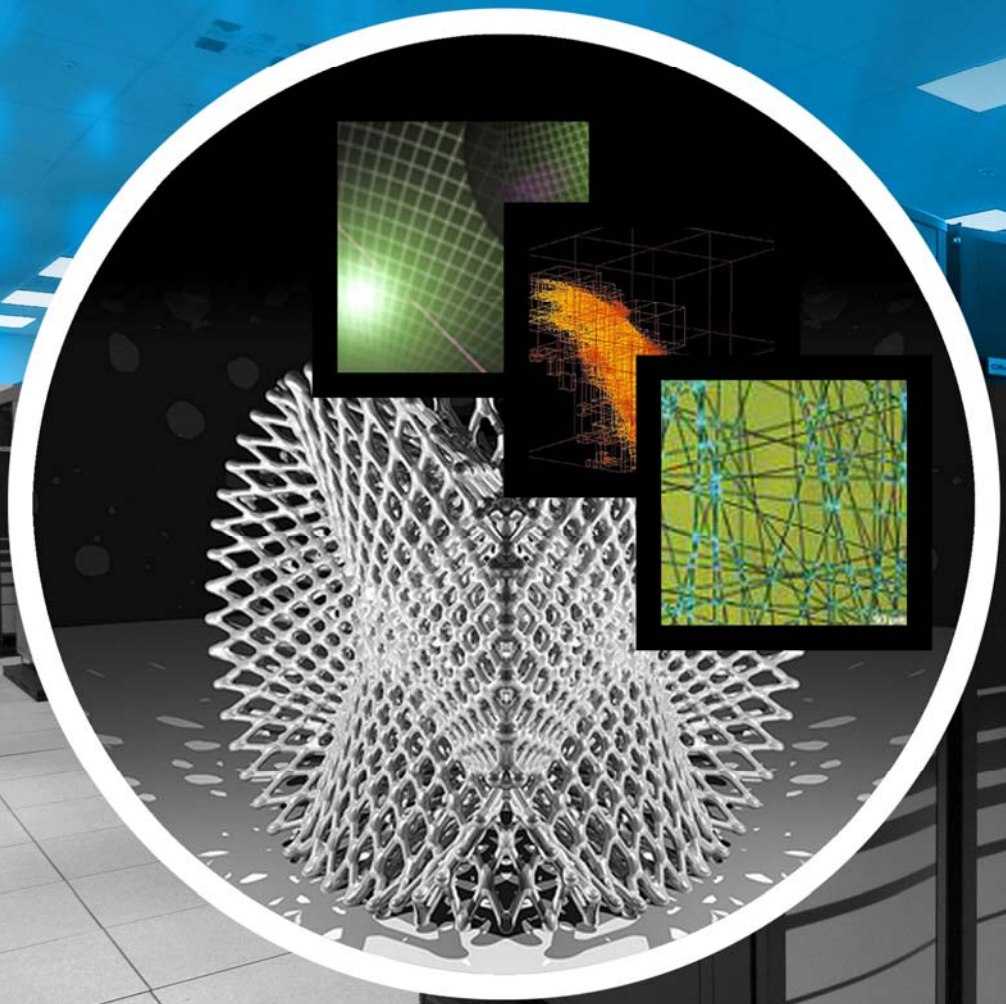
Scientific Grand Challenges

CROSSCUTTING TECHNOLOGIES FOR COMPUTING AT THE EXASCALE

February 2-4, 2010 • Washington, D.C.



U.S. DEPARTMENT OF ENERGY

Sponsored by:
Office of Advanced Scientific Computing Research, Office of Science
Office of Advanced Simulation and Computing, National Nuclear Security Administration

DISCLAIMER

This report was prepared as an account of a workshop sponsored by the U.S. Department of Energy. Neither the United States Government nor any agency thereof, nor any of their employees or officers, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of document authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof. Copyrights to portions of this report (including graphics) are reserved by original copyright holders or their assignees, and are used by the Government's license and by permission. Requests to use any images must be made to the provider identified in the image credits.

On the cover: Lawrence Berkeley National Laboratory's Cray XT4™ supercomputer. The computer is named Franklin in honor of Benjamin Franklin, one of the first American scientists. The Franklin is at the National Energy Research Scientific Computing Center, which is managed by Lawrence Berkeley National Laboratory for the U.S. Department of Energy's Office of Science. The Franklin's computing power makes it one of the largest and most powerful supercomputers in the world. Future reports in the Scientific Grand Challenges workshop series will feature different Office of Science computers on their covers.

SCIENTIFIC GRAND CHALLENGES: CROSSCUTTING TECHNOLOGIES FOR COMPUTING AT THE EXASCALE

Report from the Workshop Held February 2-4, 2010

Sponsored by the U.S. Department of Energy, Office of Advanced Scientific Computing Research, Office of Science; and the Office of Advanced Simulation and Computing, National Nuclear Security Administration

Chair, **David L. Brown**
Lawrence Livermore National Laboratory

Chair, **Paul Messina**
Argonne National Laboratory

Theme I: **Domain Science and System Architecture**

Principal Lead, **David Keyes**
King Abdullah University of Science and Technology and Columbia University

Co-Lead, **John Morrison**
Los Alamos National Laboratory

Co-Lead, **Robert Lucas**
University of Southern California

Co-Lead, **John Shalf**
Lawrence Berkeley National Laboratory

Theme II: **System Software**

Principal Lead, **Pete Beckman**
Argonne National Laboratory

Co-Lead, **Ron Brightwell**
Sandia National Laboratories

Co-Lead, **Al Geist**
Oak Ridge National Laboratory

Theme III: **Programming Models and Environment**

Principal Lead, **Jeffrey Vetter**
Oak Ridge National Laboratory and Georgia Institute of Technology

Co-Lead, **Bradford L. Chamberlain**
Cray, Inc.

Co-Lead, **Ewing Lusk**
Argonne National Laboratories

Applied Mathematics Topical Leads

Partial Differential Equations

Co-Lead, **John Bell**
Lawrence Berkeley National Laboratory

Co-Lead, **Mark S. Shephard**
Rensselaer Polytechnic Institute

Data/Visualization

Lead, **Mihai Anitescu**
Argonne National Laboratory

Uncertainty Quantification/Stochastic Systems

Lead, **Don Estep**
Colorado State University

Discrete Math

Co-Lead, **Bruce Hendrickson**
Sandia National Laboratories

Co-Lead, **Ali Pinar**
Sandia National Laboratories

Optimization and Solvers

Lead, **Michael A. Heroux**
Sandia National Laboratories

Representative, Office of Advanced Scientific Computing Research, **Barbara Helland**
Representative, Office of Advanced Scientific Computing Research, **Daniel Hitchcock**
Representative, Office of Advanced Scientific Computing Research, **Walter M. Polansky**
Representative, *Office of Science*, **Michael Strayer**
Representative, National Nuclear Security Administration, **Sander Lee**
Representative, National Nuclear Security Administration, **Thuc Hoang**

EXECUTIVE SUMMARY

The development of an exascale computing capability with machines capable of executing O(10^{18}) operations per second in the 2018 time frame will be characterized by significant and dramatic changes in computing hardware architecture from current (2010) petascale high-performance computers. From the perspective of computational science, this will be at least as disruptive as the transition from vector supercomputing to parallel supercomputing that occurred in the 1990s. Similar to that transition, the achievement of scientific application performance commensurate with the expected improvement in computing capability will require identifying and/or developing mathematical models and numerical algorithms that map efficiently onto exascale architectures. Significant reengineering of scientific application codes will also be required, supported by the corresponding development of new programming models and system software appropriate for these new architectures. To achieve these increases in capability by 2018, significant acceleration in both hardware and software development is required. This could be accomplished through an intensive "co-design" effort where system architects, application software designers, applied mathematicians, and computer scientists work interactively to characterize and produce an environment for computational science discovery that fully leverages these significant advances in computational capability.

The goal of the "Scientific Grand Challenges - Crosscutting Technologies for Computing at the Exascale" workshop[1] in February 2010, jointly sponsored by the U.S. Department of Energy's Office of Advanced Scientific Computing Research, Office of Science; and the Office of Advanced Simulation and Computing, National Nuclear Security Administration, was to identify the elements of a research and development (R&D) agenda that will address these challenges and create a comprehensive exascale computing environment. This exascale computing environment will enable the science applications identified in the eight previously held Scientific Grand Challenges Workshop Series.[2] Many of the new applications will tackle predictive simulation of complex, multiphysics problems, while new mathematical models, algorithms, and programming models must be developed in addition to the exascale computing environments. The R&D agenda must address system architecture, software, and tools as well as the models, algorithms, and application software to enable scientific discovery and problem solving at the exascale. To meet the goal of creating computing environments that will enable future applications by 2018, these R&D efforts must be initiated now.

EXASCALE SYSTEMS AND APPLICATIONS WILL DIFFER FROM PETASCALE COMPUTING

Over the past two decades, performance improvements in high-performance computing hardware have been achieved through an increase in both processor speed and parallelism. The programming model for this high-performance parallel computing environment has largely been bulk synchronous, where each programming element executes the same code with regular communication between processors. Energy constraints now dictate that processor clock rates (currently around 1-3 gigahertz [GHz]) cannot increase significantly with current materials and fabrication technologies, and may even decrease slightly in future

[1] "Scientific Grand Challenges - Crosscutting Technologies for Computing at the Exascale," February 2-4, 2010, Washington, D.C.
[2] http://www.er.doe.gov/ascr/Misc/GrandChallenges.html

leading-edge architectures. Significant increases in hardware concurrency, billion-way concurrency—a factor of 10,000 greater than on current platforms—may be required to achieve exaflop (10^{18} floating-point operations per second) computing. This increased concurrency is likely to occur predominantly on-chip, meaning that nodes with 1000- or even 10,000-way concurrency can be expected on an exascale platform. This development will require significant changes in algorithms, their implementation, and features of programming models for the new architectures.

As total concurrency and node concurrency increase dramatically, cost constraints dictate that neither system memory nor interconnect bandwidth can increase commensurately unless new technologies emerge. Thus, high-performance computers at the exascale will exhibit a substantially different balance among processor speed, interconnect bandwidth, and system memory. Those changes will require a significant redesign and reimplementation of current codes that are supported by new algorithms, programming models, and systems software. An indispensable role of the co-design approach will be to determine an acceptable balance between flops and memory for classes of high-end scientific and engineering applications. In some cases, co-design may identify new functionality that can be added to the properties of existing cores to facilitate transition of application codes onto exascale systems.

FINDINGS AND OBSERVATIONS

The R&D efforts identified in this workshop are crosscutting research topics fundamental to enabling many exascale applications. Advances in applied mathematics—in areas such as mathematical modeling, numerical analysis, mesh generation, and adaptive algorithms—will be essential enablers of exascale applications and provide crucial input to the co-design process. Often, research in these areas takes years to produce results, and the implementation of those results requires additional time. Thus, as noted, the R&D that leads to those advances can and must start now. Indeed, early results stemming from research on data movement-avoiding algorithms indicate such algorithms can be developed and will provide effective ways to deal with the foreseen change in architectural balance.

A substantial fraction of the exascale systems workload is expected to have workflows and computing requirements that differ from today's applications. For example, uncertainty quantification will include statistical techniques involving large ensemble calculations and other statistical analysis tools with significantly different dynamic resource allocation requirements than in the past. The design and scheduling of complex, distributed, interconnected national infrastructures (e.g., communication, transportation, energy, and water distribution) will require analysis, simulation, and optimization that entail enormous discrete algorithms and combinatorial problems. Graph algorithms are also increasingly central to cyber security and knowledge discovery in huge data sets.

An important observation is the building blocks of exascale systems will be similar to those of mass-market servers. As a result, challenges posed by exascale architectures will be faced by everyone over the next 10 years, exascale or not. This means that many of the R&D efforts undertaken for exascale systems will also benefit scientific and engineering applications that require less powerful systems.

The co-design approach will be a crucial tool for creating an effective and affordable exascale computing environment, and again, must be initiated now and continued throughout the development of exascale systems and beyond. If the co-design approach is to be included in systems that will be deployed in 2018, component and architectural features that could facilitate implementation of exascale applications must be identified soon.

SUMMARY OF PRIORITY RESEARCH DIRECTIONS IDENTIFIED BY THE WORKSHOP PANELS

Workshop panel members identified essential priority research directions (PRDs) for developing exascale computing environments, which are grouped below under three topic areas: 1) Algorithm and Model Research and Development Needed to Support New Architectures; 2) Research and Development for Programming Models to Support Exascale Computing; and 3) Research and Development for System Software at the Exascale.

Topic Area 1: *Algorithm and Model Research and Development Needed to Support New Architectures*

PRD 1.1: Recast critical applied mathematics algorithms to reflect impact of anticipated macro architecture evolution, such as memory and communication constraints.

PRD 1.2: Take advantage of exascale architectural evolution to design new algorithms for uncertainty quantification and discrete mathematics.

PRD 1.3: Develop new mathematical models and formulations that effectively exploit anticipated exascale hardware architectures.

PRD 1.4: Address numerical analysis questions associated with moving away from bulk-synchronous programs to multitask approaches.

PRD 1.5: Adapt data analysis algorithms to exascale environments.

PRD 1.6: Extract essential elements of critical science applications as "mini-applications" that hardware and system software designers can use to understand computational requirements.

PRD 1.7: Develop tools to simulate emerging architectures and performance modeling methods for use in co-design.

Topic Area 2: *Research and Development for Programming Models to Support Exascale Computing*

PRD 2.1: Investigate and develop new exascale programming paradigms to support "billion-way" concurrency.

PRD 2.2: Develop tools and runtime systems for dynamic resource management.

PRD 2.3: Develop programming models that support memory management on exascale architectures.

PRD 2.4: Develop new scalable approaches for input/output (I/O) on exascale architectures.

PRD 2.5: Provide interoperability tools to support the incremental transition of critical legacy science application codes to an exascale programming environment.

PRD 2.6: Develop language support for programming environments at the exascale.

PRD 2.7: Develop programming model support for latency management.

PRD 2.8: Develop programming model support for fault tolerance and resilience.

PRD 2.9: Develop integrated tools to support application performance and correctness.

PRD 2.10: Develop a new abstract machine model that exposes the performance-impacting design parameters of possible exascale architectures.

Topic Area 3: *Research and Development for System Software at the Exascale*

PRD 3.1: Develop new system software tools to support node-level parallelism.

PRD 3.2: Provide system support for dynamic resource allocation.

PRD 3.3: Develop new system software support for memory access (global address space, memory hierarchy, and reconfigurable local memory).

PRD 3.4: Develop performance and resource measurement and analysis tools for exascale.

PRD 3.5: Develop new system tools to support fault management and system resilience.

PRD 3.6: Develop capabilities to address the exascale I/O challenge.

TABLE OF CONTENTS

EXECUTIVE SUMMARY V
INTRODUCTION 1
Workshop Logistics 1
Crosscutting Findings 2
Accelerated Scientific Discovery Through Co-design 5
FUTURE ARCHITECTURE DEVELOPMENT 9
Implications of Nonincreasing Processor Speed 10
Total Memory Capacity is Limited by Fabrication Cost 12
Memory Bandwidth Constraints 14
Other Considerations 15
MATHEMATICAL MODELS AND ALGORITHMS 19
Introduction 19
General Considerations for Mathematical Models and Algorithms at the Exascale 19
Overview of Exascale Considerations for Mathematical Topical Areas 24
Crosscutting Challenges and Research Needs in Algorithm and Model Research and Development Needed to Support New Architectures 49
PROGRAMMING MODELS AND ENVIRONMENTS 51
Understanding Programming Model Requirements at the Exascale in the Context of Critical Mathematical Models, Methods, and Algorithms 52
Crosscutting Challenges and Research Needs for Programming Models to Support Exascale Computing 61
SYSTEM SOFTWARE 65
Introduction 65
Understanding Mathematical Models, Methods, and Algorithms in the Context of System Software 65
Crosscutting Challenges and Research Needs for System Software at the Exascale 71
CONCLUSIONS AND RECOMMENDATIONS 75
Topic Area 1: Algorithm and Model Research and Development Needed to Support New Architectures 75
Topic Area 2: Research and Development for Programming Models to Support Exascale Computing 76
Topic Area 3: Research and Development for System Software at the Exascale 76

REFERENCES 77
APPENDIX 1: THE INTERNATIONAL EXASCALE SOFTWARE PROJECT ROADMAP ..... APPENDIX 1-1
APPENDIX 2: WORKSHOP PROGRAM/AGENDA ..... APPENDIX 2-1
APPENDIX 3: WORKSHOP PARTICIPANTS ..... APPENDIX 3-1
APPENDIX 4: ACRONYMS AND ABBREVIATIONS ..... APPENDIX 4-1

INTRODUCTION

A technical workshop, "Scientific Grand Challenges – Crosscutting Technologies for Computing at the Exascale" was held February 2-4, 2010, in Rockville, Maryland. The U.S. Department of Energy (DOE) Office of Advanced Scientific Computing Research (ASCR), Office of Science; and the Office of Advanced Simulation and Computing, National Nuclear Security Administration (NNSA) co-sponsored this collaborative workshop. This workshop report is one of a series resulting from the Scientific Grand Challenges Workshops hosted by DOE ASCR in partnership with other DOE Office of Science programs. The workshop series focuses on the grand challenges of specific scientific domains and the role of extreme-scale computing in addressing those challenges. Dr. Paul Messina, interim director of science at the Argonne Leadership Computing Facility, is overseeing the workshop series.

The workshop's purpose was to provide a forum for discussing the creation of a comprehensive exascale computing environment by 2018 that will enable the science applications identified in eight previous Scientific Grand Challenges Workshops (held in 2008 and 2009) and, in particular, to address the following:

- Determine the research and development (R&D) required in mathematical models, applied mathematics, and computer science to meet the needs of a broad spectrum of applications at the exascale

- Provide recommendations on exascale architectures and configurations that would effectively support the applications identified in the Grand Challenge Science workshops

- Provide recommendations on how to structure the co-design process where system architects, application software designers, applied mathematicians, and computer scientists work closely together to define and produce a computational science discovery environment, including the role of test beds

- Outline the R&D needed for co-design of system architecture, system software and tools, programming frameworks, mathematical models and algorithms, and scientific application codes at the exascale.

WORKSHOP LOGISTICS

A total of 148 participants registered for and attended this workshop. Workshop participants included scientists representing multiple areas of investigation in computer science and applied mathematics, as well as scientists who use computational approaches to conduct research in applications of interest to DOE. Participants represented 29 U.S. universities, 8 corporations, 9 DOE national laboratories, and 3 federal agencies. Several DOE Headquarters program managers were also present as observers throughout the duration of the workshop.

These workshop participants provided the multidisciplinary expertise required to identify and address crosscutting challenges in high-performance computing (HPC), with an emphasis on using extreme-scale computing for scientific research, advances, and discoveries.

To establish a context for breakout session discussions, the crosscutting technologies workshop opened with a series of plenary presentations. These presentations addressed mathematical models and algorithms, computer architecture, system software, programming models and environments, and co-design strategies. The workshop agenda and the plenary presentations are on the DOE Scientific Grand Challenges Workshop website at http://extremecomputing.labworks.org/crosscut/.

In the breakout sessions, technical discussions focused on five main theme areas:

- Future Architecture and Software Development
- Mathematical Models and Algorithms
- System Software and Tools
- Programming Models and Environments
- Co-design.

Workshop participants rotated through these discussions while relating them to five mathematical models and algorithms topics:

- Partial Differential Equations (PDEs)
- Data Analysis and Visualization
- Uncertainty Quantification (UQ) and Stochastic Systems
- Discrete Mathematics
- Optimization and Solvers.

CROSSCUTTING FINDINGS

The development of an exascale computing capability with machines capable of executing $O(10^{18})$ operations per second in the 2018 time frame will be characterized by significant and dramatic changes in computing hardware architecture from current (2010) petascale high-performance computers. From a computational science perspective, this will be at least as disruptive as the transition from vector supercomputing to parallel supercomputing that occurred in the 1990s. Similar to that transition, the achievement of scientific application performance commensurate with the expected improvement in computing capability will require identifying and/or developing mathematical models and numerical algorithms that map efficiently onto exascale architectures and significant reengineering of scientific application codes supported by the corresponding development of new programming models and system software appropriate for these new architectures. To achieve these increases in capability by 2018, significant acceleration in both hardware and software development is required. This could be accomplished through an intensive co-design effort, where system architects, application software designers, applied mathematicians, and computer scientists work interactively to characterize and produce

an environment for computational science discovery that fully leverages these significant advances in computational capability.

Over the past two decades, performance improvements in HPC hardware have been achieved through an increase in both processor speed and parallelism. The programming model for this high-performance parallel computing environment has largely been bulk synchronous (supported by the message passing interface [MPI] programming model), where each programming element executes the same code with regular communication between processors. Energy constraints now dictate that processor clock rates (currently around 1-3 GHz) cannot increase significantly with current materials and fabrication technologies and, in fact, may decrease slightly in future leading-edge architectures, meaning that exascale performance instead will be reached through significant increases in hardware concurrency. Allowing for latency hiding, billion-way concurrency—a factor of 10,000 greater than on current platforms—may be required to achieve exaflop (10^{18} floating-point operations per second) computing. This increased concurrency is likely to occur predominantly on-chip, meaning that nodes with 1000- or even 10,000-way concurrency can be expected on an exascale platform. This development will require major changes in the implementation of the MPI-based HPC programming model; development of alternative programming models; or most likely, a hybrid-programming model that combines MPI with other models. Those other models might be OpenMP or OpenCL, but other possibilities must be considered as well. The introduction of massive parallelism at the node level will be a significant new challenge to applications programmers. While MPI may continue to be the internode programming model, the intranode programming model likely will be quite different because of the need to manage very large-scale threaded parallelism within relatively small memory capacity per node.

It is important to note that significant increase in on-chip hardware concurrency will be a challenge not only for the high-end computing environment, but will also be faced by the commodity market as multicore chips become increasingly common in consumer products such as laptops and handheld devices. Thus, computational science applications targeted at laptop and desktop computers will face many of the same issues as HPC applications and will also need to be redesigned and re-engineered over the next decade.

As total concurrency and node concurrency increase dramatically, cost constraints dictate that neither system memory nor interconnect bandwidth can increase correspondingly—unless new technologies emerge. Thus, high-performance computers at the exascale will exhibit a substantially different balance among processor speed, interconnect bandwidth, and system memory. Current high-performance scientific application codes will not easily "port" to this dramatically different environment. As such, a significant redesign and reimplementation of those codes, supported by new algorithms, programming models, and systems software, will be required. A key role of the co-design approach will be to determine an acceptable balance between flops and memory for classes of high-end scientific and engineering applications. In some cases, co-design may identify new functionality that can be added to the properties of existing cores and facilitate the transition of application codes onto exascale systems. Today's petaflops systems incorporate several such advances that were inspired by scientific application needs.

Because of this architectural balance where a flop is by far cheaper to provision and to power up relative to a byte of memory or an improvement in memory transfer rate, the metric for the quality of a computation must decisively move to one that favors achievement of the requisite accuracy with the requisite confidence at some combination of lowest energy and shortest execution time. In particular, the

percentage of peak flop/s achieved is therefore a useless and counter-productive metric because it penalizes for over-provisioning something cheap. Good designs will over-provision flops, so they are never the limiting factor in the course of a computation, relative to the cost of storing and communicating their operands. Percentage of the instantaneously limiting resource, typically memory bandwidth, is the most interesting figure of merit for a given execution of the "right problem." It is a grand challenge of numerical analysis to equidistribute the multitudinous errors in a computation to achieve the target result without "oversolving" or "overresolving" some contributing step. Because the balance of resources needed to accommodate a computation within a given memory and energy budget is increasingly delicate, research in the propagation of numerical errors throughout an end-to-end computation, always timely in numerical analysis at any scale, takes on a greater importance at the exascale.

While for some science fields this redesign and reimplementation of scientific application codes will be quite disruptive, it does present a significant opportunity to reconsider how computational science might be performed at the exascale. Driven in many cases by improvements in the quality and number of physical models employed rather than increased computational resolution, the expectation is that exascale science applications will be characterized by increases in both functionality and physical fidelity. Exascale computing also will create an opportunity to develop codes that can provide increased understanding through predictive science, establishing clear levels of confidence in the calculations by directly incorporating UQ techniques. In addition, simulations increasingly will be used not only for discovery science, but also as a basis for informing policy decisions through, for example, risk analysis or the determination of optimal results, risk analysis, and designs.

During the course of the workshop, a number of important themes emerged from the breakout group discussions:

1. **Significant new model development, algorithm redesign, and science application code reimplementation supported by (an) exascale-appropriate programming model(s) will be required to exploit effectively the power of exascale architectures**. The disruptive transition from current sub-petascale and petascale computing to exascale computing will be at least as disruptive as the transition from vector to parallel computing in the 1990s and will require an intensive co-design effort where system architects, application software designers, applied mathematicians, and computer scientists collaborate to produce a computational science discovery environment that leverages significant advances in computational capability available at the exascale.

2. **Uncertainty quantification will permeate the exascale science workload**. The demand for predictive science results will drive the development of improved approaches for establishing levels of confidence in computational predictions. Both statistical techniques involving large ensemble calculations and other statistical analysis tools will have significantly different dynamic resource allocation requirements than in the past, and the major code redesign required for the exascale will present an opportunity to embed UQ techniques in exascale science applications.

3. **Increasing imbalance among processor speed, interconnect bandwidth, and system memory will drive the development of more "holistic" science applications**. As execution of operations on the cores and provisioning of local storage progressively become cheaper than interprocessor communication or input/output (I/O), exascale simulation codes increasingly will see analysis capabilities, UQ calculations, and performance measurements and predictions embedded in the

application codes themselves. To adapt codes to the new architectural balances, new data movement-avoiding algorithms also need to be developed.

4. **Memory management will be a significant challenge for exascale science applications due to deeper, complex memory hierarchies and relatively smaller capacities. Dynamic, latency-tolerant approaches are a promising way to deal with those issues**. Tools and interfaces must be developed to manage and control memory for runtime systems, as well as to provide information to assist the compiler in optimizing memory management. Science applications must learn to exploit the deeper, more complex memory hierarchy expected at the exascale.

5. **Science applications will need to manage resilience issues more actively at the exascale**. The mean time between failures will decrease with increased concurrency, while the time required for the output of a conventional checkpoint file will increase if current projections of storage systems technologies are accurate. Thus, a co-design effort will be required to provide fault recovery tools that can be tailored to specific application needs. Memory technologies and system architectures also may play a role, incorporating large capacity, nonvolatile random-access memory (RAM) in exascale architectures that may provide both new challenges and opportunities for managing resiliency.

6. **Automated, dynamic control of system resources will be required**. Adaptive runtime systems and programming model support for dynamic control will be important to address dynamic load balancing issues, the potential need for dynamic power allocation among system components, and the ability to reconfigure around faults.

7. **System support issues will be even more challenging at the exascale**. File system scalability and robustness will continue to be the weakest link at the exascale. Developing a hierarchical debugging tool capable of dealing with 10,000 nodes will be a significant research challenge. Performance tools that address the expected heterogeneous architectures, possibly vertically integrated within science applications, must also be developed.

ACCELERATED SCIENTIFIC DISCOVERY THROUGH CO-DESIGN

Extreme-scale computing has the potential to usher DOE disciplines and missions into an era of greatly enhanced discovery, prediction, and design capability. The scientific community's ability to predict outcomes with far greater fidelity and quantified confidence measures will be crucial metrics of success. This exciting goal can only be achieved by a deliberate strategy of advancing key enabling technologies in theory and modeling, experiments and observational tools, and simulation science. These advancements need to be guided by computing system architectures likely to provide the needed capability as well as the needs of the scientific simulations and models. To the extent possible, the computer architectures and configurations fielded must be guided by the needs of the scientific and engineering projects that require extreme-scale computing. These intertwined R&D activities are the essence of what is known as the "co-design methodology." The issues tackled in co-design of extreme-scale systems and applications are too complex to be resolved by thought experiments alone. The new architectures and computational science approaches that result from the research should be tested on existing systems and, where possible, on software and hardware simulators that implement the new architectures and simulations. At the extreme scale, such simulators must be developed as soon as possible so new approaches can be tested before huge investments in hardware and software are made. By involving a broad set of applications in the design effort, the resulting computer systems will be

general purpose. Special purpose extreme-scale systems will probably be developed and their design will be guided by the co-design approach; however, these systems were not in the purview of this workshop, which focused on the development of general-purpose exascale computer systems.

Co-design is not a new concept. Historically, most new computer systems have included features and capabilities that arose from application needs and component technology advancements. What differs for extreme-scale computing is that co-design will need to encompass almost every hardware and software component, as opposed to only the processor instruction set or the memory hierarchy as was typically the case in the past.

In addition to co-design efforts for developing exascale simulation capabilities, the goal of greatly enhanced discovery, prediction, and design capability also depends critically on similarly deliberate strategies for co-design of other national assets, such as accelerators, data collections generated by experiments, and collaborative infrastructures to use them. This effort will accelerate feedback and time-to-result, but, more importantly, it will increase the scale of challenges to which the scientific method can aspire to tackle.

Algorithms and codes must be agile and more modular for the single-physics and integrated packages that will support the science, technology, and engineering problems identified in the eight Scientific Grand Challenge workshops (http://extremecomputing.labworks.org/) that focused on science and engineering applications that would benefit from extreme-scale computing environments. Next-generation architectures must be able to handle these codes. Vice versa, the algorithms and codes must take advantage of the architectures that new technologies can permit. Early joint planning is essential for the following:

- Designing computer architectures and system configurations that will be both affordable and an appropriate match for current and future high-end science applications with reasonable implementation effort

- Devising mathematical models, numerical algorithms and software, programming models, and system software that enable implementation of complex simulations and achieve good performance on the new architectures.

Panel members at the crosscutting technologies workshop discussed many areas in which co-design will be needed to achieve the following results:

- Guide the evolution of hardware architectures for Scientific Grand Challenges

- Characterize balanced hardware/software configurations in the exascale regime

- Identify and pursue R&D topics in applied mathematics and computer science that must be addressed to pursue future complex simulations on exascale systems.

The co-design methodology is iterative, requiring frequent interactions among hardware architects, systems software experts, programming model designers, and designers and implementers of the science applications that provide the rationale for building extreme-scale systems. As new ideas and approaches are identified and pursued, some will fail. Per past experience, there may be breakthroughs in hardware

technologies that result in different micro and macro architectures becoming feasible and desirable, but their use will require rethinking of certain algorithmic and system software implementations.

Although application developers' preferences and algorithmic requirements may not be able to affect the creation of desirable device technology properties and costs, configuration flexibility can at least provide the option of fielding configurations that come closer to meeting application needs. For example, a facility that is configuring a new system may opt to trade maximum peak flops for larger memory per node or pay for faster memory access and interconnection network instead of buying more memory per node. More optimistically, by identifying features that would facilitate programming applications on extreme-scale systems and increase efficiency of their execution, system architects and component developers might be inspired to invent technologies that create those features affordably.

As simulation capability and fidelity has improved, the interplay among simulation, observation, and experiment has become increasingly important for scientific discovery. Thus, there is an even greater generational opportunity to align this co-design with planning for the next generation of experimental/data facilities so the data being acquired have maximum relevance to the theory- and data-driven models that extreme computing is simulating. Conversely, theory, modeling, and extreme computing must have the greatest synergy with what can be measured because of new experimental and engineering technologies.

The opportunity to move beyond better observation, measurement, and simulation capabilities to a true co-design strategy for accelerated discovery, prediction, and control is an exciting Grand Challenge. Through co-design, the evolution of each capability will be guided by that of the others. As such, observation, measurement, and simulation will be well matched to tackle a cooperative study of complex phenomena. If scientists are to meet the great scientific, national security, and societal challenges of our time—health, energy, defense, and information management—achieving this qualitatively higher level of interdisciplinary cooperation and integration of major assets is essential.

FUTURE ARCHITECTURE DEVELOPMENT

Understanding the parameters of computer architectures at the exascale will be essential for the designers of application software, systems software, and programming models for these machines. The *Scientific Grand Challenges: Architectures and Technology for Extreme Scale Computing, December 8-10, 2009, San Diego, CA* (DOE 2011) report discussed possible designs for exascale computers that could be deployed in the 2018 time frame. The constraints on the possible high-performance computers were a desired performance of 1 exaflop (10^{18} floating-point operations per second), an energy budget of $20 million per year, hardware cost of $200 million, and total cost over 10 years, not including facility acquisition and operation costs of approximately $200 million. Assuming power availability of about $1 million per megawatt (MW) per year over a 10-year lifecycle, this translates to an annual power budget of 20 MW. A clear conclusion from the Architectures and Technology for Extreme Scale Computing workshop (DOE 2011) was an exascale computer under these financial and energy constraints will look significantly different from current petascale computers. Energy constraints dictate that processor clock rates cannot increase significantly with current materials and fabrication technologies, and in fact, may decrease. Therefore, exascale performance will be achieved through substantial increases in hardware concurrency, much of it on-chip. Communication speeds are also power limited. As such, latency and bandwidth issues will continue to be significant, further driving concurrency requirements. Memory costs are not decreasing as fast as central processing unit (CPU) costs, so total memory also will be cost limited. These constraints will drive significant changes in the balance of high-performance computers at the exascale, resulting in daunting challenges to application programmers who must develop simulation codes that can use these machines effectively.

In the past, processor speeds have increased along with system size, and times-to-solution have remained more or less constant, even as the problem sizes have grown to fit the new machines. However, under the exascale scenarios, larger problems that require more sequential operations (e.g., time steps or iterations) will now require a correspondingly larger time-to-solution. Regarding the memory constraints, for important scientific applications that are extremely memory-intensive (such as data analysis and bio-informatics problems), it is not the increase in flop count but the increase in memory size that determines how much larger problems can be tackled. For these applications, a sublinear scaling of memory means a sublinear increase in capability, and "exascale" will not translate into a one thousand-fold increase in capability. The constraints on communication bandwidths imply that total computational cost will be measured in terms of efficiency of communication rather than flop count. These three points are discussed in more detail in the following sections.

The significant increase in on-chip hardware concurrency will be a challenge not only for the high-end computing environment, but also the commodity market as multicore chips become increasingly common in consumer products, such as laptops and handheld devices. As in the past, while these commodity chips will be used both in the high-end computing and consumer markets, there will be little transfer of software technology from the commercial marketplace to scientific and engineering communities. Thus, computational science applications targeted at laptop and desktop computers will face many of the same issues as high-performance computing (HPC) applications and will need to be redesigned and re-engineered over the next decade. Lessons learned from developing math models and algorithms for next-generation laptops will be directly applicable at the exascale. Overall success at the exascale will require a constructive co-design process, which will benefit from all parties thoroughly understanding the considerations involved in developing the entire exascale simulation environment, including the

hardware, software stack, mathematical models and algorithms, and science and engineering application codes.

IMPLICATIONS OF NONINCREASING PROCESSOR SPEED

Climate, combustion, nuclear energy, national security, and other applications critical to decision support and technology advances that require exascale simulation and data analysis depend algorithmically upon a common core of mathematical formulations. These are multiscale-, multiphysics-based problems in three spatial dimensions and time with computational work requirements that scale roughly as the four-thirds power of the memory capacity. This is a scaling that fundamentally departs from the classical Amdahl-Case rule of computer architecture (Bell 1996), which would provide for linear scaling of bytes and byte/s with flop/s. The argument for the four-thirds power law scaling is simple: these simulations are for three space dimensions where the time steps that can be taken, either for stability or accuracy reasons, scale with the smallest one-dimensional mesh spacing. In an application area such as climate, a more powerful supercomputer makes it possible to perform more highly resolved simulations (with a correspondingly smaller mesh size), but requires correspondingly more time iterations to reach the same final physical time. For example, because of this time step constraint, a computation with 5 times the resolution will require 125 times the memory and 5 times as many time iterations to get an improved answer. For a scientist to get the better answer in approximately the same time, the time to execute each time step must decrease by a factor of five, which has conventionally translated to a need for an approximately five-fold increase in processor speed.

Until just a few years ago, as parallel computers became larger, the required processor speed increase with memory size occurred sufficiently so that simulation wall-clock times have remained more or less the same for the "same" problem run at higher resolution. For example, the climate community has relied on this behavior to get increasing resolution and more "physics" into simulations that continue to run with a turnaround time of a day or two on the largest hardware available. However, in the last few years, processor speeds have "stalled out" due to energy constraints. The energy required to power a processor scales as the cube of the frequency and, hence, the operation speed of the processor.

The current largest supercomputers operate at around 2 petaflops, or $2x10^{15}$ floating-point multiply-adds (FMA) per second. (The system parameters for the "Jaguar," a Cray XT5 supercomputer located at Oak Ridge Leadership Computing Facility, are shown in Table 1 [ORNL 2009]). Current chips require about 10^{-10} joules of energy for a single FMA (Shalf et al. 2011). Thus, the power required to run just the processors on the machine is roughly 20 kilowatts (kW), a minor fraction of the total power (approximately 7 MW for the Jaguar Cray XT5) consumed by these computers (Bland et al. 2009).

To scale correctly for these multiphysics calculations, an exaflop computer (500 times as fast as the 2-petaflop machine) would require 105 times as many processors, operating at 4.73 times the frequency of current processors. Because of the cubic scaling law, the total power required to operate these processors would be about 11,000 times that of current machines, or 220 MW, an untenable number that does not even include energy demands of other system components. Thus, absent fundamentally new developments in processor technology, processor speed must remain constant or decrease for future systems.

Table 1. System parameters for the Jaguar supercomputer at Oak Ridge National Laboratory

Jaguar (2009) System Parameters	
System peak	2 petaflops
Power	6 MW
System memory	0.3 PB
Node performance	125 GF
Interconnect latency	1-5 μsec
Memory latency	150-250 clocks (~70-100 ns)
Node memory BW	25 GB/s
Node concurrency	12
Total node interconnect BW	3.5 GB/s
System size (nodes)	18,700
Total concurrency	225,000
Storage	15 PB
I/O	0.2 TB

MW = megawatt; PB = petabyte; GF = gigaflop; μsec = microsecond; ns = nanosecond; GB/s = gigabytes per second; TB = terabyte

Instead, an exaflop can be reached by increasing the total processor count. In principle, this would require an additional factor of 4.73 in processor count. However, because of latency considerations (discussed in this panel report in later sections), such a machine also will need a redundancy of about 100 to compensate for the time it takes to move a computed floating-point number to or from memory. Possible exascale scenarios discussed by panel members at the Architectures and Technology for Extreme Scale Computing workshop (DOE 2011) suggested machines with a processor count on the order of 1B, or 4400 times that of the 2-petaflop machine. In the 2018 time frame, it also is expected the energy consumption per FMA of a 1 to 2 gigahertz (GHz) processor will improve by a factor of about 10 (Shalf et al. 2011). As such, an exaflop machine would require only 440 times the energy of the petaflop machine, or about 8.8 MW for the CPUs, which is safely within the energy parameters assumed for this machine. This estimate can be further improved by decreasing the processor speed. Because of the cubic behavior of the energy, using chips running at half speed would gain another one-quarter factor (double the number of processors running at one-eighth of the frequency) to arrive at the same theoretical floating-point performance.

While the result is a machine that, in principle, has enough memory and raw flops to scale up a simulation correctly, trading sequential flops for parallel flops results in a machine that does not match time-stepping algorithms that have been designed and perfected over the last half century. For partial differential

equations (PDE)-based problems where the object of scaling up is to provide increased resolution, parallelization of the temporal dimension seems necessary. This is an important research topic in its own right and at many scales smaller than the exascale. However, opportunities for parallelization in time are limited at the exascale by memory considerations. Current time parallelization approaches typically require simultaneous storage of multiple copies of the spatial discretization at successive temporal stages. This presents a significant challenge to the applied mathematics research community—if increased spatial resolution is a design requirement for future simulations, research will be required to develop new approaches for time parallelization.

Another alternative for addressing this challenge is to change the problem. Considering again the climate simulations example, an exascale machine using current time-stepping algorithms will not be able to compute the "same" problem at increased spatial resolution in the same wall-clock time. As illustrated in the example, wall-clock times could be expected to increase by a factor of five. Thus, a one- to one-and-a-half-day turnaround becomes 1 week at the exascale. Another alternative is to consider using the increased flop count for purposes other than increased spatial resolution. Adding additional "physics" to a problem at the same spatial resolution translates into more equations and more flops required, but not necessarily more time steps. Another alternative would be to use the increased capacity to run many realizations of a problem in parallel, as might be done in a brute-force uncertainty quantification (UQ) study. Again, if the spatial resolution of each problem stays the same, the total wall-clock time also can remain the same for the overall computation.

TOTAL MEMORY CAPACITY IS LIMITED BY FABRICATION COST

A system's dynamic random access memory (DRAM) capacity is limited primarily by cost, which is defined by the dynamics of a broad-based, high-volume commodity market. The commodity market for memory makes pricing of the components highly volatile, but the centroid of the market is approximately $1.80/chip. Figure 1 illustrates that the rate of memory density improvement has gone from a 4X improvement every 3 years to a 2X improvement every 3 years (a 30% annual rate of improvement). Consequently, the cost of memory technology is not improving as rapidly as the cost of floating-point capability. Given the new rate of technology improvement, 8-gigabit memory parts will be widely available in the commodity market in the 2018 time frame, and 16 gigabit parts also will have been introduced. However, it is unclear which density will be the most cost effective in that time frame.

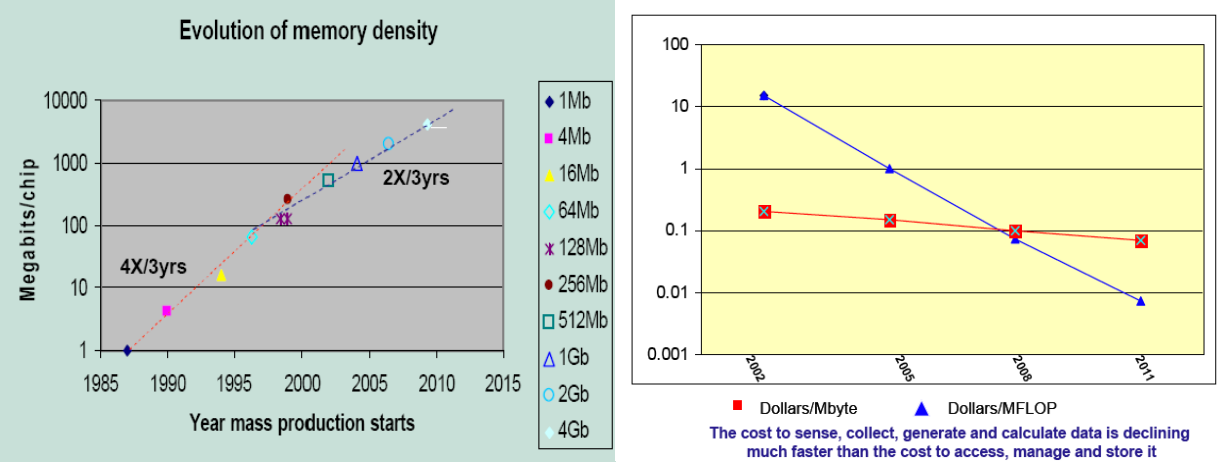


Figure 1. The rate of improvement in memory technology is improving at slower rates and now is approaching 30% per year (Shalf et al. 2011). Image courtesy of International Business Machines Corporation, © International Business Machines Corporation.

If the memory cost should not exceed 50% of the assumed $200 million cost of the entire computer system, then Figure 2 shows the approximate memory capacity that can be afforded assuming either 8-gigabit chips or 16-gigabit chips are somewhere between 50 and 100 petabytes.

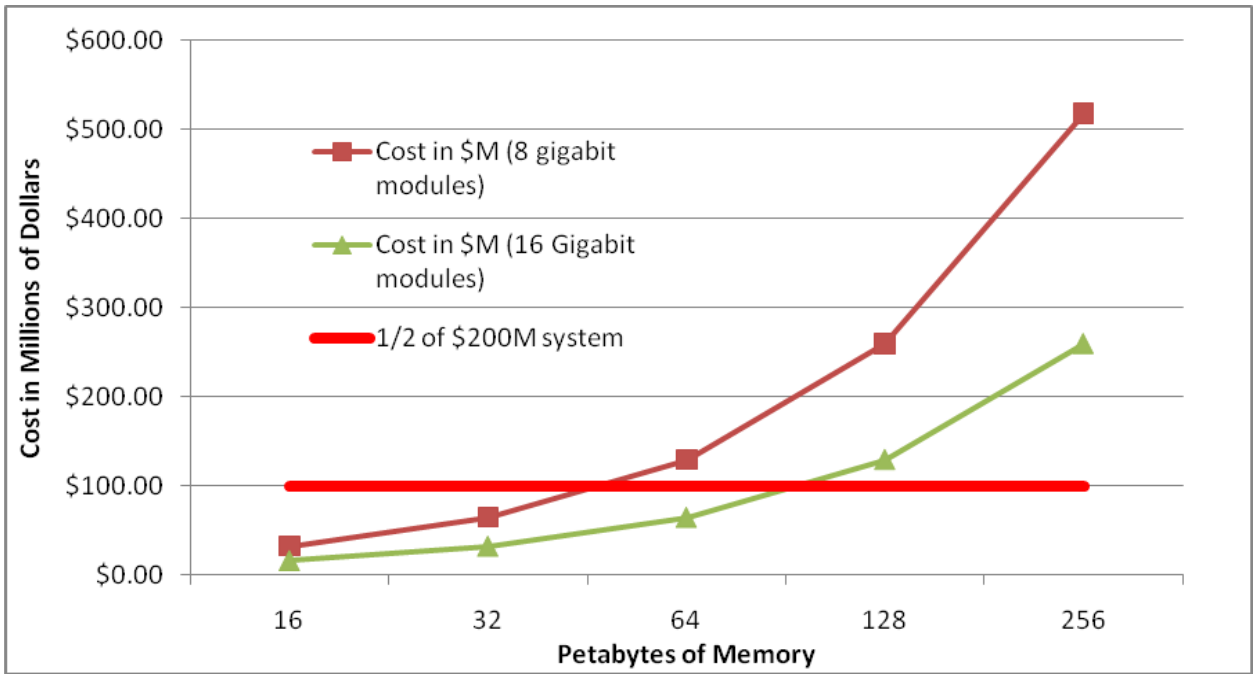


Figure 2. Two different potential memory chip densities are possible in the 2018 time frame. It is less certain which option will be at the apex of the commodity cost scaling (Shalf et al. 2011). Image courtesy of John Shalf (Lawrence Berkeley National Laboratory).

Memory capacity is limited not only by fabrication and purchase cost; memory is also a large part of operational cost, until **the scientific community** move**s** to a disruptive technology, because it consumes a great deal of power. Because memory and **input/output** (I/O) bandwidth is primarily power limited, it is

not possible, within the specified cost constraints, to obtain a memory capacity that is at parity with the conceived exascale system (parity defined by today's standards). Should **all** of the FLOPs in the system be given up to get better memory or I/O bandwidth balance, the memory bandwidth could be increased by at most 20% for the same cost.

The four-thirds scaling law for large-scale simulations suggests that a factor of 105 increase in memory capacity should be sufficient to scale from 2 petaflops to 1 exaflop. Fifty petabytes of DRAM is more than 150 times that on the Jaguar Cray XT5—well within that specification. Measured in terms of memory per node, this looks like a step in the right direction. With an anticipated 1 million nodes at the exascale, this corresponds to 50 gigabytes(GB) DRAM/node as compared to 16 GB for the Jaguar Cray XT5. However, because total flop count is achieved by massively increasing the number of processing elements in lieu of increasing processor speed, this number looks unbalanced from the perspective of memory per core. The Jaguar Cray XT5 has a total concurrency of 225,000, while an exascale machine is likely to have nearly 1 billion cores. Therefore, while the Jaguar Cray XT5 has about 1.3 gigabytes per core, a machine used in 2018 will have only 50 megabytes per core. This provides a significant indication that the current message passing interface (MPI)-based bulk synchronization approach for programming supercomputers must be replaced at the exascale and something akin to massive multithreading must be used to take advantage of the on-chip processing power. Nonetheless, message passing will be necessary between nodes, so the programming model must provide support for this capability as well. The development of practical programming models and corresponding software tools to support them will be essential for success at the exascale. Clearly, this is an important co-design topic.

MEMORY BANDWIDTH CONSTRAINTS

The power consumed by data movement will dominate the power budget of future systems. The power consumed in moving data between memory and processor is of particular concern. Historically, a bandwidth/flop ratio of around 1 byte/flop has been considered a reasonable balance; i.e., the machine can load about one double precision operand for every 8 floating-point operations. With current double data rate three (DDR-3) memory technology, the energy cost to load a double-precision operand to memory is about $5x10^{-9}$ joules (Shalf et al. 2011). For a current computer operating at 2 petaflops and accounting for the 8 bytes required to store a double-precision operand, the power required to maintain a 1 byte/flop ratio is about 1.25 MW. Extrapolating the JEDEC roadmap to 2018 when DDR-5 technology can be expected, a double-precision load should improve slightly to about $2.1x10^{-9}$ joules. However, with 500 times as many operations per second to balance with memory loads, the total power consumption of the memory system would jump to 260 MW, well above the posited parameters for an exascale system. Even reducing the byte/flop ratio to 0.2—considered by some experts to be the minimum acceptable value for large-scale modeling and simulation problems—power consumption of the memory subsystem still would exceed 50 MW. The options remaining are to accept a less healthy byte/flop ratio or to make significant investments to bring improved memory technology to market. The Architectures and Technology for Extreme Scale Computing report (DOE 2011) indicates that with a significant financial investment in more efficient memory interface protocols, advanced memory technology could be deployed with double-precision load costs between $.29x10^{-9}$ and $.50x10^{-9}$ joules, which would bring the byte/flop ratio closer to 0.2. Alternatively, sticking with the current JEDEC roadmap, total memory cost could be maintained around 10 MW by accepting a byte/flop ratio of 0.04. In either case, it is clear that a significant research investment would be required to develop algorithms that increase the amount of computation between memory accesses. This has not been a priority for algorithm development in the

past. In fact, "modern" programming practices often take advantage of considerable indirection (hence increased memory accesses) to improve code readability. Again, significant interaction between the algorithm development, programming models, and system software communities will be required to address these issues successfully.

OTHER CONSIDERATIONS

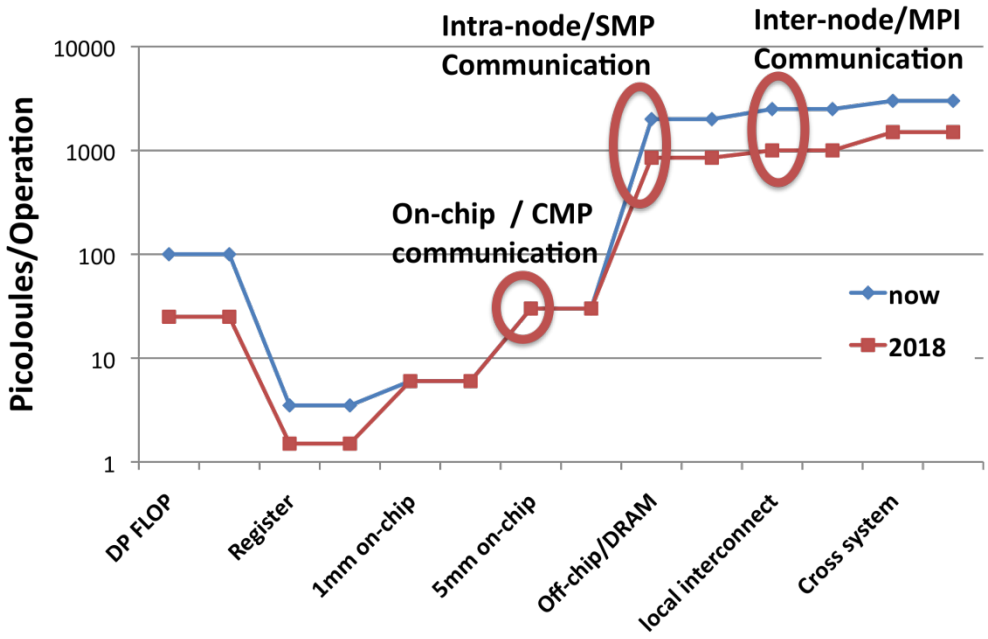


Figure 3. Energy cost of data movement relative to the cost of a flop for current and 2018 systems. The biggest delta in energy cost is movement of data off-chip. Therefore, future programming environments must support the ability of algorithms and applications to exploit locality, which in turn, will be necessary to achieve performance and energy efficiency (Shalf et al. 2011). Image courtesy of John Shalf (Lawrence Berkeley National Laboratory).

There are many other issues that present themselves in the prospective exascale architectures, which are addressed in more detail in articles such as "Exascale Computing Technology Challenges" (Shalf et al. 2011). A few of these issues are described in the following subsections.

Memory Locality Management

Because the cost of flop may improve by 5x to 10x while the cost of data movement will stay the same, the penalty for using distant memory will continue to increase. Thus, future algorithms will need to be attentive to both on-chip and off-chip memory locality management. Caches are the most convenient to program, but they virtualize the notion of on-chip versus off-chip memory, which complicates vertical locality management as users must reverse-engineer the behavior of the cache hierarchy. Likely, cache-coherence strategies can scale to dozens of processing elements, but the cost and latency of data movement on-chip would make cache coherence an inefficient method for inter-processor communication for future chip designs. In all likelihood, cache coherence could be used effectively in clusters or subdomains of the chip, but it is unlikely to be an effective approach if extended across a chip containing thousands of cores. Instead, it is possible that global memory addressing without cache coherence will be

supported with synchronization primitives to manage memory consistency explicitly. Lightweight explicit data movement protocols (such as global address space) may improve the ability to control horizontal data locality, but it currently is unclear whether partitioned global address space (PGAS) programming models improve horizontal locality management in practice.

Latency and Latency Hiding

Off-chip and cross-system latencies are unlikely to improve substantially over existing systems. Because the total node interconnect bandwidth will continue to increase, the bandwidth-latency product and, by Little's Law (Little 1961), the total number of outstanding memory requests at any time will increase correspondingly. This means that considerable attention must be focused on latency hiding both in algorithms and hardware designs.

Accelerators and Heterogeneous Multicore Processors

Accelerators and heterogeneous processing offer some opportunity to increase computational performance within a fixed power budget while still retaining conventional processors to manage more general-purpose components of the computation, such as operating system services. Currently, such accelerators have disjoint memory spaces, but it is expected that by the 2015 time frame it will be feasible from a market standpoint to integrate scalar cores with accelerators so that copying between memory spaces can be avoided.

Opportunities for Co-design of Exascale Architectures Driven by Algorithm and Application Needs

Through a co-design effort, architectures (hardware and software environments) could usefully be evolved to meet algorithms and applications partway. The following are among the promising areas for influence that were identified by the workshop; additional co-design opportunities are identified throughout this report:

- Node architecture
- Memory subsystems
- Performance modeling.

Node Architecture. It may be realistic to influence the evolution of many-core chips, whose low cost (essential to envisioned exascale machines) is tied to their commodity market volume. The key departures from the petascale systems for which valuable "legacy" code exists today are as follows:

- Heterogeneous architectures with specialty processors, usually with their own memory
- "Globally" addressable memory programming within nodes or clusters of nodes
- Mixed-precision environments with a strong incentive to use lower precision wherever possible.

The biggest challenge of exascale for scientific simulation is within the node. To some extent, the burden of exploiting the resource mix of exascale node architecture is shared with all other scales of computing,

but the pressures for scientific computing have some unique stresses. Unless the scientific simulation community works with hardware architects to motivate the incorporation of features important for simulation and modeling, multicore architectural evolution might succeed in the commercial marketplace without benefiting scientific computing. To be successful in influencing the node architecture, the scientific community must identify features with low recurring cost per unit for the commodity versions. Five such features are as follows:

- Options for global address space
- Multiple levels of synchronization support
- Fast fine-grained messaging
- Fast creation and destruction of threads
- Support for floating-point exception handling, software extended precision, and vector/single instruction multiple data (SIMD) ops
- Error correction codes.

Memory Subsystems. With respect to memory, which has a leading-order effect on the budget of exascale systems, the movement to many-core nodes will reduce the memory requirements per core (relative to flat MPI) in at least three significant ways for typical driving applications:

- By centralizing common "read-mainly" data regions, such as equation of state tables
- By reducing fraction of memory dedicated to halo data in domain-decomposed distribution of work to memory units
- By reducing the MPI message buffering.

A memory feature that some vendors already are willing to provide is a chip-wide user-controlled scratchpad cache.

MATHEMATICAL MODELS AND ALGORITHMS

INTRODUCTION

Mathematical models and algorithms are fundamental elements behind any major simulation or data analysis task. Mathematical models are the equations that describe the physical or human-engineered processes that are to be simulated or analyzed. A model also can be referred to as the "mathematical problem formulation." Mathematical algorithms include discretizations of the chosen model (e.g., the representation of a continuous model in terms of discrete elements that a computer can manipulate) as well as the organization of the data and operations involved. Much of the algorithmic technology at the exascale will be defined, as it is today, in terms of multiscale **differential-equation-based models** that are enhanced with solution- or model-adaptive mesh refinement for enhanced computational efficiency. **Data analysis and visualization** methods will be used to understand results, and linear and nonlinear **solvers** will continue to dominate the overall cost of application performance. However, current trends toward the increasing use of large-scale **optimization** and **uncertainty quantification** (UQ), necessary for taking advantage of those computational results to inform policy and decision makers, will develop more fully. **Discrete methods** also will continue to expand their role in enabling technology for improving performance and empowering larger simulations, as well as becoming the basis of large-scale applications in their own right as with their application to cyber security and knowledge discovery in very large data sets. However, because of expected major changes in architecture (discussed in the previous chapter), minor retooling these algorithm classes for exascale use may not be sufficient. In most cases, significant redesign should be anticipated. Even redesigning these algorithms may be insufficient to address the difficulties posed by these architectures. The expectation is many problems will require partial or complete reformulation in terms of new mathematical models. Successfully addressing these challenges in the time frame anticipated for exascale architectural development will require a considerable co-design effort that will involve collaborations among applied mathematicians, computer scientists, and applications scientists in developing new models and algorithms with appropriate programming models, system software, and hardware designs. This chapter focuses on mathematical modeling and algorithmic challenges faced by the computational science community in adapting computing hardware architectures to exascale computational efforts.

GENERAL CONSIDERATIONS FOR MATHEMATICAL MODELS AND ALGORITHMS AT THE EXASCALE

Climate, combustion, nuclear energy, national security, and other applications critical to decision support and technology advances depend algorithmically on a common core of mathematical formulations that are multiscale and/or multiphysics in nature and typically based in three spatial dimensions and time. Some aspects of these problems involve even more space-like dimensions. For example, transport simulations can involve three spatial dimensions and three momentum dimensions plus time. The core mathematical formulations involved in the forward problems of these driving applications, in which "first principles" or other representations typically involve particles and fields, typically are ordinary differential equations, partial differential equations (PDEs), and integro-PDEs. Particle models may use general-purpose ordinary differential equation software or take advantage of more specialized approaches that exploit the problem's unique physics. Operator split parts of PDE codes may also perform intense zero-dimensional computation in every cell (e.g., to compute chemical reaction terms and exchanges between energy groups).

It is not enough to know a problem can be specified in terms of differential equations—there are many possible ways to formulate a model for a given problem. For computational purposes, the ideal choice is a way that minimizes overall computational expense for the desired output. For example, the multiscale nature of a problem can be exploited to separate scales and to formulate a model that rigorously represents important scales and how those scales interact while ignoring others. This approach would result in significantly increased computational efficiency. A concrete example of this approach is described below. Another trend that should be exploited when possible is the replacement of PDEs with integral equations or of pre-conditioners for PDEs with integral equation-mediated operations. Often, formulating a model is a mathematical challenge that will involve substantial analysis by applied mathematicians in partnership with domain experts.

In designing mathematical algorithms for a given problem, a range of options are available. The choice of an optimal algorithm also depends on the characteristics of the target computer architecture. For many applications, exascale architectures are expected to drive different choices for algorithm optimization than would be made for current codes. In the past, algorithms typically were optimized to minimize operation count. At the exascale, it is likely algorithms that maximize the ratio of operations to data movement will be preferred. For example, higher-order discretizations may prove to be significantly more efficient at the exascale because they typically involve more computation per degree of freedom than lower-order discretizations. Local methods may be preferred over global methods as memory access patterns will correspondingly be more local. For instance, communication-reducing algorithms have been demonstrated for solvers that trade the frequency of communication against memory costs and extra flops to overcome the potential loss of stability of such methods.

Mathematical Models: *Exascale-Appropriate Mathematical Problem Formulation*

For problems that are multiscale in character (i.e., different phenomena in the applications take place at very different spatial or temporal scales), there are many possible formulations of the physical problem in terms of sets of PDEs or integral equations that can be discretized and then solved on a computer. However, the computational costs for different options can vary extremely, so a formulation that minimizes the solution cost for a particular architecture must be chosen. Typically, the chosen formulation involves exploiting the multiscale nature of the problem to reduce the number of degrees of freedom required for a computational simulation and, hence, the overall computational cost.

If the multiscale natures of interesting problems are not exploited, the smallest spatial scales determine the number of degrees of freedom that must be represented on a computer, and the fastest temporal scale determines the number of time steps needed to solve the problem. Together, they determine the total computational power required. Often, however, the phenomena of most interest and relevance occur at larger spatial and temporal scales than these smallest scales. While intuition can help understand these larger-scale phenomena, performing an actual simulation to examine the large-scale behavior still will require computational work proportional to the number of degrees of freedom needed to represent the smallest spatial scale and the time steps needed to resolve the fastest temporal scales unless special steps are taken to avoid computations at these small scales. Using mathematical analysis to allow computation at larger scales can be the key to making a computation possible at all, even on the largest available computers.

An example of such a situation occurs in the study of laboratory-scale turbulent flames. In principle, one must solve PDEs that describe a compressible reactive fluid on a uniform mesh covering the physical region of interest on time scales bounded by acoustic wave speeds and chemical kinetics. However, the recognition that the relevant flow regime for the problem is very low Mach number combustion led to the development of a multiscale approach that allows computations to be performed on the convective velocity scale (10 to 50 m/sec) instead of the much more restrictive scale corresponding to acoustic waves (1000 m/sec) (Majda and Sethian 1985; Bell et al. 2010). When integrated numerically, these equations can be advanced at a time step corresponding to the convective time scale in the problem, typically 20 to 100 times as large as the time step that would have been dictated by the time scale associated with the acoustic waves. A further exploitable feature of the problem involves multiple spatial scales in the flow. Laboratory-scale flame simulations typically require finer resolution around flame fronts than are needed in other parts of the domain. Furthermore, variations in turbulent intensities lead to variation in the resolution needed to resolve the flow. As with the time-scale issue, the mesh spacing for such a calculation is determined by the smallest spatial scales that must be resolved. When using a mesh with uniform spacing, a very fine mesh of cells must be used, even in regions where not much is happening physically. Applying adaptive mesh refinement (AMR) results in an additional factor of 10 or more reduction in the runtime for simulation of these problems. As this example illustrates, mathematical considerations are as important as raw computational power in making important simulations possible. For some laboratory flame problems, the new formulation resulted in a factor of more than 700 savings in compute power, which is equivalent to at least a decade of computing hardware improvement even at Moore's law rates.

Often, a new mathematical method is applicable to more than just the problem for which it was developed. For example, the methodology and code infrastructure used to simulate laboratory-scale turbulent flames also has been adapted to study Type-1a supernova explosions. The key to this transfer of computational technology was the recognition that both problems are amenable to using a low Mach number formulation combined with AMR.

Moving Beyond Bulk-Synchronization

Almost all scalable parallelism uses a bulk-synchronous model in which data are exchanged in large intensive bursts across the network and then the network is idle while intensive local computation occurs. Probable exascale architectures will necessarily push future application codes away from this model. This evolution may not be as dramatic as it seems. Current numerical models balance numerous non-uniformities, including physics at grid cell (e.g., table lookup, equation of state, external forcing, etc.), discretization adaptivity, solver adaptivity, and precision adaptivity. From an operational viewpoint, these sources of non-uniformity are interchangeable with those that will arise from the hardware and systems software that are too dynamic and unpredictable or difficult to measure to be consistent with bulk synchronization. From the hardware perspective, these non-uniformities include manufacturing, dynamic power management, and run-time component failure. From the systems software perspective, they include operating-system jitter, software-mediated resiliency, and TLB/cache performance variations. Network non-uniformities can arise from either hardware or software reasons.

While the total elimination of periodic synchronization appears impossible for the applications mentioned above, there is an opportunity for application writers and library designers to write code in styles that require significantly less synchronization than today. This can be illustrated with recourse to an implicit loop in a typical PDE-based application. At the core of such a computation is a loop that involves

forming a global residual, solving a system of linear equations to obtain a correction, bounding the step of the correction, and updating the physical state. This loop, which embeds various global reductions, cannot easily or apparently be avoided. However, operations that could be done less synchronously are often inserted into this path because of the limited language expressiveness available to indicate that their importance is secondary.

For example, Jacobian refresh and preconditioner refresh can be taken off the path. In the best implementations, they already are done relatively infrequently to amortize the relative high cost of their formation and application. They must be refreshed to attain theoretical convergence in nonlinear problems; however, if some processors are in arrears heading to the next synchronization, they can be deferred. This is most easily accomplished if the code to refresh them is isolated from the code that uses them and is called when cycles are available so that critical path progress does not stall for convergence reasons. The main distinction between these tasks and the tasks on the path is they are very time consuming and their load can be redistributed dynamically without halting the critical tasks. The same is true for many other tasks that are performed on the critical path today but could be done less frequently. This includes convergence testing itself. Many iterations could be performed between tests at the cost of an occasional unnecessary iteration. Similar lower-priority status can be accorded sub-threads that perform constitutive updates, algorithmic parameter adaptation, input/output (I/O), compression, visualization, data mining, UQ, etc.

In addition, in pursuit of synchronization reduction, "additive operator" versions of sequentially "multiplicative operator" algorithms often are available. Historically, such algorithms sometimes have been dismissed (e.g., "chaotic relaxation" as proposed by Chazan and Miranker [1969]). In retrospect, these algorithms were ahead of their time because the hardware of the day did not drive the demand or provide the opportunity that it does today. Sometimes, asynchronous-additive methods have been made virtually as good as multiplicative methods (e.g., "AFACx" versus "AFAC" [Lee et al. 2003]).

To take full advantage of such synchronization-reducing algorithms, greater expressiveness in scientific programming must be developed. It must become possible to create separate sub-threads for logically separate tasks whose priority is a function of algorithmic state not unlike the way a time-sharing operating system works. Some software paradigms already may exist that used in this way, such as the Asynchronous Dynamic Load Balancing construct (Lusk et al. 2010), or prioritized forms of tuple-space programming. In adopting such programming styles, it may be necessary to introduce "soft barriers" to prevent the state from becoming too stale.

Exascale programming will require prioritization of critical-path and noncritical-path tasks, adaptive directed acyclic graph scheduling of critical-path tasks, and adaptive rebalancing of all tasks with the freedom of not putting the rebalancing of noncritical-path tasks on the path itself.

Algorithmic research can pay dividends well beyond research into novel programming styles and implementations. For instance, communication-reducing algorithms have been demonstrated for solvers that trade the frequency of communication against memory costs and extra flops to overcome the potential loss of stability of such methods. Stepping up to a higher level than algorithms, better mathematical formulations may be available. One trend to be exploited wherever possible is the replacement of PDEs with integral equations or of preconditioners for PDEs with integral equation-mediated operations.

Exploiting a Mixed-Precision Arithmetic Approach

An exciting prospect in programming exascale hardware is to exploit a mixed-precision, floating-point arithmetic approach, striving to use the lowest precision required in a local tasks to achieve a given global accuracy outcome. The benefits of lower precision have been dramatically demonstrated on heterogeneous systems such as those built from general-purpose processors and graphics processing units. The potential advantages of this approach include reductions in execution and communication times, and in power consumption. These techniques are not unknown in petascale practice. In fact, for more than a decade, they have been in practice in extensively used U.S. Department of Energy (DOE) codes such as PETSc, in which preconditioner elements are stored and communicated in single precision by default, whereas the Krylov computation is performed in double precision. Algorithms, such as iterative refinement that make selective use of higher levels of precision, are classical in numerical analysis. The key is to reformulate algorithms to find corrections rather than solutions. The solutions must be stored at the desired precision level, but the corrections, which ultimately affect only the lower bits of the solution, can be calculated in lower precision.

Code Development at the Exascale

Exascale machines likely will expand current million-way flat parallelism across nodes with thousand-way parallelism within a node. This advance offers a pseudo-evolutionary path for applications in that message passing interface (MPI)-based legacy code still will be usable on the million nodes. Changes will be mainly within nodes, in which thousand-way parallelism must evolve to "MPI+*X*," where X is the programming model used within a node (e.g., thread-based). Notably, this evolution will be shared by the commercial marketplace. A decade from now, laptops may employ single nodes, but this same thousand-way parallelism will drive the development of software at all levels, from basic system software to advanced scientific and engineering applications.

Loosely speaking, under this incremental hardware path to the exascale, the algorithmic path from the petascale to the exascale preserves the focus on weak scaling between nodes and adds (mostly) strong scaling within nodes, the latter presenting the greatest challenge to future application developers. If sufficient memory is available, there is adequate concurrency in tasks such as PDEs for thousand-fold strong scaling (see Burstedde et al. [2008], "Scalable Adaptive Mantle Convection Simulation on Petascale Supercomputers," which was a 2008 Gordon Bell Prize finalist), that was mediated by MPI alone. A key consideration in expanding from current million-fold thread concurrency to billion-fold on currency is in the global reduction. The synchronization cost of global reductions propagates to the thousand intranodal threads with a much smaller coefficient than the MPI-mediated internode part. Because remote dynamic random access member (DRAM) access is not inordinately slower than local DRAM access, dynamic load balance through work stealing is relatively more sufferable in this architecture. The cost is in local memory management. The internode memory management cost is not significantly greater than the local.

For PDE-derived discrete systems, domain decomposition is the weak-scaling method of choice. It allows asymptotically linear scaling of nodes with problem size, per iteration, given log-diameter global reductions. Given a superlinearly to quadratically convergent inexact Newton method, it may have an asymptotically constant number of outer nonlinear iterations. Under an appropriate physics-, Schwarz-, or multigrid-based preconditioned Krylov method, it may have an asymptotically constant number of inner iterations in solving the linear systems that arise at each outer Newton iteration. This

asymptotically optimal scaling is nontrivial to achieve at all three levels—the per-iteration constant scaling and the constant number of linear and nonlinear iterations. However, such scaling has been approached on a number of applications that make effective use of current petascale machines. Continued research is necessary on preconditioning that extends such results for exascale computing.

Importance of Software Libraries

A natural place to look for the earliest exascale driver applications is among today's petascale applications. A factor of 1000 in capability actually is a modest step in resolution when spread into three space dimensions or even higher dimensions in problems posed in phase space, such as Boltzmann formulations. Current petascale applications are built on a large number of software toolkits, most of which were developed by DOE and are actively maintained on National Leadership Computing Facility platforms by DOE scientists. On the modeling side, these toolkits include geometric modelers, meshers, discretizers, partitioners, solvers and integrators, systems for mesh and discretization adaptivity, random number generators, libraries of subgridscale physics models and constitutive relations, UQ, dynamic load balancing, graphical and combinatorial algorithms, and compression.

Code development efforts at the petascale rely on another fleet of software toolkits, such as configuration systems, compilers and source-to-source translators, messaging systems, debuggers, and profilers. Finally, production use of these petascale applications rely on dynamic resource management, dynamic performance optimization, authentication systems for remote access, I/O systems, visualization and data analysis systems, data miners, workflow controllers, and frameworks. These toolkits will all need to be supported on emerging exascale machines, for which they will need to be supplemented by additional tools such as fault monitoring, fault reporting to the application, and fault recovery.

Software Engineering Practices

Software engineering practices already well established and in use at the petascale should remain the same throughout the transition to the exascale. One of the most important of these practices is multilayer software design. Successful software is multilayered in its accessibility. The outer layer, which is accessible to all users, is an abstract interface featuring the language of the application domain, while hiding implementation details, with conservative parameter defaults. For example, a vector should be represented and manipulated as an element of a Hilbert space without regard for how it is implemented in an array data structure partitioned across multiple memory systems. The goals of accessing the software at this layer are robustness, correctness, and ease of use. The middle layers, which can be revealed to experienced users through handles to the underlying objects, provide a rich collection of state-of-the-art methods and data structures. These layers can be exposed upon demand and are variously configurable. The goals of access are capability, algorithmic efficiency, extensibility, composability, and comprehensibility of performance and resource use through profiling. The inner layer, which is intended for developer access only, includes support for a variety of hardware and software execution environments. The goals of access are portability and implementation efficiency.

OVERVIEW OF EXASCALE CONSIDERATIONS FOR MATHEMATICAL TOPICAL AREAS

The Crosscutting Technologies for Computing at the Exascale workshop focused consideration on the five classes of algorithms already mentioned—namely PDEs, linear and nonlinear solvers and

optimization, data analysis and visualization, UQ, and discrete methods. The following sections focus on these algorithms in more detail, outlining many of the challenges and possible solutions in the context of exascale computing.

Partial Differential Equations

A range of DOE applications focus on PDE solutions, including accelerators, astrophysics, climate, combustion, defense science, fusion, nuclear energy, and subsurface flow. A characteristic common to most of these applications is they span an array of spatial and temporal scales and require the modeling of a range of physical processes with disparate mathematical properties. Computational modeling of these multiphysics, multiscale systems has evolved to include a combination of fairly sophisticated mathematical and numerical methodologies.

Many applications use some form of AMR to match spatial resolution of the discretization to local resolution requirements. For applications in which the focus of the dynamics is on the behavior of lower dimensional features of the solution, such as a distinguished interface, specialized techniques for modeling these interfaces (e.g., front-tracking or level-set approaches) have been developed.

Similarly, many applications employ some strategy for separating disparate temporal scales associated with different physical processes and treating each with discretization approaches appropriate to those particular scales. This type of scale separation can be derived from asymptotic analysis or physical intuition about the scales relevant to the problem. Coupling of the processes is then accomplished with a variety of techniques, from simple operator-splitting approaches to iterative approaches to couple the processes in a fully implicit approximation.

Another feature of many emerging application simulations is the use of different models to describe different parts of the simulation. One example of this type of simulation is the use of molecular dynamics at the tip of a crack in a solid coupled to a traditional elasticity model used in the region located away from the crack. Another example where this type of simulation arises is in climate models that include oceans, atmospheres, land surfaces, and sea ice in a single simulation.

To make effective use of exascale computers for PDE-based applications, the entire workflow associated with the PDE solution process, from mesh generation to solution and visualization, should be considered. In general, the workflow starts from a high-level problem definition in terms of a set of mathematical equations subject to boundary and initial conditions over some space-time domain where the spatial domain often is expressed in terms of a mathematical definition, such as a solid model (computational geometry). From this point, a reasonable starting mesh for that space-time domain must be constructed, and a numerical discretization on the mesh must be solved. The resulting solution would be analyzed to determine if the combination of the mesh and its discretization yielded the desired accuracy. In most cases, the initial mesh/discretization is insufficient and must be adapted. From many problem classes of interest, the steps of mesh/discretization, adaptation, solution, and evaluation are repeated many times. The discretization and solution techniques chosen can vary considerably and can range from techniques with local support, such as finite element methods, to those with global support (e.g., spectral methods). Solution methods can use implicit or explicit solvers, be focused on single or multiple types of physics, and incorporate one or more scales. The varying characteristics of these different methods will dictate how each performs on an exascale computer as well as the type of research required to address the development of new, successful methods.

Given this workflow, it is apparent the automated adaptive solution of PDEs on exascale machines will tax all aspects of the exascale system's computation and communication fabric. Thus, a primary concern with respect to the hardware is what constitutes a good, balanced system.

With respect to typical workflows for PDE solutions, some key characteristics for exascale-size problems are described below.

- For local methods, the numerical discretization and solution process is computationally dominant and includes the need to support regular communications. Most of these communications are within local neighborhoods. However, there are times when specific global communications are required. In addition, with current methods, these global communications require specific synchronizations. Although there may be opportunities to continue to reduce the synchronizations, their complete elimination is unlikely. For some problems, the communication patterns for the discretization/ solution step are fixed, and workloads can be accurately predicted; however, problems characterized by heterogeneous physics or that use adaptive technology often have unpredictable computation loads, communications, and memory load patterns. This variability requires both predictive computations and dynamic load balancing to create a well-balanced system for the next discretization solution step.

- For more global methods, such as spectral techniques, the work also is computationally dominated, but the communication patterns are global in nature (e.g., fast Fourier transforms [FFTs]). Moreover, as currently implemented, these methods may not scale well to the exascale. To improve scalability, research is underway on using real-space methods to create hierarchical approaches that localize these techniques.

- In the case of unstructured mesh generation, there are techniques that, given the mathematical definition of the domain, can automatically create the meshes. These mesh-generation processes are highly irregular and dominated by substantial communications that become increasingly local as the mesh is being formed. In the case of general geometric domains, it is impossible to determine tight predictions of local workloads, making dynamic load balancing critical.

- For problems with a multiscale or multiphysics nature, these same concerns exist, and are augmented by the need to transfer information among simulation components. These communication patterns can take a variety of forms, ranging from local on-node transfers of information to substantial many-to-many transfers of information across significant portions of the computational domain.

- Because I/O is, and will continue to be, a dominant cost in the solution process, it is imperative that all of these processes be executed on the exascale computer. One potential exception is initial mesh generation in cases where it is known that a very coarse initial mesh can be used so that the initial input is "small."

- One approach to reducing the amount of calculated data stored —and to making the simulation more effective—is support of *in situ* visualization and feedback control to change simulation conditions (e.g., boundary conditions, geometry, etc.) as the simulation proceeds. Such capabilities require the ability to interact with connected visualization systems to extract and communicate appropriate data and control effectively.

These are only a few of the critical aspects of the solution of PDEs on exascale computers that are expected to affect performance. A more exhaustive list can be determined through discussions with a broad range of scientists exploring a multitude of different solution approaches.

Interplay of Machine Characteristics with PDE Methods and Workflows

The machine characteristics most critical to solving PDEs are a function of the simulation workflow. During the discretization/solution process, there are a number of interesting node and communication characteristics that must be considered. These characteristics are described below.

- Effective use of the heterogeneous computing units (central processing units [CPUs] and graphics processing units [GPUs]) and their interactions to gain a high degree of parallelism will be critical.
- Effectively accounting for the memory hierarchy and access will be important to gaining node-level performance.
- Dealing with the memory limitations will be difficult for some techniques. Other techniques may effectively scale with the limited memory, but there will be a limit to the total size problem that can be solved.
- Unless it is essentially nonexistent, OS jitter can strongly compromise scaling.
- The current implementations of some techniques (e.g., spectral methods based on FFTs) place more demanding requirements on internode communications than can probably be sustained with emerging exascale architectures.

Although heavily dominated by local calculations, the discretization/solution process does require more than just nearest-neighbor communications and at least limited global synchronized communications.

The mesh generation and mesh adaptation processes, particularly for unstructured meshes, likely will lack the ability to take effective advantage of GPUs as they are currently designed. Therefore, it is assumed that most unstructured mesh-generation and adaptation processes will be executed on node-level CPUs. Because the level of computation required for these steps is typically "small" with respect to the solution process, this may well be acceptable if these steps can be executed with sufficient system-level scalability and node-level performance. The ability to control the communications effectively, particularly node-to-node communications, is of critical importance to these steps in the process. The communications required are irregular and dominated by small messages. The communications are to neighbors. However, the neighborhoods can evolve as the mesh is generated and/or adapted. The need to support fast dynamic load balancing to account for the fact that the inability to accurately predict local workloads for meshing operations leads to load imbalance also is important to node-level performance and system-level scalability of these steps.

Initial experience on current near-petascale machines indicates that critical mesh adaptation steps can be conducted on massively parallel computers and not become the dominant step. More research and development is needed to determine how effectively all mesh adaptation steps and the more complex mesh generations processes will be executed on these and larger machines.

Beyond simple check pointing techniques, which will not be effective on exascale machines, the requirement that applications deal with fault tolerance is new to PDE applications developers. For application developers to consider inclusion of fault-tolerant techniques, methods must be developed to inform the application that an error has occurred, including information on what type of error along with where and when it occurred. The implication of providing the needed information influences all aspects, including hardware, systems software, and application.

Algorithm Research Required to Address Hardware Constraints

As the movement toward exascale simulation progresses, development of these methodologies will require a significant research effort in applied mathematics. The following paragraphs describe some specific areas of research necessary to develop accurate, robust, and efficient simulation methodologies.

- There is an overall need to rethink how PDEs are discretized in a way that reflects the character of proposed exascale architectures. Key themes that need to be considered are described below.
  - PDE discretizations must be revised to reflect a paradigm shift from flop-constrained to memory-constrained architectures. For example, higher-order discretizations have the potential to reduce the number of degrees of freedom needed to represent the solution at the expense of requiring more flops. Quantifying these tradeoffs for potential exascale architectures could lead to a significant shift in the way fundamental discretization issues are approached.
  - Temporally implicit, fully coupled formulations often are used for multi-rate applications in which dynamics of interest are in scales that are much slower than the fastest time scales in the problems. High-order temporal discretizations are useless in the presence of loosely coupled low-order temporal operator splitting, which is one factor that drives scientists toward full coupling. In addition, stability theory for loose coupling of nonlinear problems is, at best, incomplete.
  - Similar to the above, there also is an overall need to develop algorithms that compute more for a given level of communication. For example, potential theory type ideas can be used to develop approaches for solving elliptic PDEs that require significantly less communication than traditional iterative approaches.
  - There is a need to develop algorithms that can operate more asynchronously than traditional algorithms based on a bulk synchronous execution model. A critical point is to understand how asynchronous operations might affect accuracy and stability properties of the resulting methods.
- As approaches are developed for increasingly complex physical problems, a detailed analysis must be performed of adaptive mesh algorithms for multiphysics applications. In particular, the accuracy and stability properties of such approaches to coupling different processes across different levels of refinement must be quantified.
- A detailed analysis of temporal coupling strategies for multiphysics applications must be performed. For more complex systems and potentially higher-order discretizations, lower-order operator splitting approaches will no longer be satisfactory. Improved techniques for iteratively coupling different processes will be needed.

- The increased compute power of exascale computing will enable consideration of new classes of multiscale problems in which different types of discretizations, appropriate to a particular scale in different portions of the domain, are employed. As this type of simulation expands, there is a critical need to develop systematic approaches for coupling across the range of scales and quantification of the properties of these types of coupling strategies. Similar research issues also arise when modeling problems that treat two distinct phenomena in different parts of the domain, such as ocean-atmosphere coupling in climate modeling or surface water-subsurface water coupling in hydrology.

- Within the PDE area, there is an implicit need for research to develop effective iterative methods for the sparse linear and nonlinear systems that arise in discretizing PDEs.

- Although by its nature not well-defined, there also is merit in considering alternative approaches to formulating PDEs that could offer dramatically different approaches to solving problems at the exascale.

Completely New PDE Parallelization Techniques

Current PDE approaches are based on a domain decomposition model in which the simulation domain is broken into subdomains that are distributed to processors. Within subdomains, there are still substantial levels of concurrency. Proposed exascale architectures will have nodes that have 1000 or more cores on the node. Effective use of these machines will require efficient exploitation of parallelism at the node level. For PDEs, this means that parallelism must be exploited within the subdomains inside the nodes. To support this, the applied math PDE community must work with the programming models and environments community to develop useful abstractions to expose both loop-level and data-level parallelism for computations on subdomains. PDE algorithm developers will require relatively simple approaches for specifying fine-grained parallelism that can be efficiently translated into a lightweight parallel implementation. In addition, the PDE community must work with the programming models and environments community to develop approaches for expressing asynchronous algorithms so as to hide communication latency and avoid load-imbalance issues. This likely will require the development of approaches for expressing nonstandard data layout issues as well. Issues relating to moving away from a block synchronous execution model will become increasingly important as a broader range of multiscale problems is considered. It also holds the key to achieving strong scaling on algorithms that cannot express more parallelism through spatial partitioning. Moving towards these kinds of asynchronous programming models also will require advanced load balancing methods.

Effect of Algorithm Requirements on the Hardware

Interconnect. The movement towards strong scaling will result in a much higher volume of small messages. Thus, support for handling small, lightweight messaging will be critical. Inconsistent messaging performance can make the already challenging load-balancing problem more difficult, even for nonadaptive codes. Advanced network interface card designs to support ultra-low-overhead messaging for very small messages will be essential. The network must support synchronization primitives for advanced programming languages, such as asynchronous collective operations and memory fences.

On-Chip Memory. Given the importance of minimizing unnecessary data movement, the ability to allow software control of data movement for performance-critical kernels would be beneficial. Methods

that can coexist with conventional, automatically managed caches are important because an incremental porting path is required. Support of some form of global address space (GAS) would be desirable.

PDE Solution Algorithms Research and Development

To exploit exascale machines effectively, almost every aspect of PDE solution methods will need to be readdressed. Notably, there is a corresponding unknown degree of risk with respect to determining successful methods. Some of the areas of research and development identified at the *Cross-cutting Technologies for Computing at the Exascale* workshop are listed as follows:

- **Newton-Krylov methods**. Can techniques with full approximation scheme multigrid methods that better respect hard limitations on memory be used?
- **Data layout and multi-precision algorithms for reducing cost of accessing memory**. How do these methods affect convergence and other numerical properties?
- **Resiliency**. Development of methods to effectively increase exascale simulation resiliency, including effective local check pointing, algorithms to recover for detected errors efficiently, and algorithms that use available computation resources to determine when errors have occurred in critical code segments. Can **MapReduce** transaction computing be used to increase the resiliency of PDE simulations in an effective manner on exascale computers?
- **Scalable FFT-based spectral methods**. Are two-, three-, or higher-level hierarchical methods possible; should more local real-space methods for computing transforms be used instead?
- **Mesh generation**. Mesh generation is not likely to scale fully on exascale machines, but doing it independently of a simulation could incur huge I/O costs. What is the proper balance?
- **Time parallelization**. Some work exists to extend the range of applicability of existing techniques. Research to date indicates that unless substantially more memory is used, the amount of time parallelism possible is quite limited.
- **Trading flops for communication**. Take advantage of extra flops for useful work (e.g., for resilience computations or re-compute tables rather than accessing from memory).
- **Dynamic load balancing methods**. Dynamic load-balancing methods can deal with multilevel adaptive computations, such as multiscale methods, both concurrent methods, and hierarchical methods, including sub-grid models.
- ***In situ* visualization and data analysis***:* Development of methods needed to support the execution of ***in situ*** visualization that will include effective visualization extractions and data reduction in going from the exascale machine to visualization nodes to the display system.

There is a need to identify and exploit the legacy investments already made to get to petascale. Many of them will contribute markedly to effective simulations on exascale machines.

Global Addressing. Message passing probably can continue to be used for internode communication for exascale systems, but it is not practical for interprocessor communication at billion-way parallelism. Global memory addressing is preferred over cache-coherence for managing fine-grained computation and data movement on-chip.

Fast Reductions. Virtually all implicit time stepping techniques rely on Krylov subspace methods, which require fast inner-product summations. At the system scale, this appears as a local summation

followed by an "all-reduce" global collective operation. As such, there is a critical need for hardware support of fast all-reduce because such steps are latency limited. Blue Gene has demonstrated the value of hardware support for fast global collectives. Given the move toward asynchronous execution models, this capability would need to be generalized to support for asynchronous collective operations.

Additional Co-Design Opportunities for PDE Methods and Workflows

Working on quantification of the parameters that define a balanced exascale computer for various classes of PDE methods and workflows would provide extremely valuable input for co-design. At the heart of the co-design iterations will be examining possible trade-offs and alternatives, some of which are described below.

- Can the applications use specific redundant calculations to trap various faults? What aspects of the node architecture can be used to support these redundant computations?

- How do applications maximize more "neighborhood" communications and reduce global communications and synchronizations, including understanding possible increases in computation and/or memory required?

- Understanding key characteristics of PDE solution algorithms (e.g., some algorithms require variable node-to-node communications dominated by small messages) provides an opportunity for hardware developers to support those needs effectively. Obvious examples include using local neighborhood information to support lower overhead communications, low-latency modes for small messages, etc.

- In developing and optimizing the best methods to support check-pointing, understanding which and how much information must be check-pointed is critical information for the hardware designers.

- Fast dynamic load balancing, potentially both intra- and internode, for operations requiring moving limited amounts of work and data within neighborhoods. For example, the need to move small numbers of elements between neighboring mesh parts to support a mesh adaptation operation.

- Consideration of methods to move data and modify tasks as needed to support *in situ* visualization and analysis and/or simulation steering.

The Role of Mini-Applications and Hardware Emulators

To support experimentation and evaluation of the possible hardware configurations, simulation tools that emulate node architectures must be made available to application developers in the near term. In return, the application developers must provide component operations and mini-application implementations to test system simulations. Mini-applications (or mini-apps, sometimes referred to as "compact apps") are smaller-scale versions of full-scale application science codes, representing the algorithms being used within the full-scale applications but in a simplified setting with reduced complexity. By definition, mini-apps are not being constantly modified to experiment with new scientific developments. Thus, mini-apps represent a simplified and stable basis for the exploration of computer science and mathematical ideas that, if successful, may be extended to be included in the full-scale applications. Importantly, mini-apps provide a basis for non-application scientists to study the application codes within a simplified setting. Mini-apps are appropriate for algorithm scaling studies, evaluation of alternative solvers, exploration of

new programming models, and other uses. Mini-apps can be developed manually either from scratch or through careful simplifications of original full-scale applications; either way, their development requires significant application-specific expertise.

Examples of basic component tests for PDEs are described below.

- Sparse matrix vector multiply and dot product (global and node-level matrices)
- Interactions between CPUs and GPUs for optimizing on-node parallelism and performance for various operations
- Migration of data for AMR; performance of iterators across levels (e.g., on-core versus on-node versus off-node)
- Operations that can compare data interleave verse blocked data operations
- Operations, such as selected mesh adaptation functions, to test the effectiveness of the neighborhood communications and fact neighborhood dynamic load balancing.

A primary goal of the mini-apps is to ensure they have features of the entire workflow that must be supported by the application.

Mini-apps may *still* be too large to be run on hardware simulators. A significant gap exists between the complexity of mini-apps and the required inputs to drive hardware research and exascale co-design. Thus, *skeleton applications* must also be developed that isolate one or more specific features of an application (either a mini-app or a full-scale application). These skeleton applications are designed to serve as inputs to hardware simulators. Skeleton applications can potentially be developed automatically using code analysis tools such as ROSE.

Solvers and Optimization

There are a number of well-known issues in solvers and optimization for scalable computing, and anticipated issues going forward. Some of these issues described below.

- **Poor scaling of collectives***:* Collective operations (e.g., global vector norms and dot products) are ubiquitous in numerical linear algebra and require global synchronization across the parallel machine. On very large systems, the operations incur a large cost.
- **Poor floating-point performance of sparse matrix-vector multiplication (SpMV)**: SpMV is the single most expensive kernel in many applications. It typically has a very low computation-to-communication ratio, little temporal locality, and typically employs gather/scatter patterns.
- **Poor scaling of coarse grid solves**: Scalable pre-conditioners tend to be multilevel methods, which traverse a hierarchy of fine–to-coarse grids or graphs. The coarsest grid typically is very small, easily fitting on a few processors, and requires a precise solution, often by a direct solver. This phase becomes an increasingly serious serial bottleneck in the computation as processor counts increase.

- **Lack of robustness and reliability**: Solution algorithms are not always robust or reliable, requiring user intervention and knowledge of tuning parameters. This increases when attempting to scale on large systems because scalability often comes from sophisticated algorithms that are very sensitive to tuning parameters.

- **Repeatable results (non-determinism)**: Use of shared memory parallel algorithms introduces non-determinism in floating-point addition. In particular, collectives, when executed with optimal performance, will typically produce varying results for each run of an application, leading to verification issues.

- **Resilience in the presence of soft errors**: Many algorithms fail if a single calculation is wrong. Dealing with soft errors (e.g., bit-flips) is important.

Solvers scale more poorly than other portions of an application on a multi-core node. Because of good temporal locality, computation of physics, construction of global operators, and other application-specific computations tend to scale well on multi-core nodes. Solvers tend not to scale well when transitioning from using one core to all cores on a node, because the algorithms are bandwidth-limited and there is insufficient bandwidth to support computation on all cores.

One way to address the poor performance of sparse iterative methods is to avoid forming a matrix, relying instead on matrix-free formulations and nonlinear methods such as full approximation scheme multigrid methods. These algorithms tend to be more problem specific, but it is possible to develop a general framework that can be customized for specific problems.

Data dependent parallelism such as level-scheduling for sparse factorization and forward-back solves requires explicit thread barriers between levels of node sets. However, for efficiency a thread that arrives at the barrier must be kept active so as to immediately resume execution past the barrier after all other threads arrive. In contrast, a passive barrier must be used at the completion of a library kernel, so that hardware resources can be released once the library function is complete (especially if the library function is using a different thread application programming interface [API] than the application). As a result, we need two different kinds of barriers.

Exascale architecture designs are exascale only in operation count. All other measures (e.g., bandwidth, memory size, etc.) are likely to be improved by factors ranging only from 10 to 100 times over petascale systems. To address this disproportionate growth in capabilities, solvers can be redesigned and implemented to improve computation-to-bandwidth performance, and computation-to-memory size ratios. These opportunities have always been available, but they are more important today.

The basic algorithmic approaches used today for solvers and optimization will continue to work at the exascale, perhaps with some scaling limitations. Changes in node architecture (e.g., efficient threading) could make sparse kernels more efficient. Changes to algorithms and data structures can improve compute-to-communication ratios; however, current forward solves will not fully scale because of the lack of concurrency.

Dense eigensolvers currently do not scale well. Sparse eigensolvers that have communication-avoiding strategies and regular grids and that do not use pre-conditioners will scale in a few years. FFTs and

sparse linear solvers will encounter difficulties, but progress could be made with co-design efforts involving algorithm designers and programming model designers.

Optimization algorithms, a broader set of UQ algorithms, and first-principle calculations in some regimes may be enabled at the exascale because of sheer increase in capability. However, memory size limitations may restrict the scope of first principle calculations.

Resiliency is an important consideration that also demands much research in moving from petascale to exascale applications; however, it should not be showstopper despite the significant factor of a thousand times more threads on the critical path of the computation. First, the number of physical nodes is not much greater because most of the parallelism occurs within a node. Hardware failure rates, after the chips are burned in and certified following manufacture, do not increase as strongly with the complexity of the chip as with the number of sockets. More importantly, in the face of decreasing time between failures, driving physical applications generally possess a time-dependent working set that varies in size periodically and predictably, reaching minimal size at each time step or evolutionary step towards equilibrium when the physical state is updated.

The vast number of workspaces carried around, such as tables for equation of state on the physics side or stored and factored Jacobian matrices on the numerical side, could be eliminated. They are ephemeral and can be regenerated from the minimal physical state variables, which are dumped often for visualization/analysis. To be sure, they can be reused with efficiency, and are in good implementations, but in the case of failure, the computation can be rolled back to the last dumped state and restarted with some overhead of recalculation. Some faults such as soft memory errors occur in parts of the code that are not on the critical path of state evolution, but are carried to improve performance or to aid interpretation. While it is inefficient to lose them, and while their loss may cost a frame of a visualization or an error bar in an uncertainty quantification, or an optimization step that must be discarded along the way, the overall forward model may remain intact, and not all errors must trigger restart. For this advanced form of application-mediated resiliency, significant new tools for monitoring faults and reporting them to the user are required, and users must be willing to enter into the recovery process in a collaboration of co-design that is unprecedented to date at the petascale.

In contemplating the direct map of petascale solver algorithms to the exascale, the following issues appear to be the greatest challenges. Each of these challenges is an exacerbation of an issue that already exists at the petascale:

- Poor scaling of global reductions because of internode latency
- Poor performance of SpMV because of shared intranode bandwidth
- Poor scaling of coarse grid solutions because of insufficient concurrency
- Lack of reproducibility because of floating-point noncommutativity and algorithmic adaptivity (including auto-tuning) in efficient production mode
- Difficulty in understanding and controlling vertical memory transfers.

Data Analysis and Visualization

Extracting knowledge from increasingly large data sets, both experimentally and computationally produced, continues to be a significant challenge for scientific discovery. Only two years ago, a DOE workshop considered the mathematical challenges faced in data analysis at the petascale and concluded that "… extracting scientific knowledge from these massive data sets has become both increasingly difficult and increasingly necessary, as computer systems have grown larger and experimental devices more sophisticated" (DOE 2008a). Many of the research challenges identified at that workshop only recently have begun to be addressed, and the difficulties will only increase in the coming years.

Experimental fields, such as high-energy physics, report that experimental data sets are expected to grow by six orders of magnitude or so in coming years (DOE 2008b), driving the need for more *in situ* analysis and exploratory tools, such as visualization, to understand them. In addition, a major bottleneck is presented by the unfavorable scalability characteristics of current analysis algorithms. Higher order correlations scale naively as N^k where k is the order of the correlation function. For current data sets, 3-point statistics (k=3) have been measured on 50 to 100,000 sources in 10,000 CPU hours (estimating the covariance of these measures requires multiple realizations of these or simulated data sets).

The climate community has identified the need for performing certain analysis tasks near to the data, such as monthly means or other summarization operations (DOE 2008c). The availability of spatiotemporal data of unprecedented frequency and resolution on climate time scales will result in analysis problems of scales never before tried and for which the availability of appropriate techniques is unclear. Similar frequency and resolution density also has been identified for nuclear energy systems (DOE 2009).

For fusion science, estimates based on mean time between failure concerns when running on a million cores indicate that fusion energy codes will need to output 2 gigabytes/second per core or 2 petabytes/ second of checkpoint data every 10 minutes (DOE 2010a). This amounts to an unprecedented I/O rate of 3.5 terabytes/second. Data from fusion experiments are expected to increase to terabyte sizes. Therefore, robust synthetic diagnostic tools need to be developed that are cross-platform scalable and based on forthcoming community standard data formats.

Data sets produced from modeling and simulation at the exascale will be correspondingly larger than those produced today; the flop-to-I/O imbalances of those machines will increasingly drive the need for more *in situ* or "on-the-fly" analysis in future simulation codes.

These examples raise important analysis and systems software issues. Managing large-scale I/O volume and data movement and optimizing I/O performance automatically based on hardware characteristics are important endeavors. For efficient use of exascale architecture, real-time monitoring of simulations and runtime metadata generation are essential. Workflow must be scaled and enhanced to support summarization of results in real time and/or permit the monitoring software to manage automatically simulations that do not progress correctly. In addition, the issue of novel data formats that support advanced operations, with a particular emphasis on comparison between simulation and experiment, has been introduced. Detection and tracking of interest areas, such as coherent structures and fronts, when they likely are spread across many processors and processing techniques to reduce overall data size before these data are output by the simulation also are important issues. Even after this reduction, the volume of data left can be enormous. Thus, an open question is reduction and mapping of terabytes or petabytes of data into meaningful visualization. Finally, many of the requisite technologies for large-scale data

analysis are likely to require new advances in fundamental applied mathematics. There is a significant opportunity to engage the applied mathematics community to consider many of these challenges.

General issues facing data analysis at the petascale will only be more stringent at extreme scales. Techniques must be identified, adapted, and developed for several factors, including the following:

- Accommodating large-scale, heterogeneous, high-dimensional data while avoiding complexities posed by the curse of dimensionality
- Performing data and dimension reduction to extract pertinent subsets, features of interest, or low-dimensional patterns
- Analyzing and understanding uncertainty, especially in the face of messy and incomplete data
- Identifying anomalies in streaming and evolving data in near real time. This is needed to detect and respond to phenomena that are either short lived (e.g., supernova onset) or urgent (e.g., power grid disruptions).

In turn, the need for novel techniques generates new conceptual and mathematical challenges. One of the fundamental new challenges for data analysis at the extreme scale is the development of algorithms that only require a single pass through the data and/or are able to handle distributed data *in situ*. Ideally, such algorithms should incur a minimum loss of information. In the case of non-stationary spatio-temporal data of high frequency and resolution, suitable correlation functions needed for statistical representation are unknown. Finally, analysis of such data sets will pose computational challenges that may be flop-count limited and for which new, scalable approaches need to be developed, including optimization methods for regression or maximum likelihood analysis and improved sampling methods, such as adaptive sampling.

Discussion of Operational Modalities (Hardware Configurations) for Data Analysis at the Exascale

The predominant approach to data analysis and visualization for large-scale simulation in the sub-petascale era has been to perform these functions after the simulations are complete, often after moving the computed data to specialized visualization or analysis computers. However, the sheer size of data sets that potentially will be produced by exascale computations suggests this approach may be untenable. Bandwidths will not be sufficient to move data of this quantity around, so alternative paradigms must be considered. The *Cross-cutting Technologies for Computing at the Exascale* workshop considered three main operational modalities that might be employed for data analysis and visualization at the exascale, including:

1. *In situ computing,* where the exascale platform concurrently handles all visualization and analysis tasks while the simulation is taking place
2. *Heterogeneous in situ computing*, where a subset of nodes on the exascale platform are organized differently to act as I/O servers and embedded data analysis engines
3. *Post-processing of results*, either with a system custom built around the requirements of visualization and analysis applications or the exascale platform itself.

Variations on these three modalities are considered in the following subsections.

In Situ Analysis Modality

In situ analysis modality primarily involves running analysis concurrently with the computing application, co-resident on-node with the application as it is running. It offers the advantage of minimizing data movement to disk by collocating the analysis with the application in node memory. As this approach is not in common use today, there is little software infrastructure available to support it. In addition, there are no common data structures shared across application codes, which will make it difficult to create reusable data-analysis algorithms for different *in situ* environments. The development of reusable components is critical to a robust analysis capability, so this approach presents the double challenge of developing new software infrastructure and common data structures to support it.

Given that nodes in exascale systems will be extremely memory-limited, having analysis tasks competing with the simulation for resources may prove to be a significant challenge. Overcoming these memory limits creates opportunities for developing hierarchical algorithms for data analysis and visualization.

Because not all analysis is load balanced, there is some concern it will interact poorly with application-integrated load-balancing schemes and make an extremely challenging problem even worse. However, it also may lead to new opportunities to create both hardware and software structures that more comprehensively treat and mitigate load-balancing issues.

The assumption is that *in situ* analysis can reduce the data flow by allowing rapid in-core summarization of complex data. However, if *in situ* analysis does not reduce data flow, is it advantageous?

Streaming Heterogeneous In Situ Analysis Modality

Streaming heterogeneous *in situ* analysis modality is a slight variation on the homogeneous approach in which an intelligent I/O subsystem processes bytes as they stream by. The advantage of stream processing is that it involves highly localized processing, which can be more energy efficient because of simpler computational elements and minimal data movement. Stream processing also can be space efficient because it shares minimal node memory (given its streaming nature). The approach is much like Netezza's field programmable gate array-optimized query engine, where the processing is placed next to or in line with the data-storage device.

The advantage is that the specialization offers cost and energy efficiency and minimizes data movement by operating only on data in the data path. The disadvantage is the restricted programming model. Some approaches include dedicating a subset of hardware onboard, or it can be emulated with multicore by dedicating a subset of cores to the stream-processing functionality. From a programming standpoint, it could take advantage of stackable I/O libraries, as demonstrated by the SysV streaming POSIX interfaces.

An intelligent I/O data path (highly programmable I/O data path) could offer a good approach for summarization of Performance Application Programming Interface data for scalable performance monitoring. Anomaly detection in the data stream could act as a kill switch to save resources or as a trigger mechanism for fine-grained fault resilience/recovery mechanisms. Stream processing might also be used to leave "cookie crumbs" (metadata tags) for interesting events that can accelerate later secondary data analysis tasks.

Heterogeneous In Situ Modality

Heterogeneous *in situ* modality is a variant of homogeneous *in situ* modality that defines subsets of nodes with differentiated function. In this situation, the data is not saved to disk but pays the cost of traversing the system interconnect. By isolating data analysis to a mutually exclusive set of nodes, this option mitigates the potential of analysis tasks crashing the main simulation or directly competing for scarce node resources. By creating nodes that are differentiated substantially to improve memory capacity and/or offer different capabilities than the compute nodes, this option also could be supported by the hardware architecture.

To reduce or reorganize data before it is transferred to disk, particular parts of the analysis could be sent through these nodes as a streaming process. For example, a streaming process could be programmed to filter "on the fly" to reduce storage requirements. This approach also can support real-time visualization requirements by preventing overload of off-system network for the path to remote clients. It also can support real-time debugging which is desirable because, when running in batch mode, it is difficult to debug in existing computing facilities.

Again, with this option, the lack of a common data model across application domains (even ones that *should* have a common data model) inhibits development of a common, reusable analysis infrastructure.

Issues With All In Situ Modalities

There are a number of limitations to these *in situ* approaches. First, there are many cases where the analysis domain decomposition does not match the compute domain decomposition. For example, many analysis tasks involve tracking features across multiple time steps or tracking features backwards in time from a specific event. In another example, K-point correlation is unlikely to scale well because it requires a massive amount of global communication.

Integrating analysis with simulation pre-supposes that the analysis objective is known. If this is not the case, integrating analysis with the simulation code may not be of substantial benefit because it eliminates human cognition from the analysis process. In addition, some data-analysis approaches might need data from all of the simulation's time steps. An *in situ* approach cannot move backward in time without accessing data on disk (which obviates its advantages). Because an expensive resource cannot be held hostage by scheduling dedicated time for interactive access, secondary analysis is out of the question for interactive analysis.

Post-Processing Modalities On The Exascale Platform Versus Using Specialized Hardware

For post-processing analysis or visualization, the data is stored to disk or some other I/O system and read back into the system memory for secondary processing in a separate analysis/visualization phase. While I/O bandwidth requirements are not reduced, well-understood methods for interactive analysis can be employed in this approach. Post-processing also addresses the need, in some cases, to perform integrative data analysis across simulations (i.e., simulation comparison) or comparison of simulation and sensor data, neither of which can be accomplished with purely *in situ* or off-line analysis techniques. Archival data storage also is important when data provenance preservation is a priority or when secondary data analysis reveals features that require re-running or enhancing the primary analysis. In spite of arguments

that data is much cheaper to re-compute than store, support for archival storage cannot be completely eliminated from future computing environments.

Post processing might occur either on a separate dedicated resource or the exascale platform itself. Investing in a separate resource for interactive analysis has the advantage that it could prevent interactive tasks from taking the larger machine "hostage" during analysis periods. A separate dedicated resource also would allow the option of designing hardware specialized to the needs of analysis algorithms. However, cost and efficiency considerations may preclude this development. A common baseline design allows for cost efficiencies in hardware development. Creation and support of two divergent architectures would mean more overall expense in initial hardware design, loss of manufacturing cost efficiencies, and additional expense for the ultimate support of two different architectures.

Many of the leading-edge computational facilities (e.g., National Energy Research Scientific Computing Center, Oak Ridge Leadership Computing Facility, and Argonne Leadership Computing Facility) have separate analysis platforms; however this is not the norm for scientific computing systems. Most current facilities do not have separate data-intensive systems. Those that do have separate visualization/analysis clusters (created more for policy issues, not architectural differentiation). The Advanced Simulation and Computing (ASC) program allocated 10% of its hardware budget for resources dedicated to interactive and offline data analysis. This worked because, at the time, visualization and analysis of simulation results at the required scale were under development and required significant debugging and experimentation. Having separate systems for developing and running those new tools provided faster response than if the main ASC systems had been used. In addition, the systems of a decade ago had a different balance between flops, data movement, and memory size than those projected for exascale systems. However, future technology constraints on the cost of memory and energy cost of I/O bandwidth will make it difficult for a separate cluster to have substantially different balance from the main platform.

Co-Design Opportunities Resulting from Data Analysis and Visualization Requirements

Hardware architecture potentially can be modified to better support data analysis and visualization requirements, and data analysis and visualization system implementations may be reformulated to better fit within known hardware constraints of exascale computing platforms. There are very few degrees of freedom available to substantially change the system architecture balance to make it more favorable for I/O or memory-intensive applications. For example, given that memory and I/O bandwidth are primarily power limited, even if *all* of the flops in the system were given up to get better memory or I/O bandwidth balance, the memory bandwidth could be increased by, at most, 20% for the same cost. To obtain a memory capacity that is at parity (as defined by today's standards) with the conceived exascale system, it would cost approximately $100 million.

Given the narrow space for affecting technology parameters related to system balance, the primary opportunity for hardware/software co-design resides in optimization around architectural features (the system organization as opposed to modifying the speeds and feeds of memory and I/O subsystems). A number of ways that system architecture could be modified are discussed in the following sections.

Co-Design Considerations: Hardware Constraints

As there is limited opportunity to trade computational capability for more bandwidth in a power-constrained architecture, the primary design optimization will be to reorganize the storage hierarchy to minimize data movement. Shared (limited bandwidth) and off-system (accessing an archival and/or dedicated analysis system) I/O will be increasingly challenging to support because power consumption is proportional to both bandwidth and length of the wires that connect to the I/O subsystem. Even the protocols for providing consistent state across a shared file system (demanded by POSIX semantics) will be difficult to scale.

Future system designs can incorporate more node local non-volatile random access memory (NVRAM) memory to supplement apparent localized I/O bandwidth and the otherwise limited DRAM capacity on each node. To accommodate this change, considering how to reorganize data analysis algorithms will be necessary.

New Data Analysis and Visualization Algorithms

Multi-resolution processing algorithms are advantageous as they conserve memory space and reduce I/O pressure on machines with constrained I/O. Research is required to understand how to take advantage of highly-hierarchical machine architecture by designing hierarchical approaches for handling multi-resolution data processing.

In situ and stream processing can reduce I/O load and localize computation. In addition, research into better sampling algorithms will be on the critical path for computing summary statistics or localizing data output to "interesting" events more efficiently. One example of this uses *in situ* analysis processing to leave a trail of metadata identifying "interesting events" to facilitate more efficient secondary analysis. By representing high-dimensional data with fewer coefficients, non-linear optimization can enable substantial data reduction. Other data compression techniques, either algorithm-specific or pattern-matching, may play a more prominent role in data reduction because they allow use of prodigiously over-provisioned computing capability to reduce the I/O requirements.

Approximate algorithms also could be important in reducing data capacity and bandwidth requirements. An example is a hierarchical approach for three-point correlation where the point-to-point correlation is not performed at finest scale. Instead, the data is sub-sampled, and the k-point algorithm is performed on lower-resolution data, trading computation for data and bandwidth reduction.

Node local NVRAM can do more I/O operations per second than a conventional disk. One research opportunity entails considering what kinds of data-analysis algorithms can take full advantage of NVRAM performance characteristics. A significant example is out-of-core (OOC) algorithms for data analysis, which should be re-examined. Potentially, OOC algorithms will be much more effective using NVRAM technology because they would not suffer from the poor I/O operations per second performance found in typical disks. If OOC techniques can be sufficiently refined, node-local NVRAM can be a panacea for the lack of DRAM growth.

There is a significant opportunity for co-design interactions between the performance and debugging research communities and the data-analysis community. Recent trends in performance analysis and debugging suggest analysis techniques will play a much larger role in those areas at the exascale

(Bronevetsky et al. 2010; Gamblin et al. 2010; Ahn et al. 2009). For example, performance counter data collected at each node could benefit from data mining for automating the co-design process. It also could change the approach to performance monitoring by exploiting hooks in hardware to perform *in situ* analysis. *In situ* data analysis techniques also could be valuable in anomaly detection for debugging purposes.

Uncertainty Quantification

Recently, UQ has emerged as an essential component in the investigation of complex, multiphysics, multiscale system behavior. UQ is the end-to-end study of the accuracy and reliability of scientific inferences (DOE 2010b).

"End-to-end" means UQ starts by describing (input) error and uncertainty affecting a model, how the effects of those propagate through the model, and how they affect the model's output. The particular details of UQ are quite specific to the application, the kinds of error and uncertainty affecting models, and the goals underlying the use of the models.

UQ is a relatively new field, and as such, there is less experience with and understanding of the methodologies and workloads that UQ will generate in the exascale computing regime. The UQ discussion in this report includes some ideas that have not yet been tested. Nevertheless they are included because UQ goals and results are recognized as extremely important for many application areas. The topic was covered in other reports from the Extreme Scale Computing Grand Challenges workshop series. Among the most comprehensive discussions of UQ are the ones in the reports of the workshop *Science Based Nuclear Energy Systems Enabled by Advanced Modeling and Simulation at the Extreme Scale*[1] and the workshop *Scientific Grand Challenges in National Security: The Role of Computing at the Extreme Scale*.[2] Instead of duplicating the UQ material in those reports, we encourage interested readers to consult those documents.

UQ promises to become more important as high-end computational power increases for the following reasons.

- The scale of computation required to conduct systematic UQ analysis for complex systems will become available.
- There will be increasing ability to use computation to access complex physical systems that are progressively more difficult to understand through physical intuition or experiment.
- Exascale capability promises to increase the ability of computational science and engineering to inform policy and design decisions in situations where substantial resources are involved. The quantified confidence measures that UQ will provide are essential to support these decisions.

Currently, routine use of UQ is hampered by the associated computational costs, which is an issue that is more pressing given that current computational capacity is used largely for producing one-off solutions of complex models. This showcases a need for exascale power for UQ in terms of both capacity and capability. Planning for the exascale platform provides an opportunity for coordination and co-design

[1] http://www.sc.doe.gov/ascr/ProgramDocuments/Docs/SC-NEWorkshopReport.pdf
[2] http://www.sc.doe.gov/ascr/ProgramDocuments/Docs/NNSAGrandChallengesReport.pdf

between the needs of UQ algorithms; algorithms for simulation of complex multiscale/multiphysics models; hardware; and system software.

Current UQ approaches typically perform a huge number of ensemble simulations followed by statistical analysis of results stored on disk. With the increasing cost of data movement, the development of methods that execute computations associated with statistical analysis directly on-chip rather than wrapping statistical computations on the outside of simulations must be developed. In addition, running ensembles on widely distributed or un-integrated resources is inefficient because of the energy cost to perform statistical/data summarization. Thus, UQ would be supported by a high-performance network capable of handling small messages to enable efficient and effective cross-comparisons across ensemble runs. Local NVRAM may be useful to support localized analysis computations for the statistical summarization.

For the following reasons, analyzing and predicting the behavior of complex systems depends increasingly on a fusion of experimental and observational data with modeling and simulation results.

- Experiments are expensive, and observations are limited
- Models present only a partial physical understanding
- Computing high-fidelity solutions of models is expensive
- Many environmental conditions are poorly known or understood
- Often, it is desirable to "extrapolate" beyond known or observed states.

As a consequence, uncertainty and error affect every scientific analysis or prediction.

UQ starts by identifying all sources of error, uncertainty, and variation that may affect a model. The options generally considered are identified below.

- Systematic and stochastic measurement error (epistemic uncertainty)
- Limitations of theoretical and phenomenological models (what can models describe, and what do they fail to describe?)
- Limitations of representations of data and models (how are models and data simplified to be used for analysis?)
- Accuracy of computation and approximation (includes both deterministic and stochastic approximation errors)
- Random phenomena (aleatoric or contingent uncertainty).

Once the sources of uncertainty are identified, computation is used to understand how uncertainty and error propagate through a model. This goal is complicated by the great amount of feedback and interaction between different sources of uncertainty in the solution of a model and is particularly true for multiphysics, multiscale models where the component physics are each contributing particular kinds of

uncertainty and error to the overall system. The final act of UQ analysis is quantifying the effects of uncertainty and error on model analysis and predictions. Describing UQ results in terms that are accessible can be extremely challenging when the dimension of the space describing uncertainty is large and the model under consideration is complex. Additionally, a UQ analysis should be accompanied by inventories of the error sources and uncertainty accounted for in the analysis, (potential) sources not included, and assumptions under which the analysis is performed.

Description of UQ Research Challenges

No one UQ approach or representation will apply to all problems or even to treatment of different components of a single multiphysics, multiscale system. Hence, research across a broad front of UQ-associated topics is necessary. Some of the fundamental research challenges associated with modern UQ are identified below.

The following sections describe these research challenges in more detail.

Representation of Uncertainty and Error

Representing uncertainty and error in ways that are both descriptive and useful for computation is an ongoing research challenge. Research is particularly needed in high-fidelity representations of random quantities (e.g., polynomial chaos and probability distributions). Research also is needed in determining how to combine different representations of uncertainty in one form and the effect of changing scales between representations.

Sensitivity Analysis

Sensitivity analysis is the systematic study of how model inputs—parameters, initial conditions, and boundary conditions—affect key model outputs. There is both a forward problem of propagating variation through a model and an inverse problem in determining the input variation that yields an observable variation in the model's output. Both stochastic and deterministic methods are useful.

Multiphysics, Multiscale Systems

Multiscale, multiphysics systems bring about well-known challenges for accurate modeling and simulation, and these all carry over to UQ. Particular challenges include combining various descriptions of uncertainties from different kinds of physical models and varied data, treating the coupling and feedback between the uncertainties and errors generated in each component, and dealing with the effects of scale on uncertainty representations. Scale differences in behavior of different physical components may lead to scale issues in computing and representing uncertainties.

Model Uncertainty

An especially difficult area of UQ is the treatment of uncertainty in a model and/or the use of phenomenological models. A major difficulty is quantifying what is not represented by the model. In addition, changes to a model may dramatically alter the behavior of solutions. This topic also includes the determination of actual values for data and parameters to use in specific applications of a model, where the data and parameters may not be readily accessible to experiment or observation.

Verification and Validation

Verification and validation have been part of UQ for computational science and engineering for many years. Verification of a computer code is a critical component of a UQ analysis as it provides confidence in the basic scientific platform used for computational science and engineering. In turn, UQ is a critical component of validation because comparison between quantities with uncertainty and error requires knowledge of the uncertainty and error. Traditional validation has expanded to a range of research tasks (described as follows).

UQ for Inverse Problems

UQ arises naturally in inverse problems associated with the prediction of the behavior of a complex system. A prevalent example, depending on the field of application, is called variously "parameter calibration," "model calibration," "data-model fusion," and "data assimilation." The general problem is to adjust data and parameters input into a model to match an imposed or observed output that will minimize some notion of error in the data and parameter values. A challenge for multiscale, multiphysics models is that, often, only observations of functionals of the model's output are available rather than the entire state.

Decision-Making and Optimization Under Uncertainty

Large-scale computational models are a powerful tool for planning and decision-making. These models can be used to assess important questions regarding management of a particular system. Often, such issues are posed as optimization problems. Using models for this purpose requires understanding the uncertainties and errors in predicted behavior.

Detection and Forecasting of Rare Events

Another difficult challenge arises in accurately representing the effects of events that have a very low probability of occurring but yield a significantly large or catastrophic effect when they do.

Dealing with Large Dimension Probability Spaces

This affects both representation of uncertainty and all stochastic sampling (e.g., Monte Carlo) sampling techniques.

Software Support for UQ and Necessary Technologies

Until now, algorithms and methods for UQ, along with enabling technologies typically were developed outside of application codes, which can lead to significant inefficiencies.

UQ/Architecture Co-Design Opportunities

This section considers how hardware architecture could be modified to better support UQ applications and how UQ calculations could be reformulated to better fit within known hardware constraints of exascale computing platforms.

Hardware/Software Co-Design

The proposed exascale architectures favor locality because writing to disk is about 100 times as expensive as summarizing in memory. Therefore, instead of storing intermediate results to disk and summarizing later (i.e., the current practice), UQ would benefit from fully integrated systems that minimize data movement costs by enabling statistical summarization and analysis to be performed across ensemble runs while still in system node memory. A high-performance network capable of handling small messages to enable efficient and effective cross-comparisons across ensemble runs would also be available. Local NVRAM may be useful to support localized analysis computations for the statistical summarization. Support from compilers and programming models to transparently convert conventional scalar arithmetical expressions into arithmetic on statistical distributions, rather than scalar/single-valued quantities, also could be quite useful.

Computations for Uncertainty Carrying Variables

Relative to modern systems, future systems will have severely constrained memory bandwidth. Therefore, strategies that explicitly manage data movement using software-managed on-chip memories rather than caches can greatly reduce pressure on off-chip memory bandwidth. How UQ implementations could be modified to make use of such on-chip scratchpad memories and how these scratchpads should ideally be organized to support UQ applications are pertinent considerations.

Co-Design Example: Inverting the UQ Computational Paradigm

The inference of the sensitivity of a model's prediction to changes in the input data and parameter set is one of the critical computational problems associated with UQ. This may be tackled using stochastic techniques, typically running many different instances of the simulation with statistically perturbed inputs (e.g., ensembles of simulations used for climate). The results are statistically analyzed to describe how the model varies as a function of uncertainties in the input. While there are many specific approaches, modern stochastic techniques applied to complex models demand exascale computing capability to support a huge number ensemble simulations and statistical analysis of the results. Sensitivity analysis also could be tackled by techniques that use derivative information about the model, which generally requires intrusive changes to simulation code as well as raising an enormous computational demand (e.g., requiring the solution of associated adjoint problems). Applying intrusive methods to complex multiphysics, multiscale systems also requires exascale computing capability.

An overarching goal for UQ relative to extreme-scale computing is to "move the statistics as far inside the loops as possible." As such, the typical model of UQ computations could be inverted to use the prodigious on-chip flops to perform computations associated with statistical analysis directly on-chip rather than wrapping statistical computations on the outside of simulations. Instead of loading a scalar value for each parameter or point in the computational space, one would load a parameterized description of the statistical distribution of each scalar variable. The statistics can be expanded into a full table of the statistical distribution on-chip using the prodigious on-chip flops (conserving memory bandwidth) to enable direct computation of a new statistical distribution, which, in turn, can be recompressed as a parameterized representation for the next time step. This reformulation of UQ calculations corresponds much better with the architectural constraints anticipated for future systems, which involve limited memory bandwidth and many more flops on-chip. This approach also enables fewer simulations to be run for gathering accurate UQ statistics. Finally, the approach would benefit greatly from architectural

extensions, such as on-chip scratchpad memories to support expansion of scalar values into tables of statistical distributions.

Discrete Mathematics

Most applications that run on the fastest computers are performing simulations of complex engineering and physical processes. However, discrete or combinatorial algorithms have important roles to play as enablers of traditional applications and as the fundamental algorithms for many emerging fields of national importance.

In the enabling category, discrete algorithms have long been part of high-performance computing environments as tools for improving performance, facilitating higher concurrencies, and empowering larger simulations. Graph and hypergraph partitioning algorithms are widely used to divide data structures and workloads among the processors of a parallel machine. Graph coloring is used to identify parallelism in complex data structures. Graph algorithms are central to sparse direct solvers and some pre-conditioners, and graph-ordering methods have been used to improve memory locality. In approaching the exascale, combinatorial algorithms likely will be an important tool for resiliency, contributing to dynamic re-routing, optimal data duplication strategies, and other areas.

In addition to their ubiquitous supporting role, combinatorial applications also are becoming important as the fundamental algorithms supporting emerging applications in applied energy and national intelligence. Graph algorithms are increasingly central to cyber security and knowledge discovery in large data sets. Combinatorial optimization is the key to many complex scheduling and design problems. Much of the national infrastructure consists of complex, distributed, interconnected systems (used for communication, transportation, energy and water distribution, etc.), and analysis, design, simulation, and optimization of these networks entail enormous combinatorial problems.

Implementing discrete algorithms on exascale architectures will pose even more of a challenge than it does today. Most importantly, the characteristics of discrete problems are quite different from those involved in modeling physical phenomena, so the current high-performance computing environment often is poorly suited to solving combinatorial problems.

- Often, the underlying data dependencies in discrete problems are radically different from those in physical simulations. Because physics occurs in three dimensions, computational structures for physics usually reflect natural locality that comes from the underlying geometric space. This locality enables problems to be partitioned and memory hierarchies to be exploited. Discrete computations that model the internet or social interactions lack this natural locality. Instead, computations can involve seemingly random accesses to global memory addresses. Thus, discrete applications can be difficult to partition and may struggle to make effective use of memory hierarchies.

- Many discrete kernels also have low computational intensity. This is particularly true of graph algorithms. In a typical graph algorithm, an edge list is queried, a neighbor is visited, its edge list is queried, and so on. Most of the time is spent reading lists and following pointers with little computational work to mask the data access time.

- Graph algorithms can have a high degree of parallelism, but it is often fine-grained and asynchronous. Graph algorithms are not naturally bulk synchronous, so they work best with lightweight, one-sided communication and synchronization mechanisms.

- Algorithms for combinatorial optimization often involve exploration of a dynamic search tree within a branch-and-bound algorithmic framework. The nodes of the tree represent sub-problems being considered and can be exponentially large. Moreover, the choice of sub-problems is a critical step, and making clever decisions for choosing the right sub-problems can be computationally expensive. It is worth stressing that branch-and-bound algorithms are amenable to massive parallelism, but not embarrassing parallelism. Branching in a brute-force way leads to a great deal of redundant work. Real parallelism is variable during the course of the run, and the amount of memory and computational power required also varies greatly. Maximizing efficiency requires coordination among tree nodes, and associated communication involves small messages and can be performed in a non-blocking way.

- Simulation on networks is of growing importance. Agent-based simulation, where interacting entities evolve over the course of the simulation, is of particular interest. Memory access patterns of these applications are similar to those of graph algorithms (poor locality), but the models of the agents can be computationally intensive.

- These characteristics combine to generate tremendous difficulty in getting discrete algorithms to perform and scale well with traditional architectures and programming models. However, the community has had notable success with non-traditional approaches, such as the massively multithreaded Cray XMT computer. The XMT provides latency tolerance and an adaptive runtime that supports dynamic parallelism.

- Unlike in many computational science communities, some discrete math codes tend to be small and there is not a large body of legacy software to sustain. As such, the discrete math community will be eager and willing to explore emerging approaches to parallel computing and can serve as early adopters of new ideas.

Discrete Math/Architecture Co-Design Opportunities

Discrete mathematical algorithms and applications exhibit a range of computational characteristics, so any hard-and-fast characterizations will be necessarily incomplete. However, a variety of themes and characteristics commonly recur, and several of these are relevant to architecture co-design.

- Graphs are a common and powerful abstraction, and graph algorithms have distinctive features. Generally, graph algorithms involve following sequences of pointers with very little computational work. Many graphs relevant to exascale computing are associated with meshes, and these graphs will inherit some locality structure from the underlying geometric space. Yet, graphs that arise from different applications, such as cyber security or social networks, have limited exploitable structure. For these graphs, algorithms will be making frequent visits to essentially random locations in global memory.

- Among their many uses, graph algorithms can improve the efficiency of many scientific applications. Examples include exploiting structure in sparse direct solvers, identifying potential parallelism

through graph coloring, and partitioning or repartitioning work and data across the cores of a parallel computer. For such tools to be effective, they must run quickly enough to improve the overall application runtime.

- Often, combinatorial optimization problems are solved by some kind of branch-and-bound technique. These techniques involve the dynamic construction and exploration of a search tree. The tree's leaf nodes require computationally intensive optimization solutions with quite variable memory and computational demands. In parallel, these calculations require significant bookkeeping and are difficult to load balance.

In many of these settings, the underlying combinatorial object (e.g., the graph) can be rather difficult to partition effectively. Thus, a programming model that supports a GAS is highly advantageous. In addition, many combinatorial algorithms have a high degree of fine-grained parallelism, but the available parallelism is dynamic. Generally, parallel algorithms for these problems are not naturally bulk synchronous.

Many discrete math applications are good candidates for co-design because they are relatively agile in comparison to the large multiphysics applications and the community is willing to experiment with new approaches. Discrete math codes will challenge the memory subsystem and interconnect messaging rates.

Architectural features that can assist with these computations include the following:

- Ability to support a GAS
- Lightweight, one-sided communication for small messages
- Lightweight, fine-grained, and very flexible synchronization mechanisms
- Ability to turn off the memory hierarchy for accesses that cannot make good use of it
- High degree of threading to tolerate the inevitable latency in some discrete applications.

Key research challenges include the following:

- Can discrete algorithms be expressed well via message-driven computing models, and how should they be implemented in hardware?
- What memory consistency models are necessary to support discrete algorithms?
- What are the right load-balancing abstractions for exascale machines?

CROSSCUTTING CHALLENGES AND RESEARCH NEEDS IN ALGORITHM AND MODEL RESEARCH AND DEVELOPMENT NEEDED TO SUPPORT NEW ARCHITECTURES

The challenges identified for mathematical models and algorithms are summarized in the following priority research directions (PRDs):

PRD 1.1: **Re-cast critical applied mathematics algorithms to reflect impact of anticipated macro-architecture evolution, such as memory and communication constraints.**

- Develop new PDE discretizations that reflect the shift from flop-constrained to memory-constrained hardware by computing more for a given level of communication.

- Because growth in parallelism will be on-chip where latency of the all-gather operation is low, developing solvers that are scalable within a node should be possible. However, focus on recasting linear, nonlinear solvers, eigensolvers, FFTs, and optimizing algorithms for exascale architectures.

- Develop algorithms that exploit variable-precision arithmetic approaches for increased efficiency. For example, some algorithms can use a reduced-precision arithmetic approach (requiring less storage) without loss of accuracy, while other algorithms (e.g., very large inner product calculations) may require higher-precision arithmetic approach to maintain overall computation accuracy.

PRD 1.2: **Take advantage of exascale architectural evolution to design new algorithms for uncertainty quantification and discrete mathematics**

- Because simulation codes may need to be substantially rewritten for exascale computers, there is an opportunity for redesign that includes embedded techniques and tools to carry out UQ. For example, statistics can increasingly be moved inside computational simulation algorithms.

- Develop new approaches for algorithms that are memory and communication intensive, such as discrete mathematics algorithms.

PRD 1.3: **Develop new mathematical models and formulations that effectively exploit anticipated exascale hardware architectures.**

- Simply re-casting algorithms may be insufficient to leverage exascale architectures completely. New mathematical models of physical processes may be required. This challenge will be supported by a significant co-design approach involving scientists who understand the domain science and applied mathematicians and computer scientists who can help translate the domain science into mathematical models that are computable at the exascale.

PRD 1.4: **Address numerical-analysis questions associated with moving away from bulk-synchronous programs to multi-task approaches.**

- With very large processor counts, expect an evolution toward simulation codes that are more multiphysics or multiscale in nature, with different processes executing on different parts of the hardware. Thus, research in the following areas will be required:

  - Quantification of accuracy and stability for methods that couple different processes, possibly across different kinds of discretizations and levels of discretization resolution
  - Development of high-order operator splitting and decomposition methods for coupling across space and time
  - Methods for coupling across a range of scales.
- Develop theory and algorithms that apply operators more asynchronously (move away from bulk-synchronous algorithms).

PRD 1.5: **Adapt data-analysis algorithms to exascale environments.**

- Reorganize data-analysis algorithms to leverage increased node-local, NVRAM availability.
- Rewrite data-analysis algorithms to leverage GAS.
- Develop analysis techniques for streaming data.
- Develop analysis methods that can use reduced-precision arithmetic for performance without compromising accuracy.
- Conduct research to determine the best overall approach for performing data analysis at the exascale, including:
  - *In situ* (part of simulation code) analysis
  - Post-processing on the exascale platform
  - Dedicated separate platform for post-processing—can the I/O bottleneck be avoided?
- Conduct research on the development of common data structures or data access APIs across science application space to improve reuse of data analysis software.

PRD 1.6: **Extract essential elements of critical science applications as "mini-applications" that hardware and system software designers can use to understand computational requirements.**

- "Mini-applications" will expose critical performance issues to architecture and system software designers in the co-design process.

PRD 1.7: **Develop tools to simulate emerging architectures and performance modeling methods for use in co-design**

- Develop predictive capabilities that emulate node architectures to test new algorithms, possibly using discrete-event simulation approaches.

PROGRAMMING MODELS AND ENVIRONMENTS

For the past decade, existing programming environments for writing parallel scientific applications have sustained high-performance computing (HPC) application software development and have been successful for petascale computing. However, they were architecturally designed for coarse-grained concurrency largely dominated by bulk-synchronous algorithms. Future hardware trends will likely require a mass migration to new algorithms and software architectures that support hierarchies of parallelism and dynamic behavior, whether it is used to address power management, resiliency, or application requirements. This change will be as broad and disruptive as the migration from vector to massively parallel (MPP) computing systems that occurred 15 years ago. Moreover, the applications and algorithms will need to rely increasingly on fine-grained parallelism, strong scaling, and support fault resilience.

The interface between applications and architectures (Figure 4) programming environments play a pivotal role in making exascale applications efficient, resilient, and productive on exascale architectures. To address these challenges, there is a renewed opportunity to introduce a higher level of software engineering into current computational science applications that will enhance the modularity, portability, and performance of codes while extending their scientific capabilities. Simultaneously, past investments must be protected, and a migration path from current to future environments must be embraced.

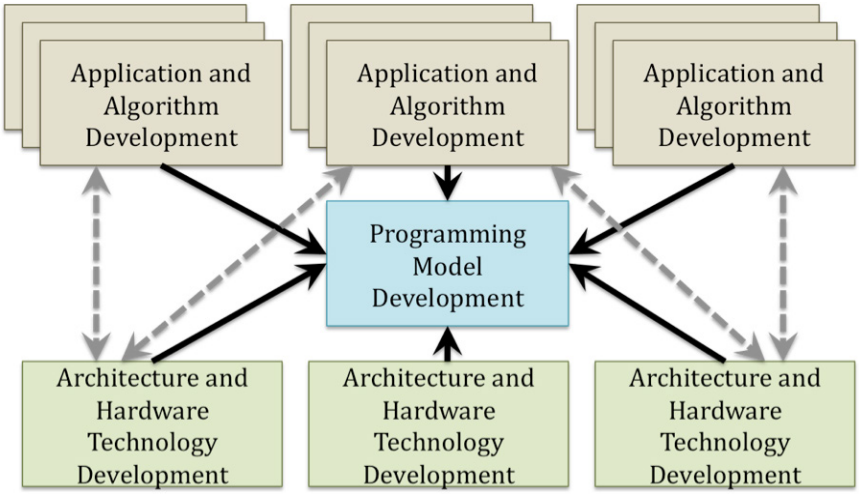


Figure 4. Structure of programming model activity relative to the hardware, application, and algorithms activities. The programming model must be responsive to these other activities to serve its purpose successfully. Image courtesy of Katherine Yelick (Lawrence Berkeley National Laboratory, University of California at Berkeley).

Outstanding questions for the programming models research community include the following:

- Will the planned programming models provide suitable abstractions and tools for a spectrum of exascale applications? If not, how should any plans be modified?
- What computer science research and development (R&D) is needed to provide the necessary functionality?

- What R&D in algorithms needs to be performed to use the programming models effectively?

UNDERSTANDING PROGRAMMING MODEL REQUIREMENTS AT THE EXASCALE IN THE CONTEXT OF CRITICAL MATHEMATICAL MODELS, METHODS, AND ALGORITHMS

Partial Differential Equations

Typically, partial differential equations (PDE) solvers are characterized by using domain decomposition with nearest-neighbor communication patterns. Elliptic PDE solvers require smaller, nonlocal communication patterns, and multilevel solvers with finer grids tend to create smaller groups of communicating processors that may be scattered across the machine. The communication characteristics in terms of frequency and message size also vary, from large, frequent nearest-neighbor messages to smaller, less frequent messages to farthest neighbors. Solvers also require frequent, small global reductions—perhaps as many as several thousand per time step. All of these characteristics place incredible demands on the network to deliver performance in terms of low latency, high bandwidth, and high message rate for point-to-point and collective communication operations. To optimize communication performance by reducing network contention, it is highly desirable that the logical layout of the communication patterns, from the application perspective, match the physical topology of the machine as closely as possible.

The number of network endpoints in an exascale system is expected to be several orders of magnitude larger than today's most powerful machines. The network topology will certainly be subject to dynamic reconfiguration as a result of component failures in the system. As such, it will be necessary to explore approaches and mechanisms that allow for the application and system software to work together to maximize the network's performance potential. Possible approaches include programming interfaces that allow the system software to convey information about underlying network topology to the application so the application can structure communication appropriately, and, conversely, providing interfaces that allow the application to inform system software about the application's communication structure so communicating elements can be placed and scheduled optimally. In the latter approach, mechanisms that support moving the computation to the data when most efficient also need to be explored.

In general, PDE solvers also have an abundance of fine-grain, node-level parallelism that can be exploited. A hierarchical machine model with two-level parallelism is appropriate for both structured and unstructured grids, as well as AMR and linear solvers. System software that could exploit hints from the application regarding spatial and temporal locality could be extremely beneficial. Cache memory is energy expensive, so alternative approaches to providing fast local memory access, such as scratchpad memory, can potentially offer a substantial performance improvement and energy savings provided there is system software support from the programming environment and the associated runtime system to supply effective resource allocation and management strategies.

Given the abundance of node-level parallelism, a low-cost mechanism for creating and destroying threads will be critical for PDE-solver performance at the exascale. Significant investigation is needed to determine the most appropriate threading model. System software will play a crucial role in providing the infrastructure necessary to support a global namespace where threads are first-class objects that are named and manipulated directly by the application. Potentially, creating more threads than available processing elements is a strategy that will work well for PDE solvers and provide the system software an opportunity

to hide memory latency and be more effective at scheduling processing resources. The dynamic nature of computation for solvers will require a highly dynamic threading model. In addition, new strategies in system software for creation and management of large numbers of dynamic threads will be necessary.

As with existing petascale platforms, ensuring the correctness of and debugging problems with applications at the exascale will continue to be a fundamental productivity challenge to application development. This debugging challenge will be further compounded by the impact of faults on the system because application errors will be indistinguishable from errors generated by the system. Current strategies that analyze the state of processes to determine equivalence and identify outliers can potentially be effective. However, given the expected dynamic and asynchronous nature of systems and applications, such an approach may be ineffective. New approaches to debugging will be critical, not only to recognize application-generated errors but to help identify and isolate system-generated errors. Potential approaches to error identification and isolation include support for applications that allow the system software to check the validity of data flowing into and out of computational routines, as well as mechanisms that will permit computing of the same function multiple times to determine the causality of an error.

There are many fundamental research questions with respect to data analysis and input/output (I/O) for PDE solvers on an exascale system. Provided there is system software support to manage memory and computational resources effectively, new approaches that perform data analysis simultaneously with the calculation can potentially be effective. On-the-fly analysis may make effective use of the abundance of compute cycles available. The insertion of nonvolatile storage into the memory hierarchy also will play a key role in data analysis. Capturing analysis data as part of a regular application checkpoint may be possible, and the potential for using nonvolatile storage for background data staging is attractive. Further investigation into the appropriate resource management techniques for managing this extra layer in the memory will be needed.

Currently, solvers are taking advantage of multiprecision algorithms that use reduced precision to increase computational performance and reduce stress on the memory subsystem. This approach likely will continue to be effective on future-generation platforms. The shift to simplified computing elements should not have a negative impact on solvers. For example, these computing elements must have some baseline functionality, such as branches (or conditionals), to support the needs of applications on an exascale system. From a system software perspective, managing heterogeneous resources based on the potentially dynamic needs of the application provides a significant challenge and opportunity for co-design.

Observations and Recommendations

1. This community currently is in transition from using a separate message passing interface (MPI) process on each core to employing hybrid approaches. Thus far, MPI+OpenMP has had some success, particularly in fusion particle codes, strong-scaling climate codes, and in AMR-based codes for astrophysics and combustion. Graphics processing units (GPUs) have been successfully exploited with MPI+Computer Unified Device Architecture (CUDA), but the CUDA programming model is cumbersome and unlikely to become the basis for long-term scientific code development. Needed is a programming model that hides much of the architectural complexity of clusters of heterogeneous, multicore nodes.

2. There is also interest in global view languages (Chapel, UPC, Co-Array Fortran) on multicore nodes used in a hybrid approach with MPI.

3. Fault tolerance is a major issue. Two efforts—the Coordinated Infrastructure for Fault Tolerant Systems (CIFTS) Scientific Discovery through Advanced Computing (SciDAC) project's "fault-tolerant backplane" approach and MPI 3.0 Fault Tolerance Working Group—remain research projects at this point. In the worst-case scenario, tools that automatically identify and minimize the checkpoint state of PDE applications will be necessary.

4. Tools are needed to understand and tune integrated, hybrid programs. Some efforts to encourage tools development are being conducted by the tools working group of the MPI-3 Forum.

5. Support is needed for querying as in other applications for exascale systems, support for cost querying would enable trade-offs. As in other exascale systems applications, support is needed for routines that can query the relative costs of various operations on the target hardware; e.g., how much it will cost to access this variable or type of memory, or how fast can this processor type perform this computation relative to another one. This query support would allow codes to be written in a way that they can make space versus time tradeoffs in an informed and flexible way from one machine or execution to the next. These queries could be static in the sense that they could make queries about the target architecture, such as clock speed; or they could be dynamic, in which case the results might make queries about variables whose location may not be known at compile time or might reflect aspects of system load that may vary from one point in time to another. This is a potential co-design opportunity between the applications groups, the programming models groups, and the hardware architects.

6. Power-aware programming models are a speculative concept. It is conceivable a programming system could allow a programmer to specify when peak computing power, network usage, or I/O capability is not needed during some part of a computation, allowing the operating system to deschedule certain resources to conserve power. However, this could lead to complexity in the programming model, but it may be necessary for efficient use of exascale resources.

7. There are significant co-design opportunities. For example, applications will define data structures appropriate for application parallelism, but compilers and other software need to know usage patterns so they can optimize the low-level operations that need not be the concern of the applications. The design of an application program interface that allows applications to communicate usage patterns to compilers and libraries requires a cooperative design effort involving both applications developers and system software developers. In turn, the system software developers need to work with hardware architects to produce architectural features that will enable significant optimizations.

Solvers and Optimization

For Solvers and Optimization, three specific areas in relation to programming environments were identified: mini-applications, data-dependent parallelism support, and the need for a new abstract machine model.

Mini-Applications

In previous efforts, these types of codes have been called compact applications or synthetic applications, so the idea is not new. The difference is that application teams should be the primary developers of the mini-applications, with the idea that a mini-app provides a concrete collaboration point in co-design by exposing a critical performance area of the full-scale application, or even a family of applications. Mini-apps do not have to be working models. Instead, they need to correlate to performance issues, so that mini-app performance in various settings (including if rewritten) can be correlated back to the real application. Mini-apps are not a replacement for the real application, but provide a working code far upstream in the design process where it is infeasible to use the real application.

Data-Dependent Parallelism Support

A substantial amount of fine-grain parallelism is expressible in terms of data dependency diagrams such as directed acyclic graphs (DAGs). Efficient execution of DAGs is challenging, and there is no uniform application programming interface (API) for forming and executing through a DAG. The PLASMA[1] project is developing DAG capabilities for dense linear algebra, and intends to make its interfaces available to other developers. However, it would be useful to have general programming environment support for this kind of execution model.

Abstract Machine Model

There is general consensus that the entire community needs a new abstract machine model, especially one that exposes the performance-impacting design parameters an algorithm developer must use. This model is an especially important co-design opportunity, given the design of the programming environment must expose the architecture's execution model to the application efficiently and productively.

Data Analysis and Visualization

There are many varieties of data analysis, and they do not necessarily share the same computational characteristics. Thus, it is not always possible to generalize characterizations of data analysis algorithms.

One concern about data analysis at the exascale is that the machine scale and complexity, and expected code complexity are such that it can be difficult to distinguish between a bug and a "potential new scientific discovery for which no model has yet been developed." For this reason, the ability to validate analyses is important. This can be extremely difficult for areas of science in which the model cannot be empirically tested, such as simulations of future behaviors over long time scales. In addition, for scientific simulations with the potential to affect public policy, it is important to create a provenance that will back up scientific claims and permit others to reproduce and verify conclusions. For all of these reasons, one useful programming environment concept should have **analysis tools automatically capture intermediate results between their distinct stages**. This would provide a valuable means of performing verification, validation, and retracing after the fact. It could serve as provenance and permit scientists to double-check themselves or document how they arrived at a particular state. However, given the potentially large sizes of these states to be recorded, it is important to note that creating such artifacts along the way could add a significant I/O burden.

[1] Parallel Linear Algebra for Scalable Multi-core Architectures (PLASMA) Project: http://icl.cs.utk.edu/plasma/

There are two abstractions that could be useful for programming models. The first is **support for natural data abstractions**; i.e., abstractions that closely match the scientific abstractions rather than systems-oriented concepts. For example, scientists and data analysts think in terms of multidimensional variables with physical units associated with them rather than files, data linearizations, and data formats. This also should help reduce the frequency of coding errors because the higher-level concepts embodied in the abstractions would presumably be closer to the natural terms for the programmer and expressible with fewer lines of code. Of course, supplying such higher-level concepts requires the concepts themselves be implemented correctly and validated. Moreover, in the specific case of exascale, there is a concern that for certain analyses, it would be crucial for the abstractions to work consistently when executed across different processor types. For example, for an analysis that is sensitive to numerical error, if the CPU generated a different result than the GPU, those differences in results across architectures should be automatically detected and highlighted for the user.

The second abstraction of interest is the **ability to run parallel tasks locally on the node that owns a particular piece of data**. This suggests programming models that support richer and more dynamic execution models than current single program multiple data (SPMD)/bulk synchronous parallel (BSP) models. High-productivity computing systems (HPCS) languages support this concept. In Chapel, this pattern could be written as follows:

```
on A(i) do begin myAnalysis(A(i), B, epsilon).
```

This says "on whichever node owns array element `A(i)`, start a parallel task to execute the `myAnalysis()` function call shown here while the original task continues executing." Removing the `begin` statement would cause the execution to be synchronous, and the originating task to block until the remote invocation of `myAnalysis()` had completed and returned. Chapel's `on` -clauses can take any expression that has storage associated with it, causing the computation that follows to be run on the node owning that storage. If file/disk-based data are represented using a similar syntax as data structures in memory, the same concept could be used to express active storage operations in which computation takes place near the disk in question. For example, if the aforementioned array `A` was considered to describe a file's data, the statement could be interpreted as calling the `analysis()` on the CPUs that are closest to the disk in question; i.e., potentially the CPUs of a (noncompute) dedicated I/O node.

Data analysts use a variety of special-purpose languages and tools (special purpose in the sense they are not necessarily widely used within HPC applications programming). Examples include R, Weka, IDL, SPSS, Scout, SAS, and Kepler (these are not necessarily equivalent tools that provide similar functionality). The data analysis community would find it attractive to run these tools and languages in parallel, either on smaller petascale extreme clusters used for secondary analysis or, ideally, on the exascale hardware itself. Some of these tools wrap standard library solvers that could benefit from algorithmic advances, such as those Jack Dongarra (University of Tennessee/Oak Ridge National Laboratory) described in his plenary talk on the first day[1] of the "Scientific Grand Challenges - Crosscutting Technologies for Computing at the Exascale" workshop. From a programming models perspective, these tools and languages should be studied and their value proposition understood, with the

[1] Dongarra J. 2010. "Impact of Architecture and Technology for Extreme Scale on Software and Algorithm Design." Presented at the "Scientific Grand Challenges - Crosscutting Technologies for Computing at the Exascale," February 2-4, 2010, Washington, D.C. Available at http://extremecomputing.labworks.org/crosscut/presentations.stm.

goal of seeing whether or not their attractive features and concepts could be cleanly integrated into general HPC programming models.

One reason this last point makes sense is that data analysts also use standard HPC programming models, such as C+MPI, to perform their computations. Thus, being able to subsume the features or desirable characteristics of the special-purpose languages and tools would reduce the number of languages that users would need to learn or support. In discussing data analysis, it was clear that analysts wrestle with many of the same problems as standard HPC programmers: load balancing, software engineering, validation, testing, etc. They also have some less common challenges, such as having workflows in which data sets flow through analysis stages, some of which could be done in parallel. They also have the issue of generating a provenance (previously described) and a greater need for interactivity to perform computational steering and discovery. It would be worthwhile to explore which programming language concepts and abstractions could better support these novel features.

Due to the huge volume of data on exascale machines, it will be crucial to support forms of *in situ* or on-the-fly analysis. That is, researchers can perform some processing on the exascale machine to take advantage of the computational resources, interleave analysis with computation, and reduce the data volume for storage purposes. This indicates that good support is needed for scalable implementations of several common data transforms. Examples include transposes (e.g., changing an array of records into a record of arrays) and creating indexes for exascale data.

One observation—given the massive amount of performance counter data that will be generated by exascale machines—is that traditional roles will reverse. In addition to computer science supporting data analysis as it traditionally has done, it will also become more of a first-class client of data analysis because performance data must be digested and reduced, using techniques like statistical clustering, to not overwhelm the human. The digested information may be fed back into the application and/or runtime and/or compiler to steer, tune, abort, or otherwise take a responsive action within the computation.

The map-reduce programming model, recently popularized by Google and Yahoo!®[1], is an interesting option to consider. The map-reduce model appears to be too restrictive to be a crucial tool for exascale applications in that its parallel activities have to be independent, whereas the exascale mission-critical applications are not so embarrassingly parallel. However, it may be worth further investigation. Even if the map-reduce model does not come into general use at the exascale, the properties that have made it successful—support for very general reduction operations, fault tolerance through redundant storage and resilient master-worker paradigms, and good disk/file storage strategies—may be important concepts and characteristics to support in future HPC programming models.

Traditionally, data analysis and visualization have been relegated to the "10% of the whole machine cost" rule. Nevertheless, with the advent of exascale computing capabilities and experimental facilities of unprecedented resolution, data analysis and visualization are likely to be employed to address deep scientific questions, which in turn, may justify increasing the resources allocated to them. The 10% rule, however, also results in the fact it may be more difficult to identify limiting factors that data analysis will impose for certain system software components. For example, fault tolerance may not be a major issue if data analysis and visualization are performed on 1/10-size machines. The prediction is that *in situ* data analysis and visualization need to use the same checkpoint-restart mechanism, whereas not-*in situ* data

[1] Yahoo! is a registered trademark of Yahoo, Inc.

analysis and visualization will be constrained by bandwidth to run on a smaller scale for which fault-tolerance is less critical than for others' applications.

Conversely, data analysis and visualization needs and aims open the possibility of massive use of *in situ* and near I/O analysis, which results in a unique set of questions for systems software and programming models. For *in situ* analysis, what are the programming models and operating system (OS) mechanisms that separate the analysis from the simulation in a way that makes sense? This can be done either as part of the same computation or by using a shared address space mapping. For a near I/O data analysis approach, the computation does not necessarily operate in the same cluster, but it operates on the same cluster storage. Some issues to be solved include performance isolation (what the visualization does should not cause simulation to slow down) and partitioning visualization to execute near the data.

In addition, the projected massive use of nonvolatile random access memory (NVRAM), with its two orders of magnitude superior performance compared to disk, offers the unprecedented opportunity of it being exposed by the system software for data analysis that does not need sequential access. Such cases appear, for example, when correlation information is computed hierarchically across multiple nodes for only a small subset of the data resident on the NVRAM. Because the primary intent of NVRAM use is check pointing, appropriate programming models and services provided by the OS must be identified while also providing for checkpoint success.

Developing exascale data formats that allow for better use of the bounded I/O bandwidth is an immediate need. One approach is to use database technology approaches, such as columnar store and highly compressed data indexing (the latter possibly computed *in situ*). This will allow for efficient access to the output of multiphysics simulations when accessing only individual variables, as well as higher compression ratios due to better scaling compatibility. It should be possible to accomplish in a way that allows the user to still see this as a standard format, such as network Common Data Form (NetCDF).

Co-design will play an important role for the interaction between data analysis and visualization. Better data models are a fundamental issue for both system software and data analysis because specialized data formats may highly reduce extensibility. A possible solution is using object models for data, which raises several challenges, such as performing I/O on objects, having application-relevant data models, and mapping them efficiently onto exascale storage. Getting this correct would greatly simplify the caching of relevant attributes and increase the efficiency of storage systems. Seamless memory/storage is another joint-interest topic because, depending on the size of data, the analysis can either be performed in the random-access memory (RAM) of the visualization nodes or the computation can be moved to storage if it does not fit in RAM.

Appropriate management of this situation is both a programming model challenge and system software challenge.

Uncertainty Quantification

Given that large-scale simulations will be increasingly used to inform public policy, uncertainty quantification (UQ) will play an important role in exascale computing. Moreover, many UQ computations are themselves potential exascale computations as they use similarly compute-intensive processes to generate results as the simulations they are analyzing. The disruptive nature of moving to exascale actually could be empowering for the UQ field because programming models already will be

under reconsideration and application codes revisited. This provides an opportunity for UQ to become a more active part of the field's standard operating procedure rather than an afterthought.

Special Type Qualifiers for Uncertainty Quantification

One concept discussed at the "Scientific Grand Challenges - Crosscutting Technologies for Computing at the Exascale," workshop for supporting UQ within programming models is the notion of having a special-type qualifier within programming languages to identify "uncertainty-carrying variables." These are variables whose values are identified as being uncertain and would carry along some measure of that uncertainty in addition to their traditional value. As uncertainty-carrying variables are combined, their measures of uncertainty would be combined. A slight variation on this would be to support a means of overlaying such UQs on preexisting code without modifying it (e.g., via an annotation overlay file) to apply UQ analysis in a nonintrusive manner to preexisting codes. The challenge to this concept is, because of the relative youth of the UQ field, there currently is no single agreed-upon standard for representing these uncertainty factors. For example, some UQ representations use intervals, while others use stochastic distributions. Even when two approaches use the same representation, such as intervals, they may define the semantics on these concepts in different ways. This suggests that if a language did support uncertainty-carrying variables, they should probably support an opaque auxiliary data structure that could be specified orthogonally to the computation such that each UQ practitioner could plug in a representation and semantics that matched their philosophy. It is likely that a calculus of distributions of uncertainties needs to be developed and implemented in a library or to develop software engineering practices that embrace it.

Support of Interval Types

A related side topic was whether programming languages should support interval types. In the discussions during the workshop, such support was generally considered to be a reasonable idea because intervals are a fairly well understood and commonly used concept. However, doing so raises the issue of where to draw the line because intervals are just one of several UQ-appropriate representations. In addition, a mathematically sound basis needs to be developed for defining and combining interval types; otherwise, the result would not be useful for UQ.

Dynamic Execution Models

UQ is among the application areas that have a need for more dynamic execution models than SPMD/BSP styles. UQ would benefit from workflow-based execution models, DAG-style task scheduling, dynamic spawning of new simulations, and efficient replication of tasks. Similarly, UQ could itself benefit from research into development of increasingly asynchronous algorithms for UQ that avoid bulk synchronizations during analysis by delaying and reducing the frequency of synchronization. A motivating example can be found in variance reduction in Monte Carlo sampling.

Fault-Tolerant MPI-Collectives

It was noted that UQ would benefit from fault-tolerant MPI collectives. For example, if a task dies before entering a barrier, it is reasonable for the rest of the tasks to complete the barrier and proceed rather than deadlocking the program or requiring it to be torn down.

The UQ community believes it would benefit from hardware that supported an additional network for control and collective-style communications in addition to the main network for data transfers. The programming model would need to expose these multiple networks to the user so the appropriate one could be chosen.

Some UQ techniques rely on source-to-source transformation methods. These techniques would benefit from **compilers and libraries that support an open-source infrastructure** because several of the analyses the techniques use are similar to those supported by traditional compiler optimizations. Obviously, it would be preferable to reuse these analyses for a given language/compiler than to reinvent them on a case-by-case basis. Another wish-list item for these techniques is programming model support for data structure linearization in which an arbitrarily complex data structure can be linearized—i.e., written sequentially according to some conventions—to targets such as a file, memory buffer, or a network socket, thus providing much more efficient data movement.

Adjoint computations users, a specific subset of UQ, wanted some additional capabilities from the programming model, such as a **mechanism for recording the control flow** taken during the execution of a program that could be reversed after the fact. A similar capability would be a dataflow reversal mechanism where stores are converted to loads, sends to receives, etc.

Workshop attendees discussed a concept in which a user could define a metric that should be evaluated throughout the computation of a large program, and, upon exceeding some threshold, that code region would be identified as being a region where UQ was needed.

Discrete Math

Discrete mathematical algorithms and applications exhibit a range of computational characteristics so any rigid characterizations are necessarily incomplete. However, a variety of themes and characteristics commonly recur, and several of these are relevant to programming models: graph data structures, graph algorithms, and combinatorial optimization algorithms.

Graphs and Graph Algorithms

Graphs are a common and powerful abstraction, and graph algorithms have distinctive features. Generally, graph algorithms involve following sequences of pointers with very little computational work. Many graphs relevant to exascale computing are associated with meshes, and these graphs will inherit some locality structure from the underlying geometric space. However, graphs that arise from different applications, such as cyber security or social networks, have especially limited exploitable structure. For such graphs, algorithms will be making frequent visits to essentially random locations in global memory.

Among their many uses, graph algorithms are used to improve the efficiency of many scientific applications, including exploiting structure in sparse direct solvers, identifying potential parallelism through graph coloring, and partitioning or repartitioning work and data across the cores of a parallel computer. For these tools to be effective, they must run quickly enough to improve the overall application runtime.

Combinatorial Optimization Problems

Often, combinatorial optimization problems are solved by some kind of branch-and-bound technique, involving the dynamic construction and exploration of a search tree. The tree's leaf nodes require computationally intensive optimization solutions with quite variable memory and computational demands. In parallel, these calculations require significant bookkeeping and are difficult to load balance.

In many of these settings, the underlying combinatorial object (e.g., the graph) can be difficult to partition effectively. Thus, a programming model that supports a global address space is highly advantageous. In addition, many combinatorial algorithms have a high degree of fine-grained parallelism, but the available parallelism is dynamic. Generally, parallel algorithms for these problems are not naturally bulk synchronous, necessitating programming support for task parallelism with load balancing.

In this context, programming language features that can assist with these computations include the following: support for a global address space; lightweight, one-sided communication for small messages; lightweight, fine-grained synchronization; and support for maintaining logical state even when objects are migrated between processors. Key research challenges include the following:

- How can memory performance be maximized for applications that lack natural locality?
- Instead of focusing only on computational load, how can memory/network/power load be balanced across processors?
- How can discrete applications make use of language features supporting the dynamic control of computational resources?

Co-design opportunities include collaboration between application developers and language/library implementers to confirm discrete mathematical algorithms are provided for in exascale hardware and software development.

CROSSCUTTING CHALLENGES AND RESEARCH NEEDS FOR PROGRAMMING MODELS TO SUPPORT EXASCALE COMPUTING

The challenges identified for programming models to support exascale computing are summarized in the following PRDs:

PRD 2.1: **Investigate and develop new exascale programming paradigms to support "billion-way" concurrency.**

- Investigate hybrid programming models (different internode and intranode programming models)
  - A likely solution will be MPI+X (hybrid) programming model: MPI internode programming model, X = an intranode programming model, where X could be MPI, OpenMP, PThreads, CUDA, etc.

- Develop effective abstractions to expose loop-level and data-level parallelism for computation on subdomains, including relatively simple approaches for specifying fine-grained parallelism that can be translated into lightweight parallel implementations.

- Take advantage of programming model changes to design new (e.g., intrusive) UQ techniques.

- Develop programming model support that allows using multiple networks within a single application (e.g., network for data transfer/network for control and collective-style communication).

- Develop mechanisms for specifying idealized concurrency and hierarchical affinity.

PRD 2.2: **Develop tools and runtime systems for dynamic resource management.**

- Develop language support for dynamic control of system resources in response to performance, energy efficiency, and resiliency hints.

- Develop methods to record control flow of execution and data so flow can be reversed in adjoint methods.

- Develop load-balancing abstractions.

- Provide support for dynamic load balancing with increased concurrency.

PRD 2.3: **Develop programming models that support memory management on exascale architectures.**

- Provide partitioned global address space (PGAS) language support.

- Develop programming models that address nonuniform memory access characteristics on nodes by giving more direct memory access control to the programmer.

- Develop rich memory consistency models that reflect the requirements of various applications and support for expressing memory locality issues.

- Develop memory management models that support alternative memory hierarchies based on new technologies, such as NVRAM.

PRD 2.4: **Develop new scalable approaches for I/O on exascale architectures.**

- Research a possible move to database techniques for I/O; e.g., object models for data.

- Develop new I/O models and tools that target alternative storage hierarchies based on new technologies, such as NVRAM.

- Provide programming model support for "data structure linearization"; i.e., mapping complex data structures to file systems, buffers, network sockets, etc.

- Develop extensions to programming models that support *in situ* data analysis, rather than traditional post-processing workflows.

PRD 2.5: **Provide interoperability tools to support the incremental transition of critical legacy science application codes to an exascale programming environment.**

- Interoperability with old programming models/languages will be required to promote adoption of the new programming models/paradigms (PDE). This may be provided through tools to support interoperability between languages or cross-compilers that translate from "old" to "new" languages.

PRD 2.6: **Develop language support for programming environments at the exascale.**

- Continue research on new languages to support evolving computer architectures.

- Investigate domain-specific languages as a mechanism to provide portability and high performance for particular application capabilities.

- Consider language support for "uncertainty carrying" variables; e.g., typing qualifiers.

PRD 2.7: **Develop programming model support for latency management.**

- Provide language support for asynchronous latency-tolerant algorithms.

- Provide automated capability to overlap computing, analysis, communication, and I/O.

PRD 2.8: **Develop programming model support for fault tolerance/resilience.**

- Develop mechanisms for specifying hints for resiliency and power management to compilers and runtime systems.

- Provide programming model support for fault management.

- Develop fault-tolerant MPI collectives.

PRD 2.9: **Develop integrated tools to support application performance and correctness.**

- Develop analysis and engineering tools that integrate performance and correctness information from multiple sources into a single, application-oriented view.

- Co-design appropriate hooks in hardware, system software, and programming models that will provide requisite performance and correctness information to integrated tools.

- Design new techniques that address the increasing complexity of performance and correctness analysis caused by the factors of scale, software complexity, application sophistication, and hardware complexity.

PRD 2.10: **Develop a new abstract machine model that exposes the performance-impacting design parameters of possible exascale architectures.**

- Develop new abstract machine models to provide new algorithm and software systems designers information on the expected hardware targets in terms of scale, bandwidths, latencies, and capacities of possible exascale architectures.

- Develop a common methodology for capturing, exchanging, and evaluating these models and parameters on new algorithms and software systems.

SYSTEM SOFTWARE

INTRODUCTION

Often, system software is unseen. It is the layer of software from the low-level machine control system through the operating system, file system, message-passing libraries, and performance tools. From the scientist's perspective, system software provides services, common interfaces, and tools for developing and running applications.

As the community looks to exascale computing as a path toward scientific discovery, several key hardware trends have been identified that will significantly impact the design of system software, as well as the functionality available to science teams. The explosion of on-socket parallelism is a good example of how hardware trends for exascale computing will impact system software.

Likewise, there are trends in computational science affecting the designs of system software in anticipation of exascale systems. For example, the growing role of uncertainty quantification (UQ) and the need to understand the shift toward data-centric analysis models for some domains.

It is the confluence of these trends—from hardware up and application on down—that presents a significant challenge for system software as it must adapt to the changing architecture while providing new functionality and use modes to ever-larger application frameworks.

The sections below present a summary of the topics discussed and conclusions reached during the breakout sessions of the Crosscutting Technologies for Computing at the Exascale workshop.

In addition, Appendix 1 provides an overview of the activities of the *International Exascale Software Project* (IESP)[1] and the IESP roadmap developed by that project. The IESP[1] brings together representatives from the major segments of the global HPC community—software research and development, leading-edge scientific applications, leadership-class facilities, major system vendors, and relevant government funding agencies—in a proactive effort to plan and create a new software infrastructure for the extreme scale systems that represent the future of computational science. Through a series of workshops, initiated in spring 2009 and rotating between the U.S., Europe, and Asia, the IESP and its international partners have endeavored to provide leadership in formulating a long-term roadmap to an exascale-capable software infrastructure and initializing the corresponding research and development process necessary for its production.

UNDERSTANDING MATHEMATICAL MODELS, METHODS, AND ALGORITHMS IN THE CONTEXT OF SYSTEM SOFTWARE

Partial Differential Equations

This section includes meshing, PDE discretization, computational geometry, and multiphysics issues. Typically, PDE solvers are characterized by using domain decomposition with nearest-neighbor communication patterns. Elliptic PDE solvers require smaller, non-local communication patterns, and multilevel solvers with finer grids tend to create smaller groups of communicating processors scattered

[1] http://www.exascale.org

across the machine. In terms of frequency and message size, the communication characteristics also are varied, from large, frequent nearest-neighbor messages to smaller, less frequent messages to farthest neighbors. Solvers also require frequent, small global reductions—perhaps as many as several thousand per time step. All of these characteristics place incredible demands on the network to deliver performance in terms of low latency, high bandwidth, and high message rate for point-to-point and collective communication operations. From the application perspective, it is highly desirable that the logical layout of the communication patterns match the physical topology of the machine as closely as possible to optimize communication performance by reducing network contention.

The number of network endpoints in an exascale system is expected to be several orders of magnitude larger than today's most powerful machines. As a result of component failures in the system, network topology most certainly will be subject to dynamic reconfiguration. As such, it will be necessary to explore approaches and mechanisms that allow the application and system software to work together to maximize the network's performance potential. Possible approaches include programming interfaces that allow system software to convey information about the underlying network topology to the application so the application can structure communication appropriately and, conversely, providing interfaces that allow the application to inform system software about the application's communication structure so communicating elements can be placed and scheduled optimally. In the latter approach, mechanisms that support moving the computation to the data when most efficient also need to be explored.

Fault Tolerance

Fault tolerance and resilience are significant issues for applications moving to larger systems. From an application perspective, the system software should handle the fault management transparently. However, scientists realize that a partnership designing the system software will be required. It is unlikely that technology will evolve to respond to faults transparently, and it is also understood that simple checkpoint-restart scales poorly. New techniques that possibly use local NVRAM may prove helpful.

However, fault notification must be part of any solution that shares the responsibility for managing problems with the underlying system. At the most basic level, an application must be notified that a fault has occurred, what type of fault it is, and what resources are impacted. A co-design effort is needed between the system software and application developers to define a standard fault notification API and nomenclature. One possibility is a publish/subscribe interface, where applications, libraries, and even system software components could subscribe only to the fault events they are interested in and avoid being inundated with a continuous stream of fault messages. In some cases, local recovery from a fault using checkpoint/restart is possible. Moreover, the checkpoint files typically are small and could be stored in a neighbor's memory or non-volatile RAM.

Debugging

Applications debugging is related to fault tolerance. As with existing petascale platforms, the ability to debug applications at the exascale is one of the most important requirements and fundamental to application development. The impact of faults on the system will further complicate the debugging challenge. In a sense, application errors will be indistinguishable from errors generated by the system. Potentially, current strategies that analyze the state of processes to determine equivalence and identify outliers can be effective. However, such an approach may not extend to systems and applications that are

more dynamic and asynchronous than those that exist today. New approaches to debugging, not only to recognize application-generated errors, but also to identify and isolate system-generated errors, will be critical. Potential approaches for error identification and isolation include support for applications that allow the system software to check the validity of data flowing into and out of computational routines, as well as mechanisms that permit computing for the same function multiple times to determine the causality of an error.

Memory Management is Another Important Area for Future PDE Applications

As systems grow toward exascale, the memory hierarchy continues to become deeper and more complex. From the perspective of an application, tools and interfaces that can hint, manage, and control memory for runtime systems are inadequate. PDE solvers typically have an abundance of fine-grain, node-level parallelism that can be exploited. A hierarchical machine model with two-level parallelism is appropriate for both structured and unstructured grids, as wells as AMR and linear solvers. System software that could exploit hints from the application regarding spatial and temporal locality could be extremely beneficial. Cache memory is energy expensive, so alternative approaches to providing fast local memory access, such as scratchpad memory, can potentially provide a substantial performance improvement and energy savings, provided there is system software support from the compilation environment and the associated runtime system to provide effective resource allocation and management strategies.

Given the abundance of node-level parallelism, a low-cost mechanism for creating and destroying threads will be critical for the performance of PDE solvers at the exascale. System software will play a key role in providing the infrastructure necessary to support a global namespace where threads are first class objects that are named and manipulated directly by the application. Significant investigation is needed to determine the most appropriate threading model. Creating more threads than available processing elements is potentially a strategy that will work well for PDE solvers and will provide the system software an opportunity to hide memory latency and be more effective at scheduling processing resources. The dynamic nature of computation for solvers will require a highly dynamic threading model and new strategies in system software for creation and management of large numbers of dynamic threads will be necessary.

Solvers and Optimization

Heterogeneous multicore-based chip designs will radically change how solvers are designed and written. In addition to the traditional functions of system software, a better language for describing the underlying architecture is required. OpenCL has been one attempt at defining a machine model that can be probed by solvers. However, the feeling of the scientific community is that OpenCL may only be an early start in what will require significant research and exploration. Furthermore, as solvers increasingly embrace hybrid programming models and seek ways to use heterogeneous resources, automatic optimization and auto tuning will need dramatic improvement.

Data Analysis and Visualization

Traditionally, data analysis and visualization has been relegated to the “10% of the whole machine cost” rule. Nevertheless, exascale computing capabilities and experimental facilities of unprecedented resolution open the possibility that data analysis and visualization will address very deep scientific questions, which, in turn, justifies turning more resources to them. For example, they could approach the

unresolved issue of statistical uncertainty in the subgrid models used by the highest performing fluid simulation codes. The 10% rule, however, also results in the fact that it may be harder to identify limiting factors that data analysis will impose for certain system software components. For example, it is not envisioned that fault tolerance will be a major issue. It is likely that in-situ data analysis and visualization will have to use the same check-point restart mechanism, whereas data analysis and visualization not-in-situ will be constrained by bandwidth to run on a smaller scale, for which fault-tolerance is less critical than for full-scale applications.

Another possibility for future data analysis and visualization is the possibility of massive use of *in situ* and near-I/O analysis, which results in a unique set of questions for systems software. For *in situ* analysis, it will be important to consider what programming models and OS mechanisms will separate analysis from the simulation in a way that makes sense. Possible approaches are to do both, either as part of the same computation or by using a shared address space mapping. For a near-I/O data analysis approach, the two computations do not necessarily operate in the same cluster, but they do operate on the same cluster storage. Some issues to be solved include performance isolation (what the visualization does should not cause the simulation to slow down) and partitioning visualization to execute near the data.

In addition, the projected massive use of NVRAM, with its two orders of magnitude superior performance compared to disk, offers the unprecedented opportunity of being exposed by the system software for data analysis that does not need sequential access. Such cases appear, for example, when correlation information is computed hierarchically across multiple nodes for only a small subset of the data resident on the NVRAM. As the likely primary use of NVRAM is for check pointing, appropriate programming models and services provided by the OS must be identified while also ensuring checkpoint success.

An immediate need is the one of developing exascale data formats that allow for better use of the bounded I/O bandwidth. One approach is to use database technology approaches, such as columnar store and highly compressed data indexing, the latter possibly computed *in situ*. This will allow for efficient access to the output of multiphysics simulations when accessing only individual variables, as well as higher compression ratios due to better scaling compatibility. Potentially, this can be accomplished in a way that still allows the user to see this as a standard format, such as NetCDF.

Co-design will play an important role for the interaction between data analysis and visualization. Improved data models development is a fundamental issue for both system software and data analysis since specialized data formats may highly reduce extensibility. A possible solution is using object models for data, which raises several challenges, such as performing I/O on objects, having application-relevant data models, and mapping them efficiently onto exascale storage. Getting this right would greatly simplify the caching of relevant attributes and increase the efficiency of storage systems. Seamless memory/storage is another joint-interest topic since, depending on the size of data, the analysis can either be performed in the RAM of the visualization nodes or the computation can be moved to storage if it does not fit in RAM. Appropriate management of this situation is both a programming model challenge and system software challenge.

Uncertainty Quantification

Traditionally, UQ includes error arising from the hardware and software environment in the approximation error of numerical algorithms. However, unlike with today's petascale systems, the increasing component count of an exascale platform will require novel mitigation strategies to manage

errors and faults. Hard errors, soft errors, and silent errors will impact every aspect of the computing platform, including applications and system software. This will occur on the smallest scale, e.g., bit errors in computations. Additionally, systems will be subject to much more non-deterministic behavior, both as a result of random hardware failures and new system software for managing scarce resources, such as power. Investigation is needed to determine approaches that allow applications to communicate the need for determinism and accuracy to the system software so it can act appropriately. All of these factors will need to be included in UQ at the exascale.

Several characteristic approaches of UQ involve ensemble calculations, which have dynamic resource allocation requirements that differ significantly from the traditional, single large-simulation application. Ensemble calculations typically use a client/server model, where a set of coordinating processes launch larger, sometimes independent, computations. This model impacts the system software in several ways. Scheduling of jobs needs to be much more dynamic. The traditional space-shared, batch-scheduled usage model likely will not be effective at the exascale. The client/server model also requires a different failure model. Ideally, the system would respond differently based on if a failure impacts a client or server process. In many cases, the clients can simply be re-run, while a failed server process is much more difficult to recover.

Discrete Math

Discrete mathematical algorithms in graph theory, integer programming, and combinatorial optimization are characterized by a large amount of data movement and very few floating-point operations. The data movement is governed by the structure of the graph and, as such, is seldom regular and, often, quite random. The architectural specifications related to efficient data movement and local memory volumes are more important for the performance of discrete math problems than the total number of flops.

Several system software features will be required by discrete math applications executing on future computers. This is not an exhaustive list, but it indicates those system software features that should be improved to meet the future needs of discrete math applications.

Adaptive Runtime

Discrete algorithms, such as branch-and-bound, often begin by needing a small number of resources, and the resource needs grow as the problem evolves. Thus, the runtime system must be able to grow dynamically and shrink the number of resources required based on the application needs. Dynamic load balancing on a node is another capability that must be supported by the runtime system due to the shifting resource needs of many discrete math problems. To reconfigure the computer around faults and more generally adapt to a dynamically changing system configuration, an adaptive runtime also is needed. As much as possible, the system software is expected to handle faults through transparent migration of processes or restarting processes from a checkpoint.

Resource Management

Power will be a primary constraint in an exascale system. Because discrete math applications require few flops, it would be beneficial if the resource manager could reapportion the power budget so power would primarily go into data movement. This feature has an impact on the architecture because it requires the system to be over-provisioned, i.e., some applications may want 20 MW to move data fast, while others

may want to use 20 MW to do flops. If an application tried to do both, the system software would need to step in and constrain the total system power to 20 MW.

Using graph algorithms to schedule tasks on and across nodes offers another co-design opportunity. The dynamic scheduling algorithms should be memory efficient, or they could use up the limited memory on the node. They should be fast so dynamic scheduling does not impede the application. Ideally, they would be able to do incremental updates so scheduling across a million nodes does not have to be recalculated every time one node changes. The co-design contributions from the system software include: dynamic load balancing schemes, scheduling work on heterogeneous nodes, and handling dynamically changing resources. The resources could be changing due to energy efficiency constraints, faults, or application needs.

I/O

Discrete math applications often deal with irregular data streams and moving sparse data packets. For future systems, the MPI-I/O, or its future equivalent, needs to be able to handle irregular data efficiently.

Simulation

Using discrete event simulation to study the effects of various hardware choices on different algorithms provides another opportunity for co-design of hardware and software. Simulation provides two opportunities for discrete math and system software researchers to work together. First, discrete math can help with the development of the simulator itself. Second, a simulator is needed to study design choices for memory and resource management within the system software.

Crosscutting System Software Themes

File System

A main concern is the scalability and robustness of the file system. Historically, the file system has been the weakest link of the largest supercomputers. At the exascale, I/O will be different from today's systems and will include more *in situ* analysis to reduce data and speed up the discovery process.

Debug Tool

A very detailed single node debugger will form the base layer of the required debugger suite. A debugger that can work on a modest-size system with 10,000 nodes composes the middle of the suite. There is skepticism that a debugger can be built that will work at the full size of an exascale system. However, it is well-known with today's applications that bugs which appear on large-scale systems do not surface on modest-size systems. Two gaps in the current debugging tools are: 1) how to find large-scale bugs and 2) what to do when a bug does not materialize until an application has run several hours.

As with existing petascale platforms, debugging applications at the exascale is one of the most important capabilities and fundamental to application development. The impact of faults on the system will further complicate the debugging challenge. In a sense, application errors will be indistinguishable from errors generated by the system. Potentially, current strategies that analyze the state of processes to determine equivalence and identify outliers can be effective. However, given the expected dynamic and asynchronous nature of systems and applications, this approach may be ineffective. New approaches to

debugging, not only to recognize application-generated errors, but to identify and isolate system-generated errors, will be critical. Potential approaches to error identification and isolation include support for applications that allow the system software to check the validity of data flowing into and out of computational routines, as well as mechanisms that will permit computing the same function multiple times to determine the causality of an error.

Performance Tools

Today's performance tools have not been able to keep pace with the scale increase of the largest systems. This creates a gap in understanding how applications will perform on an exascale system or how to improve their performance. The shift to heterogeneous systems will make the existing performance tools even less useful. A holistic performance measurement tool that vertically integrates an application's performance model is needed—along with a detailed understanding of the (potentially heterogeneous) node and the system as a whole. To better understand the performance characteristics of an application, this tool would use the "dynamic system information space" (previously described) to collect and analyze configuration and performance data during a run.

Performance Modeling

Performance monitoring and engineering is a research emphasis that is useful at any scale, but it becomes much more essential at exascale. The purpose of performance modeling in co-design is to focus human attention on critical sections and high duty kernels, with the philosophy that one cannot optimize what one cannot measure or understand. Performance profiling also is essential for use in auto-tuning. An entire spectrum of methodologies ranging from instrumentation to measurement to modeling will be required. Modeling includes the various flavors of deterministic, stochastic, and simulation of the machine itself.

CROSSCUTTING CHALLENGES AND RESEARCH NEEDS FOR SYSTEM SOFTWARE AT THE EXASCALE

The challenges identified for system software to support exascale computing are summarized in the following PRDs:

PRD 3.1: **Develop new system software tools to support node-level parallelism.**

- Provide support for handling small, lightweight messaging.
- Provide lightweight, fine-grained, and flexible synchronization mechanisms.
- Provide support for high degree of threading to tolerate latencies.
- Provide system calls for node-level parallelism.
- Provide low-cost mechanisms to create and destroy threads.
- Provide software control of on-chip data movement for performance-critical kernels.

- Provide support for fast all-reduce to support fast inner products.
- Provide tools to manage communication patterns, e.g., two-way communication between application code and system.
- Provide support for moving away from bulk-synchronous applications.
- Provide support for maintaining local state when objects migrate between processors.

PRD 3.2: **Provide system support for dynamic resource allocation.**

- Provide support for dynamic load balancing (e.g., for adaptive methods, which have unpredictable resource requirements).
- Research client-server model for UQ ensemble calculations
  - develop dynamic job scheduling tools to replace current batch approaches
  - provide support for dynamic spawning of new simulations.
- Provide support for directed acyclic graph-style task scheduling.
- Provide support for efficient task replication.
- Develop adaptive runtime systems to address:
  - dynamic resource requirements
  - dynamic load balancing
  - ability to reconfigure around faults or changing system configurations
  - adaptive power allocation to network versus central processing unit.
- Research the use of graph analysis for fast dynamic scheduling.

PRD 3.3: **Develop new system software support for memory access (global address space, memory hierarchy, reconfigurable local memory).**

- Provide support for global address space to replace cache-coherence as a mechanism.
- Research the use of global address space in partitioning of graphs.
- Provide tools to manage memory hierarchies.
- Provide ability to turn off memory hierarchy for accesses that cannot use it sufficiently.
- Provide hooks for direct access to memory management.

- Provide support to allow local memory to be configured in either scratchpad or cache mode.
- Provide system support for data provenance to support data analysis.

PRD 3.4: **Develop performance/resource measurement and analysis tools for exascale.**

- Research and develop new performance analysis tools, particularly for hybrid programs and heterogeneous environments.
- Provide system calls to query relative costs of various operations
  - both static and dynamic information are needed.
- Leverage data mining methods in the development of new performance measurement tools.
- Add functionality to runtime layers to provide information about the system state.

PRD 3.5: **Develop new system tools to support fault management/system resilience.**

- Develop tools to support fault tolerance management, e.g., a fault notification application programming interface.
- Perform research to support debugging at scale.
- Research the fault-tolerance implications of UQ.
- Develop a taxonomy of faults to support advanced fault handling.
- Understand the role of system software in resilience.
- If a smaller system (e.g., 10%) is used for data analysis, observe that:
  - Because it will not be possible to move large amounts of data off the main computing platform, resilience will be less of a problem.

PRD 3.6: **Develop capabilities to address the exascale input/output challenge.**

- Research file system scalability and robustness.
- Research the input/output bandwidth issue.
- Research input/output for irregular data.

CONCLUSIONS AND RECOMMENDATIONS

Development of an exascale computing capability with machines capable of executing O(10^{18}) operations per second in the 2018 time frame will be characterized by significant and dramatic changes in computing hardware architecture from current (2010) petascale high-performance computers. From the perspective of computational science, this will be at least as disruptive as the transition from vector supercomputing to parallel supercomputing that occurred in the 1990s. This report identifies many of the issues that will arise and makes recommendations on research and development (R&D) activities that will be necessary to assure the achievement of scientific application performance commensurate with the expected improvement in computing capability. The required R&D must identify and/or develop mathematical models and numerical algorithms that map efficiently onto exascale architectures, and must significantly re-engineer scientific application codes supported by the corresponding development of new programming models and system software appropriate for these new architectures. Achieving these increases in capability by 2018 will require a significant acceleration in the development of both hardware and software. An intensive "co-design" effort, where system architects, application software designers, applied mathematicians, and computer scientists work interactively to characterize and produce an environment for computational science discovery that fully leverages these significant advances in computational capability, is recommended as the approach for reaching this challenging goal.

The workshop identified priority research directions (PRDs) that will be essential for developing exascale computing environments. They are grouped here under three topic areas: 1) Algorithm and Model Research and Development Needed to Support New Architectures; 2) Research and Development for Programming Models to Support Exascale Computing; and 3) Research and Development for System Software at the Exascale.

TOPIC AREA 1: ALGORITHM AND MODEL RESEARCH AND DEVELOPMENT NEEDED TO SUPPORT NEW ARCHITECTURES

PRD 1.1: Recast critical applied mathematics algorithms to reflect impact of anticipated macro architecture evolution, such as memory and communication constraints.

PRD 1.2: Take advantage of exascale architectural evolution to design new algorithms for uncertainty quantification and discrete mathematics.

PRD 1.3: Develop new mathematical models and formulations that effectively exploit anticipated exascale hardware architectures.

PRD 1.4: Address numerical analysis questions associated with moving away from bulk-synchronous programs to multitask approaches.

PRD 1.5: Adapt data analysis algorithms to exascale environments.

PRD 1.6: Extract essential elements of critical science applications as "mini-applications" that hardware and system software designers can use to understand computational requirements.

PRD 1.7: Develop tools to simulate emerging architectures and performance modeling methods for use in co-design.

TOPIC AREA 2: RESEARCH AND DEVELOPMENT FOR PROGRAMMING MODELS TO SUPPORT EXASCALE COMPUTING

PRD 2.1: Investigate and develop new exascale programming paradigms to support "billion-way" concurrency.

PRD 2.2: Develop tools and runtime systems for dynamic resource management.

PRD 2.3: Develop programming models that support memory management on exascale architectures.

PRD 2.4: Develop new scalable approaches for input/output (I/O) on exascale architectures.

PRD 2.5: Provide interoperability tools to support the incremental transition of critical legacy science application codes to an exascale programming environment.

PRD 2.6: Develop language support for programming environments at the exascale.

PRD 2.7: Develop programming model support for latency management.

PRD 2.8: Develop programming model support for fault tolerance and resilience.

PRD 2.9: Develop integrated tools to support application performance and correctness.

PRD 2.10: Develop a new abstract machine model that exposes the performance-impacting design parameters of possible exascale architectures.

TOPIC AREA 3: RESEARCH AND DEVELOPMENT FOR SYSTEM SOFTWARE AT THE EXASCALE

PRD 3.1: Develop new system software tools to support node-level parallelism.

PRD 3.2: Provide system support for dynamic resource allocation.

PRD 3.3: Develop new system software support for memory access (global address space, memory hierarchy, and reconfigurable local memory).

PRD 3.4: Develop performance and resource measurement and analysis tools for exascale.

PRD 3.5: Develop new system tools to support fault management and system resilience.

PRD 3.6: Develop capabilities to address the exascale I/O challenge.

REFERENCES

Ahn DH, BR de Supinski, I Laguna, GL Lee, B Liblit, BP Miller, and M Schulz. 2009. "Scalable Temporal Order Analysis for Large Scale Debugging." In *Proceedings of the ACM/IEEE Conference on High Performance Computing*, *SC 2009*, November 14-20, 2009, Portland, Oregon. Last accessed January 4, 2011, at http://ftp.cs.wisc.edu/par-distr-sys/papers/Miller09ScalableDebugging.pdf.

Bell G. 1996. "Great and Big Ideas in Computer Structures." In *Mind Matters: A Tribute to Allen Newell*, eds. DM Steier and TM Mitchell, Psychology Press, London.

Bell JB, MS Day, X Gao, and MJ Lijewski. 2010. "Simulation of Nitrogen Emissions in a Low Swirl Burner." Presented at *SciDAC 2010*, July 11-15, 2010, Chattanooga, Tennessee. Last accessed January 4, 2011, at https://ccse.lbl.gov/people/jbb/.

Bland AS, RA Kendall, DB Kothe, JH Rogers, and GM Shipman. 2009. "Jaguar: The World's Most Powerful Computer." Presented at *CUG 2009*, May 4-7, 2009, Atlanta, Georgia. Last accessed January 4, 2011, at http://www.nccs.gov/wp-content/uploads/2010/01/Bland-Jaguar-Paper.pdf.

Bronevetsky G, I Laguna, S Bagchi, BR de Supinski, DH Ahn, and M Schulz. 2010. "AutomaDeD: Automata-Based Debugging for Dissimilar Parallel Tasks." Presented at the 40th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2010), June 28-July 1, 2010, Chicago, Illinois. (LLNL-CONF-426270*).

Burstedde C, O Ghattas, M Gurnis, G Stadler, E Tanz, T Tu, LC Wilcox, and S Zhong. 2008. "Scalable Adaptive Mantle Convection Simulation on Petascale Supercomputers." In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, *SC '08*, November 15-21, 2008, Austin, Texas. Last accessed January 4, 2011, at http://delivery.acm.org/10.1145/1420000/1413434/a62-burstedde.pdf?key1=1413434&key2=1037296921&coll=DL&dl=ACM&CFID=8918139&CFTOKEN=89409116.

Chazan D and W Miranker. 1969. "Chaotic Relaxation." *Linear Algebra and its Applications* 2(2): 199-222.

DOE. 2008a. *Mathematics for Analysis of Petascale Data: Report on a Department of Energy Workshop – June 3-5, 2008, Washington, D.C.* U.S. Department of Energy, Office of Science, Washington, D.C. Last accessed January 4, 2011, at http://www.sc.doe.gov/ascr/ProgramDocuments/Docs/PetascaleDataWorkshopReport.pdf.

DOE. 2008b. *Scientific Grand Challenges: Challenges for Understanding the Quantum Universe and the Role of Computing at the Extreme Scale – December 9-11, 2008, Menlo Park, CA*. PNNL-18775, U.S. Department of Energy, Office of Science, Washington, D.C. Last accessed January 4, 2011, at http://www.sc.doe.gov/ascr/ProgramDocuments/Docs/HEPReport.pdf.

DOE. 2008c. *Scientific Grand Challenges: Challenges in Climate Change Science and the Role of Computing at the Extreme Scale – November 6-7, 2008, Washington, D.C.* PNNL-18362, U.S. Department of Energy, Office of Science, Washington, D.C. Last accessed January 4, 2011, at http://www.sc.doe.gov/ascr/ProgramDocuments/Docs/ClimateReport.pdf.

DOE. 2009. *Scientific Grand Challenges: Science Based Nuclear Energy Systems Enabled by Advanced Modeling and Simulation at the Extreme Scale – May 11-12, Washington, D.C.* U.S. Department of Energy, Office of Science, Washington, D.C. Last accessed January 4, 2011, at http://www.sc.doe.gov/ascr/ProgramDocuments/Docs/SC-NEWorkshopReport.pdf.

DOE. 2010a. 2009. *Scientific Grand Challenges: Fusion Energy Sciences and the Role of Computing at the Extreme Scale – March 18-20, 2009*, Washington, D.C. PNNL-19404, U.S. Department of Energy, Office of Science, Washington, D.C. Last accessed January 4, 2011, at http://www.sc.doe.gov/ascr/ProgramDocuments/Docs/FusionReport.pdf.

DOE. 2010b. *Scientific Grand Challenges for National Security: The Role of Computing at the Extreme Scale – October 6-8, 2009, Washington, D.C.* PNNL-19379, U.S. Department of Energy, Office of Science, Washington, D.C. Last accessed January 4, 2011, at http://extremecomputing.labworks.org/nationalsecurity/report.stm.

DOE. 2011. *Scientific Grand Challenges: Architectures and Technology for Extreme Scale Computing – December 8-10, 2009, San Diego, CA.* U.S. Department of Energy, Office of Science, Washington, D.C. Last accessed February 1, 2011, at http://extremecomputing.labworks.org/hardware/reports/FINAL_Arch&TechExtremeScale1-28-11.pdf.

Dongarra J. 2010. "Impact of Architecture and Technology for Extreme Scale on Software and Algorithm Design." Presented at *Crosscutting Technologies for Computing at the Exascale*, February 2-4, 2010, Washington D.C. Last accessed January 4, 2011, at http://extremecomputing.labworks.org/crosscut/presentations.stm.

Gamblin T, BR de Supinski, M Schulz, RJ Fowler, and DA Reed. 2010. "Clustering Performance Data Efficiently at Massive Scales." Presented at the *Twenty-Fourth International Conference on Supercomputing* (ICS 2010), June 1-4, 2010, Tsukuba, Japan. (LLNL-CONF-422684).

Lee B, SF McCormick, B Philip, and DJ Quinlan. 2003. "Asynchronous Fast Adaptive Composite-Grid Methods: Numerical Results." *SIAM Journal on Scientific Computing* 25(2):682-699. DOI: 10.1137/S1064827502407536.

Little JDC. 1961. "A Proof for the Queuing Formula: L= λ W." *Operations Research* 9(3):383-387.

Lusk EL, SC Pieper, and RM Butler. 2010. "More Scalability, Less Pain." *SciDAC Review* (17):30-37. Last accessed January 4, 2011, at http://www.scidacreview.org/1002/html/adlb.html.

Majda A and JA Sethian. 1985. "The Derivation and Numerical Solution of the Equations for Zero Mach Number Combustion." *Combustion Science and Technology* 42(3-4):185-205. DOI: 10.1080/00102208508960376.

Oak Ridge National Laboratory - ORNL. 2009. "Oak Ridge 'Jaguar' Supercomputer is World's Fastest." *Science Daily*. Last accessed January 4, 2011, at http://www.sciencedaily.com-/releases/2009/11/091116204229.htm.

Shalf J, S Dosanjh, and J Morrison. 2011. "Exascale Computing Technology Challenges." In *Proceedings of VECPAR 2010*, LNCS 6449, Springer-Verlag, Berlin, Germany. Last accessed January 4, 2011, at http://www.nersc.gov/projects/reports/technical/ShalfVecpar2010.pdf.

APPENDICES

APPENDIX 1: THE INTERNATIONAL EXASCALE SOFTWARE PROJECT ROADMAP

APPENDIX 2: WORKSHOP PROGRAM/AGENDA

APPENDIX 3: WORKSHOP PARTICIPANTS

APPENDIX 4: ACRONYMS AND ABBREVIATIONS

APPENDIX 1: THE INTERNATIONAL EXASCALE SOFTWARE PROJECT ROADMAP

The *International Exascale Software Project* (IESP)[1] brings together representatives from major segments of the global high-performance computing (HPC) community: software research and development, leading-edge scientific applications, leadership-class facilities, major system vendors, and relevant government funding agencies. The IESP is a proactive effort to plan and create a new software infrastructure for the extreme-scale systems that represent the future of computational science. Through a series of workshops starting in the spring of 2009 and rotating between the United States, Europe, and Asia, the IESP and its international partners have endeavored to provide leadership in formulating a long-term roadmap to an exascale-capable software infrastructure, and initializing the corresponding research and development process necessary to produce it. As expressed in the IESP Roadmap Version 1.0 (referenced hereafter as "IESP Roadmap" and published in 2011[2]), the project mission commits to a vision of the future of scientific computing that involves two key objectives:

> The guiding purpose of the IESP is to empower ultrahigh resolution and data intensive science and engineering research through the year 2020 by developing a plan for 1) a common, high quality computational environment for peta/exascale systems and for 2) catalyzing, coordinating, and sustaining the effort of the international open source software community to create that environment as quickly as possible. (IESP Roadmap, p. 3*)*

In providing an overview of the IESP work for the *Scientific Grand Challenge: Crosscutting Technologies for Computing at the Exascale* report, the authors focused on summarizing the IESP Roadmap's attempt to provide a comprehensive account of all different areas of software research and development that need to be addressed to achieve this goal. However, in establishing a high-level summary of the software plan, it is important to highlight a few key aspects of the current HPC environment that continue to affect the IESP in all of its work, particularly to the level of openness and diversity of participation the authors hope to achieve in organizational and community building efforts. Three deserve particular attention:

Science today requires software as infrastructure: Over the last decade, the idea that some software should be treated as infrastructure—and invested in as such—has become widely accepted. Computational science today depends on a large, complex and integrated ensemble of software components and tools, replicated in many places and maintained across successive generations of hardware. The formation of the IESP is a response to the recognition by many in the HPC community that its software infrastructure would need to be overhauled to address the changes now occurring in processor and system architecture, and the exponential growth in the quantity of research data that scientists are producing and consuming.

"Software ecology" is an apt metaphor: The concept that software depends upon something analogous to an ecological system —a "software ecology"— has gained currency as a way of conceptualizing the complex set of factors that need to be addressed to create and sustain it over time. Software ecology is the complicated network of people, processes, organizations,

[1] See http://www.exascale.org.
[2] "The International Exascale Software Roadmap," Dongarra, J., Beckman, P. et al., to appear in Volume 25, Number 1, 2011, *International Journal of High Performance Computer Applications*, ISSN 1094-3420.

technologies, markets, laws, and other software on which the continued existence and growth of a given type (or instance) of software depends. The software ecology of high-end computational science encompasses large science projects, major computing facilities and resource providers, government and academic laboratories, software research communities and private HPC vendors.

Global collaboration in science is rapidly becoming the norm: Given the ways in which scientific communities with expansive research agendas tend to self-organize in the Internet era, a new software infrastructure for high-end scientific computing must address the requirements of transnational collaborations; i.e., a situation in which the instruments and simulations that produce the essential data of interest, the computational resources to be applied to it, and the multidisciplinary teams who bring it all together, are distributed around the globe. The international character of the IESP reflects this reality and seeks to leverage the global software research and development (R&D) community that complements and supports it.

With this perspective, IESP leaders have sought to elicit the participation of qualified people from several different countries, are part of the HPC software ecosystem and share the project's goals, and are ready to work toward its objectives. The following is a high-level summary of the technical elements in the IESP Roadmap, which represents the main result of the project's work to date. The aim is to provide readers with a sense of the IESP Roadmap's content and comprehensiveness. The conclusion section briefly describes the three aspects of HPC software ecology addressed in the IESP Roadmap:

1. The "co-design vehicles" that are being developed as a shared context for the work of software R&D teams, scientific application communities, hardware designers, and resource providers

2. The range of issues that must be ironed out with HPC vendors, such as reasonable licensing and support policies for infrastructure that will include a mix of open source and proprietary software solutions

3. The concerns of national funding agencies that must be addressed by any future governance model proposed for the IESP.

OVERVIEW OF THE IESP ROADMAP

The purpose of the IESP Roadmap is to lay the foundation for a plan to create a common, open source software infrastructure for scientific computing on the exascale platforms of the future. The term "X-stack" was adopted as a convenient way of referring to the integrated collection of software infrastructure that a broad research and development effort would be charged with producing. The range of components and component types to be encompassed by a comprehensive X-stack is substantial. It includes the following: 1) **system software** (operating systems, runtime systems, input/output [I/O] systems, systems management, and external environments); 2) **development environments** (programming models, frameworks, compilers, numerical libraries, and development tools); 3) **application foundations** (algorithms, data analysis and visualization, and scientific data management); and 4) **crosscutting elements** (resilience, power management, performance optimization, and programmability).

The challenge of compiling such an X-stack derives not only from its scope, but also from the minimum set of required characteristics that this collection, taken as a whole, must possess. As described in Section 2 of the IESP Roadmap, the necessary properties are as follows:

- The X-stack must enable suitably designed science applications to exploit the full resources of the largest systems

- The X-stack must scale both up and down the platform development chain—from departmental clusters to supercomputers—because leading research applications are developed and used at all levels of that chain, from departmental clusters to supercomputers

- The X-stack must be highly modular, both because well designed modularity is essential to scalability and sustainability, and because—for practical and political reasons—it is important to be able to accept and offer alternative component designs and/or implementations in various areas

- The X-stack must offer open source alternatives for all its components, although vendors may also offer proprietary alternatives for critical functions: reasonable schemes for supporting open source components will need to be worked out.

With those characteristics in mind, the core of the IESP Roadmap (Section 4) titled, "Formulating paths forward for X-stack component technologies," presents the IESP's attempt to translate critical technology and science trends into concrete (if skeletal) plans for R&D for all parts of the X-stack. Because the IESP Roadmap is supposed to ultimately lead to a software infrastructure capable of bringing up an exascale system and doing exascale science on it, there was a concerted effort to make its treatment of the X-stack as comprehensive as possible. Even the first public version of the IESP Roadmap offered a relatively complete idea of the scope of the software R&D required to field a fully operational system in the expected time frame. The following sections review in turn all the main categories of X-stack software presented in the IESP Roadmap and briefly summarize the types of components they encompass.

System Software

The first general category of X-stack software —system software— includes the basic parts of the infrastructure that manage system resources on behalf of the application, and will therefore bear much of the brunt of the radical changes that scientific computing on exascale systems will introduce. It includes the operating system, runtime system, I/O subsystems, systems management and external environments. The last two component classes in this list—systems management and external environments—provide a good illustration of the IESP Roadmap's goal of comprehensiveness. These classes are intended to group software components that will be indispensable to the operation of any extreme-scale system of the future, but which are not necessarily in use or readily apparent while the application is executing. However, unless the IESP Roadmap included all the components of system software that users, administrators and wide-area collaborators require to interact with an exascale system, its description of the X-stack would be essentially incomplete. Each of the component areas of X-stack system software can be briefly described as follows:

Operating Systems: Because an operating system (OS) provides the fundamental set of mechanisms through which applications and users work with system resources, tomorrow's extreme-scale systems pose a steep challenge for OS designers. For example, while today's

systems contain hundreds of thousands of nodes with a small number of homogeneous cores per node, exascale systems will increase the number of cores by a few orders of magnitude and include both traditional and nontraditional (e.g., stream-based) cores. To make effective use of these resources, an exascale-capable OS will have to enable programmers to exploit parallelism at levels a thousand times greater than currently employed, and do so while coping with unprecedented constraints on power consumption. Consequently, critical goals in this area include providing mechanisms for internode and intranode communication and thread management, and a comprehensive memory management hierarchy. One of the proposed R&D strategies suggested by the IESP Roadmap authors involves the creation of a full system simulation environment to test alternative designs.

Runtime Systems: Runtime systems help match the characteristics of application algorithms to available system resources. When they work well, they help optimize performance and efficiency, while at the same time allowing developers to think about the science they are trying to accomplish rather than about task scheduling and low-level resource management. Many exascale challenges (e.g., parallelism, power, programmability, heterogeneity) converge on this area. Among the key design considerations for new extreme-scale runtime systems are unprecedented power/energy constraints and application development cost. The first constraint establishes the new objective for X-stack runtimes: *maximization of the achieved ratio of performance to power/energy consumption*, instead of raw performance alone. The second constraint means that X-stack runtimes must focus on supporting the execution of the *same program* at all levels of the platform development chain. Alternative R&D approaches divide according to the degree of hierarchy they involve: flat models that use message passing throughout (e.g., more evolved forms of message passing interface [MPI]) and hierarchical models that distinguish internode and intranode systems, which might hybridize MPI with a shared memory model.

I/O Systems: As compared to most other parts of the X-stack, the I/O libraries, which serve to move data between external storage devices and the core of the computer (i.e., processors and main memory), are as much affected by exponential increases in the amount of data involved in extreme-scale science as they are by the architectural changes in extreme-scale computers. I/O performance bottlenecks, already a concern at petascale, are likely to become alarmingly narrow as we start the ascent to the exascale. Radical solutions, such as abandoning the traditional file system paradigm for more "purpose-driven" I/O software layers that use lighter-weight semantics and are better able to expose parallelism to applications, are already being advanced as necessary to solve the problems looming in this area.

Systems Management: Systems management encompasses a broad range of technical areas that pertain, roughly speaking, to all those elements of system software infrastructure necessary to manage and monitor system operations in general, as opposed to those elements dedicated to supporting individual applications as they execute. For example, systems management includes X-stack components for system startup and configuration, resource provisioning, security, diagnostic testing, system logging and analysis, and coordination with external resources, such as grid or cloud computing infrastructures with which the system is networked.

External Environments: The IESP Roadmap authors use the term "external environments" to refer to the essential interfaces to remote computational resources (e.g., data repositories,

real-time data streams, high performance networks, and computing grids and clouds) that form a critical part of the working environment of most major research communities. These interfaces must therefore have a place in the X-stack. The IESP Roadmap focuses on interfaces to distributed data repositories as the main area of concern because, unlike most other resources, the data that advanced applications need to work on are often generated remotely (e.g., by an instrument or another supercomputer), or stored offsite (e.g., in a data grid).

Development Environments

The application development software that scientific programmers use to write, debug, and optimize research applications forms an especially difficult part of the exascale challenge. This area is challenging because the complexity of applications at extreme scale will increase by a few orders of magnitude along several different dimensions, while human intelligence and skill levels will not see corresponding increases. To address this situation, the programming models, software development frameworks, compilers, libraries, debuggers and other development tools that make up this category must all evolve. More likely, they will need to be revolutionized, because expanding the amount of parallelism that applications exploit by three orders of magnitude, while at the same time addressing processor heterogeneity and minimizing data movement to conserve power, goes well beyond anything the current programming paradigm enables programmers to do.

Programming Models: Enabling scientific programmers to express the algorithms their applications require in ways that make effective and efficient use of the resources that exascale systems make available will be a very difficult problem. In large degree, and for obvious reasons, the leading problems in this area—parallelism, power, programmability, heterogeneity, and reliability—mirror the problems that extreme-scale runtime systems must solve. Although current available approaches may be hybridized to deal with problems like node heterogeneity, a nontrivial portion of the community believes that to make extreme-scale computing viable for the vast majority of computational scientists, it will be necessary to discover or create a radically new programming paradigm that somehow hides the enormous complexity of the underlying system, especially in terms of parallelism.

Frameworks: Software development frameworks have a long history in HPC of providing a common, integrated collection of interfaces, libraries, and capabilities that help researchers encapsulate many of the computer science issues imposed by the large-scale platforms (e.g., discretizations, meshes, load balancing, I/O, scalable linear algebra). This allows researchers to focus instead on the questions involved in the science they are trying to accomplish (e.g., large-scale, multiphysics engineering, data mining for bioinformatics). By packaging application components that are reusable across a set of related applications, the scientific software community hopes to migrate a large body of existing applications toward exascale as quickly and efficiently as possible, while also preparing the ground for new applications in related science domains.

Compilers: Because compilers are at the intersection between programming languages and hardware architectures, major advances in compiler technology will be necessary to translate the new programming models into the high-quality object code that extreme-scale science will require. All the difficult issues confronting programming models are reprised for compilers, with the added complication that compilers must also support tools for adaptation, tuning, and

debugging. Achieving both high performance and developer productivity at the exascale will require compilers that go beyond the traditional "black box" approach by interoperating and interacting with other tools in the development and execution environment to systematically share information about an application and its execution behavior.

Numerical Libraries: From the perspective of the computational scientist, numerical libraries are the work horses of software infrastructure because they encode the underlying mathematical computations that their applications spend most of their time processing. Performance of these libraries tends to be the most critical factor in application performance. In addition to the architectural challenges they must address, their portability across platforms and different levels of scale is also essential to avoid interruptions and obstacles in the work of most research communities. Achieving the required portability means that future numerical libraries will not only need dramatic progress in areas such as auto-tuning, but also need to be able to build on standards—which do not currently exist—for things like power management, programming in heterogeneous environments, and fault tolerance.

Debugging Tools: The tools and methods traditionally used for the discovery and treatment of errors in program codes will have to be completely transformed to both locate and identify errors in extreme-scale environments (e.g., with hundreds of thousands or millions of processes), and to diagnose their cause and treat them in the face of incredible system complexity (e.g., machine heterogeneity, hybrid runtime systems, etc.). The general strategy is to create debuggers that can communicate and interact with other critical elements of the X-stack—compilers, libraries, OS, runtime and I/O layers—so that the search and diagnosis process can be pursued from a number of different perspectives and researchers can converge on portable standards for debugging that make the tools created maintainable over time. Examples include debuggers that interoperate with OS and runtime fault tolerance technologies to diagnose node failures, and standard of interfaces to performance information that debuggers can use to integrate performance information across software layers and in a portable way.

Applications

Although applications, more or less by definition, are not part of an exascale software infrastructure, the IESP community views it as necessary for the IESP Roadmap to address certain software components that are either foundational for exascale applications —namely, algorithms—or that provide essential support for doing exascale science with those applications; namely, tools for managing, analyzing and visualizing the data they produce. Consequently, Version 1.0 of the IESP Roadmap includes the following three application-related software components as an essential part of the plan for the X-stack:

Algorithms: By far, the most important determinant of a given application's ability to succeed at the exascale is the underlying algorithm(s) it uses to implement its mathematical model of the phenomena being studied. Although the relative importance of different factors will vary somewhat depending on the specific characteristics of a different platform, algorithms developed for exascale computing will need to address all fundamental issues:

- Create extremely high levels of concurrency for new architectures to exploit

- Increase the computation/communication ratio dramatically to help hide rapidly growing memory access latencies

- Contribute to improved resilience by helping to detect, repair, or overcome faults

- Help limit the degree to which performance must be traded away to achieve power consumption targets.

Once "strawman" exascale architectures are sufficiently defined, both evolutionary and revolutionary (i.e., clean slate) approaches will need to be explored to achieve these goals.

Data Analysis and Visualization: Given the immense profusion of data that exascale research is expected to produce, achieving insights at the exascale will inevitably require new approaches to data analysis and visualization that can penetrate and organize the welter of new information that will be flowing in. Interactive and collaborative tools and methods for data exploration are already becoming more important at the petascale—but at the exascale, they will become far harder to build. Advances in mathematical and statistical methods, in algorithms, in auto-adaptive and proactive software strategies, and even in cognitive science, will be necessary to create the data analysis and visualization support infrastructure necessary to enable exascale science.

Scientific Data Management: As already noted, the sheer volume and growing complexity of the data produced and consumed by extreme-scale science has made software for data management one of the community's most important (and increasingly urgent) needs. The challenges involved are myriad. Examples include the following:

- Future data analysis and data mining tools will need to be not only more scalable, but also easier to use

- Future data analysis and data mining tools must be able to cope with the increasingly multivariate, multidimensional and hierarchical data sets that advanced research is generating

- X-stack must provide essential workflow infrastructure that integrates with both the exascale I/0 subsystems and with the wide area data repositories described above

- Software researchers must develop more scalable data formats and high-level libraries that nonetheless preserve, as much as possible, backward compatibility with the formats already in use.

Crosscutting Dimensions

For the most part, the IESP Roadmap analyzes the X-stack in terms of horizontal layers that separate different domains of functionality, with higher levels in the stack depending on lower ones. The goal of organizing things this way is to prepare for a modular design that provides for an appropriate division of labor and clean interfaces between components. However, at the exascale, it is clear certain critical issues cut vertically through the layers of the stack so that in addition to the work that must happen at each layer, there must also be coordination and cooperation across different layers to address these issues

successfully. The IESP Roadmap identifies four areas in which common or standardized, cross-layer methods, protocols, or interfaces are likely to be essential: resilience, power management, performance optimization and programmability.

Resilience: Two facts combine to paint a realistic picture of the problem of fault tolerance at the exascale. First, exponential increases in the number of sockets, cores, threads, disks, data volumes, etc. means the need for resilience will increase very sharply. Second, existing techniques (global checkpoint/global restart) will be impractical at the exascale; in fact, these techniques are already proving inadequate at petascale. Because various kinds of errors (e.g., soft, silent, transient and permanent software and hardware errors) will affect software at various layers, designing and agreeing upon common, low overhead methods and protocols for detecting, managing, and recovering from faults—as well as finding fault avoidance and fault oblivious algorithms and techniques—will be essential to success at the exascale.

Power Management: One of the hallmarks of the extreme-scale era in supercomputing is the emergence of power consumption as a first-class design constraint. The design compromises likely to be required to control a system's power budget, which is a leading factor in its total cost of ownership, will also tend to reduce system bandwidth, application performance, and the system's overall scope and effectiveness. To enable software at different layers of the X-stack to contribute to effective power management while minimizing impact on performance and resiliency, the community will need (among many other things) to settle on common interfaces for representing control system sensor data, for expressing system/node power management policy, and for communicating control decisions system wide, and at all layers of the stack. The absence of such standards today means, that at this point, there are no tools to support the integration of power management into libraries and applications going forward.

Performance Optimization: Over and above the sheer complexity of exascale systems and applications, increasing operational dynamism caused largely by power management schemes and the proliferation of faults, will make the modeling, prediction, measurement, analysis and optimization of application performance far more difficult to perform. Various R&D strategies will need to be pursued to solve this problem. Possible approaches range from extending current tools with new and more scalable methods to building performance-awareness, self-monitoring and self-tuning into new software infrastructure components from the outset. Perhaps most importantly, components and layers of the X-stack will have to be designed for performance transparency, so that higher levels of the stack, and especially the application itself, can detect and respond effectively to the sources of performance degradation.

Programmability: A simplified picture of programmability divides the application development process into three interrelated parts: algorithm capture and implementation, program correctness and debugging, and performance optimization. Because all levels of a system, hardware and software, affect one or more parts of this process in a substantial way, a programming model and language suitable for the exascale must provide semantics for parallelism and abstractions for controlling the underlying system that, under reasonable assumptions, enable application developers to actually cope with their fantastic complexity and scale. Perhaps because this area touches more directly on the normal human limitations of programmers, the path forward in this area of X-stack R&D divides into two broad camps: those that favor an evolutionary approach based on extensions (or perhaps hybridized versions) of conventional programming models, and

those that favor a revolutionary approach, arguing an entirely new (and as yet unknown) paradigm will be required to enable application scientists to address the incredible rigors of exascale computing.

THREE ASPECTS OF X-STACK SOFTWARE ECOLOGY

The three concluding sections of the IESP Roadmap—"Application Perspectives and Co-Design Vehicles," "Perspectives on Cooperation between IESP and HPC Vendor Communities," and "Initial Thoughts on IESP Organization and Governance" —highlight the IESP's concerted effort to ensure the plan it develops reflects the needs and views of all IESP stakeholders. This includes the scientific communities that aim to use exascale systems, the major supercomputing sites around the world that will have to purchase and operate them, the commercial providers who will design and build them, and the government agencies that fund and take political responsibility for these critical national investments. The goal of this part of the plan is to ensure the inevitable tradeoffs involved in creating an exascale-capable software stack can be embraced by all the stakeholders in this complex and highly dynamic environment.

Application Co-Design Vehicles

From its inception, the IESP has sought out and brought in the ideas and concerns of computationally intensive applications from different domains; this involvement of leading-edge computational science communities is bound to grow as the climb toward exascale computing proceeds. The input of these domain sciences to the initial version of the IESP Roadmap has already been significant. In particular, the identification of various keys to success at the exascale for most application communities, and especially the formulation and discussion of the concept of application co-design vehicles have been influential. Co-design vehicles (CDVs) are science applications that are well suited to provide targets for and feedback to the exascale hardware and software research, design, and development process. By laying out the criteria that make an application suitable as a CDV, and providing analyses of how a variety of application areas (e.g., High Energy Physics, Fusion Energy, Materials Science, Earth Systems, Health Sciences) match up with these criteria, the IESP Roadmap shows how CDVs provide a natural way to orient and focus concrete exascale design and development efforts moving forward.

Subsequent to the publication of Version 1.0 of the IESP Roadmap, and inspired in part by its account of application CDVs, the U.S. Department of Energy (DOE) has solicited proposals for Exascale Co-Design Centers to help organize its push to the next level of Computational Science. These Exascale Co-Design Centers would involve interdisciplinary teams collaborating to scale-up applications particularly in domain sciences, while at the same time informing the efforts of vendor hardware architects, X-stack R&D teams, and facility managers in the process. Because the fundamental rationale for exascale computing is to empower exascale science, there are good reasons to expect such domain-oriented CDV projects, both national and international, to take form. Moreover, the idea of the co-design process, which continues to evolve rapidly and clearly has an intimate connection with the development, testing and deployment of extreme-scale software infrastructure, is also improving the framework around which the future work of the IESP itself can coalesce.

COOPERATION WITH HIGH-PERFORMANCE COMPUTING VENDOR COMMUNITIES

Like the leaders of the application domains, the major HPC vendors have been well represented in the work of the IESP, although they clearly occupy a part of the HPC software ecology where very different conditions are obtained, and very different concerns—both technical and nontechnical—must be addressed. Achieving the goals of the IESP will require this group not only provides major enhancements to the functionality of their systems, but that they also work with the rest of the community to develop mutually agreeable social and economic models for creating, integrating, testing, and supporting the software infrastructure these systems will need. Beyond their substantial technical contributions to the plan, vendors' input into the current IESP Roadmap also lays the foundation for concrete progress in a variety of important nontechnical areas, including the following: analysis of development/support models for software, both proprietary and open source, created for future extreme-scale systems; presentation of expected effects of different intellectual property choices on the achievability of diverse goals of computing center management, the software research community and scientific application users; and suggestions on how segments of the IESP community could coordinate and define effective methodologies for managing the development, testing, deployment and ongoing support for all of the components of the X-stack.

The Challenge of IESP Governance

Finally, although focused discussions of the long-term organization and governance model for the IESP were only started in spring 2010, a relatively large group of governmental agencies in key nations (e.g., the U.S. Department of Energy, National Security Foundation, and Defense Advanced Research Projects Agency from the United States; MEXT and RIKEN from Japan; the European Commission and national funding agencies from the United Kingdom, France, Germany and the Netherlands) have actively engaged in addressing those issues. One key element, shaping both the governance model of the IESP and the cooperation strategy that national funding agencies will eventually adopt, is the business case the IESP community needs to make; i.e., the budget estimates, timelines, expected actors, roadmaps, risks and contingency plans for a new software infrastructure for exascale science. Because IESP governance is the area in which national differences —budgetary environments, funding mechanisms and regulatory frameworks (e.g., intellectual property and export control)— play the most prominent role, it may prove to be the steepest hill the international scientific community will have to climb on the road to exascale.

APPENDIX 2: WORKSHOP PROGRAM/AGENDA

Monday, February 1, 2010

Time	Session	Lead	Room
7:00-9:00 p.m.	Pre-Workshop Dinner Meeting (Organizers, Panel Leads, and Speakers)		Jefferson
9:00 p.m.	Adjourn		

Tuesday, February 2, 2010

Time	Session	Lead	Room
7:45 a.m.	Registration/Working Breakfast		Foyer
8:15-8:45 a.m.	Welcome	Michael Strayer, Dimitri Kusnezov	Regency
Plenary Talks			
8:45 a.m.	Exascale Initiative Overview	Andrew White	
9:15 a.m.	Co-design for the Exascale	Sudip Dosanjh	Regency
9:45 a.m.	Mission & Science Needs for Applied Math at the Exascale	Phil Colella	
10:25 a.m.	General Discussion		Foyer
10:45 a.m.	Mission & Science Needs for Computer Science at the Exascale	Rick Stevens	Regency
11:25 p.m.	Impact of Architecture and Technology for Extreme Scale on Software and Algorithm Design	Jack Dongarra	
12:05 p.m.	Working Lunch: Plenary Talk Discussion and Preparation for Breakouts		Foyer
12:40 p.m.	Software Challenges in Extreme Scale Systems	Vivek Sarkar	
1:10 p.m.	Programming Models at the Exascale	Bradford L. Chamberlain	Regency
1:40 p.m.	Charge to the Participants	David Brown	
2:00-3:45 p.m.	**Breakout Sessions**		
	Math A x Theme IA	Bell, Shalf	Washington
	Math C x Theme IB	Shephard, Morrison	Wilson
	Math D x Theme IIA	Brightwell, Estep	Truman
	Math E x Theme IIB	Pinar, Geist	Madison

Time	Session	Lead	Room
	Math B x Theme IIIA	Anitescu, Chamberlain	Roosevelt
	Math F x Theme IIIB	Heroux, Lusk	Jackson
3:45 p.m.	General Discussion		Foyer
4:00-5:30 p.m.	Continue Breakout Sessions (return to breakout rooms)		
6:00-7:30 p.m.	Working Dinner: Organizational Meetings		
7:30 p.m.	Adjourn		

Wednesday, February 3, 2010

Time	Session	Lead	Room
8:00-9:00 a.m.	Working Breakfast and Preparation for Breakouts		Randolph
9:00-10:00 a.m.	Breakout Reports (10 minutes each in length)		
	Math Track A x Theme IA	Bell, Shalf	
	Math Track C x Theme IB	Shephard, Morrison	
	Math Track D x Theme IIA	Estep, Brightwell	Regency
	Math Track E x Theme IIB	Pinar, Geist	
	Math Track B x Theme IIIA	Anitescu, Chamberlain	
	Math Track F x Theme IIIB	Heroux, Lusk	
10:00 a.m.	**General Discussion**		Randolph
10:30 - 12:30 p.m.	Breakout Sessions		
	Math Track B x Theme IA	Anitescu, Shalf	Washington
	Math Track F x Theme IB	Heroux, Lucas	Wilson
	Math Track A x Theme IIA	Bell, Geist	Truman
	Math Track C x Theme IIB	Shephard, Brightwell	Madison
	Math Track D x Theme IIIA	Estep, Chamberlain	Jefferson
	Math Track E x Theme IIIB	Hendrickson, Lusk	Jackson
12:30-1:30 p.m.	Working Lunch: Continue Breakout Work		Foyer
1:30-3:30 p.m.	Breakouts Resume (Return to Breakout Rooms)		
3:30 p.m.	**General Discussion**		
4:00-5:30 p.m.	Day 2 Breakout Reports (10 minutes each in length)		
	Math Track B x Theme IA	Anitescu, Shalf	
	Math Track F x Theme IB	Heroux, Lucas	
	Math Track A x Theme IIA	Bell, Geist	
	Math Track C x Theme IIB	Shephard, Brightwell	Regency
	Math Track D x Theme IIIA	Estep, Chamberlain	

Time	Session	Lead	Room
	Math Track E x Theme IIIB	Hendrickson, Lusk	
5:30-6:30 p.m.	**Breakout Sessions**		
	Math Track D x Theme IA	Estep, Shalf	Regency
	Math Track E x Theme IB	Hendrickson, Lucas	Wilson
	Math Track B x Theme IIA	Anitescu, Brightwell	Truman
	Math Track F x Theme IIB	Heroux, Geist	Madison
	Math Track A x Theme IIIA	Bell, Chamberlain	Jefferson
	Math Track C x Theme IIIB	Shephard, Lusk	Jackson
6:30 p.m.	Working Dinner: Continue Breakout Work		
9:30 p.m.	Adjourn		

Thursday, February 4, 2010

Time	Session	Lead	Room
7:30-8:00 a.m.	Working Breakfast and Preparation for Breakouts		Randolph
8:00-9:30 a.m.	**Breakout Reports**		
	Math Track D x Theme IA	Estep, Shalf	Regency
	Math Track E x Theme IB	Hendrickson, Lucas	
	Math Track B x Theme IIA	Anitescu, Brightwell	
	Math Track F x Theme IIB	Heroux, Geist	
	Math Track A x Theme IIIA	Bell, Chamberlain	
	Math Track C x Theme IIIB	Shephard, Lusk	
9:30 a.m.	**General Discussion**		Randolph
9:45 - 10:30 p.m.	**Initial Theme Roll-up Presentations (15 minutes each)**		
	Theme I	David Keyes	Regency
	Theme II	Pete Beckman	
	Theme III	Jeffrey Vetter	
10:30-1:30 p.m.	**Final Breakouts to Summarize by Theme**		
	Theme I: Math Models and Algorithms	David Keyes	Jefferson
	Theme II: System Software	Pete Beckmann	Madison
	Theme III: Programming Models	Jeffrey Vetter	Washington

Time	Session	Lead	Room
11:30 a.m.	**Working Lunch Available**: **Continue Breakout Work**		Foyer
1:30-2:00 p.m.	Theme 1: Conclusions	David Keyes	Regency
2:00-2:30 p.m.	Theme 2: Conclusions	Pete Beckman	
2:30-3:00 p.m.	Theme 3: Conclusions	Jeffrey Vetter	
3:00 p.m.	General Session Ends (Leads remain to outline final report, refreshments available)		Foyer
3:00 - 6:00 p.m.	Report writing session for leads and editors		
6:00 p.m.	Meeting Adjourns		

Applied Mathematics and Algorithms Tracks

Track A: PDEs I

- Multiscale Methods
- Multiphysics Problems
- ODE/DAE
- PDE Discretization Methods.

Track B: Data/Visualization

- Statistical Analysis of Large-Scale Data Sets
- Scalable Data Processing Methods
- Machine Learning
- Data Reduction
- Informatics.

Track C: PDEs II

- PDE Discretization Methods
- Multiphysics Problems
- Computational Geometry
- Meshing.

Track D: Uncertainty Quantification/Stochastic Systems

- Statistical Representation of Uncertainty
- Uncertainty Propagation
- Sampling Methods
- Sensitivity/Automatic Differentiation.

Track E: Discrete Math

- Graph Theory and Algorithms
- Combinatorial Optimization/Integer Programming
- Complex Networked Systems
- Load Balancing.

Track F: Optimization and Solvers

- Continuous Optimization
- Stochastic Programming
- Linear Algebra /Iterative Solvers
- Nonlinear Solvers
- Multigrid.

Workshop Theme Tracks

Theme 1: Impact of domain science needs and architectural developments on mathematical models, algorithms and programming models. Conversely, how domain science problems, mathematical models, and algorithm needs at the exascale can influence architectural development.

- Architecture/hardware
- Networks
- Power
- Memory subsystems
- Performance modeling.

Theme 2: System software functionality required by applications at the exascale, based on the mathematical models, algorithms, and programming models that they expect to use. Conversely, what tasks traditionally handled by system software will domain science applications at the exascale have to take on; e.g., resiliency handling, and what computer science R&D is needed to provide the necessary functionality?

- Fault tolerance
- Parallel I/O
- Grid computing
- Systems software.

Theme 3: Programming models and environment that are proposed to be developed for exascale systems (and their 100-300 PF precursors). Will the planned programming models provide suitable abstractions and tools for a spectrum of exascale applications? If not, how should the plans be modified? What computer science R&D is needed to provide the necessary functionality? What R&D in algorithms needs to be carried out to use the programming models effectively?

- Programming models
- Software engineering
- Development tools: performance measurement, debuggers, autotuners, etc.

APPENDIX 3: WORKSHOP PARTICIPANTS

Last /First Name	E-mail	Affiliation
Ahrens, James	ahrens@lanl.gov	Los Alamos National Laboratory
Allen, Gabrielle	gallen@cct.lsu.edu	Louisiana State University
Anitescu, Mihai	anitescu@mcs.anl.gov	Argonne National Laboratory
Ashby, Steven	steven.ashby@pnl.gov	Pacific Northwest National Laboratory
Balaji, Venkatramani	v.balaji@noaa.gov	Princeton University
Beckman, Pete	beckman@mcs.anl.gov	Argonne National Laboratory
Bell, John	JBBell@lbl.gov	Lawrence Berkeley National Laboratory
Brightwell, Ron	rbbrigh@sandia.gov	Sandia National Laboratories
Brown, David	dlb@llnl.gov	Lawrence Livermore National Laboratory
Camp, William	william.j.camp@intel.com	Intel Corporation
Canning, Andrew	acanning@lbl.gov	Lawrence Berkeley National Laboratory
Cappello, Franck	fci@lri.fr	INRIA
Chamberlain, Bradford L.	bradc@cray.com	Cray, Inc.
Chang, C. S.	cschang@cims.nyu.edu	New York University
Chapman, Barbara	bmchapman@earthlink.net	University of Houston
Chiang, Patrick Yin	pchiang@eecs.oregonstate.edu	Oregon State University
Childs, Hank	hchilds@lbl.gov	Lawrence Berkeley National Laboratory
Choudhary, Alok	a-choudhary@northwestern.edu	Northwestern University
Colella, Phillip	Pcolella@lbl.gov	Lawrence Berkeley National Laboratory
Collis, Scott	sscoll@sandia.gov	Sandia National Laboratories
Critchlow, Terence	terence.critchlow@pnl.gov	Pacific Northwest National Laboratory
Dally, William	dally@stanford.edu	Stanford University
Daly, John	john.t.daly@ugov.gov	The Laboratory for Physical Sciences
D'Azevedo, Ed	dazevedoef@ornl.gov	Oak Ridge National Laboratory
Dean, David	deandj@ornl.gov	Oak Ridge National Laboratory
Diachin, Lori	diachin2@llnl.gov	Lawrence Livermore National Laboratory
Domyancic, David	Domyancic1@llnl.gov	Lawrence Livermore National Laboratory
Dongarra, Jack	dongarra@eecs.utk.edu	University of Tennessee
Dorr, Milo	dorr1@llnl.gov	Lawrence Livermore National Laboratory
Dosanj, Sudip	ssdosan@sandia.gov	Sandia National Laboratories
Dubey, Anshu	dubey@flash.uchicago.edu	University of Chicago
Efstathiadis, Efstratios	stratos@bnl.gov	Brookhaven National Laboratory
Epperly, Tom	epperly2@llnl.gov	Lawrence Livermore National Laboratory
Estep, Don	estep@math.colostate.edu	Colorado State University
Evans, Katherine	evanskj@ornl.gov	Oak Ridge National Laboratory
Fann, George	fanngi@ornl.gov	Oak Ridge National Laboratory
Feng, Wu-chun	feng@cs.vt.edu	Virginia Tech
Gao, Guang	ggao@capsl.udel.edu	University of Delaware
Gara, Alan	alangara@us.ibm.com	International Business Machines
Garaizar, Xabier	garaizar1@llnl.gov	Lawrence Livermore National Laboratory
Geist, Al	gst@ornl.gov	Oak Ridge National Laboratory

Last /First Name	E-mail	Affiliation
Germann, Timothy	tcg@lanl.gov	Los Alamos National Laboratory
Gibson, Garth	garth.gibson@cs.cmu.edu	Carnegie Mellon University
Giles, Roscoe	roscoe@bu.edu	Boston University
Glimm, James	glimm@ams.sunysb.edu	Stony Brook University
Gottlieb, Steven	sg@denali.physics.indiana.edu	University of Indiana
Grider, Gary	ggrider@lanl.gov	Los Alamos National Laboratory
Grinberg, Leopold	lgrinb@dam.brown.edu	Brown University
Hack, James	jhack@ornl.gov	Oak Ridge National Laboratory
Hammond, Glenn	glenn.hammond@pnl.gov	Pacific Northwest National Laboratory
Hansen, Charles (Chuck)	hansen@cs.utah.edu	University of Utah
Hartman-Baker, Rebecca	hartmanbakrj@ornl.gov	Oak Ridge National Laboratory
Hatton, Angela	angela.hatton@pnl.gov	Pacific Northwest National Laboratory
Helland, Barbara	barbara.helland@science.doe.gov	U.S. Department of Energy, Office of Science
Hendrickson, Bruce	bahendr@sandia.gov	Sandia National Laboratories
Hereld, Mark	hereld@mcs.anl.gov	Argonne National Laboratory
Heroux, Michael A.	maherou@sandia.gov	Sandia National Laboratories
Hill, Judith	hilljc@ornl.gov	Oak Ridge National Laboratory
Hirata, So	hirata@qtp.ufl.edu	University of Florida
Hitchcock, Daniel	daniel.hitchcock@science.doe.gov	U.S. Department of Energy, Office of Science
Hoang, Thuc	thuc.hoang@nnsa.doe.gov	National Nuclear Security Administration
Hoisie, Adolfy	hoisie@lanl.gov	Los Alamos National Laboratory
Hovland, Paul	hovland@mcs.anl.gov	Argonne National Laboratory
Jablonowski, Christiane	cjablono@umich.edu	University of Michigan
Jansen, Kenneth	kjansen@scorec.rpi.edu	Center for Industrial Innovation
Johnson, Fred	fcj@acm.org	SAIC
Johnson, Gary	gmj@computationalsciencesolutions.com	Computational Science Solutions
Joo, Balint	bjoo@jlab.org	Jefferson Laboratory
Kale, Laxmikant	kale@cs.uiuc.edu	University of Illinois
Kamath, Chandrika	kamath2@llnl.gov	Lawrence Livermore National Laboratory
Kaplan, Larry	lkaplan@cray.com	Cray, Inc.
Karpeev, Dmitry	karpeev@mcs.anl.gov	Argonne National Laboratory
Keyes, David	kd2112@columbia.edu	King Abdullah University of Science and Technology and Columbia University
Khaleel, Moe	moe.khaleel@pnl.gov	Pacific Northwest National Laboratory
Koch, Kenneth	krk@lanl.gov	Los Alamos National Laboratory
Lamb, Donald	lamb@oddjob.uchicago.edu	University of Chicago
Landsberg, Sandy	sandy.landsberg@science.doe.gov	U.S. Department of Energy, Office of Science
Larzelere, Alex	alex.larzelere@nuclear.energy.gov	U.S. Department of Energy, Office of Science
Lee, Lie-Quan (Rich)	liequan@slac.stanford.edu	Stanford University

Last /First Name	E-mail	Affiliation
Lee, Sander	sander.lee@nnsa.doe.gov	National Nuclear Security Administration
Lee, Steven	slee@ascr.doe.gov	U.S. Department of Energy, Office of Science
Lin, Guang	guang.lin@pnl.gov	Pacific Northwest National Laboratory
Livny, Miron	miron@cs.wisc.edu	University of Wisconsin
Lucas, Robert	rflucas@isi.edu	University of Southern California
Lusk, Ewing	lusk@mcs.anl.gov	Argonne National Laboratory
Malony, Allen	malony@cs.uoregon.edu	University of Oregon
Marathe, Madhav	mmarathe@vbi.vt.edu	Virginia Bioinformatics Institute
Matthews, Hope	hope.matthews@pnl.gov	Pacific Northwest National Laboratory
McCoy, Michel	mccoy2@llnl.gov	Lawrence Livermore National Laboratory
McInnes, Lois	mcinnes@mcs.anl.gov	Argonne National Laboratory
Messer, Bronson	bronson@ornl.gov	Oak Ridge National Laboratory
Messina, Paul	messina@mcs.anl.gov	Argonne National Laboratory
Mezzacappa, Tony	mezzacappaa@ornl.gov	Oak Ridge National Laboratory
Morrison, John	jfm@lanl.gov	Los Alamos National Laboratory
Najm, Habib	hnnajm@sandia.gov	Sandia National Laboratories
Negrut, Dan	negrut@wisc.edu	University of Wisconsin
Nichols, Jeff	nicholsja@ornl.gov	Oak Ridge National Laboratory
Panda, Dhabaleswar	panda@cse.ohio-state.edu	Ohio State University
Pao, Karen	karen.pao@science.doe.gov	U.S. Department of Energy, Office of Science
Papka, Michael	papka@anl.gov	Argonne National Laboratory
Parker, Steven	sparker@nvidia.com	NVIDIA
Peery, James	jspeery@sandia.gov	Sandia National Laboratories
Pinar, Ali	apinar@sandia.gov	Sandia National Laboratories
Plata, Charity	charity.plata@pnl.gov	Pacific Northwest National Laboratory
Pothen, Alex	apothen@purdue.edu	Purdue University
Prudencio, Ernesto	prudenci@ices.utexas.edu	University of Texas, Austen
Riley, Katherine	riley@mcs.anl.gov	Argonne National Laboratory
Roche, Kenneth	kenneth.roche@pnl.gov	Pacific Northwest National Laboratory
Roderick, Oleg	roderick@mcs.anl.gov	Argonne National Laboratory
Ross, Robert	rross@mcs.anl.gov	Argonne National Laboratory
Sarkar, Vivek	vsarkar@rice.edu	Rice University
Schreiber, Robert	rob.schreiber@hp.com	Hewlett Packard
Schulz, Martin	schulz6@llnl.gov	Lawrence Livermore National Laboratory
Seager, Mark	seager1@llnl.gov	Lawrence Livermore National Laboratory
Shalf, John	jshalf@lbl.gov	Lawrence Berkeley National Laboratory
Shephard, Mark S.	shephard@scorec.rpi.edu	Rensselaer Polytechnic Institute
Shoshani, Arie	shoshani@lbl.gov	Lawrence Berkeley National Laboratory
Silva, Claudio	csilva@cs.utah.edu	University of Utah
Sterling, Thomas	tron@cct.lsu.edu	Louisiana State University
Stevens, Rick	stevens@mcs.anl.gov	Argonne National Laboratory

Last /First Name	E-mail	Affiliation
Straatsma, Tjerk	tps@pnl.gov	Pacific Northwest National Laboratory
Strayer, Michael	michael.strayer@science.doe.gov	U.S. Department of Energy, Office of Science
Streitz, Fred	streitz1@llnl.gov	Lawrence Livermore National Laboratory
Strout, Michelle	mstrout@cs.colostate.edu	Colorado State University
Swart, Pieter	swart@lanl.gov	Los Alamos National Laboratory
Tang, William	tang@pppl.gov	Princeton Plasma Physics Laboratory
Tartakovsky, Daniel	dmt@ucsd.edu	University of California, San Diego
Tong, Charles	chtong@llnl.gov	Lawrence Livermore National Laboratory
Trease, Harold	harold.trease@pnl.gov	Pacific Northwest National Laboratory
Tufo, Henry	tufo@ucar.edu	National Center for Atmospheric Research
Turnbull, Susan	susan.turnbull@science.doe.gov	U.S. Department of Energy, Office of Science
Turner, John	turnerja@ornl.gov	Oak Ridge National Laboratory
Underwood, Keith	keith.d.underwood@intel.com	Intel Corporation
Van Straalen, Brian	bvstraalen@lbl.gov	Lawrence Berkeley National Laboratory
Vander Wiel, Scott	scottv@lanl.gov	Los Alamos National Laboratory
Vannoy, Lucy	lucy.vannoy@pnl.gov	Pacific Northwest National Laboratory
Vetter, Jeffrey	vetter@computer.org	Oak Ridge National Laboratory and Georgia Institute of Technology
Wampler, Cheryl	clw@lanl.gov	Los Alamos National Laboratory
White, Andrew	abw@lanl.gov	Los Alamos National Laboratory
Wisniewski, Robert	bobww@us.ibm.com	International Business Machines
Worley, Patrick	worleyph@ornl.gov	Oak Ridge National Laboratory
Xiu, Dongbin	dxiu@purdue.edu	Purdue University
Yang, Ulrike	yang11@llnl.gov	Lawrence Livermore National Laboratory

APPENDIX 4: ACRONYMS AND ABBREVIATIONS

µsec	microsecond
AFAC	Asynchronous Fast Adaptive Composite Grid (algorithm)
AMR	adaptive mesh refinement
API	application programming interface
ASC	Advanced Simulation & Computing (National Nuclear Security Administration)
ASCR	Office of Advanced Scientific Computing Research, U.S. Department of Energy
BSP	bulk synchronous parallel
CDV	co-design vehicles
CIFTS	Coordinated Infrastructure for Fault Tolerant Systems
CPU	central processing unit
CUDA	Compute Unified Device Architecture (NVIDIA)
DAG	directed acyclic graph
DOE	U.S. Department of Energy
DRAM	dynamic random access memory
FFT	Fast Fourier Transform
FLOPS	FLoating point OPerations per Second
FMA	floating-point multiply-adds
GAS	global address space
GB/s	gigabytes per second
GF	gigaflop
GHz	gigahertz
GPU	graphics processing unit
HPC	high-performance computing
HPCS	high productivity computing systems
I/O	input/output
IDL	Interactive Data Language
IESP	International Exascale Software Project
kW	kilowatt

MEXT	(Japanese) Ministry of Education, Culture, Sports, Science and Technology
MPI	message passing interface
MW	megawatt
NetCDF	Network Common Data Form
NNSA	National Nuclear Security Administration
ns	nanosecond
NVRAM	nonvolatile random-access memory
OOC	out-of-core
OS	operating system
PB	petabyte
PDE	partial differential equation
PGAS	partitioned global address space
PLASMA	Parallel Linear Algebra for Scalable Multi-core Architectures
POSIX	Portable Operating System Interface [for Unix] (current meaning: a family of Institute of Electrical and Electronics Engineers standards for operating systems)
PRD	priority research direction
R&D	research and development
RAM	random-access memory
RIKEN	Rikagaku Kenkyūsho, which, in Japanese, means: "The Institute of Physical and Chemical Research"
SAS	SAS language (originally "Statistical Analysis System")
SPMD/BSP	single program, multiple data
SpMV	sparse matrix-vector multiplication
SPSS	IBM SPSS language (originally "Statistical Package for the Social Sciences")
TB	terabyte
TLB	translation lookaside buffer
UPC	Unified Parallel C
UQ	uncertainty quantification

U.S. DEPARTMENT OF
ENERGY

Office of Science

Office of Advanced Scientific Computing Research,
Office of Science

Office of Advanced Simulation and Computing,
National Nuclear Security Administration

Prepared for the U.S. Department of Energy under Contract DE-AC05-76RL01830

Production support provided by Pacific Northwest National Laboratory,
Fundamental & Computational Sciences Directorate

PNNL-20168