CROSSCUTTING TECHNOLOGIES FOR COMPUTING AT THE EXASCALE

DRAFT

TABLE OF CONTENTS

CROSSCUTTING TECHNOLOGIES FOR COMPUTING AT THE EXASCALE 1
TABLE OF CONTENTS 2
EXECUTIVE SUMMARY 4
Topic Area 1: Algorithm and Model Research and Development Needed to Support New Architectures 4
Topic Area 2: Research and Development for Programming Models to Support Exascale Computing 4
Topic Area 3: Research and Development for System Software at the Exascale 5
AUTHORS 6
INTRODUCTION 8
Workshop Logistics 8
Crosscutting Findings 9
Accelerated Scientific Discovery Through Co-design 12
FUTURE ARCHITECTURE AND SOFTWARE DEVELOPMENT 15
MATH MODELS AND ALGORITHMS 21
Introduction 21
Critical mathematical models, methods and algorithms that support scientific discovery through computing 23
Partial Differential Equations 23
Applied Math Challenges in Mission-Driven Data Analysis and Visualization 32
Uncertainty Quantification/Stochastic Systems 43
UQ SIDEBAR----------------------- 50
Discrete Mathematics 52
Optimization and Solvers 55
Opportunities for Co-design of Exascale Architectures Driven by Algorithm and Application Needs 59
Cross-cutting Challenges and Research Needs in Algorithm and Model Research and Development Needed to Support New Architectures 62
PROGRAMMING MODELS AND ENVIRONMENTS 65
Introduction 65
Understanding programming model requirements at the exascale in the context of critical mathematical models, methods and algorithms 66
Partial Differential Equations 66
Data Analysis and Visualization 70
Uncertainty Quantification 74
Discrete Math 76
Solvers and Optimization 78
Cross-cutting Challenges and Research Needs for Programming Models to Support Exascale Computing 78
SYSTEM SOFTWARE 82
Introduction 82
Understanding mathematical models, methods and algorithms in the context of system software ..82

DRAFT

Understanding PDEs in the context of System Software ... 82
Understanding Data Analysis and Visualization in the context of System Software ... 84
Understanding Uncertainty Quantification (UQ) in the context of System Software ... 86
Understanding Solvers and Optimization in the context of System Software ... 86
Understanding Discrete Math in the context of System Software ... 86
Co-design Opportunities between Discrete Math and System Software ... 87
Cross-cutting system software themes for all break-out areas ... 88
Cross-cutting Challenges and Research Needs for System Software at the Exascale ... 89

CONCLUSIONS AND RECOMMENDATIONS ... 92
Topic Area 1: Algorithm and Model Research and Development Needed to Support New Architectures ... 92
Topic Area 2: Research and Development for Programming Models to Support Exascale Computing ... 92
Topic Area 3: Research and Development for System Software at the Exascale ... 93

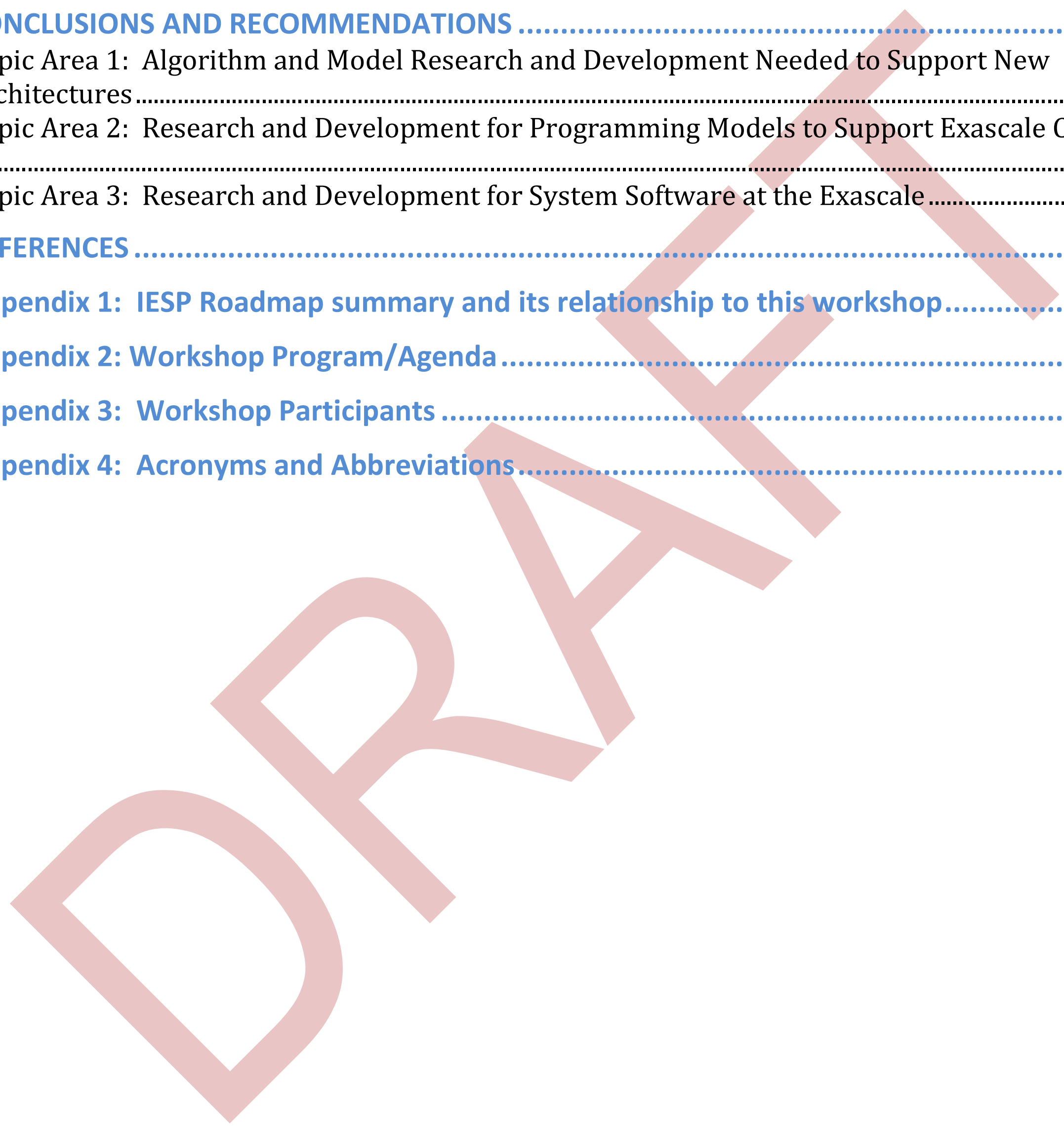
REFERENCES ... 94

Appendix 1: IESP Roadmap summary and its relationship to this workshop ... 95

Appendix 2: Workshop Program/Agenda ... 96

Appendix 3: Workshop Participants ... 97

Appendix 4: Acronyms and Abbreviations ... 98



EXECUTIVE SUMMARY

The Priority Research Directions (PRDs) that the breakout panels identified in their workshop discussions are summarized as follows. Note that the details provided here are preliminary and subject to change in the final workshop report.

[Final Executive Summary will be written later]

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

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

PRD 1.2: *Develop new mathematical models and formulations that effectively exploit anticipated exascale hardware architectures*

PRD 1.3: *Address numerical analysis questions associated with moving away from bulk-synchronous programs to multi-task approaches*

PRD 1.4: *Adapt data analysis algorithms to exascale environments*

PRD 1.5: *Extract essential elements of critical science applications as "mini-applications" that hardware and system software designers can use to understand computational requirements*

PRD 1.6: *Develop tools to simulate emerging architectures 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 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*

DRAFT

PRD 2.8: *Develop programming model support for fault tolerance/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; reconfigurable local memory)*

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

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

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

DRAFT

DRAFT

AUTHORS

Workshop chairs:

- David L. Brown, *Lawrence Livermore National Laboratory*
- Paul Messina, *Argonne National Laboratory*

Workshop theme leads:

- Peter Beckman, *Argonne National Laboratory*
- David Keyes, *Columbia University and King Abdullah University of Science and Technology*
- Jeffrey Vetter, *Oak Ridge National Laboratory*

Workshop breakout session leads:

- Mihai Anitescu, *Argonne National Laboratory*
- John Bell, *Lawrence Berkeley National Laboratory*
- Ronald Brightwell, *Sandia National Laboratories*
- Brad Chamberlain, *Cray, Inc.*
- Donald Estep, *Colorado State University*
- Al Geist, *Oak Ridge National Laboratory*
- Bruce Hendrickson, *Sandia National Laboratories*
- Michael Heroux, *Sandia National Laboratories*
- Rusty Lusk, *Argonne National Laboratories*
- John Morrison, *Los Alamos National Laboratory*
- Ali Pinar, *Sandia National Laboratories*
- John Shalf, *Lawrence Berkeley National Laboratory*
- Mark Shephard*, Rensselaer Polytechnic Institute*

DRAFT

INTRODUCTION

A technical workshop to discuss Crosscutting Technologies for Computing at the Extreme Scale was held February 2-4, 2010, in Rockville, Maryland. The U.S. Department of Energy (DOE) Office of Advanced Scientific Computing Research and the National Nuclear Security Administration co-sponsored this collaborative workshop. The purpose of the workshop 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 the previous eight Grand Challenge Science workshops held in 2008 and 2009 and, in particular, to address the following:

- Determine the research and development required in mathematical models, applied mathematics, and computer science to meet the needs of a spectrum of applications at the exascale;

- Provide recommendations on exascale architectures and configurations for those architectures; and,

- 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 produce a computational science discovery environment), including the role of testbeds.

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 also were present as observers.

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

The crosscutting technologies workshop opened with a series of plenary presentations to establish a context for breakout session discussions. These presentations addressed the following: mathematical models and algorithms, computer architecture, system software, programming models and environments, and co-design strategies. The workshop agenda and the plenary presentations may be found on the DOE Scientific

DRAFT

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
- Data Analysis and Visualization
- Uncertainty Quantification 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 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 significant re-engineering of 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 would require a significant acceleration in the development of both hardware and software. This can only be accomplished through an intensive "co-design" effort, where system architects, application software designers, applied mathematicians, and computer scientists work interactively to produce an

DRAFT

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 a combination of increased processor speed and increased 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 computing. Likely, this increased concurrency will occur predominantly on-chip, meaning that nodes with 1,000- 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; alternative programming models; or, most likely, a hybrid programming model that combines MPI with other models, such as OpenMP or OpenCL. 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 inter-node programming model, the intra-node programming model likely will be quite different because of the need to manage very large-scale threaded parallelism.

As total concurrency and node concurrency increase dramatically -- unless new technologies emerge --, cost constraints dictate that neither system memory nor interconnect bandwidth can increase correspondingly. Thus, high-performance computers at the exascale will exhibit 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 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.

While this redesign and reimplementation of scientific application codes will be disruptive, it does present a significant opportunity to reconsider how computational science might be performed at the exascale. Driven 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

DRAFT

predictive science, establishing clear levels of confidence in the calculations by directly incorporating uncertainty quantification techniques. In addition, simulations will increasingly be used not only for discovery science, but also as a basis for informing policy decisions through, for example, determination of optimal results or designs.

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

1. *Significant new model development, algorithm re-design and science application code re-implementation, supported by (an) exascale-appropriate programming model(s), will be required to exploit effectively the power of exascale architectures.* The 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 1990's. An intensive "co-design" effort will be essential for success, where system architects, application software designers, applied mathematicians, and computer scientists work closely together to produce a computational science discovery environment that fully leverages the significant advances in computational capability that will be 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 significant code redesign required for the exascale will present an opportunity to embed uncertainty quantification 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.* Exascale simulation codes will increasingly see analysis capabilities, uncertainty quantification calculations, and performance measurements and predictions embedded in the application codes themselves as operations computation and local storage increasingly become cheaper than interprocessor communication or I/O.

4. *Memory management will be a significant challenge for exascale science applications due to their deeper, complex hierarchies and relatively smaller capacities, and dynamic, latency-tolerant approaches must be developed.* Tools and interfaces must be developed to manage and control memory for run-time systems, as well as to provide information to assist the compiler in optimizing

DRAFT

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 for application-specific fault recovery. Moreover, future exascale systems may be forced to use alternative I/O architectures, such as large-capacity, non-volatile RAM, that may provide both new challenges and opportunities for managing resiliency.

6. *Automated, dynamic control of system resources will be required.* Adaptive run-time 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, also must 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 uncertainties 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. In addition to co-design efforts for developing exascale simulation capabilities, this goal also depends critically on similarly deliberate strategies for co-design of other national assets, such as accelerators and 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 change the scale of challenges to which the scientific method can aspire.

Algorithms and codes must be agile and more modular for the single-physics and integrated packages that will support the science, technology, and engineering

DRAFT

problems identified in the eight Grand Challenge Science 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:

a) 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;

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

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 configurations in the exascale regime

- Identify and pursue research and development 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, designers of programming models, 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. As with past experience, there may be breakthroughs in hardware technologies that result in different micro and macro architectures becoming feasible and desirable, but they will require rethinking of certain algorithmic and system software implementations.

As simulation capability and fidelity improves, 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 has 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.

DRAFT

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 the study of complex phenomena cooperatively. Achieving this qualitatively higher level of interdisciplinary cooperation and integration of major assets is essential if scientists are to meet the great scientific, national security, and societal challenges of our time—health, energy, defense, and information management.

DRAFT

FUTURE ARCHITECTURE AND SOFTWARE DEVELOPMENT

Passage to the exascale introduces interesting incentives for taking application codes beyond their nearly universal roots in classical bulk synchronization. There are sources of nonuniformity from the hardware and the systems software that are too dynamic and too unpredictable or difficult to measure to be consistent with bulk synchronization. On the hardware side, these include: manufacturing nonuniformities, dynamic power management, and runtime component failure. On the systems software, they include: OS jitter, software mediated resiliency, and TLB/cache performance variations. Network nonuniformities can arise from either hardware or software reasons. The nonuniformities that we have traditionally sought to balance in the numerical models, including: physics at gridcell (e.g., table lookup, equation of state, external forcing), discretization adaptivity, solver adaptivity, precision adaptivity, are – from an operational viewpoint – interchangeable with these other sources. We note that any efforts that we need to make at the exascale to accommodate the union of these sources of nonuniformity will help us at the petascale and the terascale, where the systems-related concerns are much smaller, but the opportunities available if we could better tolerate model-related nonuniformities are extensive!

Co-design of architecture, software, and algorithms has emerged as a necessity to negotiate the physical, power, and cost limitations of exascale systems. Though it inveighs against the computer science aesthetic of “separation of concerns,” co-design has been practiced for two decades by message-passing programmers, who, in effect, write a piece of the runtime system when they write software libraries and applications. The attitude and aptitude required for co-design is therefore *not* entirely unfamiliar and unwelcome, though it does greatly expand in scope when the hardware design, itself, becomes involved. As in the case of scientific software toolkits based on message-passing, we aim to isolate applications programmers from many hardware and software architectural details of exascale execution, just as we do today from message-passing details at the petascale.

One factorization of hardware resources that achieves exascale simulation is the product of billion-way concurrency with gigaHertz cores. Compared to the (nearly) million-way parallelism of gigaHertz processors that achieves petascale simulation today, the main development that must be accomplished is to expand today’s million-way flat parallelism at the node level with thousand-way parallelism within a node. This offers a pseudo-evolutionary path for applications, in that MPI-based legacy code will still be usable on the million nodes. Changes will be mainly within node, where we will need to evolve thousand-way parallelism: “MPI+*X*”. Under the MPI+*X* model, the principal challenges from peta to exa are within the node, and the burden of migration

DRAFT

from a handful to a million process threads per node is shared by the marketplace, at all incremental scales of node aggregation. Under this incremental hardware path to the exascale, loosely speaking, the algorithmic path from peta to exa preserves the focus on weak scaling between nodes and adds (mostly) strong scaling within nodes. The big challenge is the strong scaling within the nodes. There is sufficient concurrency in tasks like PDEs for thousand-fold strong scaling if sufficiently memory is available (see., *e.g.,* the 2008 Gordon Bell Prize finalist paper by Burstedde, *et al.*), which was mediated by MPI alone. A key consideration in expanding from today's million-fold thread concurrency to billion-fold 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. Dynamic load balance through work stealing is relatively more sufferable in this architecture, since remote DRAM access is not inordinately faster than local DRAM access. The cost is in the vertical and the horizontal is not significantly greater.

Domain decomposition is the weak-scaling method of choice, for PDE-derived discrete systems. It allows asymptotically linear scaling of nodes with problem size, per iteration, given log-diameter global reductions. It may have an asymptotically constant number of outer nonlinear iterations, given a superlinearly to quadratically convergent inexact Newton method. Under appropriate physics-based, Schwarz-based, or multigrid-based preconditioned Krylov, 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 today's petascale machines. Continued research is necessary on preconditioning that extends such results from where the beacon of theory shows how to aim to more and more complex mathematical operators.

Resiliency is an important consideration that also demands much research in going from petascale to exascale; however, it should not be showstopper despite the fearsome factor of a thousand times more threads on the critical path of the computation. First, the number of physical nodes is not much greater since most of the parallelism occurs within a node. Hardware failure rates, after the chips are burned in and certified following manufacture, do not grow 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 workingset that varies in size periodically and predictably, reaching minimal size at each timestep 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, can afford to be lost. 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 like soft memory errors occur in parts of the code that are not on the critical path of state evolution, but 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 in tact, 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 dance of co-design that is unprecedented to date at the petascale.

One of the most important requirements of the exascale for applications is the reduction of synchronization. While the elimination of periodic synchronization appears impossible, application writers and library designers can write code in styles requires significantly less *artifactual* synchronization than today. This can be illustrated with recourse to an implicit loop in a typical PDE-based application. On the critical path of such a solve is a loop that consists of forming a global residual, performing a solve to obtain a correction, bounding the step of the correction, and updating physical state. This loop, which embeds various global reductions, cannot easily or apparently be avoided. However, we often insert into this path things that could be done less synchronously, because we have limited language expressiveness to indicate that their importance is secondary.

For instance, Jacobian refresh and preconditioner refresh can be taken off the path. In the best implementations, they are already done infrequently relative to use, to amortize their relative high cost of formation and application. They must be refreshed for theoretical attainment of convergence in nonlinear problems, but if some processors are in arrears heading to the next synchronization, they can be deferred. This is most easily done if the code to refresh them is isolated from the code that uses them, and 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 that they are very time-consumptive 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 subthreads that perform constitutive updates, algorithmic parameter adaptation, I/O, compression, visualization, data mining, uncertainty quantification, and so forth.

In addition, in pursuit of synchronization reduction, “additive operator” versions of sequentially “multiplicative operator” algorithms are often available. Such algorithms

DRAFT

have sometimes received a bad rap historically, e.g., "chaotic relaxation" as proposed by Chazan & Miranker in 1969, for instance. In retrospect, these were ahead of their time in that the hardware of the day did not drive the demand or hold out the opportunity that it does today. It has been shown that asynchronous additive methods can sometimes be made virtually as good as their multiplicative cousins, for instance "AFACx" versus "AFAC," (Lee, McCormick, Philip & Quinlan, 2003, for instance).

To take full advantage of such synchronization-reducing algorithms, we need to develop greater expressiveness in scientific programming. We need to be able to create separate subthreads 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 may already exist that can be put to use in this way, for instance the Asynchronous Dynamic Load Balancing (ADLB) construct of Lusk, et al., 2009, or prioritized forms of tuple-space programming. In adopting such programming styles, we may need to introduce "soft barriers" to prevent state from getting too stale.

Exascale programming will require prioritization of critical path and noncritical tasks, adaptive DAG 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 be 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. Backing up still further, of course, better mathematical formulations may be available. A trend to be exploited where possible, for instance, is the replacement of partial differential equations with integral equations, or of preconditioners for partial differential equations with integral equation-mediated operations.

A very exciting prospect in programming exascale hardware is to exploit mixed precision floating point arithmetic, striving to use the lowest required in a local tasks to achieve a given global accuracy outcome. The benefits of lower precision have been dramatically demonstrated on heterogeneous processors such as those built from GPGPUs. The promises include reduced execution times, reduced communication times, and reduced power consumption. Such techniques are not unknown in petascale practice. In fact, for over a decade they have been in practice in DOE warhorse codes such as PETSc, in which the preconditioner elements are stored and communication in single precision by default, whereas the Krylov computation is performed in double precision. Algorithms such as iterative refinement, which make use selective use of higher precision are classical in numerical analysis. The key is to reformulate algorithms to find corrections, rather than solutions. The solutions must be stored in

DRAFT

desired precision, but the corrections, which ultimately affect only the lower bits of the solution, can be calculated in lower precision.

Progress in programming exascale machines will be impeded if we continue to employ suboptimal metrics as the arbiter of the quality of a computation. Since, at the architectural balance, a flop is by far cheaper to provision and to power up, relative to a byte memory or an improvement in memory transfer rate, good designs over-provision flops, so that they are never limiting in the course of a computation, relative to the cost of storing and communicating their operands. Percentage of peak flop/s is therefore a counter-productive metric, since it penalizes for over-provisioning something cheap. Percentage of the instantaneously limiting resource, typically memory bandwidth, is the most interesting figure of merit for a given execution of the "right problem."

The "right problem" is the one that provides the requisite accuracy with the requisite confidence at some combination of lowest energy and shortest execution time. It is a grand challenge of numerical analysis to equidistribute the multitudinous errors in a computation as to achieve the target result without "oversolving" or "overresolving" some contributing step. 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, since the balance of resources required to fit a computation into a given memory and energy budget is increasingly delicate.

In contemplating the direct map of petascale applications to the exascale, the following issues present themselves as the greatest challenges. Each one of them is an exacerbation of an issue that already exists at the petascale:

- Poor scaling of global reductions due to internode latency

- Poor performance of sparse matrix-vector multiply (SpMV) due to shared intranode bandwidth

- Poor scaling of coarse grid solutions due to insufficient concurrency

- Lack of reproducibility due to floating point noncommutativity and algorithmic adaptivity (including autotuning) in efficient production mode

- Difficulty of understanding and controlling vertical memory transfers

Software engineering practices that have already become well established at the petascale for cyberinfrastructure in common use today should remain the same as we transition to the exascale, to leading order. However, the opportunities of co-design may force some compromises. One of the most important of these practices is multilayer software design.

DRAFT

Successful software is multilayered in its accessibility. The outer layer, which is accessible to all users, is an abstract interface featuring language of application domain, hiding implementation details, with conservative parameter defaults. For instance, 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 opened up to experienced users through handles to the underlying objects, provide a rich collection of state-of-the-art methods and data structures, exposed upon demand, and variously configurable. The goals of access are capability, algorithmic efficiency, extensibility, composability, and comprehensibility of performance and resource use through profiling. The inner layer is intended for developer access only. It includes support for variety of hardware and software execution environments. The goals of access are portability and implementation efficiency.

DRAFT

MATH MODELS AND ALGORITHMS

Introduction

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 4/3rds of the memory capacity, in a naïve stability-limited explicit context. This superlinear scaling is consistent with the direction of memory-thin architectures being proposed for exascale architecture. These architectures fundamentally depart from the classical Amdahl-Case Rule of computer architecture, which would provide for linear scaling of bytes and byte/s with flop/s. They take to further extremes the departures of the terascale and petascale eras, during which application writers have reluctantly and with difficulty, in many cases, weaned themselves of the luxury of large memories.

The core mathematical formulations involved in the forward problems of these driving applications, whose "first principles" or other representations typically involve particles and fields, are: ordinary, partial, and integro-partial differential equations. (The rubric of "ordinary differential equations" here includes particle models, which may, in fact, be solved through means more specialized to exploit their physics than general-purpose ODE software. "ODEs" here also represents the operator split parts of PDE codes that involve intense zero-dimensional computation in every cell, such as chemical reaction terms and exchanges between energy groups.) After discretization, the computational formulations of these problems depend further upon a range of discrete mathematics for manipulating objects like grids, sparse matrices, task graphs, and the source code, itself.

There is intrinsic interest in passage to the exascale for individual codes in these driving scientific and technological campaigns (capability). There is additional interest in their multiphysics combination (complexity). There is still further interest in solving inverse problems, sensitivity analysis, and uncertainty quantification (understanding).

Today, these applications scale to the edge of National Leadership Class Facilities (NLCF) platforms, using primarily bulk synchronous MPI-based implementations on statically load-balanced partitions. Such codes are entered in the Gordon Bell Prize competition annually, and they occasionally win the "special" prize, on the basis of their scaling, though they generally do not achieve a high percentage of peak flop/s. This is due to known memory bandwidth limitations of current hierarchical memories, in which access to DRAM is approximately two orders of magnitude longer than a processor

DRAFT

cycle, yet elements of (typically) the largest workingset in the problem, a Jacobian matrix, have poor locality.

For the earliest exascale driver applications, we naturally look to today's petascale applications. (We anticipate that new applications will join them as creative scientists and engineers are afforded headroom, but petascale applications are already ripe – a factor of one thousand in capability is actually a *modest* step in resolution when spread into three space dimensions, or even higher dimensions in problems posed in phase space, such a Boltzmann formulations.) Today's petascale applications are built on a large number of software toolkits, most of which have been developed by the Department of Energy and are actively maintained on NLCF 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, uncertainty quantification, dynamic load balancing, graphical and combinatorial algorithms, and compression.

Code development effort at the petascale relies 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 systems, data miners, workflow controllers, and frameworks. These toolkits will all need to be supported on emerging exascale machines, where they will need to be supplemented beyond their counterparts today by additional tools, such as: fault monitoring, fault reporting to the application, and fault recovery.

We conclude this section by recalling some general characteristics of these applications that are relevant to challenges that await at the exascale.

Fully Implicit Coupling. Temporally implicit, fully coupled formulations are often preferred for multirate applications whose dynamics of interest are in scales that are much slower than the fastest wavespeeds 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 towards full coupling. Also, stability theory for loose coupling of nonlinear problems is incomplete, at best. Synchronization at every (macro) time step at which we wish to observe the solution cannot be avoided, and more frequent synchronization is built into today's algorithms (artifactually and removably in some cases). Implicit methods potentially improve on explicit by offering lots of useful work *between* synchronizations. They advance to a given physical time on a much smaller "budget" of synchronizations.

DRAFT

Lack of Amenability to Time Parallelism. Scientists often express the desire for strong speed-up in order to reduce the execution time of large-scale simulations, and sometimes propose parallelization in the temporal dimension of the model in pursuit of this aim. Temporal parallelization is an important research topic in its own right, at many scales smaller than exascale. However, opportunities for parallelization in time are limited at the exascale by memory considerations. Parallelization in time typically requires simultaneous storage of multiple copies of the spatial discretization at successive temporal stages.

Adaptivity. Solution-based adaptivity, which is highly desirable for numerical efficiency, is another challenge at the exascale, at least the way adaptation is practiced today. The adaptive step typically performs it own synchronization and must be followed by a dynamic *in situ* load balance, and since the classical SPMD bulk synchronous paradigm requires statically balanced work per physical degree-of-freedom per major iteration.

Critical mathematical models, methods and algorithms that support scientific discovery through computing

Partial Differential Equations

A wide range of DOE applications focus on the solution of partial differential equations, including accelerators, astrophysics, climate, combustion, defense science, fusion, nuclear energy, and subsurface flow. A common characteristic of most of these applications is that they span a broad range of spatial and temporal scales and they 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 adaptive mesh refinement (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 theses interfaces such as 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 separation of scales 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 range of techniques ranging from simply operator-splitting approaches to iterative approaches to couple the processes in a fully implicit approximation.

Another feature of many emerging applications is simulations that include different models to describe different parts of the simulation. One example of this type of

DRAFT

simulation would be the use of molecular dynamics at the tip of a crack in a solid coupled to a traditional elasticity model away from the crack. Another type of example in which this type of simulation arises is climate models that include oceans, atmospheres, land surfaces and sea ice in a single simulation.

Co-Design: Effect of Hardware Constraints on the Algorithms

Development of these methodologies as we move toward exascale simulation will require a significant research effort in applied mathematics. Some specific areas of research that will be needed to develop accurate, robust and efficient simulation methodologies include:

1. There is an overall need to rethink how we discretize PDE's that reflect the character of proposed exascale architectures. There are several key themes that need to be considered:
    a. Rethink discretization PDE's to reflect a paradigm shift in architectures from FLOP constrained to memory constrained. 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 we approach fundamental discretization issues.
    b. Similar to the above, there is also an overall need to try to develop algorithms that compute more for a given level of communication. For example, is it possible to use potential theory type ideas to develop approaches for solving elliptic PDE's that require significantly less communication than traditional iterative approaches
    c. Related to (b), there is a need to think about how to develop algorithms that can operate more asynchronously than traditional algorithms based on a bulk synchronous execution model. A key here is to understand how asynchronous operations might affect accuracy and stability properties
2. As we develop approaches for increasingly complex physical problems we need to perform a detailed analysis of adaptive mesh algorithms for multiphysics applications. In particular, we need to quantify the accuracy and stability properties of approaches to coupling different processes across different levels of refinement.
3. Similar to 2, we need to perform a detailed analysis of temporal coupling strategies for multiphysics applications. As we move to 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.

DRAFT

4. The increased compute power of exascale computing will enable us to consider new classes of multiscale problems in which we employ different types of discretizations, appropriate to a particular scale in different portions of the domain. As this type of simulation expands, there is a critical need to develop systematic approaches for coupling across the range of scales and for a 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.
5. Implicit within the PDE area is also a need for research to develop effective iterative methods for the sparse linear and nonlinear systems that arise in discretizing PDEs.
6. Although by its nature not well-defined, there is also significant merit in considering alternative approaches to formulating PDE's that could offer dramatically different approaches to solving problems at the exascale.

Completely New approaches to discretizing PDE's

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 utilization of these machines will require that we be able to effectively exploit parallelism at the node level. For PDEs this means that we need to be able to exploit parallelism within the subdomains inside the nodes. This observation suggests several major co-design opportunities in the PDE area as well, primarily with the programming models and environments area. In particular, the applied math PDE community needs to work with the programming models and environments community to develop effective abstractions to expose both loop-level parallelism and data-level parallelism for computations on subdomains. In particular, PDE algorithm developer need to be able to have relatively simple approaches to specifying fine-grained parallelism that can be efficiently translated into a light-weight parallel implementation. In addition, the PDE community needs to work with the programming models and environments community to develop approaches for expressing asynchronous algorithms. The ability to specify asynchronous operation is a critical issue for hiding communication latency and for avoiding load-imbalance issues. Issues relating to moving away from a block synchronous execution model become increasingly important as we consider a broader range of multiscale problems.

Co-Design: Effect of Algorithm and Hardware Requirements on the Programming Environment

The PDE community needs to work with the programming models and environments community to develop approaches for expressing asynchronous algorithms. The ability to specify asynchronous operation is a critical issue for hiding communication latency and for avoiding load-imbalance issues. Issues relating to moving away from a block synchronous execution model become increasingly important as we consider a broader

DRAFT

range of multiscale problems. 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 will require advanced load balancing methods.

Co-Design: Effect of Algorithm Requirements on the Hardware

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

On-chip memory: Given the importance minimizing unnecessary data movement, the ability to allow software control of data movement for performance-critical kernels would be beneficial. Methods that can co-exist with conventional automatically managed caches are important because an incremental porting path is required. Support of some form of global address space would be desirable.

Global Addressing*:* There was agreement that message passing could probably be used for inter-node communication for exascale systems, but is clearly not practical for inter-processor communication at billion way parallelism. Global memory addressing is preferred over cache-coherence for managing fine-grained computation and data movement on-chip. It would be even more beneficial for supporting advanced algorithm development if global addressability could be extended across the system.

Fast Reductions: Virtually all implicit timestepping methods rely on Krylov subspace methods, which require fast inner product summations. At system scale, this appears as a local summation followed by an “all reduce” global collective operation. There is a critical need for hardware support for fast allreduce because such steps are latency limited. Hardware support for fast global collectives has been demonstrated to be valuable on BlueGene. Continue to advance such support for this important class of numerical algorithms. Given the move towards asynchronous execution models, this capability would need to be generalized to support for asynchronous collective operations.

The PDE II math area considered the following aspects of the PDE solution:

- Discretization methods
- Multiple physics simulations

DRAFT

- Computational geometry
- Meshing

The value in considering this list of topics is that the entire work flow associated with the PDE solution process, from mesh generation to solution and visualization, can be discussed together. In general, the work flow starts with 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 given 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. The numerical discretization is constructed from this space time mesh and then solved. The resulting solution is analyzed to determine if the combination of the mesh and its discretization has yielded the desired level of accuracy. In almost all cases, the initial mesh/discretization is not sufficient and must be adapted. From many problem classes of interest the steps of mesh/discretization adaptation, solve and evaluation are repeated many times. We note that the discretization and solution techniques chosen can vary considerably and range from techniques with local support such as finite elements methods to those with global support such as spectral methods. Solution methods can use implicit or explicit solvers, can 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 will perform on an exascale computer and the type of research that will be required to address the development of new methods to ensure success.

Some key characteristics with respect to typical work flows for PDE solution for exascale size problems are:

- 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.) During a discretization/solve step the communication patterns are fixed and work loads can be accurately predicted.

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

DRAFT

- In the case of unstructured mesh generation there are techniques that can automatically create the meshes, given the mathematical definition of the domain. These mesh generation processes are highly irregular and dominated by substantial communications which become increasingly local as the mesh is being formed. In the case of general geometric domains it is not possible to determine tight predictions of local work loads thus making dynamic load balancing critical.

- The mesh adaptation process has many of the same characteristics with the advantage that the existence of the mesh to be adapted provides more localization of the processes. However, it is important to note that although the operations are local, the specifics are not known until operations are attempted, thus causing unpredictable computation loads, communication 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 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 from local on-node transfers of information to substantial many-to-many transfers of information across significant portions of the computational domain.

- Since 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. The one potential exception to this is initial mesh generation in those cases where it is known that a very coarse initial mesh can be used so that the initial input is “small”.

- One means to reduce the amount of simulation data stored and to make simulations more effective is support of in situ visualization and feed back control to change simulation conditions (boundary conditions, geometry, etc.) as simulations proceed. Such capabilities require the ability interact with connected visualization systems to effectively extract and communicate appropriate data and control.

These are but a few key aspects of the solution of the PDEs on exascale computers that are likely 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.

As a side note, we would like to point out that members of the PDE II group have been successful in getting key portions of this work flow for local, finite-element based methods on unstructured meshes running on current massively parallel computers. In

DRAFT

particular, they have demonstrated the ability to discretize and solve, using implicit methods, problems with excellent strong scaling to 288K cores. They have also executed key aspects of unstructured mesh adaptation on 32K cores.

Given this work flow, it is clear that the automated adaptive solution of PDEs on exascale machines will tax all aspects to the exascale system's computation and communication fabric and that our key concern with respect to the hardware is what constitutes a good balanced system.

PDE Applications R&D

Clearly there are many developments that will need to be done by PDE application developers to effectively use exascale machines. Note that it is easy to say applied mathematicians need to be disruptive and define new methods/algorithms for exascale machines. However, there is an unknown degree of risk with respect to determining successful methods. Some of the areas of research and development identified include:

- Newton-Krylov methods - Can we use techniques with FAS methods that better respect hard limitations on memory?
- Data layout and multi-precision algorithms for reducing cost of accessing memory. How do such methods affect convergence and other numerical properties?
- Development of methods to effectively increase the resiliency of exascale simulations including effective local check pointing, algorithms to effectively recover for detected errors, and algorithms that use available computation resources to help determine when errors have occurred in critical code segments.
- Can map reduce transaction computing be used to increase the resiliency of PDE simulations in an effective manner on exascale computers.
- More scalable spectral methods that use FFTs. Are 2, 3 or more level hierarchical methods possible; are more local algorithms (real space algorithms to split the system)?
- Mesh generation on exascale machines – not likely to scale fully, but must be done "effectively" on the machine to avoid I/O costs.
- Parallel in time; 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.
- Loosely coupling current parallel methods by developing interface conditions

DRAFT

- Taking advantage of extra flops for useful work (e.g. for resilience computations or re-compute tables rather than accessing from memory)
- More effective parallel implementations of multiphysics problems that look carefully at the best execution of node level multiphysics operations.
- Fast dynamic load balancing algorithms for dealing with operations that are local to neighborhoods and do not involve substantial computation or date exchange.
- Dynamic load balancing methods to deal with multilevel adaptive computations such as multiscale methods both concurrent methods and hierarchic methods (including subgrid models).
- Consideration of the most scalable discretization methods to deal with the increased levels of parallelism required with specific consideration of the node level parallelism.
- Development of the 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 the visualization nodes to the display system.
- There is a need to recognize the legacy investments that have already been made to get to petascale. Many of them will be part of getting to effective simulations on exascale machines.

Interplay of Machine Characteristics with PDE methods and workflows:

The machine characteristics that are most critical in the solution of PDEs are a function of the simulation workflow.

During the discretization/solution process the node characteristics of interest are:

- Effective use of the heterogeneous computing units (CPU's and GPU's) and their interactions to gain a high degree of parallelism is critical.
- Effectively accounting for the memory hierarchy and access will be important to gaining node level performance.
- Some techniques will have difficulties dealing with the memory limitations. Other techniques can effectively scale with the limited memory, but there is a limit to the total size problem that can be solved.
- OS jitter must be essentially non-existent, otherwise scaling will be strongly compromised.
- Some techniques (e.g., spectral methods based on FFTs) place much heavier requirements on inter-node communications with currently know techniques.

DRAFT

Although heavily dominated by local calculations the discretization/solution process does require neighbor communications and at least limited global synchronized communications.

The mesh generation and mesh adaptation processes, particularly for unstructured meshes, are unlikely to be able to take effective advantage of GPU's as they are currently designed. Thus it is currently assumed that most unstructured mesh generations and adaptation processes will be executed on node level CPU's. Since the level of computation required for these steps is typically "small" with respect to the solution process, this may well be acceptable of these steps can be executed with sufficient system level scalability and node level performance. Of critical importance to these steps in the process is the ability to effectively control the communications, particularly node to node communications. 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/adapted. Also of importance to node level performance and system level scalability of these steps is the need to support fast dynamic load balancing to account for the fact that the inability to accurately predict local work loads for meshing operations leads to load imbalance.

Initial experience on current near petascale indicates that key mesh adaptation steps can done 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 able to be executed on these and larger machines.

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

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 is a key opportunity for co-design. At the heart of the co-design iterations will be examining possible trade-offs and alternatives such as:

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

DRAFT

- 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 (for example, some algorithms require variable node-to-node communications that are dominated by small messages) gives hardware developers the opportunity to effectively support those needs. Obvious examples include use of local neighborhood information to support effective communications, low latency modes for small messages, etc.
- Understanding which information and how much must be check pointed is critical information for the hardware designers to know in developing and optimizing the best methods to support check pointing.
- Fast dynamic load balancing, potentially both intra-node and inter-node, 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 supports a mesh adaptation operation.
- Consideration of methods to move data and modify tasks as needed to support *in situ* visualization and/or simulation steering.

To support experimentation and evaluation of the possible hardware configurations, it is absolutely critical that simulation tools that emulate node architectures become available to the developers of applications in the very near term. In return, the application developers must provide component operations and mini-application implementations to test system simulations. Examples of basic component tests for PDEs include:

- 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 vs on node vs off node).
- Operations that can compare data interleave verse blocked data operations.
- Operations, like selected mesh adaptation functions, to test the effectiveness of the neighborhood communications and fact neighborhood dynamic load balancing.

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

Applied Math Challenges in Mission-Driven Data Analysis and Visualization

Data analysis and visualization are playing an ever-more-prominent role in the set of tools necessary for the mission of the application areas that drive the exascale initiative.

DRAFT

Data analysis and visualization technologies needs identified in previous extreme scale reports.

The *High Energy Physics Report* has identified extremely large data sets that will require end-to-end solutions for data management, analysis, and visualization, and the management and automation of the workflows associated with the overall simulation execution. Indeed, the size of data is expected to grow by 6 orders of magnitude or more for extreme-scale problems. To deal with such data complexity, data analysis and visualization in situ seem to be some of the more promising avenues to the report authors. For discovery in such large data sets, visualization is essential, the HEP community requires the development of exploratory algorithms to examine the entire physical domain and display some of the interesting region from the large dataset. 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 50-100,000 sources in 10,000 CPU hours (to estimate the covariance of these measures requires multiple realizations of these or simulated data sets).

The *Climate Science at the Extreme Scales* Report also identifies the need for performing certain analysis tasks near to the data, such as monthly means or other summarization operations. The availability of spatio-temporal data of unprecedented frequency and resolution on climate time scales which will result in analysis problems of scales never before tried for which the availability of appropriate techniques is unclear. Similar frequency and resolution density has also been identified in the *Nuclear Energy Systems at the Extreme Scale Report.*

The Fusion Energy Sciences at Extreme Scale Letter Report estimates that, based on mean-time-between-failure concerns when running on a million cores, these fusion energy codes will need to output 2 gigabytes/second per core or 2 petabytes/second of checkpoint data every 10 minutes. This amounts to an unprecedented input/output rate of 3.5 terabytes/second. This raises very important analysis and systems software issues. Managing large-scale input/output volume and data movement and optimize input/output performance automatically based on hardware characteristics are important endeavors. Real-time monitoring of simulations and run-time metadata generation are essential to efficient use of exascale architecture for fusion energy science needs. Workflow needs to be scaled and must support summarization of results in real time, and/or permit the monitoring software to automatically manage simulations that do not progress correctly. In addition, the issue of novel data formats that support advanced operations was clearly identified, with a particular emphasis on experiment-simulation data comparison. Experimental data are expected to grow to terabyte sizes, and therefore robust synthetic diagnostic tools need to be developed that are cross-platform scalable and based on forthcoming community standard data formats. Finally, the fusion community has identified important bottlenecks in the current practice of data analysis and the visualization. Important issues are the detection and tracking of areas of interest— such as coherent structures and fronts—when they are likely to be spread across many processors and techniques to process the data to reduce overall size before these data are output by the simulation. Even after this reduction, the volume of

DRAFT

data left can be enormous, so an open question is the one of reduction and mapping of terabytes or petabytes of data into meaningful visualization.

Challenges in Data Analysis and Visualization

General issues that are facing data analysis now will only be more stringent at extreme scales. We are still in need of identifying, adapting, and developing techniques for

- accommodating large-scale, heterogeneous, high-dimensional data while avoiding the 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.
- near real-time identification of anomalies in streaming and evolving data. This is needed in order to detect and respond to phenomena that are either shortlived (e.g., supernova onset) or urgent (e.g., power grid disruptions).

In turn, this 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 need 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 nonstationary spatio-temporal data of high frequency and resolution it is not known what are the suitable correlation functions to be used for statistical representation. Finally, the 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. They include optimization methods for regression or max likelihood analysis and improved sampling methods, such as adaptive sampling.

Hardware codesign opportunities resulting from data analysis and visualization requirements

This session considered how hardware architecture could be modified to better support data analysis and visualization requirements, and how the data analysis and visualization system implementations could be reformulated to better fit within known hardware constraints of Exascale computing platforms.

There are very few degrees of freedom available to us to substantially change the balance of a system architecture to make it more favorable for I/O or memory intensive applications. For example, given the memory and I/O bandwidth is primarily power limited, were we to give up **ALL** of the FLOPs in the system to get better memory or I/O bandwidth balance, we could afford to increase the memory bandwidth by at most 20%. Given memory capacity is cost-limited, in order to get a memory capacity that is at party

DRAFT

with HPC system would be approximately $100M. Creating a system with 10% of the memory capacity of the Exascale system would dominate the budget for said system.

Given the narrow parameter space for affecting the technology parameters that affect system balance, the primary opportunity for hardware/software co-design lies in optimization around architectural features (the organization of the system as opposed to modifying the speeds and feeds of memory and I/O subsystems). Therefore, we considered how system architecture could be changed in the following areas.

Storage locality: Currently, most I/O subsystems on systems consist of large shared filesystems such as Lustre and GPFS. However, the power considerations of future systems favor highly localized I/O bandwidth. So it is desirable for future HPC systems to offer more node-local storage. We considered what applications could take advantage of node-local storage, and how one could make more effective use of such storage. We also considered how to support off-system I/O to a dedicated "analysis machine" or to tape/archival storage

System configurations: There were a three different system configurations discussed,

1) a separate data intensive computing system that was custom-built around the requirements of visualization and analysis applications

2) In-situ computing, where the exascale computing platform is also used to handle all visualization and analysis tasks

3) heterogeneous in-situ where a subset of nodes are organized differently to act as I/O servers and embedded data analysis engines.

Co-Design Considerations: Hardware Constraints

This describes what kind of hardware architectural support could be targeted at the requirements of visualization, analysis and I/O intensive computing.

It is clear that 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 (getting to archival and/or dedicated analysis system) I/O will be increasingly challenging to support given the power consumption of scaling I/O bandwidth is proportional to both bandwidth and length of the wires that connect to the I/O subsystem. Even the protocols for ensuring consistent state across a shared filesystem (demanded by POSIX semantics) will be difficult to scale.

DRAFT

Future system designs can incorporate more node local (NVRAM) memory to supplement apparent localized I/O bandwidth and to supplement the otherwise limited DRAM capacity on each node. Hardware software co-design should consider how to reorganize our data analysis algorithms to accommodate this

Hardware developers and analysis experts must work together to determine can we reorganize. What is **not** amenable to hierarchical organization (and why)?

Co-Design Opportunities: Designing Hardware to Better Support Algorithms

We consider that dedicating 10% of the overall procurement cost to data/analysis systems has a track record as reasonable percentage of full machine configuration cost. According to Hardware Marketplace.com, the US computer market in 2003 was US $136B and visualization market was $20B. This suggests a rough estimate for the size of an analysis machine, which dictates cost ($20M)

So we consider: What can we do better for analysis and visualization using $20M than simply reusing the same machine that is used for computing?

- Because of memory issues global addressability is an absolute requirement.
- Perhaps more nonvolatile memory (but cost differential is problematic).
- Dedicated small-message (global reduction) is even MORE important for data analysis algorithms than for mainstream compute algorithms.
- Reduced precision is very common for data analysis applications. However, we cannot give up double precision: computing summary statistics may make it more of a requirement for double (and ill-conditioned matrices).
-

Global Addressing vs. Cache-Coherent SMPs: Large Cache-coherent Shared memory multiprocessors are the staple of many modern visualization systems. However, in the future, it will be increasingly difficult to scale up cache-coherence approaches – even within a single chip – as we moved towards chips with thousands of processors on board.

There is a lot of code base written in OpenMP or Pthreads. Without cache-coherence there is concern about movement to a new model. PGAS models are here today, but will require further exploration to determine if they are sufficient. PGAS languages are currently poorly understood and under-utilized by the analysis community. PGAS languages are also is very weak in terms of parallel I/O support).

Hardware architects consider it very feasible to support global addressability within massively parallel chips and even across the machine, but require guidance from the community regarding the details of how this would work. One member of the analysis

DRAFT

community said explicitly "I don't want cache coherence!" (seconded by a colleague who said "I liked E-registers better. Cache coherence muddied the waters.") So there is a lot of opportunity for interaction between algorithm designers and hardware designers to specify the requirements for a globally addressable non-cache coherent system.

For example streamlines particle advection requires global addressing support and also I/O capacity to support time-dependent data analysis. Perhaps have globally addressable memory to support visualization techniques with particle advection.

However, we would need to rewrite algorithms for global addressing. The conclusion was that global addressing would be very useful and desirable for analysis, but requires hardware support.

Localizing I/O Bandwidth: I/O bandwidth is huge concern. There was general understanding that physical constraints greatly limit the headroom for increasing I/O bandwidth. One approach is to move towards node-local NVRAM for localized checkpointing, but also for highly-localized bandwidth for analysis I/O. However, there is a huge software gap for NVRAM. It will require a lot of close interaction between the node designers and analysis algorithm designers and system software architects to come up with a more optimal solution.

I/O Capacity: Data analysis really needs more memory per node, for example, to support analysis of time-dependent phenomena. However, we know that it cannot be supplied by DRAM due to cost and power constraints. Perhaps DRAM could be supplemented by using NVRAM. Again, the organization of millions of disparate NVRAM filesystems creates a daunting challenge both for system software and for general programmability of such systems.

Perhaps node-local NVRAM could be included in the global address space (HPGAS). This would enable global unified visibility for the independent NVRAM filesystems, but enable partitioned use of node-local NVRAM bandwidth that optimizes the costs of local vs. non-local data accesses.

DRAFT

Co-Design Opportunities: Algorithm Support for Hardware Constraints

Multi-resolution processing algorithms required to conserve memory space and to reduce I/O pressure on machine with constrained I/O. A good way to take advantage of highly hierarchical machine architecture is to have a hierarchical approach to handling multi-resolution data processing.

In-situ and stream processing can reduce I/O load and localize computation.

Better sampling algorithms will also be on the critical path for more efficiently computing summary statistics or localizing data output to “interesting” events. One example of this is using in-situ analysis processing to leave a trail of metadata to identify “interesting events” to facilitate more efficient secondary analysis. Non-linear optimization can enable substantial data reduction by representing high-dimensional data with fewer coefficients. 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 may also play an important role in reducing data capacity and bandwidth requirements. An example is the one of a hierarchical approach for three point correlation where you do not perform point-to-point correlation at finest scale. Instead subsample data and then do k-point on lower-resolution to trade computation for data and bandwidth reduction.

Single precision results can be exploited in many data analysis computations.

Node local NVRAM can do more IOPs than a conventional disk. There needs to be more consideration of what kinds of data analysis algorithms can take full advantage of the performance characteristics of NVRAM. For example, out-of-core (OOC) algorithms could be much more effective because they typically suffer from the poor IOPs performance of typical disks. With that barrier removed, many OOC techniques should be re-examined. Node-local NVRAM can be a panacea for lack of DRAM growth if OOC techniques can be sufficiently refined.

Other Co-Design Opportunities

Analysis for Debugging and Tuning: Could use for processing data for Co-Design. For example, performance counter data collected at each node could benefit from data mining for automating co-design process. It could also change the approach to performance monitoring hooks in hardware could be analysis in-situ. For example, one could automate identification of sources of memory and O(>N) or of O(>N?) algorithm complexity growth (where not expected) by tying performance counter information to debug information that is embedded in the code base. In-situ data anlaysis techniques could also be valuable for anomaly detection (for debug).

There is a huge opportunity for co-design interactions between the debug/performance modeling teams and the data analysis professionals.

Operation Modalities (hardware configurations) for Data Analysis

In-Situ analysis modality

This modality primarily involves running analysis concurrent with computing application (co-resident on node with the application as it is running). It offers the advantage of minimizing data movement to disk by co-locating the analysis with the application in node memory.

This approach is not the norm today so there is very little software infrastructure. The fact that there are no common data structures shared across application codes makes it difficult to create reusable data analysis algorithms for different in-situ environments. Failure to develop reusable components will result in non-robust analysis, so it is imperative that common data models be developed to facilitate development of any reusable in-situ data analysis software infrastructure.

Given nodes are extremely memory limited, having analysis tasks competing with the simulation for resources may prove to be a challenge. Overcoming these memory limits creates opportunity for developing hierarchical algorithms for data analysis and visualization. Since not all analysis is load balanced, there is a concern that it will interact poorly with application-integrated load-balancing schemes and make a very challenging problem even worse. However, it may lead to new opportunities to create both hardware and software structures to more comprehensively treat and mitigate load balancing issues.

The assumption is that in-situ analysis can reduce the dataflow by allowing rapid in-core summarization of complex data. However, if in-situ doesn't help reduce data flow, then is it advantageous?

Streaming heterogeneous in-situ analysis modality

This is a slight variation on the homogeneous in-situ approach where an intelligent I/O processor processes bytes as they stream by, much like 'sed' works for stream processing of text data. The advantages of stream processing is that it involves highly localized processing, which can be more energy efficient due to simpler computational elements and minimal data movement, and space efficient since it shares minimal node memory (given its streaming nature). The approach is very much like Netezza's FPGA-optimized query engine where the processing is placed next to or inline with the data storage device.

The advantage is the specialization offers cost and energy efficiency, and it 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 on-board, or it can be emulated with multicore by dedicating a subset of cores

DRAFT

to the stream processing functionality. From a programming standpoint, it could take advantage of stackable I/O libraries, such as that 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 PAPI 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. Wu Feng suggests stream processing could also be used to leave cookie crumbs (metadata tags) for interesting events, that can accelerate later secondary data analysis tasks.

Heterogeneous in-situ modality

In this case is a variant of homogeneous-in-situ that defines subsets of nodes with differentiated function like the “climate coupler.” In this situation data doesn’t hit disk, but pays the cost of traversing the system interconnect. It mitigates the problem of analysis tasks crashing the main simulation, or directly competing for scarce node resources by isolating data analysis to a mutually exclusive set of nodes. It also offers the option of creating nodes with an architecture that is differentiated substantially to improve memory capacity and/or offer different capabilities than the compute nodes.

Particular parts of the analysis could be sent through these nodes as a streaming process to reduce or reorganize data before it hits disk. For example, you could program a streaming to filter on the fly to reduce storage requirements. This approach can also support realtime visualization requirements by preventing overload of off-system network for the path to remote clients. It can also support realtime debugging since it is difficult to debug in existing computing facilities when in batch.

The lack of a common data-model across application domains (even ones that **should** have a common data model) inhibit development of a common, reusable analysis infrastructure.

Issues with all in-situ modalities

There are a number of limitations to both in-situ approaches. First, there are many cases where the domain decomposition for analysis match the compute domain decomposition (not always matched). For example, many analysis tasks involve tracking features across multiple timesteps or tracking features backwards in time from a specific event. In another example, K-point correlation gets to be a huge problem because it requires a massive amount of global communication.

If you don’t know what you are looking for, then integrating analysis with the simulation code may not be of substantial benefit because it eliminates the human cognition from the analysis process. Some data analysis approaches might need data from all

DRAFT

timesteps of the simulation. An in-situ approach can't go backwards in time without hitting the disk (which obviates its advantages). Secondary analysis is out of the question for interactive analysis since you cannot hold an expensive resource hostage by scheduling dedicated time for interactive access.

Using Exascale Platform for Postprocessing Modality

In this approach, the data is stored to disk or some other I/O system and read back into the system memory for a separate analysis/visualization phase (secondary processing). It doesn't benefit from reducing I/O bandwidth requirements, but fits better with well-understood approaches to interactive analysis. In terms of the effect of hardware on the nature of analysis algorithms, this approach is more of an extrapolation of existing approaches. As mentioned in the consideration of in-situ analysis paradigms, there is a need to do integrative data analysis across simulations (simulation comparison) that cannot be accomplished with purely in-situ or offline analysis techniques. The post-processing style of data analysis is also needed to compare sensor data to simulation data.

There is a need to store archival data for provenance concerns. For example, you need to be able to go back if someone disputes your computed data. This is motivated by secondary the fact that the secondary data analysis may reveal features that require re-running or enhancing the primary analysis. Also, the compute platform is likely the best place to perform post-processing analysis because not everyone will be able to afford an analysis system that is large enough to accommodate the data produced on an exascale system.

The postprocessing would likely need to occur on the same platform that was used to run the exascale simulation codes because cost-effectiveness is a concern. Creation and support of two divergent architectures does not make sense from a design perspective (more work and less sharing of Non-recurring expense of design), and from a cost perspective (less volume reduces cost-efficiencies for manufacture and adds support burden).

If you can only get a fraction of I/O bandwidth from fraction of nodes, you will waste I/O bandwidth. It is unclear if you can use the resource effectively if you have to scale exclusively to aggregate enough I/O capability to support the analysis requirements (seems like a waste of resources).

Using Dedicated Platform for Post Processing Modality

We discussed the consequences of creating a dedicated resource for analysis and visualization that is distinct from the primary exascale computing platform. We note that

DRAFT

most facilities don't have data-intensive systems. Current vis/analysis clusters are more for policy issue (not architectural differentiation). We need to engage in a co-design process to explore what architectural differentiation COULD be done to make a distinct design point for data analysis.

The ASCI program mandated a 10% rule, where 10% of the budget for a high-end computing platform should be dedicated to resources that are devoted to interactive and offline data analysis. This may prove to be budget issue if 10% of the memory on a separate cluster on system that is already memory constrained. The future technology constraints on the cost of memory and energy cost of I/O bandwidth will make it difficult to have substantially different balance from the main platform, but investing in a dedicated subset of the system for interactive analysis will prevent interactive tasks from taking the larger machine "hostage" during periods of analysis.

There is a strong impetus to share expensive resources rather than spread on differentiated systems (except if strong justification… e.g. we cannot do X if we don't have a differently organized system). Also, industry will be concerned about the market size for analysis systems. There is a strong desire to share a common baseline design to get cost efficiencies in hardware development. So it seems unlikely that a dedicated resource would be substantially different than the main platform, but there is a need for defining such a resource to prevent resource allocation policy conflicts.

Other Issues

There was a general agreement that another meeting of visualization/analysis for exascale may be needed. There were many mismatches of terminology that impeded progress on this meeting.

Keeping data

There was concern that some exascale reports have considered getting rid of interactive analysis or post-processing data analysis modalities, and that we can no longer afford to keep data around from simulations (just recomputed). However, this group felt there was a very strong need to keep data around that cannot be ignored. There is a need to do integrative data analysis across simulations. We need to compare sensor data to simulation data. Data provenance concerns create a critical need to be able to go back if someone disputes your computed data.

Data must be archived motivated by secondary and tertiary data analysis that is motivated by the primary. What is the cost of recomputing (how many joules for a tape vs. recomputing): how about 1% reuse? Storage density of tape will be sufficient, but bandwidth constraints to tape will be energy limited (just as with BW to shared disk). However, the storage performance (and reliability) of mechanical tape subsystems in

DRAFT

the future is uncertain. Regardless, it should be a high priority to ensure that an archival storage is supported as a requirement for future platforms.

Uncertainty Quantification/Stochastic Systems

Uncertainty quantification (UQ) is an essential component of any investigation of the behavior of a complex, multiphysics, multiscale system. The importance of UQ promises to increase dramatically in the exascale computing era because: (1) exascale capability will allow investigations of increasingly complex physical systems that are increasingly inaccessible to physical intuition and experiment; (2) exascale capability will increase the reliance on computational science and engineering to make informed policy and design decisions in situations in which substantial resources are involved; and (3) exascale computing will provide the computational power required to carry out systematic UQ analysis for complex systems. However, UQ is a relatively young field that will require substantial investment in order to meet the demands of its increasingly central role.

A key observation is that analyzing and predicting the behavior of complex systems depends on a fusion of data (experimental, observational) and modeling (physics, computation), because of limitations, e.g.,

- Experiments are expensive, observations are limited
- Models present only a partial physical understanding
- Computing high fidelity solutions of models is expensive
- Many environmental conditions are poorly known
- We often wish to “extrapolate” beyond known or observed states

As a consequence, uncertainty and error affect every scientific analysis or prediction. In general terms, uncertainty quantification is the end-to-end study of the accuracy and reliability of scientific inferences [cite Higdon exascale report]. By "end-to-end", we mean that UQ starts by describing error and uncertainty affecting a model, how the effects of those propagate through the model, and then describing the effect on the output of the model. It is important to note that the particular details of UQ are very specific to the application, the kinds of error and uncertainty affecting models, and the goals underlying the use of the models.

UQ starts by describing all sources of error, uncertainty, and variation that may affect a model. Generally, there are many different kinds of error and uncertainty that are naturally described in a multitude of ways, e.g.

DRAFT

- 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 in order to be used for analysis)
- Accuracy of computation and approximation (this includes both deterministic and stochastic approximation errors)
- Random phenomena (aleatoric or contingent uncertainty)

After describing sources of uncertainty, the next challenge is determining how uncertainty and error propagate through a given model. Complicating this goal is the fact that there is generally a great amount of feedback and interaction between different sources of uncertainty in the solution of a model. This is especially complicated in a multiphysics, multiscale model, 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 consider is complex. Additionally, a UQ analysis should be accompanied by inventories of the sources of error and uncertainty accounted for in the analysis, the (potential) sources that have not been included, and the assumptions under with the analysis is performed.

The complexity of UQ means that no one approach or representation will apply to all problems, or even to treatment of one multiphysics, multiscale system. Hence, research across a broad front of UQ associated topics is needed. Some of the key research challenges associated with modern UQ include;

- Representation of uncertainty and error
- Forward and inverse sensitivity analysis
- Treatment of multiscale, multiphysics systems
- Treatment of model uncertainty
- Verification and validation
- UQ for inverse problems
- Decision making and optimization under uncertainty
- Dealing with large dimension spaces of uncertainty
- Detection and forecasting of rare events
- Software support for UQ algorithms

DRAFT

We present a brief description of these problems below.

There is one common characteristic across this diverse list of topics. Namely, each of these component aspects of UQ present extremely serious demands on computational resources. Currently, the routine use of UQ is hampered by the associated computational costs; an issue that is more pressing in the context of the fact that current computational capacity is used largely for producing one-off solutions of complex models. This points to the 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 between the needs of UQ algorithms, algorithms for simulation of complex multiscale, multiphysics models, hardware and system software.

Description of UQ Research Challenges

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 is also needed in 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 and boundary conditions – affect key model outputs. There is both a forward problem of propagating variation through a model and an inverse problem of determining the variation in input that yields an observed variation in the output of a model. Both stochastic and deterministic methods are useful.

Multiphysics, multiscale systems

Multiscale, multiphysics systems raise well known challenges for accurate modeling and simulation and these all carry over to UQ. Particular challenges include combining different descriptions of uncertainties from different kinds of physical models and different kinds of 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.

DRAFT

Scale differences in behavior of different physical components may lead to scale issues in computing and representing uncertainties

Model uncertainty

A very 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 a model in use. In addition, changes to a model may dramatically change the behavior of solutiosn. 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 very 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 below.

UQ for inverse problems

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

Decision making and optimization under uncertainty

Large-scale computational models are a powerful tool for planning and decision-making. Such models can be used to help assess important questions regarding the management of a particular system. Such issues are often posed as optimization problems. Use of models for this purpose requires understanding of the uncertainty and error in predicted behavior.

DRAFT

Detection and forecasting of rare events

A very difficult challenge is accurately representing the effects events that have very low probability of occurring but which have very large or catastrophic effects when they do occur.

Dealing with large dimension probability spaces

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

Software support for UQ and necessary technologies

Up to now, the algorithms and methods for UQ along with enabling technologies are typically developed outside application codes, which can lead to enormous inefficiencies.

UQ/Architecture Co-design opportunities

This session considered how hardware architecture could be modified to better support Uncertainty Quantification (UQ) applications, and how the UQ calculations could be reformulated to better fit within known hardware constraints of Exascale computing platforms.

Computations for Uncertainty Carrying Variables

Future systems will have severely constrained memory bandwidth relative to modern systems. 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. We considered how UQ implementations could be modified to make use of such on-chip scrachpad memories, and how such scratchpads should ideally be organized to support UQ applications.

Hardware/Software Co-Design

Co-Design / Effect of Hardware on Algorithm Design: The first example considered was computations for variables that include information about uncertainty. These calculations depend on quadrature tables for computation of convolutions and other operations. For polynomial chaos computations, the size of these quadrature tables is approximately Np where N=order and p=dimensions for a brute force implementation.

DRAFT

An efficient implementation will require 603 double precision values, which comes to about = 2MB of memory. Such tables would be shared on chip and system wide.

Co-Design / Effect of the Algorithm Requirements on Hardware Design: It is desirable to have ~2MB of shared scratch pad memory on chip. This would constitute a different class of memory than conventional scratchpads that is shared by all cores on the chip rather than being local to individual computational elements.

The mathematical operations performed on the operands in the shared scratchpad memory require fast weighted vector sums. The basic weighted daxpy operation easy using conventional vector/SIMD units, but the implementation of the math could be made much more efficient if the Vector/SIMD floating point units provided hardware support for "horizontal adds across vector registers".

Need for an Integrated System for UQ

Exascale architecture favors locality to achieve energy efficiency (100x more costly to write to disk than to summarize in memory). UQ requires fully integrated systems to minimize data movement costs by ensuring that statistical summarization and analysis is performed across ensemble runs while they are still in system node memory in lieu of storing intermediate results to disk and summarizing later.

It is inefficient to run ensembles on the widely distributed or unintegrated resources because energy cost to perform statistical/data summarization. Therefore, exchanging data directly between nodes rather than store on disk and recall to perform comparison/summarization.

Hardware/Software Co-Design

Co-Design: Effect of Hardware Constraints on Algorithm Organization: These observations are consistent with discussions in the MathB-ThemeI Analysis + Architecture crosscut break-out, where integrated in-core analysis capability can greatly reduce the I/O requirements to shared filesystems, which will be an extremely constrained resource. There must be some more effort to quanitify the data reduction that can be achieved through this approach.

Co-Design: Effect of Algorithm Constraints on Hardware Organization: The hardware requires high-performance network capable of handling small messages to enable efficient and effective cross-comparisons across ensemble runs. Also, local NVRAM may be useful to support localized analysis computations for the statistical summarization.

DRAFT

Hardware/Software Co-Design

The above scenario describes how UQ algorithms can be reformulated to fit better with future architectural constraints. Namely, the reformulation makes use of on-chip FLOPs to reduce the off-chip bandwidth requirements. Likewise, this change in the algorithm requires addition of scratchpad memories on-chip to support the expansion of scalar variables into tabular representations. Finally, both the hardware and software require support from compilers and the programming models to transparently convert conventional scalar arithmetical expressions into arithmetic on statistical distributions rather than scalar/single-valued quantities.

Co-Design Example: Inverting the UQ Computational Paradigm

One of the key computational problems associated with Uncertainty Quantification is the inference of the sensitivity of a model's prediction to changes in the input data and parameter set. 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 in order to support a huge number ensemble simulations and statistical analysis of the results. Sensitivity analysis may also 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 directly carry out computations associated with statistical analysis directly on-chip rather wrapping statistical computations on the outside of simulations. So, rather than 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 (to conserve memory bandwidth) to enable direct computation of a new statistical distribution, which in turn can be re-compressed as a parameterized representation for the next timestep. This reformulation of UQ calculations fits much better with the architectural constraints anticipated for future systems, which involve limited memory bandwidth and a much more FLOPs on chip.

DRAFT

This approach also enables much smaller number of simulations to be run to gather 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.

UQ SIDEBAR-----------------------

The following example illustrates how this approach touches on algorithms, software, and hardware support for base level UQ computations, and ultimately how UQ algorithms should be developed relative to the configuration of Exascale machines.

Consider the classic UQ problem for Laplace's equation

(1) -Laplace U = F

for a fixed domain and the boundary data. The goal of solving the problem is to compute a value of U at a point or some kind of bulk property like the average value. A classic UQ problem is to assume that F is random, or randomly perturbed, i.e., F is a statistical quantity with a probability distribution. The UQ problem is to describe the probability properties of the computed quantity obtained by solving (1), e.g. compute its probability distribution.

In the classic "nonintrusive approach", we have a Laplace solver will be used and we cannot change the code. To carry out a stochastic sampling, we choose a large number, e.g. 20,000, of sample values for F using its probability distribution, use the code to solve for 20,000 solutions U, compute the quantity of interest (point value, average), then bin and smooth the results and obtain the approximate density. In this approach, the

Dominating cost = 20,000 Laplace solves.

There are additional costs for computing the statistics which are significant but are much less relative to this, so these tend to be ignored.

Now consider an "intrusive" approaches. For example, we can use adjoint problem and Green's functions. In this case, we solve ONE adjoint problem for the Green's function,

(2) - Laplacian G = D

where the FIXED data D gives the quantity of interested in, e.g. a delta function for a point value a constant 1/volume of the domain for the average value. The "intrusive" aspect comes from the fact that in general we have to form and solve the adjoint problem corresponding to certain data. We can compute the statistics by computing the convolution of this ONE Green's function G with the data F. This can be carried out on a discrete level, by choosing the 20,000 samples of F and computing 20,000 discrete

DRAFT

convolutions, or we can devise an approximation directly using an analytic formulation. Alternatively, we could use a direct solution method, e.g. by thinking of the problem as trying to solve Laplace's equation for 20,000 simultaneous right hand sides, and adapting the code to solve this problem for the cost of one inversion, applied many times to the data. Another powerful approach would be to use polynomial chaos or the stochastic finite element method.

In all of these cases the cost is approximately

1 Laplace solve

20,000 Convolutions (or something of equivalent cost)

These approaches clearly move the statistical computations inside the main simulation algorithm of solving the Laplace's equation, with the gain of greatly lowering the overall cost of computation.

Now consider the example further in light of the discussion about trading memory for compute cycles and localization of data. We have this function F that carries this probability distribution and we want to compute a convolution with another given function. If we have a lot of memory locally at each node, we can attach 20,000 random values of F at each node as a vector and deal directly with discrete convolutions. But if we don't have that much memory, then instead we might take an analytic representation (lots of choices, all approximated using a much lower set of numbers) of the distribution and then compute an approximate analytic convolution on the fly. Here we are directly trading compute cycles for memory, which better fits the architectural constraints of future systems.

This is directly tackling the costs of the "intrusive" approach. Having the system software/programming model handle convolutions automatically would be a tremendous gain for UQ, and would provide a strong incentive to use intrusive methods. Focusing in particular on ways to compute convolutions is a great starting point because that always comes up.

There is also a feedback to algorithm design. Depending on the best way to compute the convolutions on a given computer, we might decide to attach a long list of random samples to a variable in the design of the algorithm or we might want to use a compact analytic distribution and this would lead to different algorithm designs.

Of course, this is a very simple problem. But the same kind of reasoning would apply to a complicated differential equation, it would just be much harder to figure out new algorithms to move the statistics inside the loops. That would be one exascale-related research project in UQ.

DRAFT

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 also as potential drivers for high performance machines.

In the enabling category, discrete algorithms have long been a part of the HPC ecosystems 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 preconditioners. And graph ordering methods have been used to improve memory locality. Looking to the future, combinatorial algorithms are likely to 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 can also be drivers for the most powerful computers. Graph algorithms are increasingly central to cyber security, and to knowledge discovery in huge data sets. Combinatorial optimization is the key to many complex scheduling and design problems. Much of our national infrastructure consists of complex, distributed, interconnected systems (for communication, transportation, energy and water distribution, etc.), and analysis, design, simulation , and optimization of these networks entail enormous combinatorial problems.

Discrete problems have not played a major role to date in driving the HPC agenda for a variety of reasons. Most importantly, the characteristics of discrete problems are quite different from those involved in modeling physical phenomena, and so the current HPC landscape is often poorly suited to solving combinatorial problems.

- The underlying data dependencies in discrete problems are often radically different from those in physical simulations. Since physics occurs in 3D, 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 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; his 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.

DRAFT

- Graph algorithms can have a high degree of parallelism, but it is often fine-grained and asynchronous. Graph algorithms are not naturally bulk synchronous, and so work best with light-weight, 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 subproblems being considered, and can be exponentially large. Moreover, choice of the subproblems is a critical step and making clever decisions for choosing the right subproblems 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 lot of redundant work. The 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. Of particular interest is agent-based simulation, where interacting entities evolve over the course of the simulation. 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 create great difficulty 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 like the massively multithreaded Cray XMT. 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. So 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 wide range of computational characteristics, so any hard and fast characterizations will be necessarily incomplete. But a variety of themes and characteristics commonly recur, and several of these are relevant to architecture codesign.

- Graphs are a very 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

DRAFT

structure from the underlying geometric space. But graphs that arise from different applications like cyber security or social networks have very 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. 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.
- Combinatorial optimization problems are often solved by some kind of brand-and-bound technique. These techniques involve the dynamic construction and exploration of a search tree. The leaf nodes of the tree 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 very difficult to partition effectively. Thus a programming model that supports a global address space is highly advantageous. Also, many combinatorial algorithms have a high degree of fine-grained parallelism, but the available parallelism is dynamic. Generally speaking, parallel algorithms for these problems are not naturally bulk synchronous.

Many discrete math applications are good candidates for co-design since they are relatively agile in comparison to the large multi-physics 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:

- The ability to support a global address space.
- Light weight, one-sided communication for small messages.
- Light weight fine-grained and very flexible synchronization mechanisms
- The ability to turn off the memory hierarchy for accesses that cannot make good use of it.
- A 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 these be implemented in hardware?

DRAFT

- What memory consistency models are necessary to support discrete algorithms?
- What are the right load balancing abstractions for exascale machines?

Optimization and Solvers

There are a number of well-known issues in solvers and optimization for scalable computing, and anticipated issues going forward. Issues include:

1. 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 show up as a large cost.
2. 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.
3. Poor scaling of coarse grid solves: Scalable preconditioners tend to be multi-level methods, which traverse a hierarchy of fine to coarse grids or graphs. The coarsest grid is typically very small, easily fitting on a few processors, and requires a precise solution, often by a direct solver. This phase becomes a serial bottleneck in the computation.
4. Lack of robustness & reliability: Solution algorithms are not always robust or reliable, requiring user intervention and knowledge of tuning parameters. This increases as we attempt to scale on large systems since scalability often comes from sophisticated algorithms that are very sensitive to tuning parameters.
5. 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.
6. 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.

DRAFT

Other topics:

1. Solvers scale more poorly than other portions of an application on a multicore node: Computation of physics, construction of global operators and other application-specific computations tend to scale well on multicore nodes, due to good temporal locality. Solvers tend not to scale well we go 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.
2. Advanced modeling and simulation capabilities (Stability, Uncertainty and Optimization): A single forward solve becomes a node of a much larger problem, providing ample

parallelism for exascale systems. This particular area represents a true exascale problem and provides a qualitative improvement in the value of simulation (not just a single answer, but an optimal answer, with error bars). However there are prerequisites: The forward problem must have sufficient fidelity and there must be smoothness in the sampling space. This area presents new linear algebra problems that, fortunately, are also more amenable to efficient computation. Specifically, instead of solving $Ax = b$ for a single right-hand-side we must solve $AX = B$ or $Ax^i = b^i$, with multiple simultaneous or successive right-hand-sides. Families of related preconditioners also arise, creating opportunities for computation and communication pattern reuse.

3. Miniapplications: There was substantial discussion in this session about the need for miniapplications. In previous efforts, these types of codes have been called compact applications or synthetic applications, so the idea is not brand new [ProgEnv, p. 8] . However, there is a bit of a difference in that we are asking the application teams to be the primary developers of the miniapplications, with the idea that a miniapp 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. Miniapps do not have to be working models. Instead they need to correlate to performance issues, such that if we study miniapp performance in various setting (including rewriting it) the information we gain can be correlated to back to the real application. Miniapps are not a replacement for the real application, but allow us to use working code far upstream in the design process where it is infeasible to use the real application.
4. Two-level parallelism and strong scaling: Scalable multicore systems support flat MPI-only execution, with one MPI task per core; or can be used with MPI plus a node level model below. Examples include MPI with OpenMP, Pthreads, or another level of MPI. For large subproblems, where there is a lot of data per core, there is little difference between the two approaches, but in the case of strong scaling, where the subproblem size shrinks, the two level model can scale much better, at times being 6 times or more faster. Strong scaling will be essential for effective exascale computing, so two-level models will be very important.
5. New discrete equation formulations: 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 (FAS) 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.
6. Two-track application refactoring strategy: It is clear that applications will need to refactor code to utilization the node architecture of an exascale system. However, in the 2015-2018 time frame we must rely on existing programming models and environments [ProgEnv, p.1]. At the same time we anticipate that new programming models and environments will emerge, making it easier to express parallelism in a portable and efficient way. The challenge will be to refactor for exascale and retain the investment for post-exascale environments, which will have core counts similar to the first MPP systems [NodeArch, p. 1].
7. Bulk synchronous: Almost all scalable parallelism uses a bulk-synchronous model, where data is exchanged in large intensive bursts across the network, and then the network is

DRAFT

idle while intensive local computation occurs. Finer grain communication, overlapped with computation can reduce the impact of latency and load imbalance [ProgEnv, p. 4] However, to break this model several things must happen: Networks must be able to make progress on message traffic independent of computation, and the local memory system, which are heavily involved in communication data packing and unpacking, must be able to service both communication and computation data movement. For memory intensive computations such as sparse solvers, overlapping communication and computation will be challenging.

8. Fault resilience: Presently most advance numerical algorithms will fail in the presence of a single floating point error. However, given the ability to have software-enabled highly-reliable compute regions, data paths and data storage, we can develop algorithms that are tolerant of incorrect floating point results.
9. Active vs. passive barriers: 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 immediate proceed 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 API than the application). As a result we need two different kinds of barriers.
10. Exascale in FLOPS only: Exascale architecture designs are exascale only in operation count. All other measures (bandwidth, memory size, etc.) are only a 10-100x improvement over petascale systems [NodeArch, p. 7]. In order to address this disproportionate grow in capabilities, solvers can be redesigned and implemented to improve computation to bandwidth, and computation to memory size ratios. These opportunities have always been available, but they become more important today.

The breakout group was charged to address the following questions:

What algorithms will work and what can I solve?

What we are doing today will work, perhaps with some scaling limitations. Changes in node architecture, eg, efficient threading, could make sparse kernels more efficient. Changes to algorithms, data structures can improve compute/communication ratios.

What algorithms will have difficulties?

Dense eigensolvers don't scale well today. Sparse eigensolver with communication avoiding strategies, regular grids and don't require preconditioners will scale in a few years. FFTs and sparse linear solvers will have difficulties, but progress could be made with co-design effort algorithm designers and programming model designers.

What algorithms will not work?

DRAFT

Current forward solves will not fully scale due to lack of concurrency.

Are there whole classes of problems that are enabled by exascale?

Optimization algorithms, broader set of UQ algorithms, first principle calculations in some regimes. Memory size limitations may restrict scope of first principle calculations.

How can I most effectively work in co-design process?

We have examples of co-design working today in SCIDAC multidisciplinary teams, DARPA HPCS PERCS system design, design of application specific architectures (QCD machine). Experience has shown that having teams co-located is an advantage. We need to partition the work so that subteams can make progress independently. Language is a barrier for cross disciplinary teams. We also believe that having early intermediate results is beneficial as well as access to common testbeds.

Performance modeling can be the quantitative tool for co-design of apps and architectures. Performance modeling defined in the broadest sense (includes performance models, simulations tools, and compact apps or benchmarks) can provide the interface for co-design.

Recommendations for research thrusts

Solvers that have reduced global communications

Model driven dynamic optimization of libraries

Feedback for technology development

We need mechanisms to identify regions of code where reliability is essential (implies redundant calculations which are more efficient in use of bandwidth). We also need an API to specify to the O/S what data needs to be checkpointed for a system initiated checkpoint capability.

We discussed the desirability of a specialized barrier network and agreed that this functionality could be achieved by an efficient gather operation. Movement away from bulk-synchronous applications requires HW support. We agreed that PGAS language support would help the applications developers, but we have a “chicken or egg problem”

We must move to dynamic load balancing with increased concurrency

Power gating of processors has the potential of increasing jitter.

We can mitigate this to some extent by keeping work units small

DRAFT

Autotuners have potential for improving performance and reducing complexity for applications, but technology is relatively immature

Local memory that can be configured in either scratchpad or cache mode would be desirable

DRAFT

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

The Theme I breakout group considered what R&D on mathematical models and algorithms will need to be done to support the exascale applications on the proposed architectures and how architectures (hardware and software environments) could usefully be evolved to meet the algorithms and applications partway. We considered five aspects:

- Node architecture
- Network architecture
- Power limits
- Memory subsystems
- Performance modeling

Node Architecture. We asked if it is 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 we have valuable "legacy" code today are:

- Heterogeneous architectures with specialty processors, usually with their own memory
- "Globally" addressable memory programming within node or within 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 right here within the node. To some extent the burden of exploiting the resource mix of exascale node architecture is shared with company at all other scales of computing but the pressures for scientific computing have some unique stresses. The post-clock improvement period of Moore's Law in the evolution of computing is "too big to fail," but it could succeed in the commercial marketplace without lifting scientific simulation unless we work with architects to motivate the incorporation of features we deem important. This means looking for features with low recurring cost per unit for the commodity versions. We identified five such features:

- 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, vector/SIMD ops
- Error correction codes

We also acknowledged that there are other desirable features that are hard to influence to the degree we need them, including:

- Power efficiency to meet exascale power budget
- Restartable pipeline in face of errors

Network Architecture. With regard to network architecture, today's petascale scientific codes are substantially perched between two requirements. We need high bandwidth local communication, especially for communication-reducing algorithms that pass large aggregates occasionally, and we need low latency global communication for very small messages. In the past, and notably with the IBM BlueGene, these two different requirements were sometimes met with different networks. The cost in terms of acquisition, power, and coordination of multiple networks for different objectives may shift the design at the next scale. Chastened by Bill Dally's advice that we should tell architects what *we want*, not what *they should do*, we simply acknowledge that there is

DRAFT

a tough network topology tradeoff: Can we afford a low-diameter network to enable indefinite linear weak scaling, though it is architecturally expensive to scale, or shall we compromise with a higher diameter network?

Internode network features that we believe are relative inexpensive for vendors to be influenced in are:

- Lower overhead messaging
- Topology within constraints of wires per node
- Support for global addressing
- Support for efficient asynchronous (overlapped) messages

Features that are likely to be expensive to influence are:

- Driving message overhead close to its physical limits
- Optical or other new technology interconnects (for which we believe the time horizon for commodity is beyond 2018)

The intranode network will not be significantly influenced by exascale initiative, with the following exception. We believe that optimization can be accommodated for different kinds of messages on a typical intranode packet-switched network. This calls for advance specification of typical internode/intranode communication profiles under the new hybrid MPI+*X* programming model that represents the most rapid path forward to exascale.

Power Limits. With respect to power, we acknowledge that we can exert little influence beyond the attention that power is getting from the driver of battery life in embedded devices. Therefore, significant attention to node code will be required to minimize vertical memory traffic, which is power consumptive. Everything that reduces memory and communication traffic has a leading-order effect on reducing power requirements. We believe that co-design will include *some* expectations on the application code on managing power. This could be as little as optional hints to the dynamic load balancing software about how power management software will impact the hardware supporting application.

DRAFT

Memory Subsystems. With respect to memory, which has a leading-order effect on the budget of exascale systems, the movement to many-core nodes, if we can learn to program them, reduces 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, like 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 are already willing to provide is a chip-wide user-controlled scratchpad cache. On the other hand, it may not be easy to influence all vendors to provide a scratchpad.

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 in co-design is to focus human attention on critical sections and high duty kernels, on the philosophy that one cannot optimize what one cannot measure or understand. Performance profiling is also essential for use in autotuning. An entire spectrum of methodologies from instrumentation to measurement to modeling will be required. Modeling includes the various flavors of deterministic, stochastic, and simulation of the machine, itself.

Cross-cutting Challenges and Research Needs in Algorithm and Model Research and Development Needed to Support New Architectures

[Insert introductory sentences]

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 partial differential equations (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 all-gather operation is low, developing solvers that are scalable within a node should be possible. However, aim to recast linear, nonlinear solvers, eigensolvers, fast Fourier transforms, and optimization algorithms for exascale architectures.

DRAFT

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

PRD 1.2: ***Develop new mathematical models and formulations that effectively exploit anticipated exascale hardware architectures***

- Simply re-casting algorithms may not be sufficient 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 1.3: ***Address numerical analysis questions associated with moving away from bulk-synchronous programs to multi-task approaches***

- With very large processor counts, expect an evolution towards simulation codes that are more multi-physics 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 different 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 1.4: ***Adapt data analysis algorithms to exascale environments***

- Reorganize data analysis algorithms to leverage increased node-local nonvolatile random access memory availability

DRAFT

- Rewrite data analysis algorithms to leverage Global Address Space (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:
  - *in situ* (part of simulation code) analysis
  - post-processing on the exascale platform
  - dedicated separate platform for post-processing—can the input/output (I/O) bottleneck be avoided?
- Conduct research on the development of common data structures or data access Application Programming Interfaces (APIs) across science application space to improve re-use of data analysis software.

PRD 1.5: ***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 1.6: ***Develop tools to simulate emerging architectures for use in co-design***

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

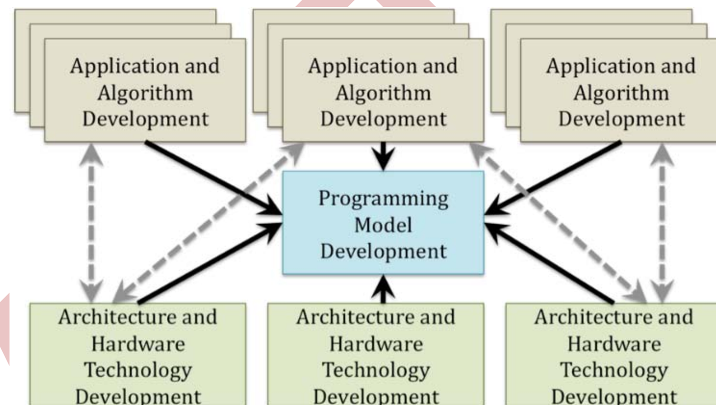
- **References**

DRAFT

PROGRAMMING MODELS AND ENVIRONMENTS

Introduction

Existing programming environments for writing parallel scientific applications have sustained HPC application software development for the past decade and have been successful for Petascale computing, but were architected 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 be for power management, resiliency, or application requirements. The group believes that this change will be as broad and disruptive as the migration from vector to parallel 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.



As the interface between applications and architectures (Figure 1), programming environments plays a pivotal role in making Exascale applications efficient, resilient, and productive on Exascale architectures. One cannot succeed without the success of the other two. To address these challenges, the group believes that 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. At the same time, past investments must be protected, and a migration path from current to future environments must be embraced.

Figure 1: 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.

In Theme III, we were particularly interested in answering the following questions. 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 CS 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?

DRAFT

Understanding programming model requirements at the exascale in the context of critical mathematical models, methods and algorithms

Partial Differential Equations

The PDE math area comprises the following aspects of the PDE solution: discretization methods, multiple physics simulations, computational geometry, and meshing. The importance of this list of topics is that consideration is being given of the entire workflow associated with the PDE solution process. The workflow starts with 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 in terms of a mathematical definition such a solid model (computational geometry). From this point a reasonable starting mesh for that space-time domain must be constructed. The numerical discretization is constructed from this space-time mesh and then solved. The solution obtained is analyzed to determine if the combination of the mesh and its discretization has yielded the desired level of accuracy. In almost all cases the initial mesh/discretization is not sufficient and must be adapted. From many problem classes of interest the steps of mesh/discretization adaptation, solves and evaluation is repeated many time. Some key characteristics with respect to this workflow for exascale size problems are:

- The numerical discretization and solution process is a computationally dominant process that does include 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 are many opportunities to continue to reduce the synchronizations, their complete elimination is unlikely.) During a discretization/solve step the communication patterns are fixed and workloads can be accurately predicted.
- In the case of unstructured mesh generation there are techniques that can automatically create the meshes given the mathematical definition of the domain. These mesh generation processes are highly irregular and dominated by substantial communications that become more dominated by local communications as the mesh is being formed. In the case of general geometric domains it is not possible to determine tight predictions of local workloads thus making the need to dynamically load balance critical.
- The mesh adaptation process has many of the same characteristics except for the advantage that the existence of the mesh to be adapted provided more localization of the processes. However, it is important to note that although the operations are local, the specifics are not known until operations are attempted, thus causing some of unpredictable communication patterns. The workload of mesh adaptation is not well balanced so that dynamic load balancing is needed

DRAFT

during adaptation in preparation of a well-balanced system for the next discretization solution step.

- Multi-physics problems can often use large memories because they must maintain multiple meshes, or scratch space for converting between mesh types. As the size of memory per node continues to become even more constrained, alternative techniques must be explored.

Since I/O is, and will continue to be, a dominant step in this process, it is assumes that all of these processes must be executed on the exascale computer. The one potential exception to this is initial mesh generation in those cases where it is known that a very coarse initial mesh can be used so that the initial input is "small".

PDE solvers are typically characterized by using domain decomposition with nearest-neighbor communication patterns. Elliptical PDE solvers require smaller, non-local communication patterns, and multi-level 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 are also 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 timestep. 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. 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 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, and the network topology will most 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 performance potential of the network. Possible approaches include programming interfaces that allow the system software to convey information about the underlying network topology to the application so that the application can structure communication appropriately, and, conversely, providing interfaces that allows the application to inform system software about the application's communication structure so that 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.

[breakout input starts here:] PDE solvers typically have an abundance of fine-grain, node-level parallelism that can be exploited. A hierarchical machine model with two-

DRAFT

level parallelism is appropriate for both structured and unstructured grids as wells as adaptive mesh refinement (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 alternatives approaches to providing fast local memory access, like 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.

As with existing petascale platforms, debugging applications at the exascale is one of the most important capabilities and is fundamental to application development. The challenge of debugging will be further compounded by the impact of faults on the system. In a sense, 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 of and applications, such an approach may be ineffective. New approaches to debugging, not only to recognize application-generated errors, but also to help identify and isolate system-generated errors will be critical. Potential approaches to error identification and isolation include support for applications to provide the ability for the system software to check the validity of data flowing into and out of computational routines, as well as mechanisms that will allow for computing the same function multiple times in order to determine the causality of an error.

There are many fundamental research questions with respect to data analysis and I/O for PDE solvers on an exascale system. New approaches to doing data analysis simultaneously with the calculation can potentially be effective, provided there is system software support to manage the memory and computational resources effectively. On-the-fly analysis may be able to make effective use of the abundance of compute cycles available. The insertion of non-volatile storage into the memory hierarchy will also play a key role in data analysis. Capturing analysis data as part of a regular application

DRAFT

checkpoint may be possible and the potential for using non-volatile 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.

Solvers are currently taking advantage of multi-precision algorithms that use reduced precision to increase computational performance and reduce stress on the memory subsystem, and this approach will likely continue to be effective on future-generation platforms. The shift to more simplified computing elements should not have a negative impact on solvers; however, the need to support conditionals is critical, so existing computational accelerators will need to evolve if they are to be used effectively in an exascale machine. 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 is currently in transition from using a separate MPI process on each core to the use of hybrid approaches. So far, some success has been had with MPI+OpenMP, particularly in fusion particle codes and strong-scaling climate codes. GPU's have been successfully exploited with MPI+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 for heterogeneous, multicore nodes that will work with MPI. (See report from MathF X PM breakout)
2. There is also interest in global view languages (Chapel, UPC, Co-Array Fortran) on multicore nodes, to be used in a hybrid approach with MPI.
3. Fault tolerance will be a major issue. Two efforts on the horizon—the CIFTS SciDAC project's "fault-tolerant backplane" approach and the fault tolerance working group of the MPI-3 Forum—remain research projects at this point. In the worst case, tools that automatically identify and minimize the checkpoint state of PDE applications will be necessary.
4. Tools will be needed to understand and tune hybrid programs. Some efforts to encourage tools development are being undertaken by the tools working group of the MPI-3 Forum.
5. A speculative concept is that of power-aware programming models. One can conceive of a programming system that allowed one to specify that peak computing power, network usage, or I/O capability was not needed during some part of a computation, allowing the operating system to deschedule certain resources in the interest of conserving power. This could lead to complexity in the programming model, but may be necessary for efficient use of exascale resources.

DRAFT

6. There are significant co-Design Opportunities: What's the interface we use to express features of data for the compilers use (apps understand the questions that are being asked; compilers/programming models understand the kinds of questions that are meaningful). For example, if an application has the responsibility to define the data structures used in the parallelism, types of things that will and won't be done (e.g. annotations for conditionals, loops, etc), automatic CUDA generating code then compiles the code in a high performance way – doesn't want to make the decisions about the particulars of what is done

Data Analysis and Visualization

The Data applications group quickly agreed that as an opening caveat, there are many varieties of data analysis and they do not necessarily share the same computational characteristics. Thus, not all characterizations made in this summary apply equally to all analyses.

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 very difficult to distinguish between bugs and "a potential new scientific discovery that you simply don't have a model for yet." For this reason, it is important to validate that your analyses are correct. However, this can be very difficult for areas of science in which you can't empirically test your model, such as simulations of future behaviors over long timescales. Related to this, for scientific simulations that have the potential to affect public policy, it is important to create a provenance which will back up scientific claims and permit others to reproduce and verify conclusions (note that there's a link here to UQ). For all of these reasons, one useful programming environment concept is to 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 had gotten to a particular state. It is important to note, however, that creating such artifacts along the way could add a significant I/O burden given the potentially large sizes of these states to be recorded.

We identified two abstractions that could be useful for programming models. The first of these is support for natural data abstractions. For example, scientists and data analysts think in terms of multidimensional variables with physical units associated with them rather than systems-oriented concepts like files, data linearizations, data formats, and so on. This suggests that programming would be more productive and natural if the programming model provided abstractions that more closely matched the user's semantics than the system's. Our hypothesis was that this would also help reduce the chances of coding errors since the code would presumably be closer to the natural terms for the programmer these higher-level concepts would result in fewer lines of code and therefore fewer places to make mistakes. Of course, supplying such higher-level concepts requires that the concepts themselves be implemented correctly and

validated. And 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, then 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 to the node that owned a particular piece of data. This suggests programming models that support richer and more dynamic execution models than our current SPMD/BSP models. It was noted that the HPCS languages support this concept. For example, 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; 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 is 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 array A above 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, for example, potentially the CPUs of a (non-compute) dedicated I/O node.

Data analysts use a wide variety of special-purpose languages and tools (special purpose in the sense that they aren't necessarily widely used within HPC applications programming). Examples included R, Weka, IDL, SPSS, Scout, SAS, and Kepler (though it should be noted that 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 seconary analysis or, ideally, on the exascale hardware itself. Some of these tools wrap standard library solvers which could benefit from algorithmic advances like those Jack Dongarra described in his plenary talk on the first day. From a programming models perspective, we ought to study these tools and languages, understand their value proposition, and see whether or not their attractive features and concepts could be cleanly integrated into our 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, so being able to subsume the features or desirable characteristics of the special-purpose languages and tools above would reduce the number of languages one would need to learn or support. In discussing data analysis, it was clear that analysts wrestle with many of the same

DRAFT

problems as standard HPC programmers: load balancing, software engineering, validation, testing, etc. That said, they also have some less common challenges such as having workflows in which datasets flow through analysis stages, some of which could be done in parallel (note that this actually did come up as a desirable idiom in other working groups; it just isn't how we typically do HPC programming using MPI today); they have the issue of generating a provenance described above; and they have a greater need for interactivity to perform computational steering and discovery. It would be worthwhile exploring what 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, one can do some amount of processing on the exascale machine in order to take advantage of the computational resources; to interleave analysis with computation; and to reduce the data volume for the purposes of storage. This suggests that we need good support for scalable implementations of several common data transforms. Examples include transposes (for example, changing an array of records into a record of arrays) and creating indexes for exascale data.

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

One brief conversation we had touched on the fact that we are already generating more data than the human eye can absorb and discern such that interesting behaviors or errors can easily go unnoticed. This problem will only get worse at exascale. (There is a tie in to performance analyses visualizations that could be made here as well).

We talked about the map-reduce programming model, popularized by Google and Yahoo. The group generally felt that map-reduce was too restrictive to be a crucial tool for exascale applications, in that its parallel activities had to be independent whereas the exascale mission-critical applications are not so embarrassingly parallel. There was one holdout in the group who felt they were not certain that they could rule it out with certainty. By and large, the group agreed that it was more important to look at what made map-reduce successful --- support for very general reduction operations; fault tolerance through redundant storage and resilient master-worker paradigms; and good disk/file storage strategies --- and strive to support those concepts and characteristics in our HPC programming models than to expect map-reduce to solve our exascale programming models issues for us.

[breakout input starts here:] Traditionally, Data Analysis/Viz 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, we do not envision fault tolerance to be a major issue. Our prediction is that in-situ data analysis and visualization will have to use the same check-point restart mechanism, whereas not-in-situ data 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.

On the other hand, data analysis and visualization needs and aims open the possibility of massive use of in-situ and near I/O analysis, which results in its unique set of questions for systems software. For in-situ analysis, what are the programming models and its 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 of 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 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. Since the primary intent of NVRAM use is checkpointing, we need to identify appropriate programming models and services provided by the OS to this end 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

DRAFT

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 compressing ratios due to better scaling compatibility. We foresee that this can be done in a way that allows the user to still see this as a standard format, such as NetCDF.

Codesign 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, since specialized data formats may highly reduce extensibility. A possible solution is the use of object models for data, which raises several challenges, such as doing 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 would increase the efficiency of storage systems. Seamless Memory/Storage is another joint interest topic, since, depending on the size of data, we can either do the analysis in the RAM of the visualization nodes or move the computation 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

Uncertainty Quantification (UQ) is likely to play an important role in exascale computing given that the driving missions are very simulation-based and will require justification of their results since they will have the potential to affect public policy. Moreover, many UQ computations are themselves potential exascale computations, as they use similarly compute-intensive processes to generate their results as the simulations that they are analyzing. One of the main themes of our overall discussion was that the disruptive nature of moving to exascale could be empowering for the field of uncertainty quantification since we will already be reconsidering our programming models and revisiting our application codes. This provides us an opportunity to make UQ more a part of our field's standard operating procedure rather than an afterthought.

One concept that we discussed to support 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 which 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

DRAFT

of overlaying such uncertainty quantifications on preexisting code without modifying it (via an annotation overlay file for example) in order to apply UQ analysis in a non-intrusive manner to pre-existing codes.

The challenge to this concept is that because of the relative youth of the UQ field, there is currently no single agreed-upon standard for representing these uncertainty factors. For example, some UQ representations use intervals while others use stochastic distributions. And even when two approaches use the same representation, like 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 suport an opaque sidecar 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 was not clear to our group whether or not this concept should be promoted to a programming language concept; it could be that using a library or software engineering practices would be sufficient. But mostly we were not certain and felt the concept had sufficient promise to deserve more attention going forward. It also struck us as being a good area for codesign between the UQ and programming models communities.

A related side topic was whether or not programming languages should <u>support interval types</u>. On one hand, this was generally considered to be a reasonable idea since intervals are a fairly well-understood and commonly-used concept. On the other hand, it raised questions about where to draw the line since intervals are just one of several UQ-appropriate representations.

One relevant item that came up for programming models or libraries is <u>support for routines to query the relative costs of various operations on the target hardware</u> --- for example, how much will it cost to access this variable or type of memory, or how fast can this processor type perform this computation relative to another one. Such support would allow codes to be written in a way that they can make space vs. 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. It's worth noting that this benefit is not specific to UQ computations; it was merely the only breakout I attended in which the concept was raised. This seems to be a codesign opportunity between the applications groups, the programming models groups, and the hardware architects.

UQ is an application area that has a need for <u>more dynamic execution models</u> 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 MC sampling.

Related, it was noted that UQ would benefit from fault-tolerant MPI collectives. For example, if a task dies before entering a barrier, it can be 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. A question that came up in our session was how a programming model would expose these multiple networks to the user so that they could specify which one they wanted to use for a particular communication.

Some UQ techniques rely on source-to-source transformation techniques. These techniques identified that they would benefit from compilers and libraries that support an open source infrastructure. The reason for this is that several of the analyses that these techniques use are similar to those supported by traditional compiler optimizations, and it would obviously be preferable to reuse these analyses for a given language/compiler than to reinvent them on a case-by-case basis. Another wishlist item for these techniques is programming model support for data structure linearization in which an arbitrarily complex data structure can be linearized to targets such as a file, memory buffer, or a network socket.

Adjoint computations, 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 such that it could be reversed after the fact. A similar capability would be a dataflow reversal mechanism in which stores are converted to loads, sends to receives, etc.

We 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 wide range of computational characteristics, so any rigid characterizations will be necessarily incomplete. But, 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.

DRAFT

Graphs are a very 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. But graphs that arise from different applications like cyber security or social networks have very 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. 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.

Combinatorial optimization problems are often solved by some kind of branch-and-bound technique. These techniques involve the dynamic construction and exploration of a search tree. The leaf nodes of the tree 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 very difficult to partition effectively. Thus a programming model that supports a global address space is highly advantageous. Also, many combinatorial algorithms have a high degree of fine-grained parallelism, but the available parallelism is dynamic. Generally speaking, 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; light-weight, one-sided communication for small messages; light-weight, fine-grained synchronization; and, support for maintaining logical state even when objects are migrated between processors. Key research challenges include the following questions. How can memory performance be maximized for applications that lack natural locality? Instead of just focusing 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?

Codesign opportunities include collaboration between application developers and language/library implementers to ensure that discrete mathematical algorithms are provided for in exascale hardware and software development.

DRAFT

Solvers and Optimization

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

Mini-applications: There was substantial discussion in this session about the need for miniapplications. In previous efforts, these types of codes have been called compact applications or synthetic applications, so the idea is not brand new. However, there is a bit of a difference in that we are asking the application teams to be the primary developers of the miniapplications, with the idea that a miniapp 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. Miniapps do not have to be working models. Instead, they need to correlate to performance issues, such that if we study miniapp performance in various setting (including rewriting it) the information we gain can be correlated to back to the real application. Miniapps are not a replacement for the real application, but allow us to use 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 API for forming and executing through a DAG. The PLASMA project is developing DAG capabilities for dense linear algebra, and intends to make their interfaces available to other developers, but 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 that the design of the programming environment must expose the architecture's execution model to the application efficiently and productively.

Cross-cutting Challenges and Research Needs for Programming Models to Support Exascale Computing

[Introductory sentences.]

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

DRAFT

- Investigate hybrid programming models (different inter- and intra-node programming models)
  - A likely solution will be MPI+X (hybrid) programming model: MPI inter-node programming model, X = an intra-node 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 rich memory consistency models that reflect the requirements of various applications.
- Develop memory management models that support alternative memory hierarchies based on new technologies, such as NV-RAM.

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

- Research a possible move to database techniques for input/output (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 NV-RAM.

DRAFT

- 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 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.

DRAFT

- 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 designers of new algorithms and software systems 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.

DRAFT

SYSTEM SOFTWARE

Introduction

System software is often unseen. It is the layer of software from the low-level machine control system up 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, and therefore the functionality available to science teams. The explosion of on-socket parallelism is a good example of how the hardware trends for exascale computing will impact the system software.

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

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

In the sections below, a summary of the topics discussed and conclusions reached during the breakout sessions of the Exascale Cross-cutting Workshop.

Understanding mathematical models, methods and algorithms in the context of system software

Understanding PDEs in the context of System Software

The PDE breakout groups explored meshing, PDE discretization, computational geometry and multi-physics issues. PDE solvers are typically characterized by using domain decomposition with nearest-neighbor communication patterns. Elliptic PDE solvers require smaller, non-local communication patterns, and multi-level 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 are also 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 timestep. 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. It is highly desirable that the logical layout of the communication patterns from the application perspective match the physical topology of

DRAFT

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, and the network topology will most 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 performance potential of the network. Possible approaches include programming interfaces that allow the system software to convey information about the underlying network topology to the application so that the application can structure communication appropriately, and, conversely, providing interfaces that allows the application to inform system software about the application's communication structure so that 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 and resilience was identified as a significant issue for applications moving to larger systems. From an application perspective, the system software should handle the management of faults transparently. However, scientists realize that a of partnership designing the system software will be required. It is unlikely that technology will evolve to transparently respond to faults, and it is also understood that simple checkpoint-restart scales poorly. New techniques, possibly that use local non-volatile memory may prove helpful.

However, it was universally recognized that fault notification must be a part of any solution that shares the responsibility for managing problems with the underlying system. At the most basic level an application has to 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 in order to define a standard fault notification API and nomenclature. A publish/subscribe interface was discussed so that applications, libraries, and even system software components could subscribe just to the fault events they are interested in and avoid being inundated with a continuous stream of fault messages. In the PDE II focus areas, local recovery from a fault using checkpoint/restart is possible. Moreover, the checkpoint files are typically small and could be stored in a neighbor's memory or non-volatile RAM.

Related to fault tolerance is the **debugging** of applications. As with existing petascale platforms, debugging applications at the exascale is one of the most important capabilities and is fundamental to application development. The challenge of debugging will be further compounded by the impact of faults on the system. In a sense, 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 of and applications, such an approach may be ineffective. New approaches to debugging, not only to recognize application-generated errors, but also to help identify and isolate system-generated errors will be critical.

DRAFT

Potential approaches to error identification and isolation include support for applications to provide the ability for the system software to check the validity of data flowing into and out of computational routines, as well as mechanisms that will allow for computing the same function multiple times in order to determine the causality of an error.

Another key area of interest for the groups focusing on PDEs was **memory management**. As systems grow toward exascale, the memory hierarchy continues to become deeper and more complex. From the perspective of an application, there are inadequate tools and interfaces to hint, manage, and control memory for run-time systems. 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 adaptive mesh refinement (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 alternatives approaches to providing fast local memory access, like 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.

Understanding Data Analysis and Visualization in the context of System Software

Traditionally, Data Analysis/Viz 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, we do not envision fault tolerance to be a major issue. Our prediction is that in-situ data analysis and visualization will have to use the same check-point restart mechanism, whereas not-in-situ data 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.

DRAFT

On the other hand, data analysis and visualization needs and aims open the possibility of massive use of in-situ and near I/O analysis, which results in its unique set of questions for systems software. For in-situ analysis, what are the programming models and its 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 of 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 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. Since the primary intent of NVRAM use is checkpointing, we need to identify appropriate programming models and services provided by the OS to this end 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 compressing ratios due to better scaling compatibility. We foresee that this can be done in a way that allows the user to still see this as a standard format, such as NetCDF.

Codesign 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, since specialized data formats may highly reduce extensibility. A possible solution is the use of object models for data, which raises several challenges, such as doing 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 would increase the efficiency of storage systems. Seamless Memory/Storage is another joint interest topic, since, depending on the size of data, we can either do the analysis in the RAM of the visualization nodes or move the computation to storage if it does not fit in RAM.

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

DRAFT

Understanding Uncertainty Quantification (UQ) in the context of System Software

Traditionally, Uncertainty Quantification (UQ) includes error arising from the hardware and software environment in the approximation error of numerical algorithms. However unlike petascale systems of today, the increasing component count of an exascale platform will require 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, but also a technique to manage scarce resources such as power. Investigation is needed to determine approaches for applications to communicate the need for determinism and accuracy to the system software so that it can act appropriately. All of these factors will have 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 jobs will have to be much more dynamic. The traditional space-shared, batch-scheduled usage model will likely not be effective at the exascale. The client/server model also requires a different failure model. Ideally, the system would respond differently based on whether 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.

Understanding Solvers and Optimization in the context of System Software

Heterogeneous multicore based chip designs will radically change how solvers are designed and written. In addition to the tradition 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 a very early start in what will require significant research and exploration. Furthermore, as solvers increasingly embrace hybrid programming models and seek ways to utilize heterogeneous resources automatic optimization and autotuning will need dramatic improvement.

Understanding Discrete Math in the context of System Software

DRAFT

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. As such the data movement 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 were identified as required by discrete math applications executing on Exascale computers. This is not an exhaustive list, but rather those system software features that need to be improved to meet the future needs of discrete math applications.

Adaptive Runtime. Discrete applications, such as branch and bound, often start out needing a small number of resources and the resource needs grow as the problem evolves. The runtime must be able to dynamically grow and shrink the number of resources required based on the application needs. Dynamic load balancing on a node is another capability required of the runtime system due to the shifting resource needs of many discrete math problems. An adaptive runtime is also needed to reconfigure the computer around faults and more generally adapt to a dynamically changing system configuration. 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 is a primary constraint in an Exascale system. Because discrete math applications require few FLOPS, it would be useful to have a resource manager that could reapportion the power budget so that the power would primarily go into data movement. This feature has an impact on the architecture since it requires that the system be over provisioned, i. e., some applications may want to use 20MW to move data fast, others may want to use 20MW to do FLOPS. If an application tried to do both then the system software would need to step in and constrain the total system power to 20MW.

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

Co-design Opportunities between Discrete Math and System Software

DRAFT

The workshop identified several opportunities where discrete mathematicians and system software developers could work together to create software of wide utility at the Exascale.

Simulation. Co-design of hardware and software requires discrete event simulation in order to study the effects of different hardware choices on different algorithms. 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 management and resource management within the system software.

Resource Management. A co-design opportunity is use of graph algorithms to schedule tasks on nodes and across nodes. The dynamic scheduling algorithms should be memory efficient lest they use up the limited memory on the node. They should be fast so that dynamic scheduling does not impede the application. Ideally, they would be able to do incremental updates so that the 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.

Cross-cutting system software themes for all break-out areas

File System. The main concern is the scalability and robustness of the file system. Historically the file system has been the weakest link of the largest supercomputers. I/O at the Exascale will be different from today and will include more in situ analysis to reduce data and speed up the discovery process.

Debug Tool. A very detailed single node debugger forms the base layer of the debugger suite. A debugger that can work on a modest size system with 10,000 nodes makes up the middle of the suite. There was skepticism that a debugger could be built that could work at the full size of an Exascale system. But several examples were given where bugs did not show up in today's applications on modest size systems. How to find large scale bugs? And what to do when a bug doesn't show up until an application has run several hours? These two questions illustrate the gap in our present debug capabilities.

As with existing petascale platforms, debugging applications at the exascale is one of the most important capabilities and is fundamental to application development. The challenge of debugging will be further compounded by the impact of faults on the system. In a sense, application errors will be indistinguishable from errors generated by

DRAFT

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

Performance Tools. Today's performance tools have not been able to keep up with the scale increase of the largest systems and this creates a gap in our understanding of 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. What is needed is a holistic performance measurement tool that vertically integrates an application's performance model with a detailed understanding of the (potentially heterogeneous) node and the system as a whole. This tool would utilize the "dynamic system information space" described above to collect and analyze configuration and performance data during a run in order to better understand the performance characteristics of an application.

Cross-cutting Challenges and Research Needs for System Software at the Exascale

[Introductory sentences.]

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

DRAFT

- 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 DAG-style task scheduling
- Provide support for efficient task replication
- Develop adaptive run-time 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 (CPU)
- 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 GAS to replace cache-coherence as a mechanism
- Research the use of GAS 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***

DRAFT

- 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 run-time 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 API
- 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 percent) is used for data analysis, observe that:
  - Because it will not be possible to move large amounts of data off of the main computing platform, resilience will be less of a problem.

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

- Research file system scalability and robustness
- Research the I/O bandwidth issue
- Research I/O for irregular data.

- **References**

DRAFT

CONCLUSIONS AND RECOMMENDATIONS

[More Introductory text here.]

The Priority Research Directions (PRDs) that the breakout panels identified in their workshop discussions are summarized as follows.

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

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

PRD 1.2: *Develop new mathematical models and formulations that effectively exploit anticipated exascale hardware architectures*

PRD 1.3: *Address numerical analysis questions associated with moving away from bulk-synchronous programs to multi-task approaches*

PRD 1.4: *Adapt data analysis algorithms to exascale environments*

PRD 1.5: *Extract essential elements of critical science applications as "mini-applications" that hardware and system software designers can use to understand computational requirements*

PRD 1.6: *Develop tools to simulate emerging architectures 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 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 fault tolerance/resilience*

DRAFT

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; reconfigurable local memory)*

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

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

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

DRAFT

REFERENCES

DRAFT

Appendix 1: IESP Roadmap summary and its relationship to this workshop

DRAFT

Appendix 2: Workshop Program/Agenda

DRAFT

Appendix 3: Workshop Participants

DRAFT

Appendix 4: Acronyms and Abbreviations

DRAFT

DRAFT