder is the other characteristic of the memory wall - latency. We are already at or beyond our ability to find enough activities to keep hardware busy in classical architectures while long time events such as memory references occur. While the flattening of clock rates has one positive effect in that such latencies won't get dramatically worse by themselves, the explosive growth in concurrency means that there will be substantially more

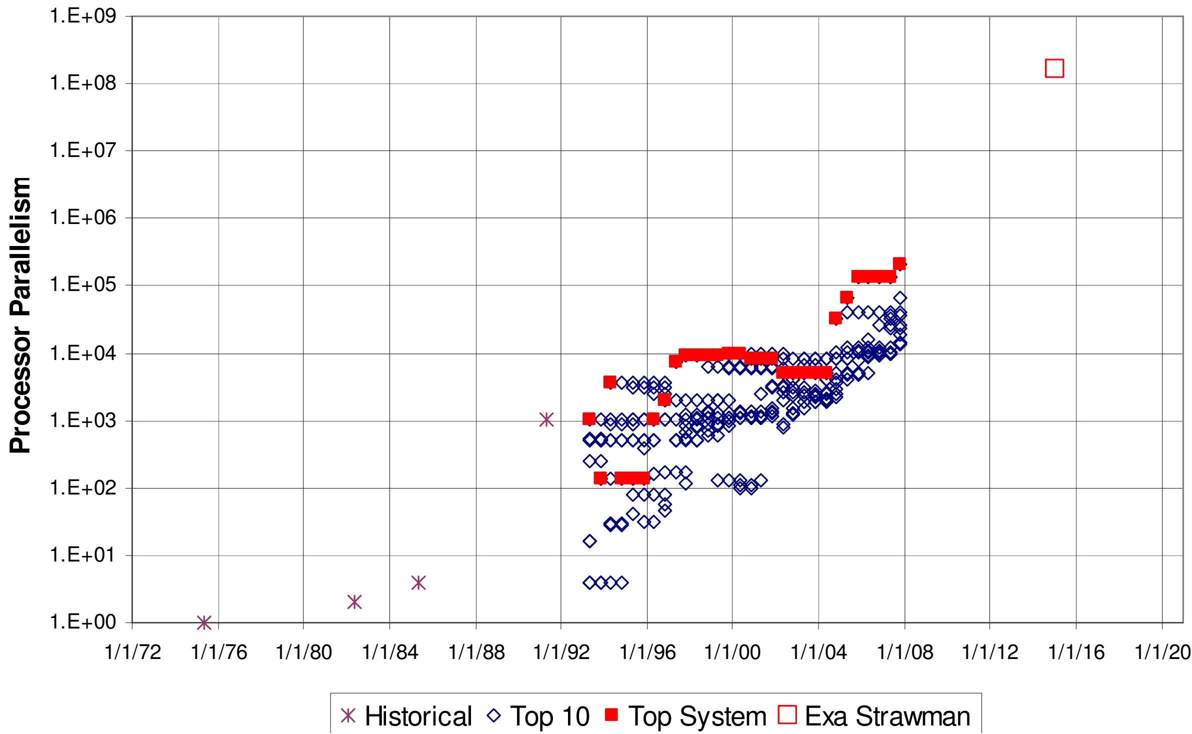


Figure 8.5: The processor parallelism challenge.

of these high latency events, and the routing, buffering, and management of all these events will introduce even more delay. When applications then require any sort of synchronization or other interaction between different threads, the effect of this latency will be to exponentially increase the facilities needed to manage independent activities, which in turn forces up the level of concurrent operations that must be derived from an application to hide them.

Further complicating this is the explosive growth in the ratio of energy to transport data versus the energy to compute with it. At the Exascale this transport energy becomes a front-and-center issue in terms of architecture. Reducing it will require creative packaging, interconnect, and architecture to makes the holders for the data needed by a computation (the memory) to be energy-wise "closer to" the function units. This closeness translates directly into reducing the latency of such accesses in creative ways that are significantly better than today's multi-level cache hierarchies.

8.1.3.2 Applications Aren't Going in the Same Direction

Section 5.8 discussed the expected future of the scalability of applications. The summary, as pictured back in Figure 5.16, is that as we look forward both applications and the algorithms behind them seem to have some definite limits in both concurrency and locality. Overlaying on this the hardware trends as we discussed above, we get Figure 8.6, where the gap between what we expect to be able to extract from applications and what hardware as we know it today seems to be growing.

Thus a significant challenge will be in developing basic architectures, execution models, and programming models that leverage emerging packaging, signalling, and memory technologies to in fact scale to such levels of concurrency, and to do so in ways that reduce the time and energy de-

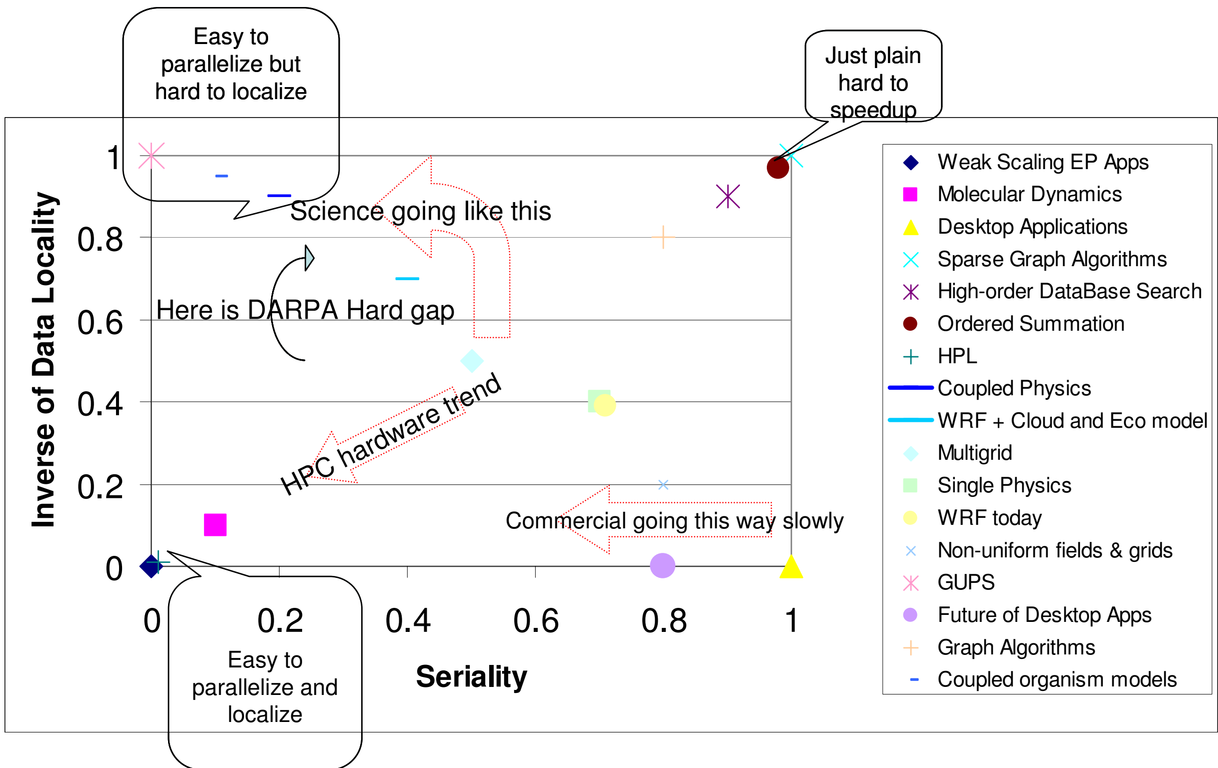


Figure 8.6: Future scaling trends present DARPA-hard challenges.

mands of access remote data in ways that applications can actually utilize the resulting concurrency in an efficient manner.

8.1.4 The Resiliency Challenge

Resiliency is the property of a system to continue effective operations even in the presence of faults either in hardware or software. The study found multiple indications that Exascale systems, especially at the data center class, will experience more and different forms of faults and disruptions than present in today's systems, including:

- Huge numbers of components, from millions to hundreds of millions of memory chips to millions of disk drives.
- Running interfaces at very high clock rates to maximize bandwidth, thus increasing both bit error rates (BER) and actual bit errors on data transmissions.
- Where leading edge silicon feature sizes are used, a growing variation in device properties across single die will increase the variation in performance characteristics of even identical circuits at different places on the die.
- In many advanced technologies there are increasing "wear-out" mechanisms in play that bring in aging effects into the fault characteristics of a device (for example, the buildup of stray charge on a gate).

- Smaller feature sizes, with less charge on a device, can in many cases increase the sensitivity of devices to single event upsets (SEU) due to cosmic rays and other radiation sources.
- Many of the technologies have significant sensitivities to temperature, making their performance characteristics a dynamic function of surrounding activity and power dissipation.
- Running silicon at lower voltages will lower power but also decreases margin, increasing the effects of noise sources such as from power supplies, and thus increasing transient errors.
- The increased levels of concurrency in a system greatly increases the number of times that different kinds of independent activity must come together at some sort of synchronization point, increasing the potential for races, metastable states, and other difficult to detect timing problems.

When taken together and placed in a system context, many of these observations tend to reinforce themselves and complicate the overall system. For example, checkpointing in high end applications requires dumping large amounts of information (in some "consistent" state) from one level of the memory hierarchy to another. How often this occurs is a function of the mean time to disruption due to a fault from which recovery is not possible. As the memory footprint of applications grow, the amount of memory that has to be fault free grows, as does the time to copy it out. The first decreases the time between checkpoints, the second increases the non-productive time to do the checkpoint, which in turn reduces the time the hardware is active, which in further turn increases the number of hardware units needed to achieve an overall performance goal, and which further increases the fault rate. Anything that an application can do to either checkpoint smaller footprints and/or indicate a willingness to ignore certain classes of errors in certain regions of code is thus clearly of huge benefit, but is currently not at all part of any application design and coding process.

Further, while many of these fault characteristics can be mitigated at design time by approaches such as ECC or duplication, not all can. As an example, variations in device parameters across a multi-core processor die results in different cores that draw different amounts of power (and thus heat differently), with different delay and thus clocking characteristics, and different sensitivities to noise, all of which may change with both local heating or aging effects. If advanced 3D chip stacking technologies are employed, then other die (with their own and different variational characteristics) will also affect and be affected by these variations. This makes the job of deciding which cores at what clock rates can be used safely a daunting real-time problem.

8.2 Research Thrust Areas

Overcoming these challenges and concerns will take a coordinated portfolio of research that is focused on some fundamental topics, but must be done within a larger context that helps direct them to Exascale-specific objectives. The study group thus looked at a whole range of topics that seemed to be of most potential impact, and grouped then into four cross-cutting **thrust areas**:

1. Co-development and optimization of Exascale Hardware Technologies and Architectures
2. Co-development and optimization of Exascale Architectures and Programming Models
3. Co-development of Exascale Algorithm, Applications, Tools, and Run-times
4. Coordinated development of Resilient Exascale Systems

This distinction between research thrust areas and challenges was deliberate; all four of the challenges are inter-related sets of problems, and solutions that address the problems represented by one challenge often affect those of other challenges. Further, even when we address the problems of just one area, solving them is not localized to a single research topic but requires co-consideration of multiple levels of the design hierarchy.

Table 8.1 overviews this relationship. The four major challenges from Section 8.1 make up the columns of Table 8.1, with the rows representing the three Thrust Areas (each discussed in detail below). Each entry in this table has two values representing criticality and gap.

Criticality is an indication by the study group as to how important effective research done in the designated research thrust is to solving the problems of the designated challenge. Thus a high value to criticality indicates the group's collective view that research in this area is absolutely crucial to solving the key problems of the challenge.

Gap is a collective view by the group of the maturity level of the current leading research in the area vis-a-vis the maturity deemed necessary to achieving Exascale systems that solve the challenge in the appropriate time frame. Thus a high value indicates the group's feeling that "business as usual" is highly unlikely to lead to viable solutions in the desired time frame.

Thus entries of the "High-High" rankings are indications that the study group believed that research into the specified areas is absolutely vital to solving the challenges, but where the current directions in research are highly unlikely to bridge the gap. These are areas where in particular additional research focus is liable to have the highest payoff.

Entries that are left blank in the Table are not areas where the group felt that there was no value to the research, only that the interaction between research and problem solution was at best indirect.

8.2.1 Thrust Area: Exascale Hardware Technologies and Architecture

In many areas it is clear that current technologies associated with the implementation of the hardware parts of systems (logic, memory, interconnect, packaging, cooling) is inadequate to solve the overall challenges, and significant research is needed. However, it is equally clear to the group that doing so in the absence of a deep understanding of how to architect such systems is liable to lead to perhaps interesting but ineffective new devices, at least for Exascale systems. Further, from experience it is also clear that today's system architectures are liable to be totally unsuited for optimizing the characteristics of such new device-level technology, and new system architectures are really needed.

Thus by grouping research topics that look at the interaction between architectures and device technologies, the group is saying that these two topics *must* be studied and developed together. Only when new device technologies are developed that blend in with system architectures that leverage their special characteristics are we liable to see overall success in solving the challenges.

As an aside, for this row in Table 8.1 an entry of "Medium" in a gap indicates that in the group's view there are laboratory demonstration devices in existence now that look promising, but that either their current path to commercialization is insufficient to get them there in time, or there is a significant lack in understanding on how to adjust Exascale architectures to leverage their properties, or (more frequently) both.

The following subsections describe several potential research topics that might fit this area.

8.2.1.1 Energy-efficient Circuits and Architecture In Silicon

Even given the promise of several non-silicon technologies, our 40 year investment in silicon means, however, that an aggressive attempt must be made to utilize silicon in some form. The challenge of building an Exascale machine (of any class) in silicon of any form that consumes an amount of power reasonable for its scale must thus be addressed at all levels. At the circuit level, there are significant opportunities to improve the energy efficiency of the basic building blocks (logic, communication, and memory) from which computing systems are built. By co-optimizing these circuits with energy-efficient architecture, even greater energy savings may be realized. Both levels of opportunities will be needed, and must be addressed together.

Most circuits and architectures today are optimized for speed, not energy. This is reflected in the supply voltage at which the circuits are operated, the threshold voltage(s) selected for the devices, the pipeline depth of the circuits, and the circuit concepts and topologies employed. We expect that substantial savings can be realized by re-optimizing all of these parameters for operations per Joule, rather than for minimum latency.

A key starting point for development of efficient circuits is in setting the power supply voltage (V_{dd}) and the device threshold voltages (V_{TN} and V_{TP}). The ITRS projections for the 32nm node project a supply voltage of 0.9V and a threshold voltage of 0.3V. Because energy scales as V^2, significant energy savings can be realized by reducing the supply voltage. The strawman of Section 7.3 reduces operation energy by a factor of nearly three by reducing V_{DD} to 0.6V. Some preliminary studies (see Section 6.2.2.3) suggest that for static CMOS logic, the supply voltage that optimizes operations per Joule is just slightly above the threshold voltage (320mV for $V_T =$ 300mV see Figure 6.10, repeated here as Figure 8.7.

The optimization is more involved, however, because threshold voltage is also a free variable. As the supply voltage is reduced, threshold voltage should also be reduced to the point where leakage power and dynamic power are balanced. The optimum setting of V_{dd} and V_T is also dependent on the circuit being optimized and its duty factor. The activity factor largely drives the balance of dynamic and static power. Also, some efficient circuit forms require some headroom - requiring higher supply voltages. Moreover a given circuit is not restricted to a single supply voltage or a single threshold voltage. Multiple threshold voltages are already used to advantage in modern circuits and multiple supply voltages may be employed as well. A study of energy efficient circuits needs to start with a careful examination of supply and threshold voltages in the context of an Exascale system.

The circuits employed in modern computer systems can be roughly broken down into those used for communication, memory, and logic. We discuss each of these briefly.

Communication circuits move bits from one location to another in a system with different circuits being employed at different levels of the packaging hierarchy. Static CMOS drivers and repeaters are typically used to drive on-chip signals, while high-speed SerDes are often used to drive signals between chips.

The circuits used to transmit global on-chip signals are a great example of the potential energy savings from optimized circuits. On-chip signal lines have a capacitance of about 300fF/mm. With conventional full-swing speed-optimal repeaters, the repeater capacitance equals the line capacitance for a total of 600fF/mm. At the ITRS supply level of 0.9V, sending a bit on chip using conventional full-swing signaling requires about 0.5pJ/mm.

If we optimize on-chip transmission for energy rather than speed, we can significantly reduce the energy required to transport them. First, we reduce the signal swing V_S to a level which minimizes energy/bit. Note that this is not the minimum possible V_S, but rather the level which balances transmit energy (that reduces with V_S) against receive energy (which increases as V_S is reduced).

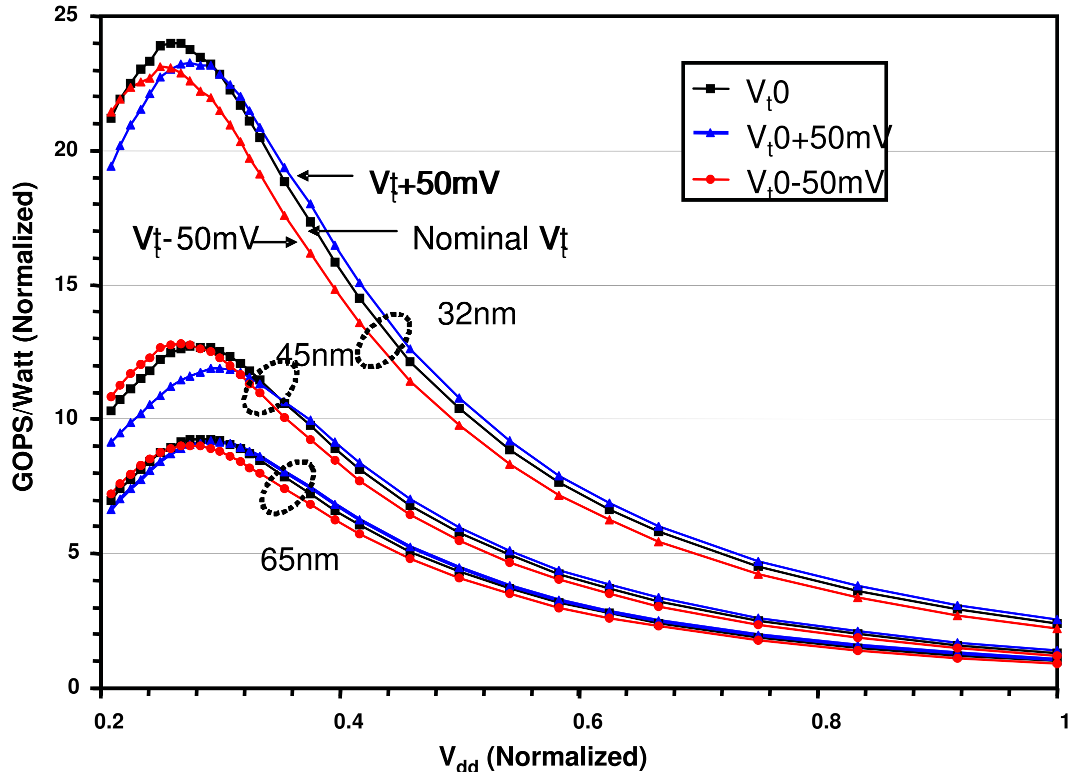


Figure 8.7: Sensitivities to changing V_{dd}.

With a V_S of 100mV, for example, generated from an efficient switching supply, the transmission energy would be about 6fJ/mm, a savings of nearly two orders of magnitude.[1] Optimizing repeater spacing, line geometry, and equalization for energy may also yield significant gains.

The serializer/deserializer (SerDes) circuits that are used for high-speed off-chip communication have energy that is dominated by timing, serialization, and deserialization. Over 2/3 of the power consumed by a recent energy-efficient SerDes was consumed by these functions - with the remaining 1/3 used by the actual transmitter and receiver. While the gains expected for off-chip communication are more modest than for on-chip, efficiency improvements of one order of magnitude or more are possible from improvements in these circuits and the protocols they implement.

It is clear that optimized communication circuits have the potential to reduce signaling energy by as much as two orders of magnitude. Such a dramatic change in communication cost changes the space of architectures that are possible, and greatly changes which architectures are optimal. Hence any architecture optimization must take optimized signaling into account.

Memory circuits move bits forward in time - storing data and later retrieving it. On the processor die memories are typically implemented today using 6-transistor SRAM circuits, while the bulk memory is usually implemented as 1-transistor DRAM. There is the potential for significant energy improvement in both. For the on-chip SRAM, access energy is dominated by charging and discharging bit lines. This can be reduced by either reducing voltage swing or by employing hierarchical bit lines to reduce the amount of capacitance switched. For the bulk DRAMs, bounding arguments suggest that one should be able to access a bit of DRAM for about 3pJ of energy, yet for conventional DRAMs access energy is about 60pJ/bit.

[1]The straw man of Section 7.3 assumes a less aggressive 20fJ/mm per bit.

As with communication circuits, optimized memory circuits can result in order-of-magnitude reductions in access energy which will have a substantial impact on machine architecture.

Logic circuits perform arithmetic and control the operation of a computer system. There are fewer opportunities to improve the energy of silicon logic circuits because static CMOS circuits are already highly efficient and designs for common arithmetic functions are already highly optimized. Gains may be realized by optimizing supply and threshold voltages. Additional gains may be realized through machine organization - e.g. using shallow pipelines and arithmetic unit organizations that reduce energy.

8.2.1.2 Alternative Low-energy Devices and Circuits for Logic and Memory

Given the criticality of driving energy per operation down, and the uncertainty surrounding silicon, the study group believes that it is equally important to develop an understanding of the potential for alternative device technologies to yield suitable low-energy high-performance circuits for both computational functions and storage. Equally important is sufficient density to meet or exceed that possible with traditional technologies.

The group believes that several such technologies may already exist, at different levels of maturity. RSFQ (Rapid Single Flux Quantum) devices have been demonstrated in several projects such as HTMT[140] in some real circuits at huge clock rates and very low power, but not in system architectures that deliver large memory capacities.

Additionally, several labs have begun discussing crossbar architectures from novel bi-state devices (see HP's SNIC[137] and IBM's SCM[61]) that can either provide very dense memory, or a dense new form of combinational logic with minimal active devices, or enable a new generation of reprogrammable logic that can work in conjunction with silicon devices underneath to reduce power by dynamic circuit reconfiguration. Again, however, what is lacking is a careful study of how best to utilize such devices in Exascale circuits to minimize total energy per operation while still having sufficient performance not to make the concurrency and capacity problems worse.

8.2.1.3 Alternative Low-energy Systems for Memory and Storage

As shown in the strawman of Section 7.3, power in the memory system is a severe problem. This power comes in several forms:

1. maintaining the actual storage arrays themselves, in whatever technology they are implemented.
2. accessing these arrays, and extracting the desired information.
3. maintaining "copies" of such data in other memories with better locality than their home.
4. moving data from arrays to off-chip (for memory-chip technologies)
5. moving data between chips, boards, and racks.
6. implementing the storage access protocols needed to trigger the transfer and synchronization of such data.

Clearly there are issues and tradeoffs to be made here in terms of density of storage medium and bandwidth, just as we have wrestled with for years. These get only dramatically worse when we attempt to use today's technologies and memory architectures in Exascale systems, especially the data center class.

While there is promise from several emerging technologies to address some of these issues, the only way to address the whole problem in a more than incremental way is to refocus in a serious fashion the whole notion of a memory hierarchy. The capabilities of some of the emerging technologies need to be blended with new architectural ideas that truly do leverage them in a way where they integrate with the other parts of the system to dramatically reduce energy. Section 7.5.4 is one example of an attempt to begin the exploration of such a space. In general, a complete exploration will require simultaneous attention to multiple dimensions, including:

- Considering new levels in the memory hierarchy where the characteristics of emerging or alternative memory technologies can bridge the gap between, for example, DRAM and disk storage, especially for some critical functions such as checkpointing and metadata management.
- Explicitly and pro-actively managing data movement between levels of the memory hierarchy to reduce power consumed by such movement.
- Rearchitecting conventional DRAMs in ways that greatly reduce the energy per bit access when used in massive memory arrays where high concurrency of access and bandwidth are required.
- Developing multi-chip and 3D packaging that allows chips of either of the above types to be efficiently stacked in ways that minimize the *total energy cost* of access, while still retaining the high access rates needed.
- Developing alternative memory access paths and protocols using photonics for example, that actually provide end-to-end reductions in both energy per bit accessed and time per access.
- Mixing and matching the right suite of storage devices, transport media, and protocols with the careful positioning of function units to minimize not only unnecessary copies, but also the total power needed to execute latency-sensitive pieces of code.

8.2.1.4 3D Interconnect, Packaging, and Cooling

The degree of difficulty posed by the packaging challenge depends on the memory bandwidth requirements of the eventual computer, as discussed in Section 7.3, and abstracted in Section 6.5. The lower end of the requirement, 44 GBps from the CPU to each of 16 DRAM die can most likely be satisfied with conventional high-end packaging. The high-end requirement of 320 GBps to each of 32 DRAMs is well beyond the means of any currently available packaging technology. Some embedded applications were investigated to obtain another perspective on the possible size requirement. As discussed elsewhere in this document, the usual requirement of 1 Byte per second and 1 Byte of memory for every FLOPS, would require larger scales of memory system but would have significant power issues. A system anywhere near this scale would require significant advances in interconnect and packaging. In particular, advances in 3D system geometries need to be considered.

Advances in 3D packaging also presents an opportunity to use geometry to reduce power consumption. With a conventional packaging approach, aggressive scaling of interconnect power would permit memory-CPU communications at an energy cost of around 2 pJ/bit. On the other hand, some 3D integration technologies, as discussed in Section 7.1.5 may permit power levels to approach 1-20 fJ/bit, depending on the length of run of on-chip interconnect required. Even at the low end bandwidth of 16 x 44 GBps, this represents a potential power savings of around 10 W per module, which could be more productively used for FPUs or additional memory capacity.

As well as provisioning interconnect, packaging also plays roles in power and ground current distribution, noise control (through embedded capacitors), heat removal and mechanical support, so as to ensure high reliability. The simultaneous roles of current delivery and heat removal create a geometric conundrum as the high power areas of the chip need lots of both at the same time. Again, this is a case where a concerted effort that involves both architecture and technology expertise es needed to arrive at practical solutions that would have real effect on achieving Exascale metrics.

Finally, it should be realized that advances in areas outside of packaging could simplify the creation of an Exascale 3D solution. In particular, if efficient voltage conversion could be incorporated within a 3D chip stack, then the "two-sided problem" is greatly simplified. Delivering 100 A at 1 V is a lot harder than delivering 1 A at 100 V. Similarly, releasing one side from a cooling function provides a similar improvement. For example, incorporating silicon micro-channel cooling into the chip stack removes the need for using one side for heat removal. Again, this is a difficult co-optimization problem if a truly useable solution is to be found.

On the topic of cooling, it might be that an initial embedded Exascale computing stack would be air-cooled. Large scale deployment of water chillers and circulating coolants, such as feasible for the data center or even departmental classes, do not lend themselves to embedded solutions in war fighting craft and vehicles. However, this global issues does not prevent designs from using liquid phase solutions locally (inside the stack). As long as the liquid does not require replenishment, then such local solutions might be reasonable. There are severe challenges in such solutions though. Leaks are still a problem in liquid cooling, and a leak-free technology would be a worthwhile investment. Also, techniques to circulate and pump the coolant are needed on the physical scale of a small 3D package. Similarly, local heat exchangers would be needed if local liquid cooling solutions are to be used. Heat pipes provide an excellent way to use liquid phase cooling locally without mechanical complexity. Thus, advances in the capacity of thin heat-pipe like solutions would be welcome in an Exascale computer.

8.2.1.5 Photonic Interconnect Research Opportunities and Goals

Photonics for long-haul data communication has long been believed to have real potential for significant power and latency reduction, especially when very high bandwidths are needed (see Section 7.5). Recent research (see for example Section 7.5.4) has opened up some tantalizing alternatives that push photonics far deeper into a system architecture than considered in the past. Given the pressing need to reduce energy and power throughout an Exascale system, it is incumbent that a complete research program fully explore the potentials for such technologies for their possible infusion, and do so in a way where the potential advantages can be fully compared on a common basis to the best of conventional wire based communication.

Further, as with many of the alternative technologies discussed earlier, it is also essential that such studies tightly integrate both architectural, circuit-level, and technological innovation and tradeoffs, and do so with metrics that cover not just power and bandwidth but also ones that are relevant to the other major challenges discussed in this report, such as resiliency. Consequently, a well-designed program in this area will need to focus on both long and short range communication. To a large extent, the long-range studies are already under way in industry, but efforts must be made to ensure their relevance to both data center and departmental class Exascale peta systems.

The goal of "short-range" photonic studies need to determine whether or not there are in fact gains to be made by creative use of new optical devices, with the key goal of establishing on a true "apples-to-apples" basis what the energy and performance gains really are possible. As before, this can only be done by considering microarchitectural alternatives concurrently with alternatives in interconnect devices and protocols for the photonic-digital interface. This will involve modeling for

performance, power, and resiliency.

8.2.2 Thrust Area: Exascale Architectures and Programming Models

Just as slowing clock and memory density growth have driven the need for alternative technologies and architectures which can incorporate them, the expected explosive growth in concurrency is driving the need to explore and develop better ways to connect what is programmed with what is left visible to programs from the underlying architectures. Thus the group feels that a second major research thrust must involve new technologies to bridge the gap, and again do so with a strong component of architectural development.

The following subsections describe several potential research topics that might fit this area.

8.2.2.1 Systems Architectures and Programming Models to Reduce Communication

The toughest of the power problems is not on circuits for computation, but on communication. Communication is insidious - a pJ wasted on moving data is not only a pJ that is not available for communication, but also a pJ that must often be matched by other pJs that must be expended to monitor the progress of the data during the movement, to correct for any errors, and to manage the suspension and reawakening of any circuits that source or sink the data.

Further, simply minimizing communication power alone may not minimize overall power. If higher power signalling can reduce latency, then perhaps more energy may be saved in maintaining the state of higher level threads, which in turn may reduce the number of threads that must be maintained in a register file, which in turn may allow the size of the register file to become smaller, which in turn may reduce the energy associated with all accesses to that file.

Hence there is a real need for the development of architectures and matching programming systems solutions that focus on reducing communications, and thus communication power. Such solutions would allow more of the total power to be used for computation.

Examples of potential techniques include:

- Over-provision the design with both function units and communications paths, but with flexible activation so as we can either use all available power in the function units if the problem is computation limited and all data is local, or use all the power for communication if the problem is global communication limited.
- Design in self-awareness of the status of energy usage at all levels, and the ability to adjust the activation of either logic or communication circuits to maintain a specific power level below a maximum while maintaining specified performance or system throughput goals.
- Provide more explicit and dynamic program control over the contents of memory structures closest to the function units so that minimal communication energy can be expended.
- Develop alternative execution and programming models where short sequences of code can be exported to data that is not resident, rather than dragging the data all across the system.

8.2.2.2 Locality-aware Architectures

The energy required to access an operand or instruction increases significantly with distance, with a particularly large increase occurring when moving from on-chip to off-chip. Hence data locality is a key element of reducing energy per operation to a level that enables Exascale systems. To operate efficiently, the bulk of data accesses must come from small registers and memories co-located with

the arithmetic units consuming the data. The number of high-energy off-chip and cross-machine accesses must be limited.

The problem of data locality spans several abstraction levels, and achieving high data locality requires a **locality-aware** architecture, a programming system that optimizes data placement and movement, and applications with inherent locality. Some applications (and portions of applications) fundamentally require large numbers of global accesses and will not benefit from locality-aware architectures and programming systems. For other applications, however, large gains can be realized by keeping data local and explicitly controlling its movement.

In this section we discuss a possible thrust to develop locality-aware architectures. Almost all architectures employ a storage hierarchy of some variety. These hierarchies are distinguished by their structure (the sizes of memory arrays and how they are connected), their management (implicit and/or explicit mapping and replacement policies), and how power is budgeted across the levels of the hierarchy.

The structure of the storage hierarchy limits what management and power policies are possible, since these policies operate over the structure. Any structure includes at least a level of registers close to the arithmetic units, one or more levels of on-chip storage arrays interconnected by a network, a level of per-chip or per-node off-chip storage, and one or more levels of global storage, also interconnected by a network. The granularity of the storage levels is often driven by efficient sizes for storage arrays and the levels are often designed to fit the packaging hierarchy. As machines increase in scale - both more cores per chip and more chips - we expect the number of levels and the complexity of certain levels to increase.

Most contemporary machines manage the hierarchy between registers and main memory *implicitly* (i.e.: as caches) and *reactively* (moving data in response to a miss at one level in the hierarchy) and use a *uniform* mapping at each level. While conceptually simple, these management policies leave large opportunities for improvement in energy efficiency and performance. Managing L2 or L3 on-chip storage uniformly, for example, results in distributing data across the entire area of the storage level - often the entire chip. Using non-uniform mapping policies can potentially reduce communication (and hence energy) by placing data nearer to its point of use.

Implicit and reactive caches wait to fetch data until it is requested - which increases the amount of concurrency required to cover the resulting long latency - and replace the data according to a simple least-recently-used policy. Managing the same memory structures in an explicit and proactive manner (i.e., explicitly transferring data from one level to another before it is needed, as in the IBM Cell) has been shown to yield large improvements in both efficiency (by eliminating unneeded data movement) and performance (by overlapping load latency with operations). The fully-associative nature of an explicitly-managed memory hierarchy means that arbitrary data can be placed in a memory structure, limited only by the capacity of the memory. Software can thus count on deterministic access latencies, and can be scheduled by the compiler to achieve high utilization. Explicit management is particularly attractive when combined with locality-enhancing program transformations so that a fragment of a data set can be produced in a local memory and consumed from the same memory without ever having to be read from or be written to main memory.

Explicit management, however, poses programming challenges for irregular applications that share mutable data. Copying data into a local memory with a separate address space can lead to incoherent data updates if proper synchronization is not employed. A hybrid management strategy that explicitly moves data while providing the needed exclusion and synchronization holds the potential to provide performance and efficiency gains even for irregular codes with mutable data. Research on hardware/software approaches for structuring and efficiently managing a storage hierarchy is likely to yield large gains in efficiency while at the same time reducing the required

concurrency and simplifying programming.

Explicit management poses other challenges for the compiler. Runtime data dependencies may make it difficult or impossible to know where a given piece of data is located. Furthermore, legacy applications may need extensive, whole-program-level transformations affecting the order in which data is traversed in order to exploit a locality-aware machine architecture. For this reason, a locality aware architecture may require (or at least be best exploited by) a locality-aware programming model. Research in compiler technologies and/or programming models for locality-aware architectures should be focused on achieving high locality without burdening the programmer with detailed control of data movement.

Power is the critical resource in emerging computer systems and a memory hierarchy is largely defined by how power is budgeted across the levels of the hierarchy. Putting the bulk of the power into arithmetic units, registers, and L1 access, for example, gives a machine that performs well on highly local problems at the expense of problems dominated by global accesses. A machine that puts the bulk of its power into global accesses is optimized in the opposite direction.

Because modern chips are power, not area, constrained, it is possible to build a multi-core computing node that can spend its entire power budget on any one level of the hierarchy. Such a *power-adaptive architecture* would be able to use its total power budget across a wide range of program locality - rather than being constrained by a fixed power allocation. Such an architecture, however, requires power management hardware and software to ensure that it does not exceed its total power budget in a way that would either collapse the power supply or result in an over-temperature situation. Research in power-aware architectures and management policies will not improve energy efficiency, but rather will allow a machine with a fixed power limit achieve a larger fraction of its peak performance by adapting its power budget to the needs of the application.

8.2.3 Thrust Area: Exascale Algorithm and Application Development

In the past two decades there has been a three order-of-magnitude growth in the amount of concurrency that application and algorithm developers must confront. Where once it was O(100) operations in a vector, it is now O(100,000) individual processors, each furnished with multiple floating-point arithmetic processing units (ALUs). This dramatic increase in concurrency has lead the dramatic lessening of efficient application scaling. Small load imbalances or synchronization events can create an Amdahl fraction that prevents most of today's applications from efficiently utilizing hundreds of thousands of processors.

As we look forward to Exascale computing, using processors whose clock frequencies will be at best marginally higher than those in use today, we anticipate a further growth in ALU count of *another* three orders-of-magnitude. On top of this will likely be another one or two orders-of-magnitude increase in the number of *threads* as processor vendors increasingly turn to multi-threading as a strategy for tolerating the latency of main memory accesses (i.e., cache misses). The challenge of developing applications that can effectively express $O(10^{10})$ threads of concurrent operations will be daunting. Equally difficult will be avoiding even the smallest unnecessary synchronization overheads, load imbalances, or accesses to remote data.

In addition, historically, software developers have striven to minimize the number of operations performed. More recently, as the memory hierarchies of today's mainstream systems have made main memory references increasingly costly, the emphasis has turned to organizing the operations so as to keep them in cache. At Exascale, where fetching a word from DRAM will cost more power than performing a double precision floating point multiply-accumulate operation with it, there will be a new imperative to minimize state size and eliminate unnecessary references to deep in the memory hierarchy. Research into a new generation of algorithms which repeat calculations to avoid

storing intermediate values or use additional calculations to compress state size will be needed.

Similar and equally vexing difficulties will arise when trying to counter the other challenges of resiliency, locality, and storage capacity.

Together, this calls for a substantial research effort to fund a variety of areas such as discussed in the following subsections.

8.2.3.1 Power and Resiliency Models in Application Models

Going forward it will be critical to understand the impacts of architectural choices on the performance and reliability of applications. For example, one will wish to measure or predict the "science results per watt" of a specific design in terms of intended uses. If an algorithm or application can be made less compute-intensive it may require fewer, or slower/cooler functional units. On the other hand if it is already memory bandwidth-bound at Petascale we want to know that and perhaps invest more of the energy budget in the memory subsystem. Related to energy and memory, if an algorithm's or application's overall memory footprint or locality clusters can be improved, then perhaps a power-aware architecture can run it for less energy. And related to concurrency, one must understand "what is the inherent concurrency of an algorithm or application?" to be able to say what sort of processors, and how many processors, will enable it to run at Exascale. Finally, as to resiliency, we must understand the fault-tolerance and checkpointing overhead of algorithms and applications to be able to say if they can run at all on proposed architectures, or if the require more development work to make them resilient enough to survive the anticipated hardware fault rates.

An initial research thrust then is simply to characterize the computational intensity, locality, memory footprints and access patterns, communications patterns, and resiliency of existing nearly-Petascale applications deemed most likely to go to Exascale, create performance models of these, and connect these performance models to power and failure-rate models from the Exascale technologists and architects. This will tell us how much power these applications will consume, how fast they will run, how often they will fail etc. at various system design points. We want to construct unified application-system models that enable critical-path analysis, thus to say where the energy budget is being expended, what are the performance bottlenecks, where will irrecoverable failures happen first, on a per-application basis.

8.2.3.2 Understanding and Adapting Old Algorithms

It is probably the case that few if any of today's applications would run at Exascale. Reasons might include: inability to scale to the new levels of concurrency, inefficient performance at whatever levels it can reach, poor energy per operation management, or inadequate checkpointing considerations that do not match the resiliency characteristics of the machine.

In most cases, we are not going to replace fundamental mathematical kernels. Therefore we will be forced to learn how to adapt them to this extraordinary scale. This will likely involve reformulating such algorithms as well as expressing them in new programming languages. An example might be to try to reformulate Krylov space algorithms so as to minimize the number of dot-products (and hence the global synchronization necessary to sum up across huge processor sections) without a subsequent loss in numerical robustness.

In addition, we will also have to learn to tune and scale these algorithms for the target machines, most probably using whatever modeling facilities might become available as discussed in the prior section. For example, it may be that applications inherently have ample fine-grained asynchronous parallelism, but that it is poorly expressed in the inherently course-grained and synchronous MPI

of today. Models will tell us the inherent parallelism and thus predict the potential suitability and benefit to converting the application to modern, more expressive and asynchronous, programming paradigms and languages.

As to locality, many applications may have inherent locality that is poorly expressed, or impossible to exploit with today's clumsy cache and prefetch management systems, or is present but ill-understood. As mentioned in Chapter 5, tools should be developed as part of this research thrust to help identify inherent locality, while programming languages should allow hooks to control the memory hierarchy of power-aware machines to exploit locality. As to checkpointing, it may be that crude checkpointing (write out entire memory image every time-step) works acceptably at Petascale but is impossible to do because of size and overheads of data at Exascale. Again, an algorithm reformulation approach that uses models can indicate this and guide development of refined smarter and cheaper application specific checkpointing strategies.

8.2.3.3 Inventing New Algorithms

Developing new algorithms that adapt to Exascale systems will involve more than just finding algorithms with sufficient amounts of useable concurrency. These systems will be extremely sensitive to starvation, and will require a new generation of graph-partitioning heuristics and dynamic load-balancing techniques that need to be developed and integrated hand-in-hand with the basic computational kernels. There will be a premium on locality to minimize unnecessary data movement and the concomitant expense of both power and latency. Exascale systems will likely have a much higher ratio of computing power over memory volume than today's systems, and this will open up room for new algorithms that exploit this surfeit of processing power to reduce their memory size.

8.2.3.4 Inventing New Applications

Applications are typically complex compositions of multiple algorithms interacting through a variety of data structures, and thus are entities separate from the underlying algorithms discussed above. Exascale systems cannot be designed independently of the target applications that will run on them, lest there be a vanishingly small number of them. It has taken a decade for applications to evolve to the point where they express enough concurrency to exploit today's trans-Petascale systems. It will be harder still for them to stretch to Exascale. Clearly it will also be important to develop a keen understanding of how to develop full-scale *applications* suitable for execution on Exascale systems. For example, future DoE applications will most likely be focused on developing massively parallel coupled-physics calculations. While these science drivers are inherently billion-way parallel they risk being less efficient, not more efficient, than today's mono-physics codes would be at Exascale, for reasons elaborated in Section 5. And, as shown in Figure 8.6 the trend may be away from easy-to-exploit locality in these applications. An important research thrust then will be to invest in scalable algorithm and application development that preserves locality. However an emphasis of this research thrust might again be to provide parameterized power/reliability/performance models to application development teams so they can proceed with such development in parallel with Exascale hardware and system research, and be ready to take advantage of new architectures when they emerge. Again the search for locality in applications is symbiotic with the ability of power-aware architectures to manage locality explicitly (cache only the things I want cached, don't prefetch the things I don't want prefetched etc.) It is important though that Exascale architecture not be carried out in a vacuum but inform and be informed by development of Exascale applications. Thus we propose a specific research thrust in this area towards developing models that can be a

Lingua Franca for communicating to applications developers the capabilities, limitations, and best usage models, for Exascale technologies and architectures.

8.2.3.5 Making Applications Resiliency-Aware

Finally, the development of **Resiliency-aware applications** will become critically valuable to achieving genuinely useful Exascale. We must understand the fault-tolerance and checkpointing overhead of algorithms and applications to be able to say if they can run at all on proposed architectures or if they require more development work to make them resilient enough to survive the anticipated hardware fault rates. Applications may need to specify where higher guarantees of correctness are necessary, and not necessary, in order to avoid such techniques as brute duplication that can rapidly run up power consumption.

8.2.4 Thrust Area: Resilient Exascale Systems

Technology trends and increased device count pose fundamental challenges in resisting intermittent hardware and software errors, device degradation and wearout, and device variability. Unfortunately, traditional resiliency solutions will not be sufficient. Extensive hardware redundancy could dramatically improve resiliency, but at a cost of 2-3 times more power. Checkpointing is effective at providing fault recovery, but when error rates become sufficiently high, the system may need to be saving checkpoints all of the time, driving down the time left for useful computation. To increase system resiliency without excess power or performance costs, we recommend a multi-pronged approach which includes innovations at each level of the system hierarchy and a vertically integrated effort that enables new resiliency strategies across the levels.

8.2.4.1 Energy-efficient Error Detection and Correction Architectures

The tradeoff between resiliency and power consumption manifests itself in several ways. Reducing power supply voltages toward the threshold voltage reduces power consumption in the primary circuits but may require more rigorous resiliency to handle higher error rates. In memories, shortening the refresh rate reduces power consumption but may increase the bit-error rate. Error rates can also be a function of the application characteristics or the temperature. We recommend research into low overhead hardware mechanisms to detect (and potentially correct) errors at the logic block level, rather than the gate level. Past research into arithmetic residual computation checking and on-line control logic protocol verification are starting points for this work. We also recommend tunable resiliency in which the degree of resiliency (and therefore the power required for it) can vary depending on the demand. Examples include selective redundant execution in software through re-execution or in hardware through on-the-fly processor pairing.

8.2.4.2 Fail-in-place and Self-Healing Systems

Because Exascale systems will experience frequent component failures, they must be able to continue operating at peak or near-peak capacity without requiring constant service. Research into fail-in-place and self-healing systems will be necessary to achieve this goal. Exploiting the redundancy inherent in Exascale systems in a manner transparent to application programmers will be critical. Hardware redundancy may include spare cores per chip, spare chips per node, memory modules, or path diversity in the interconnection network. The key to exploiting the redundancy transparently will likely be virtualization so that applications need not care precisely which resources are being used. Redundancy and virtualization techniques will be necessary to tolerate manufacturing defects,

device variation, aging, and degradation. The system must be able to reconfigure its hardware and software to make the system resilient to frequent hardware failures. This goal also reaches up to the packaging design so that system modules can be easily hot-swapped without halting the system. Emerging nanoscale devices, such as hybrid CMOS Field Programmable Nanowire Interconnect architectures, will require redundancy and automatic reconfiguration just to tolerate the inherent unpredictability in the manufacturing processes.

8.2.4.3 Checkpoint Rollback and Recovery

The dominant form of system recovery relies on checkpoint-restart and would benefit from innovations to reduce checkpointing overheads. For example, using solid-state non-volatile level of the memory hierarchy (flash memory) could provide substantial bandwidth for checkpointing so that it could be done under hardware control, drastically reducing checkpoint latency. Another avenue for research is in intelligent checkpointing, at the application or system levels. Such intelligent checkpointing schemes would select when and which data structures to checkpoint so that no unnecessary data is saved or time is wasted. Ideally, compilers or run-time systems would automatically select when and what to checkpoint, but an intermediate approach that runs semi-automatically (perhaps relying on application-level directives) would be a substantial advance over the state-of-the-art. Another approach is to shift to a programming model that is inherently amenable to checkpointing and recovery, such as transactions from the database and commercial computing domains. Finally, the community should prepare itself for a time when parallel systems must be continuously checkpointing due to the high error rates. Providing resiliency to such systems will require a range of solutions at the software system level.

8.2.4.4 Algorithmic-level Fault Checking and Fault Resiliency

Algorithmic based fault tolerance can provide extremely efficient error detection as only a small amount of computation is required to check the results for a large amount of computation. Prior research has produced algorithmic approaches for matrix multiply, QR factorization, FFT, and multi-grid methods. Further research in this area could expand the domain of algorithms suitable for efficient checking. Additional research will also be needed to automatically provide error checking for intermediate results in these algorithms, as high error rates may prevent any long-running computation from completing without correction. Another class of applications that can provide efficient correction are those that are self-healing, in which the algorithm can automatically correct errors. Some convergence algorithms already have this property as some data errors in one iteration can be corrected in subsequent iterations. Finally, applications that require less precision than the computer numerical representations provide may not care if errors exist in insignificant bits. The rise in error rates may require application and software tools writers to optimize for fast error detection and recovery in order to make good use of the inherently unreliable hardware.

8.2.4.5 Vertically-Integrated Resilient Systems

A vertically integrated approach will span all levels of the system stack including applications, programming systems, operating systems, and hardware. Achieving a system that is optimized as a whole for resiliency will require substantial research at the system level, but can result higher overall resiliency at a lower cost. As an example, a vertically integrated system might employ watchdog timers to detect faults in control logic, but use programming system controlled checking threads to provide redundancy for the critical portions of an application. Selective replication eliminates duplicate hardware and is applied only when necessary. Checkpointing would still be

required for recovery from control or data errors, but intelligent checkpointing (at the application or system level) in conjunction with fast hardware checkpointing mechanisms could reduce the overhead experienced by the application. System-level resiliency can be an enabling technology by allowing less reliable or new devices to be manufactured at lower cost; such techniques may be necessary for continued device scaling or to bridge the gap beyond traditional semiconductor technologies.

8.3 Multi-phase Technology Development

Given the relative immaturity of many of the technologies mentioned above as potential members of an ultimate Exascale technology portfolio, it made sense to the study group that bringing them to a level of technological maturity commensurate with a 2015 timeframe would require a careful phasing of the research efforts into roughly three phases:

1. System architecture explorations
2. Technology demonstrations
3. Scalability slice prototyping

The most immature of the technology solutions will have to go through all three, while others than have, for example, lab demonstrations, may simply have to accelerate their transition through the last two.

8.3.1 Phase 1: Systems Architecture Explorations

This would be the earliest phase for those challenges where the level of technological maturity is very low. The goal for research in this phase is to propose potential new alternatives, develop a coherent understanding of the interaction of the new technologies with architectures that will scale to the exa level, and identify what are the critical experiments that need to be performed to verify their applicability.

8.3.2 Phase 2: Technology Demonstrators

Research at this phase would focus on demonstrating solutions to the “long poles” of challenges using new technologies developed in the first phase. Each such demonstration would most probably be done in isolation of integration with other new technologies, so that the real properties and attributes of a single technology can be identified and measured with a fair degree of confidence.

The purpose here is not to demonstrate system solutions or even subsystems, but to develop and demonstrate targeted pieces of new technology in ways that would provide system implementers enough confidence that they could start planning for its introduction into Exascale products.

8.3.3 Phase 3: Scalability Slice Prototype

This phase would build on multiple demonstrations from the second phase to integrate different pieces of technology in ways that represent a relatively complete end-to-end “slice” through a potential Exascale system. This slice is not expected to be a complete system, but should include enough of multiple subsystems from a potential real system that the scaling to a real, complete, system integration is now feasible and believable.

Such a phase is an absolute necessity were significant deviations from existing architectures, programming models, and tools are forecast.

Appendix A

Exascale Study Group Members

A.1 Committee Members

Academia and Industry	
Name	**Organization**
Peter M. Kogge, Chair	University of Notre Dame
Keren Bergman	Columbia University
Shekhar Borkar	Intel Corporation
William W. Carlson	Institute for Defense Analysis
William J. Dally	Stanford University
Monty Denneau	IBM Corporation
Paul D. Franzon	North Carolina State University
Stephen W. Keckler	University of Texas at Austin
Dean Klein	Micron Technology
Robert F. Lucas	University of Southern California Information Sciences Institute
Steve Scott	Cray, Inc.
Allan E. Snavely	San Diego Supercomputer Center
Thomas L. Sterling	Louisiana State University
R. Stanley Williams	Hewlett-Packard Laboratories
Katherine A. Yelick	University of California at Berkeley
Government and Support	
William Harrod, Organizer	Defense Advanced Research Projects Agency
Daniel P. Campbell	Georgia Institute of Technology
Kerry L. Hill	Air Force Research Laboratory
Jon C. Hiller	Science & Technology Associates
Sherman Karp	Consultant
Mark A. Richards	Georgia Institute of Technology
Alfred J. Scarpelli	Air Force Research Laboratory

Table A.1: Study Committee Members.

A.2 Biographies

Keren Bergman is a Professor of Electrical Engineering at Columbia University where she also

directs the Lightwave Research Laboratory. Her current research programs involve optical interconnection networks for advanced computing systems, photonic packet switching, and nanophotonic networks-on-chip. Before joining Columbia, Dr. Bergman was with the optical networking group at Tellium where she headed the optical design of large-scale MEMS based cross-connects. Dr. Bergman received her B.S in 1988 from Bucknell University, the M.S. in 1991 and Ph.D. in 1994 from M.I.T. all in Electrical Engineering. She is a recipient of the National Science Foundation CAREER award in 1995 and the Office of Naval Research Young Investigator in 1996. In 1997 she received the CalTech President's Award for joint work with JPL on optical interconnection networks. Dr. Bergman led the optical interconnect effort in the HTMT Petaflops scale design project that combined advanced technologies and parallel architecture exploration. She recently served as Technical Advisor to the Interconnect Thrust of the NSA's Advanced Computing Systems research initiative. Dr. Bergman is a senior member of IEEE and a fellow of OSA. She is currently Associate Editor for IEEE *Photonic Technology Letters* and Editor-in-Chief of the OSA *Journal of Optical Networking*.

Shekhar Borkar graduated with MSc in Physics from University of Bombay, MSEE from University of Notre Dame in 1981, and joined Intel Corporation. He worked on the 8051 family of microcontrollers, the iWarp multicomputer project, and subsequently on Intel's supercomputers. He is an Intel Fellow and director of Microprocessor Research. His research interests are high performance and low power digital circuits, and high-speed signaling.

Daniel P. Campbell is a Senior Research Engineer in the Sensors and Electromagnetic Applications Laboratory of the Georgia Tech Research Institute. Mr. Campbell's research focuses on application development infrastructure for high performance embedded computing, with an emphasis on inexpensive, commodity computing platforms. He is co-chair of the Vector Signal Image Processing Library (VSIPL) Forum, and has developed implementations of the VSIPL and VSIPL++ specifications that exploit various graphics processors for acceleration. Mr. Campbell has been involved in several programs that developed middleware and system abstractions for configurable multicore processors, including DARPA's Polymorphous Computing Architectures (PCA) program.

William W. Carlson is a member of the research staff at the IDA Center for Computing Sciences where, since 1990, his focus has been on applications and system tools for large-scale parallel and distributed computers. He also leads the UPC language effort, a consortium of industry and academic research institutions aiming to produce a unified approach to parallel C programming based on global address space methods. Dr. Carlson graduated from Worcester Polytechnic Institute in 1981 with a BS degree in Electrical Engineering. He then attended Purdue University, receiving the MSEE and Ph.D. degrees in Electrical Engineering in 1983 and 1988, respectively. From 1988 to 1990, Dr. Carlson was an Assistant Professor at the University of Wisconsin-Madison, where his work centered on performance evaluation of advanced computer architectures.

William J. Dally is The Willard R. and Inez Kerr Bell Professor of Engineering and the Chairman of the Department of Computer Science at Stanford University. He is also co-founder, Chairman, and Chief Scientist of Stream Processors, Inc. Dr. Dally and his group have developed system architecture, network architecture, signaling, routing, and synchronization technology that can be found in most large parallel computers today. While at Bell Labs Bill contributed to the BELLMAC32 microprocessor and designed the MARS hardware accelerator. At Caltech he designed the MOSSIM Simulation Engine and the Torus Routing

Chip which pioneered wormhole routing and virtual-channel flow control. While a Professor of Electrical Engineering and Computer Science at the Massachusetts Institute of Technology his group built the J-Machine and the M-Machine, experimental parallel computer systems that pioneered the separation of mechanisms from programming models and demonstrated very low overhead synchronization and communication mechanisms. At Stanford University his group has developed the Imagine processor, which introduced the concepts of stream processing and partitioned register organizations. Dr. Dally has worked with Cray Research and Intel to incorporate many of these innovations in commercial parallel computers, with Avici Systems to incorporate this technology into Internet routers, co-founded Velio Communications to commercialize high-speed signaling technology, and co-founded Stream Processors, Inc. to commercialize stream processor technology. He is a Fellow of the IEEE, a Fellow of the ACM, and a Fellow of the American Academy of Arts and Sciences. He has received numerous honors including the IEEE Seymour Cray Award and the ACM Maurice Wilkes award. He currently leads projects on computer architecture, network architecture, and programming systems. He has published over 200 papers in these areas, holds over 50 issued patents, and is an author of the textbooks, Digital Systems Engineering and Principles and Practices of Interconnection Networks.

Monty Denneau is a Research Staff Member at the IBM T.J. Watson Research Center in Yorktown Heights, NY. He is has been involved in the architecture, design, and construction of a number of special and general purpose parallel machines.

Paul D. Franzon is currently a Professor of Electrical and Computer Engineering at North Carolina State University. He earned his Ph.D. from the University of Adelaide, Adelaide, Australia in 1988. He has also worked at AT&T Bell Laboratories, DSTO Australia, Australia Telecom and two companies he cofounded, Communica and LightSpin Technologies. His research interests focus on three main areas: Interconnect solutions; Application Specific Architectures and Nanocomputing circuits and structures. He developed core concepts in low-power contactless signaling (capacitive and inductive coupling) for conventional and 3D systems. Examples of these systems have flown in test satellites. He has developed MEMS-based interconnect solutions for electronic and optical applications. He led the development of the popular Spice2Ibis tools and IC Physical Design Kits that are have thousands of users world-wide. He has developed sub-25 nm wafer-scale interconnect solutions and new approaches to using nano-crystal elements in electronics. He has established new approaches to enabling extreme environment circuits. He has lead several major research efforts and published over 180 papers in these areas. In 1993 he received an NSF Young Investigators Award, in 2001 was selected to join the NCSU Academy of Outstanding Teachers, in 2003, selected as a Distinguished Alumni Professor, and in 2005 won the Alcoa Research award. He is a Fellow of the IEEE.

William Harrod joined DARPA's Information Processing Technology Office (IPTO) as a Program Manager in December of 2005. His area of interest is extreme computing, including a current focus on advanced computer architectures and system productivity, including self-monitoring and self-healing processing, Exascale computing systems, highly productive development environments and high performance, advanced compilers. He has over 20 years of algorithmic, application, and high performance processing computing experience in industry, academics and government. Prior to his DARPA employment, he was awarded a technical fellowship for the intelligence community while employed at Silicon Graphics Incorporated (SGI). Prior to this at SGI, he led technical teams developing specialized processors and ad-

vanced algorithms, and high performance software. Dr. Harrod holds a B.S. in Mathematics from Emory University, a M.S. and a Ph.D. in Mathematics from the University of Tennessee.

Kerry L. Hill is a Senior Electronics Engineer with the Advanced Sensor Components Branch, Sensors Directorate, Air Force Research Laboratory, Wright-Patterson AFB OH. Ms. Hill has 27 years experience in advanced computing hardware and software technologies. Her current research interests include advanced digital processor architectures, real-time embedded computing, and reconfigurable computing. Ms. Hill worked computer resource acquisition technical management for both the F-117 and F-15 System Program Offices before joining the Air Force Research Laboratory in 1993. Ms. Hill has provided technical support to several DARPA programs including Adaptive Computing Systems, Power Aware Computing and Communications, Polymorphous Computing Architectures, and Architectures for Cognitive Information Processing.

Jon C. Hiller is a Senior Program Manager at Science and Technology Associates, Inc. Mr. Hiller has provided technical support to a number of DARPA programs, and specifically computing architecture research and development. This has included the Polymorphous Computing Architectures, Architectures for Cognitive Information Processing, Power Aware Computing and Communications, Data Intensive Systems, Adaptive Computing Systems, and Embedded High Performance Computing Programs. Previously in support of DARPA and the services, Mr. Hiller's activities included computer architecture, embedded processing application, autonomous vehicle, and weapons systems research and development. Prior to his support of DARPA, Mr. Hiller worked at General Electric's Military Electronic Systems Operation, Electronic Systems Division in the areas of GaAs and CMOS circuit design, computing architecture, and digital and analog design and at Honeywell's Electro-Optics Center in the areas of digital and analog design. Mr. Hiller has a BS from the University of Rochester and a MS from Syracuse University in Electrical Engineering.

Sherman Karp has been a consultant to the Defense Research Projects Agency (DARPA) for the past 21 years and has worked on a variety of projects including the High Productivity Computing System (HPCS) program. Before that he was the Chief Scientist for the Strategic Technology Office (STO) of DARPA. At DARPA he did pioneering work in Low Contrast (sub-visibility) Image enhancement and Multi-Spectral Processing. He also worked in the area of fault tolerant spaceborne processors. Before moving to DARPA, he worked at the Naval Ocean Systems Center where he conceived the concept for Blue-Green laser communications from satellite to submarine through clouds and water, and directed the initial proof of principle experiment and system design. He authored two seminal papers on this topic. For this work he was named the NOSC Scientist of the Year (1976), and was elected to the rank of Fellow in the IEEE. He is currently a Life Fellow. He has co-authored four books and two Special Issues of the IEEE. He was awarded the Secretary of Defense Medal for Meritorious Civilian Service, and is a Fellow of the Washington Academy of Science, where he won the Engineering Sciences Award. He was also a member of the Editorial Board of the IEEE Proceedings, the IEEE FCC Liaison Committee, the DC Area IEEE Fellows Nomination Committee, the IEEE Communications Society Technical Committee on Communication Theory, on which he served as Chairman from 1979-1984, and was a member of the Fellows Nominating Committee. He is also a member of Tau Beta Pi, Eta Kappa Nu, Sigma Xi and the Cosmos Club.

Stephen W. Keckler is an Associate Professor of both Computer Sciences and Electrical and Computer Engineering at the University of Texas at Austin. He earned his Ph.D. from the

Massachusetts Institute of Technology in 1998. Dr. Keckler's research focuses on three main areas: high performance parallel processor architectures, distributed memory architectures, and interconnection networks. At MIT, he worked on the M-Machine, an experimental parallel computer system that pioneered multithreaded and tightly-coupled multicore processors. At UT-Austin, his group developed the NUCA cache which exploits non-uniform access time inherent in wire-dominated integrated circuits. His group recently built the TRIPS machine, a parallel computing system with numerous innovations in the processing and memory subsystems. The TRIPS machine demonstrates a hybrid dataflow instruction set and execution model and employs a logically and physically tiled implementation. Other innovations include tight coupling of routed interconnection networks into processing cores and reconfigurability to allow automatic or manual control of the memory hierarchy. Dr. Keckler has also worked as a VLSI circuit designer at Intel. Dr. Keckler has won numerous awards, including an Alfred P. Sloan Research Fellowship, the ACM Grace Murray Hopper award, an NSF CAREER award, and the 2007 President's Associates Teaching Excellence Award at UT-Austin. Dr. Keckler is a senior member of both the IEEE and the ACM, and a member of Sigma Xi and Phi Beta Kappa.

Dean Klein is the Vice President of Memory System Development at Micron Technology, Inc., where he has held a variety of executive positions since joining Micron in 1999. Mr. Klein's earliest role at Micron was the development of embedded DRAM and logic capability at Micron, a provider of semiconductor memory devices. The embedded DRAM efforts culminated in the creation of the Yukon Processor-In-Memory device, an embedded DRAM part containing 16MBytes of commodity-density DRAM closely coupled to a SIMD array of 256 processing elements. Prior to joining Micron Technology, Klein held the position of Chief Technical Officer and EVP at Micron Electronics, Inc. a personal computer manufacturer. While at Micron Electronics, Klein was responsible for the development of chip sets that exploited advanced memory technology to dramatically boost the performance of Intel's highest performing processors. Prior to Micron Electronics, Mr. Klein was President of PC Tech, Inc., which he co-founded in 1984 and which became a wholly owned subsidiary of Micron Electronics in 1995. Mr. Klein is a graduate of the University of Minnesota, with Bachelor's and Master's degrees in Electrical Engineering. He holds over 180 patents in the area of computer architecture and memory.

Peter M. Kogge (chair) is currently the Associate Dean for research for the College of Engineering, the Ted McCourtney Chair in Computer Science and Engineering, and a Concurrent Professor of Electrical Engineering at the University of Notre Dame, Notre Dame, Indiana. From 1968 until 1994, he was with IBM's Federal Systems Division in Owego, NY, where he was appointed an IBM Fellow in 1993. In 1977 he was a Visiting Professor in the ECE Dept. at the University of Massachusetts, Amherst, MA, and from 1977 through 1994, he was also an Adjunct Professor of Computer Science at the State University of New York at Binghamton. He has been a Distinguished Visiting Scientist at the Center for Integrated Space Microsystems at JPL, and the Research Thrust Leader for Architecture in Notre Dame's Center for Nano Science and Technology. For the 2000-2001 academic year he was also the Interim Schubmehl-Prein Chairman of the CSE Dept. at Notre Dame. His research areas include advanced VLSI and nano technologies, non von Neumann models of programming and execution, parallel algorithms and applications, and their impact on massively parallel computer architecture. Since the late 1980s' this has focused on scalable single VLSI chip designs integrating both dense memory and logic into "Processing In Memory" (PIM) archi-

tectures, efficient execution models to support them, and scaling multiple chips to complete systems, for a range of real system applications, from highly scalable deep space exploration to trans-petaflops level supercomputing. This has included the world's first true multi-core chip, EXECUBE, that in the early 1990s integrated 4 Mbits of DRAM with over 100K gates of logic to support a complete 8 way binary hypercube parallel processor which could run in both SIMD and MIMD modes. Prior parallel machines included the IBM 3838 Array Processor which for a time was the fastest single precision floating point processor marketed by IBM, and the Space Shuttle Input/Output Processor which probably represents the first true parallel processor to fly in space, and one of the earliest examples of multi-threaded architectures. His Ph.D. thesis on the parallel solution of recurrence equations was one of the early works on what is now called parallel prefix, and applications of those results are still acknowledged as defining the fastest possible implementations of circuits such as adders with limited fan-in blocks (known as the Kogge-Stone adder). More recent work is investigating how PIM-like ideas may port into quantum cellular array (QCA) and other nanotechnology logic, where instead of "Processing-In-Memory" we have opportunities for "Processing-In-Wire" and similar paradigm shifts.

Robert F. Lucas is the Director of the Computational Sciences Division of the University of Southern California's Information Sciences Institute (ISI). There he manages research in computer architecture, VLSI, compilers and other software tools. Prior to joining ISI, he was the Head of the High Performance Computing Research Department in the National Energy Research Scientific Computing Center (NERSC) at Lawrence Berkeley National Laboratory. There he oversaw work in scientific data management, visualization, numerical algorithms, and scientific applications. Prior to joining NERSC, Dr. Lucas was the Deputy Director of DARPA's Information Technology Office. He also served as DARPA's Program Manager for Scalable Computing Systems and Data-Intensive Computing. From 1988 to 1998 he was a member of the research staff of the Institute for Defense Analysis, Center for Computing Sciences. From 1979 to 1984 he was a member of the Technical Staff of the Hughes Aircraft Company. Dr. Lucas received his BS, MS, and PhD degrees in Electrical Engineering from Stanford University in 1980, 1983, and 1988 respectively.

Mark A. Richards is a Principal Research Engineer and Adjunct Professor in the School of Electrical and Computer Engineering, Georgia Institute of Technology. From 1988 to 2001, Dr. Richards held a series of technical management positions at the Georgia Tech Research Institute, culminating as Chief of the Radar Systems Division of GTRI's Sensors and Electromagnetic Applications Laboratory. From 1993 to 1995, he served as a Program Manager for the Defense Advanced Research Projects Agency's (DARPA) Rapid Prototyping of Application Specific Signal Processors (RASSP) program, which developed new computer-aided design (CAD) tools, processor architectures, and design and manufacturing methodologies for embedded signal processors. Since the mid-1990s, he has been involved in a series of programs in high performance embedded computing, including the efforts to develop the Vector, Signal, and Image Processing Library (VSIPL) and VSIPL++ specifications and the Stream Virtual Machine (SVM) middleware developed under DARPA's Polymorphous Computing Architectures (PCA) program. Dr. Richards is the author of the text *Fundamentals of Radar Signal Processing* (McGraw-Hill, 2005).

Alfred J. Scarpelli is a Senior Electronics Engineer with the Advanced Sensor Components Branch, Sensors Directorate, Air Force Research Laboratory, Wright-Patterson AFB OH. His current research areas include advanced digital processor architectures, real-time embedded

computing, and reconfigurable computing. Mr. Scarpelli has 32 years research experience in computer architectures and computer software. In the 1970's, he conducted benchmarking to support development of the MIL-STD-1750 instruction set architecture, and test and evaluation work for the DoD standard Ada language development. In the 1980's, he was involved with the DoD VHSIC program, and advanced digital signal processor development, a precursor to the F-22 Common Integrated Processor. In the 1990's, his research focused on the DARPA Pilot's Associate, the development of an embedded, artificial intelligence processor powered by an Associative Memory CoProcessor, real-time embedded software schedulability techniques, VHDL compilation and simulation tools, and application partitioning tools for reconfigurable computing platforms. Since 1997, he has provided technical support to multiple DARPA programs such as Adaptive Computing Systems, Polymorphous Computing Architectures, Architectures for Cognitive Information Processing, Networked Embedded Systems Technology, Mission Specific Processing, and Foliage Penetration. He holds a B.S. degree in Computer Science from the University of Dayton (1979) and an M.S. degree in Computer Engineering from Wright State University (1987).

Steve Scott is the Chief Technology Officer and SVP at Cray Inc., where he has been since 1992 (originally with Cray Research and SGI). Dr. Scott was one of the architects of the groundbreaking Cray T3E multiprocessor, focusing on the interconnect and on synchronization and communication mechanisms. He was the chief architect of the GigaRing system area network used in all Cray systems in the late 1990s. More recently, he was the chief architect of the Cray X1/X1E supercomputers, which combined high performance vector processors with a scalable, globally-addressable system architecture. He was also the chief architect of the next generation Cray "BlackWidow" system, and the architect of the router used in Cray XT3 MPP and the follow-on Baker system. Dr. Scott is currently leading the Cray Cascade project, which is part of the DARPA High Productivity Computing Systems program targeting productive, trans-petaflop systems in the 2010 timeframe. Dr. Scott received his PhD in computer architecture from the University of Wisconsin, Madison in 1992, where he was a Wisconsin Alumni Research Foundation and Hertz Foundation Fellow. He holds seventeen US patents, has served on numerous program committees, and has served as an associate editor for the IEEE Transactions on Parallel and Distributed Systems. He was the recipient of the ACM 2005 Maurice Wilkes Award and the IEEE 2005 Seymour Cray Computer Engineering Award.

Allan E. Snavely is an Adjunct Assistant Professor in the University of California at San Diego's Department of Computer Science and is founding director of the Performance Modeling and Characterization (PMaC) Laboratory at the San Diego Supercomputer Center. He is a noted expert in high performance computing (HPC). He has published more than 50 papers on this subject, has presented numerous invited talks including briefing U.S. congressional staff on the importance of the field to economic competitiveness, was a finalist for the Gordon Bell Prize 2007 in recognition for outstanding achievement in HPC applications, and is primary investigator (PI) on several federal research grants. Notably, he is PI of the Cyberinfrastructure Evaluation Center supported by National Science Foundation, and Co-PI in charge of the performance modeling thrust for PERI (the Performance Evaluation Research Institute), a Department of Energy SciDAC2 institute.

Thomas Sterling is the Arnaud and Edwards Professor of Computer Science at Louisiana State University and a member of the Faculty of the Center for Computation and Technology. Dr. Sterling is also a Faculty Associate at the Center for Computation and Technology at Cali-

fornia Institute of Technology and a Distinguished Visiting Scientist at Oak Ridge National Laboratory. Sterling is an expert in the field of parallel computer system architecture and parallel programming methods. Dr. Sterling led the Beowulf Project that performed seminal pathfinding research establishing commodity cluster computing as a viable high performance computing approach. He led the Federally sponsored HTMT project that conducted the first Petaflops scale design point study that combined advanced technologies and parallel architecture exploration as part of the national petaflops initiative. His current research directions are the ParalleX execution model and processor in memory architecture for directed graph based applications. He is a winner of the Gorden Bell Prize, co-author of five books, and holds six patents.

R. Stanley Williams is an HP Senior Fellow at Hewlett-Packard Laboratories and Director of the Advanced Studies Lab (ASL), a group of approximately 80 research scientists and engineers working in areas of strategic interest to HP. The areas covered in ASL include computing architectures, photonics, nano-electronics, micro- and nano-mechanical systems, information theory and quantum information processing. He received a B.A. degree in Chemical Physics in 1974 from Rice University and his Ph.D. in Physical Chemistry from U. C. Berkeley in 1978. He was a Member of Technical Staff at AT&T Bell Labs from 1978-80 and a faculty member (Assistant, Associate and Full Professor) of the Chemistry Department at UCLA from 1980 1995. His primary scientific research during the past thirty years has been in the areas of solid-state chemistry and physics, and their applications to technology. Most recently, he has examined the fundamental limits of information and computing, which has led to his current research in nano-electronics and nano-photonics. He has received awards for business, scientific and academic achievement, including the 2004 Joel Birnbaum Prize (the highest internal HP award for research), the 2004 Herman Bloch Medal for Industrial Research, the 2000 Julius Springer Award for Applied Physics, the 2000 Feynman Prize in Nanotechnology. He was named to the inaugural Scientific American 50 Top Technology leaders in 2002 and then again in 2005 (the first to be named twice). In 2005, the US patent collection that he has assembled at HP was named the world's top nanotechnology intellectual property portfolio by Small Times magazine. He was a co-organizer and co-editor (with Paul Alivisatos and Mike Roco) of the workshop and book "Vision for Nanotechnology in the 21st Century", respectively, that led to the establishment of the U. S. National Nanotechnology Initiative in 2000. He has been awarded more than 60 US patents with more than 40 pending and he has published over 300 papers in reviewed scientific journals. One of his patents on crossbar based nanoscale circuits was named as one of five that will "transform business and technology" by MIT's Technology Review in 2000.

Katherine A. Yelick is Professor of Electrical Engineering and Computer Sciences at the University of California, Berkeley and a Senior Research Scientist at the Lawrence Berkeley National Laboratory. Prof. Yelick co-leads and co-invented the Titanium language, which is a Partitioned Global Address Space (PGAS) language based on Java. The Titanium group has demonstrated tremendous productivity advantages for adaptive mesh refinement algorithms and immersed boundary method simulations. She also leads the Berkeley UPC project, an open source compiler project for the Unified Parallel C (UPC) language. She co-invented the UPC language with 5 other researchers from IDA, LLNL, and UC Berkeley, and co-authored both the original language specification and the main UPC textbook, *UPC: Distributed Shared-Memory Programming* (Wiley-Interscience, 2005). The Berkeley UPC compiler project is a highly portable compiler that is the used on clusters and shared memory

systems and is shipped with systems from Cray, SGI, and some Linux clusters. Prof. Yelick leads the Berkeley Institute for Performance Studies (BIPS), which involves performance analysis, modeling, tuning, and benchmarking. The groups within BIPS work on large application performance studies across vector, cluster, and ultrascale (BG/L) supercomputers as well as synthetic benchmarking and identification of architectural bottlenecks. She also co-leads the Berkeley Benchmarking and Optimization (BeBOP) group, which developed the OSKI system for automatically tuning sparse matrix kernels. Based on ideas from her earlier Sparsity system, OSKI includes optimizations for registers and caches that are tailored to a given sparse matrix structure. The group has recently developed multicore optimizations which are being integrated into OSKI. Prof. Yelick received her B.S., M.S., and Ph.D. degrees in Computer Science from the Massachusetts Institute of Technology. She has been a Visiting Researcher at ETH, Zurich and a Visiting Associate Professor at MIT, and has received teaching awards from both Berkeley and from MIT. She was a member of the WTEC team on "Assessment of High-End Computing Research and Development in Japan," and is currently a member of an NRC committee on "Sustaining the Growth in Computing Performance."

Appendix B

Exascale Computing Study Meetings, Speakers, and Guests

B.1 Meeting #1: Study Kickoff

May 30, 2007, Arlington, VA
Host: Science and Technology Associates

Committee members present: Shekhar Borkar, Dan Campbell, William Dally, Monty Denneau, William Harrod, Kerry Hill, Jon Hiller, Sherman Karp, Steve Keckler, Dean Klein, Peter Kogge, Bob Lucas, Mark Richards, Alfred Scarpelli, Steve Scott, Allan Snavely, Thomas Sterling, Kathy Yelick

Visitors

- Jose Munoz, NSF
- Rob Schreiber, HP Labs
- Barbara Yoon, DARPA

Presentations

- Peter Kogge - Introduction and Goals
- Thomas Sterling - Towards an Exascale Report
- William Dally - NRC Future of Supercomputing Study
- Dean Klein - Memory
- Steve Scott - Processor Requirements and Scaling
- Allan Snavely - Application Scaling
- Katherine Yelick - Berkeley Dwarfs

B.2 Meeting #2: Roadmaps and Nanotechnology

June 26-27, 2007, Palo Alto, CA
Host: Hewlett-Packard Laboratories

Committee members present: Shekhar Borkar, Daniel Campbell, William Carlson, William Dally, Monty Denneau, William Harrod, Jon Hiller, Sherman Karp, Stephen Keckler, Peter Kogge, Mark Richards, Allan Snavely, Stanley Williams, Katherine Yelick

Visitors

- Jeff Draper, USC/ISI
- Rob Smith, Nantero
- Norm Jouppi, HP Labs
- Richard Kaufmann, HP Labs
- Phil Kuekes, HP Labs
- Chandrakant Patel, HP Labs
- Rob Schreiber, HP Labs
- Greg Snider, HP Labs

Presentations

- Shekhar Borkar - Logic Roadmap
- Norm Jouppi - Configurable Isolation
- Stan Williams: Exascale Overview
- Greg Snider - Adaptive, Probabilistic Exacomputing
- William Dally - Future Projections Spreadsheet
- Phil Kuekes - Defects & Faults
- Richard Kaufmann - Checkpoint/Restart & Ratios
- Steve Keckler - Reliability
- Chandrakant Patel - Smart Data Center

B.3 Special Topics Meeting #1: Packaging

July 17-18, 2007, Atlanta, GA
Host: Georgia Institute of Technology

Committee members present: William Dally, Dan Campbell, Monty Denneau, Paul Franzon, William Harrod, Jon Hiller, Peter Kogge, Mark Richards, Alfred Scarpelli. By teleconference: Kerry Hill, Sherman Karp

Visitors

- Muhannad Bakir, Georgia Institute of Technology
- Robert Conn, Research Triangle Institute
- Patrick Fay, University of Notre Dame
- Paul Franzon, NCSU
- Dorota Temple, Research Triangle Institute
- Rao Tummala, Georgia Institute of Technology

Presentations

- Paul Franzon - High Bandwidth Interconnect
- Rao Tummala - System Packaging
- Robert Conn - 3D Impact on HPC
- Dorota Temple - 3D Integration
- Patrick Fay - Quilt Packaging
- Muhannad Bakir - Electrical, Optical & Thermofluidic Interconnects

B.4 Meeting #3: Logic

July 24-25, 2007, Portland, OR
Host: Intel Corporation

Committee members present: Shekhar Borkar, Daniel Campbell, William Carlson, William Dally, Monty Denneau, Paul Franzon, William Harrod, Jon Hiller, Sherman Karp, Stephen Keckler, Dean Klein, Peter Kogge, Robert Lucas, Mark Richards, Steve Scott, Allan Snavely, Stanley Williams

Visitors

- Jim Held, Intel
- Jose Maiz, Intel
- Tim Mattson, Intel
- Marko Radosavljevic, Intel

Presentations

- William Dally - Exascale Silicon Architecture
- Allan Snavely - Applications
- Shekhar Borkar - Exascale Power Performance Tradeoffs
- Steve Scott - Socket Architecture
- Stanley Williams - Nanoscale Implications for Exascale

- Steve Keckler - Reliability
- Paul Franzon - 3-D Interconnects
- Dean Klein - DRAM Challenges
- Marko Radosavljevic - Nanoelectronic Devices
- Jim Held - Terascale Computing
- Jose Maiz - Exascale Reliability
- Clair Webb - 3DIC Integration
- Tim Mattson - Programming at Exascale

B.5 Meeting #4: Memory Roadmap and Issues

August 16-17, Boise, ID
Host: Micron Technology

Committee members present: Shekhar Borkar, Daniel Campbell, Monty Denneau,William Harrod, Jon Hiller, Sherman Karp, Stephen Keckler, Dean Klein, Peter Kogge, Robert Lucas, Mark Richards, Steve Scott, Allan Snavely, Stanley Williams. By teleconference: Paul Franzon

Visitors

- Rob Schreiber, HP Labs
- Jim Hutchby, Semiconductor Research Corporation
- Terry Lee, Micron
- Dave Resnick, Micron
- Kevin Ryan, Micron
- Brent Keeth, Micron
- Mark Durcan, Micron
- Kirk Prall, Micron
- Chandra Mouli, Micron

Presentations

- Paul Franzon - 3D Memory Packaging
- Jim Hutchby - Emerging Research Memory Devices
- Steve Scott - Thoughts on a 3D Node
- Allan Snavely - Locality versus performance
- Monty Denneau - EDRAM
- Steve Scott - Iso Curves

- Dean Klein - Micron's Yukon Architecture
- Kirk Prall - NAND
- Overview Brent Keeth - DRAM Architectures & Technology
- Micron - 3D Integration Update
- Chandra Mouli - Memory Technology Trends
- Micron - Low End Memory Solutions

B.6 Special Topics Meeting #2: Architectures and Programming Environments

August 29-30, 2007, Palo Alto, CA
Host: Stanford University

Committee members present: Shekhar Borkar, Daniel Campbell, William Dally, Paul Franzon, William Harrod, Jon Hiller, Sherman Karp, Stephen Keckler, Dean Klein, Peter Kogge, Robert Lucas, Mark Richards, Alfred Scarpelli, Steve Scott, Allan Snavely, Thomas Sterling, Stanley Williams, Katherine Yelick

Visitors

- Krste Asanovic, University of California at Berkeley
- Luiz Barroso, Google
- Mark Horowitz, Stanford University
- Kunle Olukotun, Stanford University
- Mary Hall, University of Southern California
- Vivek Sarkar, Rice University

Presentations

- Allan Snavely - Isosurfaces
- Stephen Keckler - Reliability for Exascale
- Robert Lucas - Musings on Exascale
- Robert Lucas - ORNL Exascale presentation
- Katherine Yelick - Memory footprint
- William Dally - Strawman Architecture
- Steve Scott - 3D Node Thoughts
- Stephen Keckler - IBM Fault Tolerance from HotChips
- Mark Horowitz - Power & CMOS Scaling

B.7 Special Topics Meeting #3: Applications, Storage, and I/O

September 6-7, 2007, Berkeley, CA
Host: University of California at Berkeley

Committee members present: Daniel Campbell, William Dally, Paul Franzon, William Harrod, Jon Hiller, Sherman Karp, Dean Klein, Peter Kogge, Robert Lucas, Mark Richards, Alfred Scarpelli, Allan Snavely, Thomas Sterling

Visitors

- Dave Koester, MITRE
- Winfried Wilcke, IBM
- Garth Gibson, Carnegie Mellon University
- Dave Aune, Seagate Technology
- Gary Grider, Los Alamos National Laboratory
- Duncan Stewart, HP Labs
- Dave Bailey, Lawrence Berkeley Laboratory
- John Shalf, Lawrence Berkeley Laboratory
- Steve Miller, NetApp

Presentations

- David Koester - Application Footprints
- Garth Gibson - Petascale Failure Data
- Gary Grider - Gaps
- Dave Aune - Storage Trends
- Dean Klein - Memory Resiliency
- Gary Grider - ASC I/O Report
- John Shalf - I/O Requirements
- Duncan Stewart - Nano-crosspoint Memory

B.8 Special Topics Meeting #4: Optical Interconnects

September 25-26, 2007, Palo Alto, CA
Host: Stanford University

Committee members present: Daniel Campbell, William Dally, Monty Denneau, Paul Franzon, William Harrod, Jon Hiller, Sherman Karp, Stephen Keckler, Dean Klein, Peter Kogge, Robert Lucas, Steve Scott, Mark Richards, Alfred Scarpelli, Allan Snavely, Thomas Sterling

Visitors

- Ravi Athale, MITRE
- Karen Bergman, Columbia University
- Alex Dickinson, Luxtera
- David Miller, Stanford University
- Dave Koester, MITRE
- Bill Wilson, InPhase
- Mark Beals, Massachusetts Institute of Technology
- Cheng Hengju, Intel
- Krishna Saraswat, Stanford University
- Alan Benner, IBM
- Jeff Kash, IBM
- Ahok Krishnamoorthy, Sun
- Ray Beausoleil, HP Labs
- Mootaz Elnohazy, IBM

Presentations

- David Koester - Application Scaling Requirements
- Stephen Keckler - Reliability
- Mark Beals - Photonic Integration
- Jeffrey Kash & Alan Benner - Optical/electrical Interconnect Technologies
- Bill Wilson - Holographic Archive
- Ray Beausoleil - Nanophotonic Interconnect
- Krishna Saraswat - Optics and CNT
- Keren Bergman - Photonic Interconnects

B.9 Meeting #5: Report Conclusions and Finalization Plans

October 10-11, 2007, Marina del Rey, CA
Host: University of Southern California Information Sciences Institute

Committee members present: Shekhar Borkar, Daniel Campbell, William Carlson, William Dally, Paul Franzon, William Harrod, Jon Hiller, Sherman Karp, Stephen Keckler, Peter Kogge, Robert Lucas, Mark Richards, Alfred Scarpelli, Allan Snavely, Katherine Yelick

Visitors

- Keren Bergman, Columbia University
- Loring Craymer, University of Southern California Information Sciences Institute
- Phil Kuekes, HP Labs

Appendix C

Glossary and Abbreviations

AMB: Advanced Memory Buffer

AMR: Adaptive Mesh Refinement

AVUS: Air Vehicle Unstructured Solver

BCH: , Bose, Chaudhuri, Hocquenghem error correcting code.

BER: Bit Error Rate

BIST: Built-In-Self-Test

bitline: The bitline receives or delivers charge from the memory cell capacitor through the memory cell FET access device. This charge is sensed and driven by the sense amplifier.

BGA: Ball Grid Array

BER: Bit Error Rate

CAD: Computer-aided Design

CAGR: Compound Annual Growth Rate

CDR: Clock and Data Recovery
Computational Fluid Dynamics

CMOL: CMOS MOlecular Logic

CMOS: a common type of logic that employs two types of transistors whose on/off characteristics are essentially opposite.

CMP: Chip Multi-Processor

CNT: Carbon Nano Tubes

CSP: Communicating Sequential Process model

DDR: Double Data Rate DRAM. A protocol for memory chips that has a higher bit signalling rate than earlier types.

Digitline see bitline.

DIMM: Dual Inline Memory Module. The common form of packaging for memory devices. DIMM's are available for all commodity main memory types, with and without ECC for applications from desktop computers to supercomputers.

DRAM: Dynamic Random Access Memory. A memory typically composed of a one transistor, one capacitor memory cell. This memory is most commonly used as main memory in a computing system.

DWDM: Dense Wavelength Division Multiplexing

DSP: Digital Signal Processor

E3SGS: Simulation and Modeling at exascale for Energy, Ecological Sustainability and Global Security

EB: exabyte

ECC: Error Correcting Code

eDRAM: embedded Dynamic Random Access Memory

EIP: Exa Instructions Processed to completion

EIPs: Exa Instructions Processed per second

E/O: Electrical to Optical

EOS/DIS: Earth Observing System/Data Information System

FB-DIMM: Fully Buffered Dual Inline Memory Module

Fe-RAM: Ferro-electric RAM

FG: Floating Gate.

FINFET: a dual gate non-planar transistor where the major structures look like "fins" on top of the substrate.

FIT: Failures in Time. The number of expected failures in a device over a billion hour of operation. Testing for this is generally done by accelerated testing a million devices for a thousand hours. Typically performed on memory devices.

flop: floating point operation

FPGA: Field Programmable Gate Array

FPNI: Field-Programmable Nanowire Interconnect

FPU: Floating Point Unit

GAS: Global Address Space

GB: giga byte

GUPS: Global Updates Per Second

HAL: Hardware Abstraction Layer

HECRTF: High End Computing Revitalization Task Force

HT: Hyper Transport

HPC: High Performance Computing

HPCS: High Productivity Computing Systems - a DARPA program

HPL: High Performance Linpack benchmark

HTMT: Hybrid Technology Multi-Threaded architecture

HVAC: Heating, Ventilating, and Air Conditioning

ILP: Instruction Level Parallelism

IMT: Interleaved Multi-Threading

IPS: Instructions Per Second

ISA: Instruction Set Architecture

JEDEC: Joint Electron Device Engineering Council. A standards committee for many commercial commodity electronic parts.

JJ Josephson Junction

MEMS: Micro Electrical Mechanical Systems

MLC: Multi-Level-Cell. A technology for storing more than one bit in a NAND Flash cell.

MOS: see MOSFET

MOSFET: Metal-Oxide-Semiconductor Field Effect Transistor. While not totally accurate (gates today are not implemented from metal), this isle common name for the typical transistor used today.

MPI: Message Passing Interface

MPP: Massively Parallel Processor

MRAM: Magnetic Random Access Memory

MTJ: magnetic tunnel junctions

MTTI: Mean Time To Interrupt

MW: Mega Watt

NAS: National Academy of Science

nm: nano meter

NoC: Network on Chip

NVRAM: Non Volatile RAM

O/E: Optical to Electrical

PB: Peta Byte

PCB: Printed Circuit Board

PCRAM: Phase-Change RAM

PDE: Partial Differential Equation

PDU: Power Distribution Unit

pGAS: Partitioned global address space

PIM: Processing-In-Memory

pJ: pico joules, or $10^{-}12$ joules

PSU: Power Supply Unit

QK: Quintessential Kernel

qubit: quantum bit

RAM: Random Access Memory

RLDRAM: Reduced Latency DRAM

RMA: Reliability, Maintainability, and Availability

ROM: Read Only Memory

RRAM: Resistive RAM

RSFQ: Rapid Single Flux Quantum Device

RTL: Register Transfer Language

SCI: System Call Interface

SCM: System Capable Memory

SCP: Single Chip Package

SDRAM: Synchronous DRAM

SECDED: Single Error Correcting Double Error Detecting code

SEM: Scanning Electron Microscope

SER: Soft Error Rate. Failures in a component or system that are transient failures can be aggregated to compute a soft error rate. SER sources include noise and ionizing radiation.

SerDes: Serializer/Deserializer

SEU: Single Event Upset

SIMD: Single Instruction, Multiple Data

SLC: Single-Level-Cell. A flash with one bit of information stored per cell.

SMT: Simultaneous Multi-Threading

SNIC: Semiconductor Nanowire InterConnect

SNR: Signal to Noise Ratio

SOC: System On a Chip

SOI: Silicon On Insulator

SONOS: Semiconductor-Oxide-Nitride-Oxide-Semiconductor memory

SPMD: Single Program Multiple Data

SRAM: static random access memory. A memory typically composed of a six transistor storage cell. These cells can be fast and are typically used in processor caches where access time is most critical.

SSTL: Stub Series Terminated Logic signalling standard

TEC: Thermal Electric Coolers

TIA: Trans-Impedance Amplifier - a type of optical receiver

TLB: translation lookaside buffer

TLC: Thread Level Parallelism

TMR: Triple Modular Redundancy

TSV: Through Silicon Via

UPS: Uninterruptible Power Supply

VFS: Virtual File System interface

VR: Voltage Regulator

VRT: Variable Retention Time

V_{dd}: Voltage drain to drain - main operation voltage for silicon circuits

VSEL: Vertical-Cavity Surface-Emitting Laser

Wordline: The signal line that drives the gates of the memory cell FET. The wordline may be divided across several sub-arrays, but must be considered as the logical sum of its parts. Typically a wordline activates a row of 8K memory cells.

WRF: Weather Research and Forecasting code.

Bibliography

[1] BGA-scale stacks comprised of layers containing integrated circuit die and a method for making the same. US Patent Number 20070158805.

[2] Historical Notes about the Cost of Hard Drive Storage Space. http://www.littletechshoppe.com/ns1625/winchest.html.

[3] Superconducting Technology Assessment. Technical report, National Security Agency Office of Corporate Assessments, August 2005.

[4] High Productivity Computer Systems. http://www.highproductivity.org/, 2007.

[5] N.R. Adiga and et al. An Overview of the BlueGene/L Supercomputer. In *ACM/IEEE Conference on Supercomputing*, November 2002.

[6] E. Adler and et al. The evolution of IBM CMOS DRAM technology. *IBM J. Research and Development*, 39(1/2):pp. 167–188, January 1995.

[7] Guy AlLee, Milan Milenkovic, and James Song. Data Center Energy Efficiency. http://download.intel.com/pressroom/kits/research/poster_Data_Center_Energy_Efficiency.pdf, June 2007.

[8] George Almási, Călin Caşcaval, nos José G. Casta Monty Denneau, Derek Lieber, José E. Moreira, and Henry S. Warren. Dissecting Cyclops: a detailed analysis of a multithreaded architecture. *SIGARCH Comput. Archit. News*, 31(1):26–38, 2003.

[9] C. J. Amsinck, N. H. Di Spigna, D. P. Nackashi, , and P. D. Franzon. Scaling constraints in nanoelectronic random-access memories. *Nanotechnology*, 16, October.

[10] Ken Anderson, Edeline Fotheringham, Adrian Hill, Bradley Sissom, and Kevin Curtis. High speed holographic data storage at 500 Gbit/in.^{2}. http://www.inphase-technologies.com/downloads/pdf/technology/HighSpeedHDS500Gbin2.pdf.

[11] Krste Asanovic, Ras Bodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Kurt Keutzer, David A. Patterson, William Lester Plishker, John Shalf, Samuel Webb Williams, and Katherine A. Yelick. The Landscape of Parallel Computing Research: A View from Berkeley. http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.pdf, December 2006.

[12] Computing Research Association. Grand Research Challenges in Information Systems. http://www.cra.org/reports/gc.systems.pdf, 2003.

[13] Semiconductor Industries Association. *International Technology Roadmap for Semiconductors*. 2006.

[14] I. G. Baek and et al. Multi-layer cross-point binary oxide resistive memory (OxRRAM) for post-NAND storage application. *IEDM*, 2006.

[15] Gary H. Bernstein, Qing Liu, Minjun Yan, Zhuowen Sun, David Kopp, Wolfgang Porod, Greg Snider, and Patrick Fay. Quilt Packaging: High-Density, High-Speed Interchip Communications. *IEEE Trans. on Advanced Packaging*, 2007.

[16] B. Black and et al. Die stacking (3d) microarchitecture. volume 39. IEEE Micro, 2006.

[17] Defense Science Board. Task Force on DoD Supercomputer Needs. http://stinet.dtic.mil/cgi-bin/GetTRDoc?AD=ADA383826&Location=U2&doc=GetTRDoc.pdf, October 2000.

[18] Defense Science Board. Report on Joint U.S. Defense Science Board and UK Defence Scientific Advisory Council Task Force on Defense Critical Technologies. http://www.acq.osd.mil/dsb/reports/2006-03-Defense_Critical_Technologies.pdf, March 2006.

[19] Shekhar Borkar. Designing reliable systems from unreliable components: the challenges of transistor variability and degradation. *IEEE Micro*, 25(6):10–16, 2005.

[20] Arthur A. Bright, Matthew R. Ellavsky2, Alan Gara1, Ruud A. Haring, Gerard V. Kopcsay, Robert F. Lembach, James A. Marcella, Martin Ohmacht1, and Valentina Salapura1. Creating the BlueGene/L Supercomputer from Low-Power SoC ASICs. pages 188–189, San Francisco, CA, 2005. ISSCC.

[21] R. Brightwell, K. Pedretti, and K.D. Underwood. Initial performance evaluation of the Cray SeaStar interconnect. *13th Symposium on High Performance Interconnects*, pages 51–57, 17-19 Aug. 2005.

[22] Ian Buck, Tim Foley, Daniel Horn, Jeremy Sugerman, Kayvon Fatahalian, Mike Houston, and Pat Hanrahan. Brook for GPUs: Stream Computing on Graphics Hardware. In *ACM SIGGRAPH*, pages 777–786, August 2004.

[23] P. Bunyk. RSFQ Random Logic Gate Density Scaling for the Next-Generation Josephson Junction Technology. *IEEE Transactions on Applied Superconductivity*, 13(2):496–497, June 2003.

[24] P. Bunyk, M. Leung, J. Spargo, and M. Dorojevets. Flux-1 RSFQ microprocessor: physical design and test results. *IEEE Transactions on Applied Superconductivity*, 13(2):433–436, June 2003.

[25] J. Carlstrom and et al. A 40 Gb/s Network Processor with PISC Dataflow Architecture. In *Int. Solid-State Circuits Conf.*, pages 60–67, San Francisco, USA, Feb. 2004.

[26] L. Carrington, X. Gao, A. Snavely, , and R. Campbell. Profile of AVUS Based on Sampled Memory Tracing of Basic Blocks. Users Group Conference on 2005 Users Group Conference, jun,.

[27] Tien-Hsin Chao. Holographic Data Storage. http://www.thic.org/pdf/Jan01/NASAJPL.t-schao.010116.pdf.

[28] Y. Chen, G. Y. Jung, D. A. A. Ohlberg, X. M. Li, D. R. Stewart, J. O. Jeppesen, K. A. Nielsen, J. F. Stoddart, and R. S. Williams. Nanoscale molecular-switch crossbar circuits. *Nanotechnology*, 14:462–468, April 2003.

[29] Y. C. Chen and et al. An Access-transistor-free (OT/1R) non-volatile resistance random access memory (RRAM) using a novel threshold switching, self-rectifying chalcogenide device. *IEDM Report 37.4.1*, 2003.

[30] H. Cho, P. Kapur, and K.C. Saraswat. Performance Comparison Between Vertical-Cavity Surface-Emitting Laser and Quantum-Well Modulator for Short-Distance Optical Links. *IEEE Photonics Technology Letters*, 18(3):520–522, February 2006.

[31] UPC Consortium. UPC language specifications v1.2. Technical report, Lawrence Berkeley National Lab, 2005.

[32] Livermore Software Technology Corporation. Getting Started with LS-DYNA. http://www.feainformation.com/m-pdf/IntroDyna.pdf, 2002.

[33] A. DeHo, S. C. Goldstein, P. J. Kuekes, and P. Lincoln. Nonphotolithographic nanoscale memory density prospects. *IEEE Transactions on Nanotechnology*, 4:215–228, March 2005.

[34] A. Dehon. Array-based architecture for fet-based, nanoscale electronics. *IEEE Trans. Nanotechnol.*, 2(1):23–32, 2003.

[35] A. DeHon. Design of programmable interconnect for sublithographic programmable logic array. pages 127–137, Monterey, CA, February 2005. FPGA.

[36] A. DeHon. Nanowire-based programmable architecture. *ACM J. Emerg. Technol. Comput. Syst.*, 1(2):109–162, 2005.

[37] A. DeHon and K. K. Likharev. Hybrid cmos/nanoelectronic digital circuits: Devices, architectures, and design automation. pages 375–382, San Jose, CA, November 2005. ICCAD.

[38] Dell. Data Center Efficiency in the Scalable Enterprise. http://www.dell.com/downloads/global/power/ps1q07-20070210-CoverStory.pdf, February 2007.

[39] Lisa Dhar, Arturo Hale, Howard E. Katz, Marcia L. Schilling, Melinda G. Schnoes, and Fred C. Schilling. Recording media that exhibit high dynamic range for digital holographic data storage. *Optics Letters*, 24(7):487–489, 1999.

[40] DoD. White Paper DoD Research and Development Agenda For High Productivity Computing Systems. http://www.nitrd.gov/subcommittee/hec/hecrtf-outreach/bibliography/20010611_high_productivity_computing_s.pdf, June 2001.

[41] DoD. Report on High Performance Computing for the National Security Community. http://www.nitrd.gov/subcommittee/hec/hecrtf-outreach/bibliography/200302_hec.pdf, July 2002.

[42] Mattan Erez. *Merrimac – High-Performance, Highly-Efficient Scientific Computing with Streams*. PhD thesis, Stanford University, Stanford, California, November 2005.

[43] Mattan Erez, Nuwan Jayasena, Timothy J. Knight, and William J. Dally. Fault Tolerance Techniques for the Merrimac Streaming Supercomputer. In *SC05*, Seattle, Washington, USA, November 2005.

[44] John Feo, David Harper, Simon Kahan, and Petr Konecny. ELDORADO. In *CF '05: Proceedings of the 2nd conference on Computing frontiers*, pages 28–34, New York, NY, USA, 2005. ACM.

[45] Ian Foster and Carl Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Pub., 1999.

[46] Richard Games. Survey and Analysis of the National Security High Performance Computing Architectural Requirements. http://www.nitrd.gov/subcommittee/hec/hecrtf-outreach/bibliography/20010604_hpc_arch_survey_analysis_f.pdf, June 2001.

[47] G. Gao, K. Likharev, P. Messina, and T. Sterling. Hybrid Technology Multithreaded Architecture. In *6th Symp. on Frontiers of Massively Parallel Computation*, pages 98–105, 1996.

[48] A. Gara and et al. Overview of the Bluegene/L system architecture. *IBM J. of Reearch and Development*, (2/3):195–212, 2005.

[49] A. Gayasen, N. Vijaykrishnan, and M. J. Irwin. Exploring technology alternatives for nano-scale fpga interconnects. pages 921–926, Anaheim, CA, June 2005. DAC.

[50] Garth Gibson. Reflections on Failure in Post-Terascale Parallel Computing. In *Int. Conf. on Parallel Processing.*

[51] N. E. Gilbert and M. N. Kozicki. An embeddable multilevel-cell solid electrolyte memory array. *EEE Journal of Solid-State Circuits*, 42:1383–1391, June 2007.

[52] S. C. Goldstein and M. Budiu. Nanofabrics: Spatial computing using molecular electronics. pages 178–189, Goteborg, Sweden, 2001. ISCA.

[53] S. C. Goldstein and D. Rosewater. Digital logic using molecular electronics. page 12.5, San Francisco, CA, February 2002. ISSCC.

[54] S. Graham and et al. *Getting Up to Speed: The Future of Supercomputing.* National Academies Press, 2004.

[55] Jim Gray. Building PetaByte Servers. http://research.microsoft.com/g̃ray/talks/ Building%20Petabyte%20Databases%20(CMU).ppt.

[56] YJ. E. Green, J. W. Choi, A. Boukai, Y. Bunimovich, E. Johnston-Halperin, E. DeIonno, Y. Luo, B. A. Sheriff, K. Xu, Y. S. Shin, H. R. Tseng, J. F. Stoddart, and J. R. Heath. A 160-kilobit molecular electronic memory patterned at 10(̂11) bits per square centimetre. *Nature*, 445:414–417, January 25 2005.

[57] Michael Gschwind. Chip multiprocessing and the cell broadband engine. In *CF '06: Proceedings of the 3rd conference on Computing frontiers*, pages 1–8, New York, NY, USA, 2006. ACM.

[58] D. Guckenberger, J.D. Schaub, D. Kucharski, and K.T. Komegay. 1 V, 10 mW 10 Gb/s CMOS Optical Receiver Front-End. In *2005 IEEE Radio Frequency Integrated Circuits Symposium*, page 309.

[59] P. Kapur H. Cho, K-H Koo and K.C. Saraswat. Performance Comparison between Cu/Low-L, Carbon Nanotube, and Optics for On-chip Global Interconnects, 2007. manuscript.

[60] Mary Hall, Peter Kogge, Jeff Koller, Pedro Diniz, Jacqueline Chame, Jeff Draper, Jeff LaCoss, John Granacki, Jay Brockman, Apoorv Srivastava, William Athas, Vincent Freeh, Jaewook Shin, and Joonseok Park. Mapping irregular applications to DIVA, a PIM-based data-intensive architecture. In *Supercomputing '99: Proceedings of the 1999 ACM/IEEE conference on Supercomputing (CDROM)*, page 57, New York, NY, USA, 1999. ACM.

[61] Mark W. Hart. Step−and−Flash Imprint Lithography for Storage−Class Memory. http://www.molecularimprints.com/NewsEvents/tech_articles/new_articles/EIPBN2007_IBMv2.pdf, 2007.

[62] Hisao Hayakawa, Nobuyuki Yoshikawa, Shinichi Yorozu, and Akira Fujimaki. Superconducting Digital Electronics. *Proc. of the IEEE*, 92(10), October 2004.

[63] P. Hazucha, T. Karnik, J. Maiz, S. Walstra, B. Bloechel, J. Tschanz, G. Dermer, S. Hareland, P. Armstrong, and S. Borkar. Neutron Soft Error Rate Measurements in a 90-nm CMOS Process and Scaling Trends in SRAM from 0.25-mm to 90-nm Generation. In *International Electron Devices Meeting*, pages 21.5.1–21.5.4, December 2003.

[64] J. R. Heath, P. J. Kuekes, G. S. Snider, and R. S. Williams. A defect-tolerant computer architecture: Opportunities for nanotechnology. *Science*, 280:1716–1721, June 1998.

[65] C. A. R. Hoare. Communicating sequential processes. In *Communications of the ACM*, volume 21, pages 666–677, 1978.

[66] L. Hochstein, T. Nakamura, V.R. Basili, S. Asgari, M.V. Zelkowitz, J.K. Hollingsworth, F. Shull, J. Carver, M. Voelp, N. Zazworka, , and P. Johnson. Experiments to Understand HPC Time to Development. *Cyberinfrastructure Technology Watch Quarterly*, Nov. 2006.

[67] T. Hogg and G. S. Snider. Defect-tolerant adder circuits with nanoscale crossbars. *IEEE Trans. Nanotechnol.*, 5(2):97–100, 2006.

[68] T. Hogg and G. S. Snider. Defect-tolerant logic with nanoscale crossbar circuits. *JETTA*, 23(2-3):117–129, 2007.

[69] D. Hopkins and et.al. Circuit techniques to enable a 430 gb/s/mm/mm proximity communication. In *IEEE International Solid State Circuits Conference*, pages 368–369, 2007.

[70] H.Tanaka, M.Kido, K.Yahashi, M.Oomura, and et al. Bit cost scalable technology with punch and plug process for ultra high density flash memorye. pages 14–15. IEEE Symp. on VLSI technology, 2007.

[71] Endicott Interconnect. HyperBGA Technology. http://eitnpt1.eitny.com/contentmanager/literature/HYPERBGA.pdf.

[72] Mary Jane Irwin and John Shen, editors. *Revitalizing Computer Architecture Research*. Computing Research Association, December 2005.

[73] iSuppli. isuppli market tracker, q3. http://www.isuppli.com/catalog/detail.asp?id=8805.

[74] L. Jiang and et.al. Close-loop electro-osmotic micromechannel coolings system for VLSI circuits. *IEEE Trans. CPMT, Part A*, 25:347–355, September 2002.

[75] Bill Joy and Ken Kennedy. *Information Technology Research: Investing in Our Future.* National Coordination Office for Computing, Information, and Communications, Arlington, VA, February 1999.

[76] G. Y. Jung, S. Ganapathiappan, D. A. A. Ohlberg, D. L. Olynick, Y. Chen, W. M. Tong, and R. S. Williams. Fabrication of a 34 x 34 crossbar structure at 50 nm half-pitch by UV-based nanoimprint lithography. *Nano Letters*, 4:1225–1229, July 2004.

[77] G. Y. Jung, E. Johnston-Halperin, W. Wu, Z. N. Yu, S. Y. Wang, W. M. Tong, Z. Y. Li, J. E. Green, B. A. Sheriff, A. Boukai, Y. Bunimovich, J. R. Heath, and R. S. Williams. Circuit fabrication at 17 nm half-pitch by nanoimprint lithography. *Nano Letters*, 6:351–354, March 2006.

[78] P. Kapur and K.C. Saraswat. Optical Interconnections for Future High Performance Integrated Circuits. *Physica E*, 16:620–627, 2003.

[79] B. Keeth and R. Baker. *DRAM Circuit Design: A Tutorial.* IEEE Press, 2000.

[80] T. Kgil and et al. Picoserver: Using 3d stacking technology to enable a compact energy efficient chip multiprocessor. ASPLOS, 2006.

[81] J. Kim, W. Dally, B. Towles, and A. Gupta. Microarchitecture of a high-radix router. In *IProceedings 32th Annual Int. Symp. on Computer Architecture(ISCA)*, pages 420–431, June 2005.

[82] J.S. Kim, W.H. Cha, K.N. Rainey, S. Lee, and S.M. You. Liquid cooling module using FC-72 for elecrtronics cooling. In *ITHERM '06*, pages 604–611, May 2006.

[83] Graham Kirsch. Active Memory: Micron's Yukon. In *IPDPS '03: Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, page 89.2, Washington, DC, USA, 2003. IEEE Computer Society.

[84] David Koester. Exascale Study Application Footprints. http://info.mitre.org/infoservices/selfserve/public_release_docs/2007/07-1449.pdf, 2007.

[85] P. Kogge. The EXECUBE Approach to Massively Parallel Processing. In *Int. Conf. on Parallel Processing*, Chicago, Aug. 1994.

[86] P. Kogge. An Exploration of the Technology Space for Multi-Core Memory/Logic Chips for Highly Scalable Parallel Systems. In *IEEE Int. Workshop on Innovative Architectures*, pages 55–64, Oahu, HI, Jan. 2005.

[87] J. Koomey. Estimating Total Power Consumption By Servers In The U.S. And The World, February 2007. Lawrence Berkeley National Laboratory, Final Report.

[88] Christoforos Kozyrakis and David Patterson. Vector vs. superscalar and VLIW architectures for embedded multimedia benchmarks. In *MICRO 35: Proceedings of the 35th annual ACM/IEEE international symposium on Microarchitecture*, pages 283–293, Los Alamitos, CA, USA, 2002. IEEE Computer Society Press.

[89] P. J. Kuekes, G. S. Snider, and R. S. Williams. Crossbar nanocomputers. *Sci. Am.*, 293(5):72–80, 2005.

[90] Sandia National Labs. Photo of Red Storm. http://www.sandia.gov/NNSA/ASC/images/platforms/RedStorm-montage.jpg.

[91] J. H. Lee, X. Ma, D. B. Strukov, and K. K. Likharev. Cmol. pages 3.9–3.16, Palm Springs, CA, May 2005. NanoArch.

[92] J.D Lee, S.H. Hur, and J.D. Choi. Effects of floating-gate interference on NAND flash memory cell operation. *IEEE Electron. Device Letters*, 23(5):pp. 264–266, May 2002.

[93] Ana Leon, Jinuk Shin, Kenway Tam, William Bryg, Francis Schumacher, Poonacha Kongetira, David Weisner, and Allan Strong. A Power-Efficient High-Throughput 32-Thread SPARC Processor. pages 98–99, San Francisco, CA, 2006. ISSCC.

[94] K. Likharev and D. Strukov. *CMOL: Devices, circuits, and architectures*, pages 447–478. Springer, 2005.

[95] R. J. Luyken and F. Hofmann. Concepts for hybrid CMOS-molecular non-volatile memories. *Nanotechnology*, 14:273–276, February 2003.

[96] X. Ma, D. B. Strukov, J. H. Lee, and K. K. Likharev. Afterlife for silicon: Cmol circuit architectures. pages 175–178, Nagoya, Japan, July 2005. IEEE Nanotechnol.

[97] Junichiro Makino, Eiichiro Kokubo, and Toshiyuki Fukushige. Performance evaluation and tuning of GRAPE-6 towards 40 real Tflops. ACM/IEEE SC Conference, 2003.

[98] M. Mamidipaka and N. Dutt. eCACTI: An Enhanced Power Estimation Model for On-chip Caches, 2004. Center for Embedded Computer Systems (CESC) Tech. Rep. TR-04-28.

[99] J. Mankins. Technology Readiness Levels. http://www.hq.nasa.gov/office/codeq/trl/trl.pdf, April 1995.

[100] John Markoff. Pentagon Redirects Its Research Dollars; University Scientists Concerned by Cuts in Computer Project. *New York Times*, pages section C, page 1, column 2, April 2005.

[101] M. Masoumi, F. Raissi, M. Ahmadian, and P. Keshavarzi. Design and evaluation of basic standard encryption algorithm modules using nanosized complementary metal-oxide-semiconductor-molecular circuits. *Nanotechnology*, 17(1):89–99, 2006.

[102] N. Nakajimaand F. Matsuzaki, Y. Yamanashi, N. Yoshikawa, M. Tanaka, T. Kondo, A. Fujimaki, H. Terai, , and S. Yorozu. Design and implementation of circuit components of the SFQ microprocessor, CORE 1. *9th Int. Superconductivity Conf.*, 2003.

[103] P. Mehrotra and P. Franzon. Optimal Chip Package Codesign for High Performance DSP. *IEEE Trans. Advanced Packaging*, 28(2), May 2005.

[104] S.E. Michalak, K.W. Harris, N.W. Hengartner, B.E. Takala, and S.A. Wender. Predicting the number of fatal soft errors in Los Alamos National Laboratory's ASC Q supercomputer. *IEEE Transactions on Device and Materials Reliability*, 5(3):329–335, September 2005.

[105] John Michalakes, Josh Hacker, Richard Loft, Michael O. McCracken, Allan Snavely, Nicholas J. Wright, Tom Spelce, Brent Gorda, and Robert Walkup. WRF Nature Run. Int. Conf. for High Performance Computing, Networking, and Storage, nov 2007.

[106] Micron. Micron system power calculator. http://www.micron.com/support/designsupport/tools/powercalc/powercalc.aspx.

[107] N. Miura, H. Ishikuro, T. Sakurai, and T. Kuroda. A 0.14 pj/bit inductive coupling inter-chip data transceiver with digitally-controlled precise pulse shaping. In *IEEE International Solid State Circuits Conference*, pages 358–608, 2007.

[108] Jos Moreira, Michael Brutman, Jos Castaos, Thomas Engelsiepen, Mark Giampapa, Tom Gooding, Roger Haskin, Todd Inglett, Derek Lieber, Pat McCarthy, Mike Mundy, Jeff Parker, and Brian Wallenfelt. Designing a Highly-Scalable Operating System: The Blue Gene/L Story. ACM/IEEE SC Conference, 2006.

[109] R. Murphy, A. Rodrigues, P. Kogge, , and K. Underwood. The Implications of Working Set Analysis on Supercomputing Memory Hierarchy Design. Cambridge, MA, 2005. International Conference on Supercomputing.

[110] Umesh Nawathe, Mahmudul Hassan, Lynn Warriner, King Yen, Bharat Upputuri, David Greenhill, Ashok Kumar, and Heechoul Park. An 8-core, 64-thread, 64-bit, power efficient SPARC SoC. San Francisco, CA, 2007. ISSCC.

[111] John Nickolls. GPU Parallel Computing Architecture and the CUDA Programming Model. In *HotChips 19*, August 2007.

[112] R. Numrich and J. Reid. Co-Array Fortran for parallel programming. In *ACM Fortran Forum 17, 2, 1-31.*, 1998.

[113] Y. Okuyama, S. Kamohara, Y. Manabe, T. Kobayashi K. Okuyamaand K. Kubota, and K. Kimura. Monte carlo simulation of stress-induced leakage current by hopping conduction via multi-traps in oxide. pages 905–908. IEEE Electron. Devices Meeting, December 1998.

[114] A.K. Okyay, D. Kuzum, S. Latif, D.A.B. Miller, and K.C. Saraswat. CMOS Compatible Silicon-Germanium Optoelectronic Switching Device, 2007. Manuscript.

[115] A.J. Oliner, R.K. Sahoo, J.E. Moreira, and M. Gupta. Performance implications of periodic checkpointing on large-scale cluster systems. In *International Parallel and Distributed Processing Symposium*, April 2005.

[116] D. R. Stewart P. J. Kuekes and R. S. Williams. The crossbar latch: Logic value storage, restoration, and inversion in crossbar circuits. *J. Appl. Phys.*, 97(3):034301, 2005.

[117] R. Palmer, J. Poulton, W.J. Dally, J. Eyles, A.M. Fuller, T. Greer, M. Horowitz, M.Kellan, F. Quan, and F. Zarkesshvari. A 13 mw 6.25 gb/s transceiver in 90 nm cmos for serial chip-to-chip communication. In *IEEE International Solid State Circuits Conference*, pages 440–614, 2007.

[118] A. Petitet, R. C. Whaley, J. Dongarra, and A. Cleary. HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers. http://www.netlib.org/benchmark/hpl/, 2004.

[119] Daniel Reed, editor. *The Roadmap for the Revitalization of High-End Computing*. Computing Research Association, 2003.

[120] K. Reick, P.N. Sanda, S. Swaney, J.W. Kellington, M. Floyd, and D. Henderson. Fault Tolerant Design of the IBM Power6 Microprocessor. In *HotChips XIX*, August 2007.

[121] J. R. Reimers, C. A. Picconnatto, J. C. Ellenbogen, , and R. Shashidhar, editors. *Molecular Electronics III*, volume 1006. Ann. New York Acad. Sci.

[122] M. M. Ziegler C. A. Picconatto S. Das, G. Rose and J. E. Ellenbogen. *rchitecture and simulations for nanoprocessor systems integrated on the molecular scale*, pages 479–515. Springer, 2005.

[123] Karthikeyan Sankaralingam, Ramadass Nagarajan, Haiming Liu, Changkyu Kim, Jaehyuk Huh, Nitya Ranganathan, Doug Burger, Stephen W. Keckler, Robert G. McDonald, and Charles R. Moore. TRIPS: A polymorphous architecture for exploiting ILP, TLP, and DLP. *ACM Trans. Archit. Code Optim.*, 1(1):62–93, 2004.

[124] B. Schroeder and G.A. Gibson. A Large-scale Study of Failures in High-performance-computing Systems. In *International Conference on Dependable Systems and Networks (DSN)*, pages 249–258, June 2006.

[125] B. Schroeder and G.A. Gibson. Disk Failures in the Real World: What Does an MTTF of 1,000,000 Hours Mean to You? In *USENIX Conference on File and Storage Technologies (FAST)*, February 2007.

[126] Roy Schwitters and et al. Report on High Performance Computing for the National Security Community. http://fas.org/irp/agency/dod/jason/asci.pdf, October 2003.

[127] A. Shcham and K. Bergman. Building Ultralow-Latency Interconnection Networks Using Photonic Integration. *IEEE Micro*, 27(4):6–20, July-August 2007.

[128] P. Shivakumar, M. Kistler, S.W. Keckler, D. Burger, and L. Alvisi. Modeling the Effect of Technology Trends on Soft Error Rate of Combinational Logic. In *International Conference on Dependable Systems and Networks (DSN)*, pages 389–398, June 2002.

[129] Paul H. Smith, Thomas Sterling, and Paul Messina. *Enabling Technologies for Petaflops Computing*. MIT Press, 2005.

[130] A. Snavely, L. Carrington, N. Wolter, J. Labarta, R. Badia, and A. Purkayastha. A Framework for Application Performance Modeling and Prediction. pages 112–123, Baltimore, MD, November 2002. ACM/IEEE Conference on Supercomputing.

[131] A. Snavely, M. Tikir, L. Carrington, and E. Strohmaier. A Genetic Algorithms Approach to Modeling the Performance of Memory-bound Computations. Reno, NV, November 2007. ACM/IEEE Conference on Supercomputing.

[132] Allan Snavely, Larry Carter, Jay Boisseau, Amit Majumdar, Kang Su Gatlin, Nick Mitchell, John Feo, and Brian Koblenz. Multi-processor performance on the Tera MTA. In *Supercomputing '98: Proceedings of the 1998 ACM/IEEE conference on Supercomputing (CDROM)*, pages 1–8, Washington, DC, USA, 1998. IEEE Computer Society.

[133] G. Snider. Computing with hysteretic resistor crossbars. *Appl. Phys. A-Mater. Sci. Process.*, 80(6):1165–1172, 2005.

[134] G. Snider, P. Kuekes, T. Hogg, and R. S. Williams. Nanoelectronic architectures. *Appl. Phys. A-Mater. Sci. Process.*, 80(6):1183–1195, 2005.

[135] G. Snider, P. Kuekes, and R. S. Williams. Cmos-like logic in defective, nanoscale crossbars. *Nanotechnology*, 15(8):881–891, 2004.

[136] G. S. Snider and P. J. Kuekes. Nano state machines using hysteretic resistors and diode crossbars. *IEEE Trans. Nanotechnol.*, 5(2):129–137, 2006.

[137] G. S. Snider and R. S. Williams. Nano/cmos architectures using a field-programmable nanowire interconnect. *Nanotechnology*, 18(3):035204, 2007.

[138] L. Spainhower and T. A. Gregg. IBM S/390 Parallel Enterprise Server G5 fault tolerance: A historical perspective. *IBM Journal of Research and Development*, 43(5/6):863–874, 1999.

[139] M. Stan, P. D. Franzon, S. C. Goldstein, J. C. Lach, and M. M. Ziegler. Molecular electronics: From devices and interconnect to circuits and architecture. *Proc. IEEE*, 91:1940–1957, November 2003.

[140] Thomas Sterling. HTMT-class Latency Tolerant Parallel Architecture for Petaflops-scale Computation. http:www.cs.umd.edu/users/als/NGS07/Presentations/ 8am-Sunday-Session/GaoSterling.pdf, 1999.

[141] D. B. Strukov and K. K. Likharev. Prospects for terabit-scale nanoelectronic memories. *Nanotechnology*, 16:137–148, January 2005.

[142] D. B. Strukov and K. K. Likharev. Defect-tolerant architectures for nanoelectronic crossbar memories. *Journal of Nanoscience and Nanotechnology*, 7:151–167, January 2007.

[143] F. Sun and T. Zhang. Defect and transient fault-tolerant system design for hybrid CMOS/-nanodevice digital memories. *IEEE Transactions on Nanotechnology*, 6:341–351, May 2007.

[144] T. Sunaga, P. Kogge, and et al. A Processor In Memory Chip for Massively Parallel Embedded Applications. In *IEEE J. of Solid State Circuits*, pages 1556–1559, Oct. 1996.

[145] D. Tarjan, S. Thoziyoor, and N. P. Jouppi. CACTI 4.0, 2006. HP Labs Tech. Rep. HPL-2006-86.

[146] Michael Bedford Taylor, Walter Lee, Jason Miller, David Wentzlaff, Ian Bratt, Ben Greenwald, Henry Hoffmann, Paul Johnson, Jason Kim, James Psota, Arvind Saraf, Nathan Shnidman, Volker Strumpen, Matt Frank, Saman Amarasinghe, and Anant Agarwal. Evaluation of the Raw Microprocessor: An Exposed-Wire-Delay Architecture for ILP and Streams. *SIGARCH Comput. Archit. News*, 32(2):2, 2004.

[147] IBM Blue Gene team. Overview of the IBM Blue Gene/P project. *IBM J. RES. & DEV*, 52(1/2):199–220, 2008.

[148] J. Tour. *Molecular Electronics*. World Scientific, Singapore, 2003.

[149] S. Vangal and et al. An 80-Tile 1.28 TFLOPS Network-on-chip in 65 nm CMOS. pages 98–99, San Francisco, CA, 2007. ISSCC.

[150] S. Vangal and et.al. An 80-tile 1.28 tflops network-on-chip in 65 nm cmos. In *IEEE International Solid State Circuits Conference*, pages 587–589, 2007.

[151] Michael J. Voss and Rudolf Eigenmann. High-Level Adaptive Program Optimization with ADAPT. pages 93–102. Proc. of PPOPP'01, ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, 2001.

[152] David Wallace. Compute Node Linux: Overview, Progress to Date and Roadmap. http://www.nccs.gov/wp-content/uploads/2007/08/wallace_paper.pdf, 2007.

[153] T. Wang, Z. Qi, and C. A. Moritz. Opportunities and challenges in application-tuned circuits and architectures based on nanodevices. pages 503–511, Italy, April 2004. CCF.

[154] W. Wang, M. Liu, and A. Hsu. Hybrid nanoelectronics: Future of computer technology. *J. Comp. Sci. Technolog.*, 21(6):871–886, 2006.

[155] J. Weinberg, M. O. MCracken, A. Snavely, and E. Strohmaier. Quantifying Locality In The Memory Access Patterns of HPC Applications. Seattle, WA, November 2005. SC.

[156] M. H. White, D. Adams, and J. Bu. On the go with sonos. In *IEEE Circuits and Devices*, volume 16, page 22, 2000.

[157] Wikipedia. *Cray X-MP*.

[158] S. Wilton and N. P. Jouppi. An Enhanced Access and Cycle Time Model for On-Chip Caches, 1994. DEC WRL Tech. Rep. 93/5.

[159] V. Yalala, D. Brasili, D. Carlson, A. Huges, A. Jain, T. Kiszely, K. Kodandapani, A. Varadharajan, and T. Xanthopoulos. A 16-Core RISC Microprocessor with Network Extensions. pages 100–101, San Francisco, CA, 2006. ISSCC.

[160] Katherine Yelick, Luigi Semenzato, Geoff Pike, Carleton Miyamoto, Ben Liblit, Arvind Krishnamurthy, Paul Hilfinger, Susan Graham, David Gay, Phillip Colella, and Alexander Aiken. Titanium: A High-Performance Java Dialect, journal = Concurrency: Practice and Experience. 10:825–836, 1998.

[161] H.Y. Zhang, D. Pinjala, and T. Poi-Siong. Thermal Management of high power dissipation eletronic packages: from air cooling to liquid cooling. In *EPTC 2003*, pages 620–625, 2003.

[162] L. Zhang, J. Wilson, R. Bashirulla, L. Luo, J. Xu, and P. Franzon. A 32gb/s on-chip bus with driver pre-emphasis signaling. In *IEEE Custom Integrated Circuits Conference*, pages 773–776, 2006.

[163] M. M. Ziegler, C. A. Picconatto, J. C. Ellenbogen, A. Dehon, D. Wang, Z. H. Zhong, and C. M. Lieber. Scalability simulations for nanomemory systems integrated on the molecular scale. *Molecular Electronics III*, 1006:312–330, 2003.

[164] M. M. Ziegler and M. R. Stan. Cmos/nano co-design for crossbar-based molecular electronic systems. *IEEE Trans. Nanotechnol.*, 2(4):217–230, 2003.

Index

3D packaging, 131
3D stacking, 122

fused multiply-add, 57

activate, 117
activity factor, 135
adaptive mesh refinement, 66, 71
Advanced Memory Buffer, 30
aggressive strawman, 128, 175
air conditioner, 144
Air Vehicle Unstructured Solver, 66, 71, 75, 77, 78
air-cooling, 164
AIX, 36
AMB, *see* Advanced Memory Buffer
AMD K8, 28
Amdahl's Law, 65, 227
AMR, *see* adaptive mesh refinement
anti-fuse, 109
application performance, 7
applicationa
  cross-class, 11
applications
  Category I, 71, 72, 75, 79
  Category II, 71, 72, 79, 80
  Category III, 71, 72, 75, 79
  Category IV, 72
archival storage, 73, 127, 212
array multi-core, 29
ASCI, 23
ASCI Red, 17
aspect ratio, 106
asynchronous, 44
automatic parallelization, 40
autotuners, 26
Azul, 29

Backus, 62
balanced design, 6
Ball Grid Arrays, 140
bandwidth, 6, 113
  bisection, 6, 69
  checkpoint, 6
  I/O, 6
  local memory, 6
  on-chip, 6
  ratio, 7
  requirements, 72
BCH code, 108
Beowulf, 34, 40
Berkeley, 25
BGA, 140
bisection bandwidth, 6, 69, 74
Bit Error Rate, 131, 217
bit-line, 106, 116
  capacitance, 106
Brook-GPU, 28
BTBB, 152
bulk silicon, 93
bussed interconnect, 130
bytes to flops ratio, 7

C*, 40
C++, 152
CACTI, 119
CAF, 45
capability computing, 7, 13, 79, 80
capacity computing, 7, 12, 79
Carbon Nanotubes, 137
Cascade, 46
Catamount, 36, 37
Catamount System Libraries, 38
Category I applications, 71, 72, 75, 79
Category II applications, 71, 72, 75, 79, 80
Category III applications, 71, 72, 75, 79
Category IV applications, 72, 75
Cell, 38
cell modeling, 80
cell power, 116, 119
CFD:, 253

chalcogenide glass, 108
challenges, 2, 209
  concurrency, 214
  energy, 209
  locality, 214
  memory, 212
  power, 209
  resiliency, 217
  storage, 212
channel, 182
Chapel, 46, 152
charge pumps, 118
checkpoint, 149, 184
  bandwidth, 6
  in Blue Gene/L, 149
  rollback, 71, 149
chip junction temperature, 142
chip level multi-processing, 28
chip-level reliability, 147
Cilk, 43, 152
circuit switched network, 136
circuit switching, 132
clock, 55, 56
Clock and Data Recovery, 134
CM-2, 40, 41
CMFortran, 40
CMOL, 100
CMP, 29
CNL, 37
CNT, 137
Co-Array Fortran, 40, 45, 152
collective network, 172
collectives, 44
communicating sequential processes, 39
Compaq Himalaya, 149
computational fluid dynamics, 69, 71
computational rates, 5
compute card, 170
compute chip, 170
compute node, 172
Compute Node Linux, 37
concurrency, 57
concurrency challenge, 2, 214
configurable logic, 137
constant field scaling, 49
constant voltage scaling, 49
consumer-class disks, 122
core, 175, 178
CORE-1, 101
COTS architectures, 23
Cray
  Black Widow, 132
  MTA, 30
  XMP, 63
  XMT, 30
  XT, 31
  XT3, 38
  XTn, 39
criticality, 219
cross-class applications, 11
crossbar, 100, 132
crosspoint memory array, 120
crosspoints, 137
cryostat coolers, 101
CSP, 39
cubic scaling, 27
CUDA, 28, 152
current flow, 127
custom architectures, 23
Cyclops, 29, 31

data center system, 8
data center systems, 12, 13
Data parallel languages, 40
dataflow, 39
Datastar, 63
DDR, 113, 115
DDR2, 29, 113, 115
DDR3, 113, 115
Defense Science Board, 21
Delay Locked Loop, 130
delivered power, 32
departmental systems, 9, 14, 80
dependability, 26
device resiliency scaling, 147
die stacking, 115
die thinning, 115
dielectric breakdown, 147
DIMM, 29, 172
direct chip-chip interconnect, 131
direct immersion cooling, 143
disk
  transfer time, 125
  capacity, 123
  consumer, 122
  drive reliability, 145

enterprise, 123
handheld, 123
seek time, 125
storage, 184
technology, 122
transfer rate, 125
dissipated power, 32, 88
distributed memory, 199
DIVA, 31
DLL, 130
DNA, 97
domino logic, 97
dragonfly topology, 182
DRAM, 106, 110, 212
activate mode, 117
capacitor, 116
chip architecture, 118
embedded, 107
fast cycle, 107
idle mode, 117
latency, 30
modes, 116
power, 115, 116
precharge mode, 117
read mode, 117, 118
reduced latency, 107
refresh mode, 117
reliability, 109
SER, 111
stacking, 115
Voltage Scaling, 116
write mode, 117, 118

E/O conversion, 192
E3 Initiative, 202
E3SGS, 20
Earth Simulator, 31, 41
earthquake modeling, 79
ECC, 111, 112, 147
efficiency, 32, 54
EIP, 153
EIPs, 153
electrical anti-fuses, 109
electro-migration, 142, 147
embarrassingly parallel, 72
embedded DRAM, 107
embedded systems, 10, 14, 34, 80
enabling technologies, 23
encoding for resiliency, 147
endurance, 108
energy, 210
and power challenge, 209
challenge, 2
efficiency, 95
per cycle, 88
per operation, 88, 89, 178, 210
scaling, 27, 178
enterprise-class disks, 123
environments, 198
EOS/DIS, 74
Exa Instructions Processed per second, 153
exa-sized systems, 8
exaflop, 9
exaflops, 9
Exascale, 9
EXECUBE, 29, 31
execution model, 39
executive, 37
external cooling mechanisms, 142

fail-over, 149
failure rate, 110, 145
Failures in time, 110
Fast Cycle DRAM, 107
fault recovery, 149
FB-DIMM, 30, 164
Fe-RAM, 120
featherweight supercomputer, 24
feature size, 47, 104
FeRAM, 120
ferro-electric RAM, 120
FG NAND, 109
field programmable gate array, 98, 100
field-programmable nanowire interconnect, 100
file storage, 73, 162, 212
fine line technologies, 141
FINFET, 94
FIT, 110, 145
Flash memory, 107, 213
flash rewrites, 213
floating point unit, 175, 176
floating trap layer, 108
flop, 176
flops, 5
FLUX-1, 101
flying bits, 111

Fortress, 47
FPNI, 100
FPU power, 210
front-side bus, 30
full swing signaling, 130
fully depleted, 94
fully depleted SOI, 93
fuse, 112

gap, 219
GASNet, 47
gate
  capacitance, 92
  dielectric, 92
  leakage, 92
  oxide, 92
ghost node, 59
ghost region, 41
gigascale, 9, 17
global channel, 183
global interconnect, 72
GPFS, 38
GPU, 28, 152
Grape, 7
graphs, 71
grid, 10
group, 175, 183
GUPS, 82, 212

HAL, 37
handheld-class disks, 123
hard failures, 147
Hardware Abstraction Layer, 37
heating, ventilating, and air conditioning, 32
heatsink, 165
HECRTF, 23
helper flip-flop, 118
hierarchial multi-core, 29
high performance CMOS, 89
High performance Linpack benchmark, 9
High Productivity Computing Systems, 17, 20, 22, 198
high radix routers, 132
high-end computing, 9
high-K gate dielectric, 92
high-performance computing, 9
Holographic memory, 126
HPC, 9
HPCMO test case, 63
HPF, 39–41
HPL, 9, 69, 75, 77
HPUX, 36
hurricane forecast, 78
HVAC, 32
hybrid logic, 97
hybrid technology, 100
Hybrid Technology Multi-Threaded, 17, 100
HYCOM, 72
Hycom, 66
HyperBGA, 140

I/O bandwidth, 6
IBM
  Blue Gene, 31, 38, 39, 63, 170
  Blue Gene/L, 63, 149
  Cell, 30
  Cyclops, 29
  G5, 149
  Power, 39
  Power 4, 63
  Power 6, 34
  Power6, 149
ILP, 56
imprint lithography, 121
Infiniband, 31
innovation trends in programming languages, 151
instruction level parallelism, 56
instructions per second, 5
interconnect, 3, 127
  bus, 130
  direct chip-chip, 131
  energy loss, 127
  off-chip wired, 130
  on-chip and wired, 130
  studs, 141
  switched, 132
  time loss, 127
internal cooling mechanisms, 142
intrinsic delay, 89
ionizing radiation, 111
IPS, 5
Irvine Sensors, 141
Itanium, 29, 34
ITRS, 47

JAVA, 47

Java, 152
JEDEC, 113
JJ, 100
John Backus, 62
Josephson Junction, 100

Kiviat diagrams, 61

L1 cache, 29
LAPI, 47
laser drilled vias, 141
laser-trimmed fuses, 109
latency, 113, 127
latency requirements, 72
Level 1 Packaging, 139
Level 1 packaging, 140
Level 2 Packaging, 139
Level 2 packaging, 141
Level 3 Packaging, 139
life cycle of programming languages, 152
lightweight kernel, 199
link card, 172
Linpack, 53
Linux, 36
liquid cooling, 166, 173
Lisp, 40
list search, 82
Little's Law, 72, 77
load balancing, 199
Loadleveler, 38
local channel, 183
local memory bandwidth, 6
locales, 46
locality, 68, 69, 75
locality challenge, 2, 214
locality clusters, 72
locality-aware architectures, 226
logic
  hybrid, 97
  low power, 89
  nonsilicon, 97
low operating power CMOS logic, 89
low swing interconnect, 130
low voltage signaling, 180
LS-DYNA, 10
Lustre, 38
Luxterra, 135

Mach-Zender modulator, 134
macrokernel, 37
magnetic disks, 122
magnetic RAM, 120
Magnetic Random Access Memory, 109
magnetic tunnel junctions, 109
main memory, 5, 212
main memory power, 211
mantle physics, 79
manycore, 24, 26, 28
Maspar, 40, 41
Maui scheduler, 38
mean time to interrupt, 145
membrane modeling, 79
memory, 212
  bandwidth, 113, 115
  bank, 118, 157
  capacity, 59
  cell power, 116
  challenge, 2, 212
  consistency, 152
  contoller, 30
  footprint, 72
  hierarchy, 103
  intensive applications, 66
  latency, 113
  main, 5
  management, 36
  mat, 118
  module, 115
  packaging, 114
  power, 115, 119
  requirements, 72
  socket reliability, 111
  sub-bank, 118
  wall, 18, 35, 103, 216
Merrimac, 178
message passing, 39, 172
metadata, 73, 127
metrics, 5
micro-FBGA, 115
micro-FBGA package, 115
microkernel, 37
Middleware, 38
midplane, 172
minimal routing, 183
MLC, 107
MODFET, 135
module level cooling, 142

molecular logic, 97
molecular switches, 100
Moore's Law, 63, 65
MPI, 39, 44, 152
MPI-2, 44
MPICH, 44
MRAM, 102, 109, 120
MTJ, 109
MTTI, 145
multi-core, 24, 25, 28, 31, 91
multi-level cell, 107
multi-threading, 28, 30, 43
multiple drug interactions, 79
mux/demux, 121
Myrinet, 31

NAND flash, 107
NAND memory, 109
Nano-enabled Programmable Crosspoints, 137
nanomemory, 213
nanopositioners, 135
nanowire, 100
national security applications, 22
nature run, 79
new applications, 13
Niagara, 29, 30, 34
NoC, 192, 193
node, 164, 172, 175, 181
non silicon logic, 97
non-minimal routing, 184
non-volatile memory, 107, 120
nonvolatile switches, 98
north bridge, 30
NT, 37
NVRAM, 120

O/E conversion, 192
object-oriented programming, 152
OCM, 196
off-chip wired interconnect, 130
on-chip
    access, 179
    bandwidth, 6
    data transport, 180
    interconnect, 28
    wired interconnect, 130
open row, 120
OpenMP, 40, 42
OpenMPI, 44
operating environment, 35, 150, 198
Opteron, 34
optical
    alignment, 135
    circuit switch, 136
    interconnection, 191
    logic, 97
    modulators, 134, 136
    MOSFETs, 136
    packet switch, 136
    router, 192
optical receiver, 135
optically connected modules, 196
OS kernels, 151
out of core algorithms, 73
out of order execution, 28
overall concurrency, 55
Overflow, 69

p-threads, 151
packaging, 114, 139
    Level 1, 139, 140
    Level 2, 139, 141
    Level 3, 139
packet switched network, 136
packet switching, 132
page-oriented memory, 126
pages, 36
parallelism, 25, 54, 55
parity, 147
partially depleted, 94
partially depleted SOI, 93
Partitioned Global Address Space, 45
PBS, 38
PCB, 141
PCRAM, 108, 120
PDE, 71
PDU, 32, 33
peak bandwidth, 127
peak performance, 17
persistent storage, 6
persistent surveillance, 11, 25
peta-sized systems, 8
petaflops, 17
Petascale, 9, 17, 80
PGAS, 45
pGAS, 32, 40, 152, 172, 206

phase change memory, 108
phase change RAM, 120
Phased Locked Loop, 130
photonic interconnect, 192
photonics, 23
physical attributes, 6
PIM, 31
pipelined multi-core, 29
pipelining, 113, 114
PITAC, 21
PLA, 100
PLL, 130
point-to-point interconnect, 130
popular parallel programming, 24
POSIX API, 43
power, 88, 210
  challenge, 2
  delivered, 32
  density, 88, 89
  distribution, 149
  efficiency, 32
  wall, 35, 53
Power Distribution Unit, 32
Power Supply Unit, 32
power-adaptive architectures, 227
precharge, 117
Printed Circuit Board, 141
process control thread, 37, 38
process management, 36
Processing In Memory, 31
processor channel, 183
processor chip, 175
program synchronization, 152
programmable crosspoints, 137
programmable logic array, 100
programmable redundancy, 109
programming languages
  innovative path, 151
  life cycle, 152
  road map, 152
  standardization path, 152
  standardization trends, 152
programming model, 38
property checking for resiliency, 148
protein folding, 79
prototyping phase, 3
PSU, 32
PThreads, 43
Pthreads, 152
PVFS, 38

Q. Kernel, 37
QCA, 97
Quadrics, 31
quantum cellular automata, 97
quantum computing, 97
quantum well modulators, 136
quilt packaging, 181
quintessential kernel, 37

RA, 69
rack, 175, 182, 184
radar plots, 61
radiation hardness, 100, 109, 111
Random Access, 69
Rapid Single Flux Quantum, 100, 222
RAW, 29, 31
read-write cycles, 108
real-time, 7, 13
recommendations, 2
red shift, 62
Red Storm, 9, 31, 36, 164
Reduced Latency DRAM, 107
reduced latency DRAM, 113
reductions, 42
redundancy, 112
Reed-Solomon code, 108
refresh time, 110
register files, 97
reliability, 110, 112
  DRAM, 109
  socket, 111
Reliability, Maintainability, and Availability, 111
remote memory accesses, 5
replication for resiliency, 148
research agenda, 3
research thrust areas, 218
resiliency, 144, 217
  encoding, 147
  sparing, 148
  causes, 147
  challenge, 2, 217
  property checking, 148
  replication, 148
  scrubbing, 148
  techniques, 147

resistive memory, 137
resistive RAM, 120
retention time, 110, 112, 114
RFCTH, 69
ring modulator, 134
RLDRAM, 113
RMA, 111
Rmax, 53
road map for programming languages, 152
roll-back, 149
router chip, 183, 184
routers, 132
Rpeak, 53
RRAM, 120

Sandia National Labs, 9
SATA, 184
scheduler, 36
SCI, 36
SCM, 222
SCP, 141
scratch storage, 6, 73, 162, 212
scrubbing for resiliency, 148
SDRAM, 115
Seastar, 31
SECDED, 147
seek time, 125
seek times, 214
sense-amp, 106, 111, 118
SER, 106, 111
    DRAM, 111
    SRAM, 111
SERDES, 130, 220
server blades, 18
server systems, 17
SEU, 145, 147, 218
shared memory model, 41
signal to noise ratio, 131
signalling on wire, 130
signalling rate, 127
silicon carrier, 140, 141
Silicon on Insulator, 93, 94
silicon photonic integration, 134
SIMD, 39–41, 152, 179
Single Chip Package, 141
single event upset, 145, 147, 218
single mode waveguides, 134
slab allocator, 36
SLC Flash, 108
slice, 3
SMP, 31, 34
SNIC, 222
SNR, 131
socket, 145, 165
soft error rate, 106, 111
SOI, 92–94, 134
Solaris, 36
solder bump, 144
SONOS memory, 108, 109
sparing for resiliency, 148
spatial locality, 68
speedup, 61
spintronics, 23
SPMD, 45
sPPM, 80
SRAM, 97, 106
SRAM SER, 111
SSTL, 30
stacked cell DRAM, 106
static RAM, 97, 106
storage
    capacity, 5
    challenge, 2, 212
    persistent, 6
    scratch, 6
Storm-1, 30
strained silicon, 92
strawman-aggressive, 175
STREAM, 69
stream processors, 30
structured grids, 71
sub-threshold leakage, 92, 93
sub-threshold slope, 93, 94
Sun Niagara, 29, 30, 132
super-cores, 192
super-pipelining, 89
supercomputing, 9, 31
superconducting, 23
supply voltage scaling, 94
sustainable bandwidth, 127
sustained performance, 17, 150
switched interconnect, 130, 132
synchronous, 44
system architecture phase, 3
System Call Interface, 36
systolic, 39

taper, 179
TEC, 136, 142
technology demonstration phase, 3
temperature, 147
temporal locality, 68
tera-sized systems, 8
teraflop, 17
teraflops, 17
Teraflops Research Chip, 29, 31
Terascale, 9, 14, 17
thermal electric coolers, 136
thermal electric cooling, 142
thermal resistance, 142
thermal stress, 147
thread level concurrency, 55, 56
threads, 43, 199
threshold voltage, 92, 95, 147
Through Silicon Vias, 131, 140
through wafer vias, 115
thrust areas, 218
tipping point, 20
Titanium, 152
TLC, 55, 56
TMR, 149
Top 500, 53
top 500, 53
total concurrency, 57
transactional memory, 29, 152
transfer rate, 125
transfer time, 125
transistor density, 88
transport model, 92
trench cell DRAM, 106
tri-gate transistors, 94
triple modular redundancy, 149
TRIPS, 31
Turing lecture, 62
two phase cooling, 143

UltraSparc, 34
Unified Parallel C, 45
Uninterruptible Power Supply, 32
unstructured grids, 71
UPC, 40, 45, 152
UPS, 32, 33
upscaling, 13

variability, 147
Variable Retention Time, 111
VCSEL, 134
Vdd, 89
vector, 40
vertical-cavity surface-emitting laser, 134
via, 115
Virtual File System interface, 36
virtualization, 151
visualization, 74
VLIW, 179
Voltage Regulator, 32
von Neumann, 35
von Neumann bottleneck, 62
VR, 32
VRT, 111, 112

weak scaling, 71, 77
wearout, 147
Windows NT, 37
Windows Server 2003, 37
wordline, 118
working sets, 72
WRF, 63, 66, 69, 71, 75, 77–79

X10, 47, 152
X10q, 29
Xelerator X10q, 29
Xeon, 34

YARC router, 132
YUKON, 31

zettaflops, 20
ZPL, 46