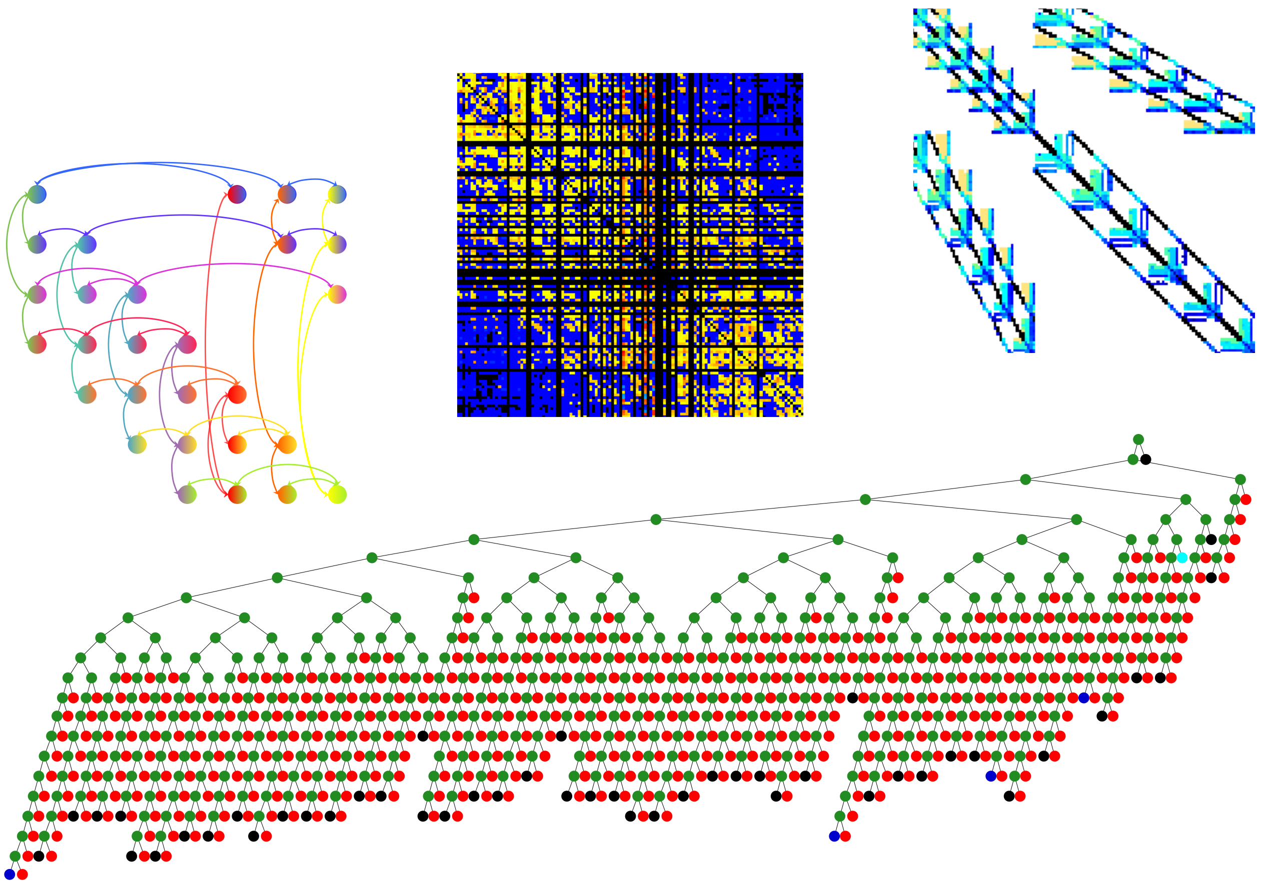
Report on the Workshop on Extreme-Scale Solvers: Transition to Future Architectures

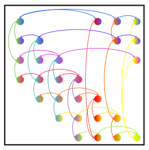
March 8-9, 2012, Washington, D.C.

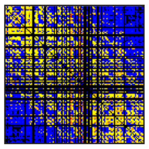


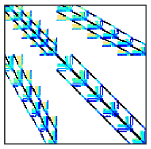
U.S. Department of Energy
Office of Advanced Scientific Computing Research

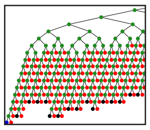
U.S. DEPARTMENT OF ENERGY

Cover image credits:


(Top left) Communication pattern between processors (each of which holds a column and a row communication group) in an implementation of the Lanczos algorithm for computing eigenvales of a sparse matrix; courtesy of H. M. Aktulga, Lawrence Berkeley National Laboratory.


(Top middle) Communication pattern for one of the coarse levels of the setup phase in an implementation of the algebraic multigrid method; courtesy of H. Gahvari and W. Gropp, Univ. of Illinois at Urbana-Champaign.


(Top right) Matrix from the solution of Maxwell's equation for a high-frequency circuit using finite-element modeling. The matrix is taken from the Univ. of Florida Sparse Matrix Collection and is courtesy of the Center for Computational Electromagnetics at the Univ. of Illinois at Urbana-Champaign.


(Bottom) A branch-and-bound tree search by MINOTAUR when solving a problem of selecting an optimal configuration for chemical synthesis; courtesy of S. Leyffer and A. Mahajan, Argonne National Laboratory.

Availability of This Report
This report is available, at no cost, at http://www.osti.gov/bridge. It is also available on paper to the U.S. Department of Energy and its contractors, for a processing fee, from:

U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831-0062
phone (865) 576-8401
fax (865) 576-5728
reports@adonis.osti.gov

Workshop on Extreme-Scale Solvers: Transition to Future Architectures

Sponsored by
U.S. Department of Energy
Office of Advanced Scientific Computing Research

March 8-9, 2012
American Geophysical Union, Washington, DC

Workshop Report Committee
Jim Ang (Sandia National Laboratories)
Katherine Evans (Oak Ridge National Laboratory)
Al Geist (Oak Ridge National Laboratory)
Michael Heroux (Sandia National Laboratories)
Paul Hovland (Argonne National Laboratory)
Osni Marques (Lawrence Berkeley National Laboratory)
Lois Curfman McInnes (Argonne National Laboratory)
Esmond Ng (Lawrence Berkeley National Laboratory)
Stefan Wild (Argonne National Laboratory)

DOE/ASCR Point of Contact
Karen Pao

Abstract

This report presents results from the DOE workshop on *Extreme-Scale Solvers: Transition to Future Architectures* held March 8–9, 2012, in Washington, D.C. The workshop brought together approximately 50 experts in the development of scalable solvers to determine research areas needed both in the near term for next-generation 100-petaflop supercomputers and in the longer term for the supercomputers expected at the end of the decade. The report also describes ways to make a smooth transition from the present solvers to the extreme-scale solvers of the future.

Contents

Executive Summary

This report presents results from the DOE workshop on *Extreme-Scale Solvers: Transition to Future Architectures* held March 8–9, 2012 in Washington, D.C. The workshop brought together experts in the development of scalable numerical solvers, with the goal of identifying research needed for solvers to effectively utilize 100-petaflop (PF) systems and beyond.

The needs of extreme-scale science are expected to drive a hundredfold increase in computational capabilities by mid-decade and a factor of 1,000 increase within ten years. These 100 PF (and larger) supercomputers will change the way scientific discoveries are made; moreover, the technology developed for those systems will provide desktop performance on par with the fastest systems from just a few years ago. Since numerical solvers are at the heart of the codes that enable these discoveries, the development of efficient, robust, high-performance, portable solvers has a tremendous impact on harnessing these computers to achieve new science. But future architectures present major challenges to the research and development of such solvers. These architectural challenges include extreme parallelism, data placement and movement, resilience, and heterogeneity.

To identify the research needed for numerical solvers to address these challenges, 50 solver experts from DOE laboratories, academia, and industry gathered at the American Geophysical Union for a $1\frac{1}{2}$-day workshop. In addition to identifying near-term and longer-term research opportunities, the participants identified what is needed from the external community to develop efficient solvers for 100 PF machines and beyond, including architecture simulators, tools, and portable programming models. The participants also discussed how to transition the user community to the new solvers. Crucial to this transition are increased interactions among the communities of hardware designers, algorithm and solver developers, and applications scientists. Also essential is encapsulation of the most promising approaches in reusable solver libraries that can bridge between lower-level issues in computer science and higher-level perspectives in applications.

Solver research opportunities include new classes of algorithms that need to be developed:

- Communication/synchronization hiding and reducing algorithms
- Mixed-precision-arithmetic algorithms
- Fault-tolerant and resilient algorithms
- Energy-efficient algorithms
- Stochastic algorithms
- Algorithms with reproducibility

These algorithmic areas as well as near-term and long-term research opportunities are discussed in detail in the report. Identified near-term research opportunities include minimization of data movement, circumvention of memory bandwidth limitations, semi-asynchronous solvers, algorithms using nonblocking collectives, exploitation of algorithmic/memory hierarchies, partitioned and parallel time integration, and improved algorithmic composition. A key characteristic of these near-term opportunities is that the results will also be useful for exascale simulations. Longer-term research opportunities include power awareness and management in solvers; selective data and computation reliability; probabilistic solvers; dynamic load balancing and scheduling of millions of solver tasks; and the need for working at higher levels of abstraction for ensemble computations, uncertainty quantification, stochastic models, and design optimization. These near- and long-term research activities will bring previously unavailable capabilities to scientific users, not simply faster or larger simulations.

The workshop participants emphasize that extreme-scale architectures require revisiting the mathematical, algorithmic, and software infrastructure that underlie today's solvers, and the community must pursue fundamentally new advances to enable next-generation extreme-scale science.

1 Introduction

Enabling extreme-scale science is a major ASCR priority. The thousandfold increase in computational capabilities expected over the next decade and a similar increase in the data volume from experimental facilities will change the way scientific discoveries are made. A DOE workshop on *Extreme-Scale Solvers: Transition to Future Architectures* was held March 8–9, 2012, at the American Geophysical Union in Washington, D.C. The workshop brought together experts in the development of scalable solvers to determine research areas needed for extreme-scale algorithms and software to effectively utilize 100-petaflop (PF) systems and beyond. Additional information is available at the workshop website: http://www.orau.gov/extremesolvers2012/.

Over fifty researchers from DOE laboratories, academia, and industry participated in the workshop. Their collective expertise spanned numerical algorithm design, deployment of numerical methods on high-performance computers, numerical libraries, science applications, computer science, and hardware architecture.

One purpose of the workshop was to explore opportunities for the solver community to influence the design of future extreme-scale computers. Helping with this exploration were attendees from Cray, IBM, Intel, and NVIDIA. The 100 PF systems appearing around 2015 are expected to be evolutionary modifications from the 20 PF-class systems being deployed today, likely employing variants of today's CPU and GPU technologies. The yet-to-be-determined architectures of extreme-scale systems are expected to be radically different from these 100 PF systems. Nevertheless, one can identify common characteristics of these extreme-scale systems, independent of the actual systems designs, that will require serious rethinking of today's numerical algorithms for large-scale scientific simulations.

Architectural features that are the most salient to the design, implementation, and deployment of all numerical algorithms for parallel, high-performance scientific computing include the following:

Extreme parallelism: Estimates of 2 to 3 orders of magnitude of parallelism over today's levels will require solvers to pay particular attention to Amdahl's law.

Data placement and movement: Optimizing data placement and movement will be key to performance, as well as a primary way for solvers to reduce power consumption.

Resilience: The number of failures is expected to increase with concurrency, requiring solvers that can run through or detect and recover from faults.

Heterogeneous architectures: Heterogeneity will be needed to meet the power requirements of the 100 PF systems; but little has been done to develop portable, hybrid codes that can run across different types of architectures.

The workshop attendees discussed how architectural features will affect the design and implementation of the numerical algorithms—especially algorithms such as direct and iterative linear solvers, nonlinear solvers, and eigensolvers—that are often the most computationally intensive parts of high-performance computer (HPC) science application codes. Attendees were divided into three breakout groups, each having a good mix of all the areas of expertise described above. The breakout groups were presented with a series of questions to discuss, including:

- What do mathematicians and computer scientists need to know about future architectures to be able to write efficient, robust, scalable, and portable high-performance solvers?
- Are specific architectural features needed or desired for the development of extreme-scale solvers?

- What do code developers need in terms of development tools and programming environments to deal with extreme parallelism and minimization of communication and data movement?
- What are some architecture-specific and architecture-independent solutions, and what are their respective strengths and weaknesses for extreme-scale solvers?
- How do we evolve from today's solvers to the extreme-scale algorithms and software needed for these 100 PF systems and architectures of the future?

The next two sections describe the potential impact of extreme-scale solvers research and the challenges faced by the solver community. Section 4 describes several overarching classes of algorithms that need fundamental research. This section also presents the near-term and long-term research opportunities in extreme-scale solvers that were identified at the workshop. Section 5 presents the experts' thoughts on how to make a smooth transition for the users from the present solvers to the extreme-scale solvers of the future.

2 Potential Impact

Applied mathematics is at the heart of all major science codes, and the numerical algorithms driving these science codes account for the majority of all floating-point operations performed on the nation's largest supercomputers. To effectively utilize the next generation of 100 PF and exascale machines, research and development must be conducted to devise numerical algorithms that can exploit the architectural characteristics of these future computers.

Increasing the efficiency of numerical solvers will significantly improve the ability of computational scientists to make scientific discoveries, because such solvers account for so much of the computation underlying scientific applications. Even theoretically scalable algorithms such as multigrid have practical limits of parallelism due to small, coarse-grid solves on large-scale systems. These fundamental issues mean that any efforts to improve the robustness and parallel execution of solvers, or reformulate problems, will have a tremendous impact on the types and fidelity of problems we can address.

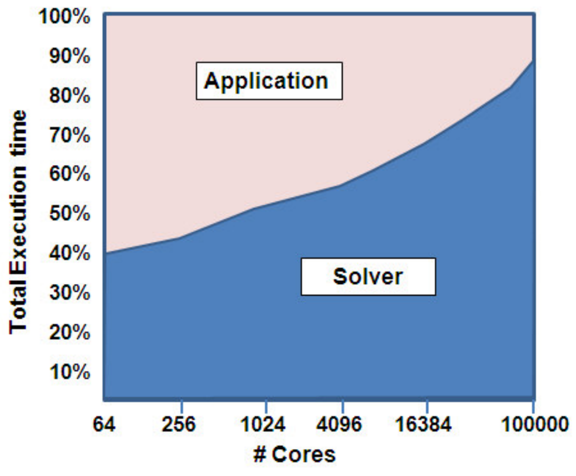


Figure 1: Comparison of time spent in the multiphysics application Charon vs. time spent in the linear solver (algebraic multigrid with GMRES), on 64 to 100,000 cores of a Cray XE6 (Cielo) using weak scaling. As core count and problem size increase, the fraction of time spent in the solver also increases. Although we use a state-of-the-art multilevel preconditioner, this trend is typical for many applications. The problem setup time is nearly constant from 64 to 100,000 cores, but solver time steadily increases. (Data courtesy of Paul Lin)

Solver Dominance. Figure 1 shows performance trends that extend a study found in [57]. These results show that as processor count and problem size increase, the time spent in the linear solver relative to the application grows and soon dominates the total execution time. These specific results are for a bipolar junction transistor (BJT) simulation using an algebraic multilevel preconditioned GMRES solver with smoothed aggregation and a local ILU smoother. Although Figure 1 focuses on weak scaling of a specific problem, these trends are typical of many science and engineering problems, and the behavior is essentially the same when strong scaling is considered.

To further demonstrate the dominance of solvers in scientific and engineering applications, two examples are given below.

Nuclear Structures. In nuclear structure calculations using an ab inito approach for light nuclei, sparse eigenvalue problems with over a billion degrees of freedom may have to be solved. In a calculation to study the reason for the isotope ^{14}C to have an extraordinary long half-life,[1] the dimension of, and the number of nonzero elements in, the matrix were, respectively, 1.1×10^9 and 39×10^{12}, and the eigensolver took over 65% of the total execution time using 216K cores on JaguarPF [60]. For nuclear structure calculations of heavier nuclei, the eigenvalue problems will become much larger. In some cases, the eigensolver is expected to take as much as 90% of the total execution time.

Atmospheric Modeling. For atmospheric climate models, petascale machines have enabled configurations of the dynamical core at fine horizontal spatial scales to resolve subglobal features by favorable weak scaling [22]. However, additional model features needed at these scales, such as the aerosol indirect effect and additional tracer quantities [64], dominate the computational cost and will need to be coupled to the current solver infrastructure. This situation creates both larger linear systems to solve and more nonlinear couplings—and also opportunities for increased parallelism.

Thus, highly scalable and efficient numerical solvers are needed and will continue to be absolutely crucial on future 100 PF and exascale supercomputers to enable these types of large-scale calculations. Additional benefits of solver redesign for 100 PF systems and beyond include the ability to add new model details and mathematical representations that have not been easily scalable, are prohibitively expensive with current machines, or both (see, e.g., [50]). Perhaps the most important contribution that extreme-scale solvers will make to DOE scientific advances is when they are embedded within new multiphysics scenarios and ensemble computations for uncertainty quantification, stochastic models, and design optimization because these will bring previously unavailable capabilities to scientific users.

Experts across the computational science landscape, from application developers to hardware designers, will be involved in enabling applications to run effectively on future supercomputers. Hence, research on extreme-scale solvers will provide a source of new ideas that will have broad benefits beyond the applied mathematics community.

3 Challenges Faced by the Solver Community

The solver community will have to address numerous challenges for the development and deployment of efficient solvers on 100 PF computers and beyond (see, e.g., [24–26]). The impact of the architectural features of these computers—such as more levels of memory, more concurrency with less memory per core, and heterogeneous cores—will be felt across the whole software stack. Applications have traditionally benefited from solvers implemented in highly optimized libraries that

[1] The half-life of ^{14}C is about $5,730$ years, but other isotopes of carbon have much shorter half-lives.

have transparently evolved over time to respond to changes in processor and system architectures. New approaches for the interplay between applications and solvers are likely to be needed in order to exploit the levels of parallelism expected on 100 PF computers.

Extreme levels of concurrency. Power and cooling constraints have already halted increases in microprocessor clock speeds; henceforth, improvements in performance will be achieved only through much higher levels of parallelism (see discussion in [71]). It is expected that major increases in parallelism will take place on chip, entailing a reorganization of CPU cores and memory. Technology trends suggest extreme-scale computers consisting of millions of nodes with thousands of lightweight cores or hundreds of thousands of nodes with more aggressive cores. In either case, solvers will need to accommodate a billion-way concurrency with small thread states.

Resilience and non-deterministic behavior. Extreme-scale systems will likely incur a high number of hardware faults because of the large number of devices on which they will be built [19]. Consequently, applications are expected to be subject to a large number of hard interrupts (failure of a device) and soft errors (change of a data value associated with faults in logic latches). Of particular concern is the dramatic growth predicted for soft errors in sub-45 nm technologies [61]. With these increases we are likely to see an increase in soft errors that are undetected by the hardware or system software, such that libraries and applications will be exposed to them.

Existing fault-tolerance approaches (i.e., approaches based on traditional, full checkpoint/restart) on these systems may incur significant overhead and not be viable options. Therefore, solvers will also have to play a role in the solution for failures (in particular if the mean time between failures is proportional to each solve); solvers will need to be equipped with mechanisms for detecting and dealing with interrupts and software errors, with minimal impact on an application, while providing appropriate feedback to the application.

Reduced memory sizes per core. The cost of memory technology is not decreasing as rapidly as the cost of devices designed for floating-point operations. Recent system trends already exhibit a small increase in the memory per node and a decrease in memory per core. Consequently, overall cost considerations are expected to limit the memory available on extreme-scale systems even further. Hence, solvers will need to exploit parallelism by performing more computation on local data, minimizing synchronization, and shifting the focus from the usual weak scaling (i.e., more resources to solve problems of larger sizes) to a scenario that favors strong scaling (i.e., more resources to solve a problem of fixed size to reduce time to solution).

Data storage and movement. On a node, data movement will be much more costly, relative to other operations. Furthermore, data access will be much more sensitive to how data are organized. Streaming memory systems, coalesced accesses, and non-uniform memory systems will require much more careful organization of data layout, in addition to concerns about efficient use of data once they are accessed. Across nodes, networks will provide similar challenges and opportunities.

Deep memory hierarchies. The cost for moving data from memory closer to the chip is comparable to a floating-point operation, while that cost is much higher for memory far from the chip. The adopted solution for reducing the overall cost of data movement across the chip has been a more complex memory hierarchy consisting of several levels (possibly up to four). This means that extreme-scale solvers may also need to be hierarchical, grouped in a way to exploit the locality of data access in the memory subsystem (reminding one of the notion of "cache-oblivious" algorithms). While solvers could potentially benefit from an explicit software management of memory, such management may not be simple and might even require changes in the hardware.

Portability with performance. Existing numerical libraries, and the solvers implemented therein, have transparently provided performance and portability, by encapsulating the complexity of system software and hardware. To a great extent that has been possible through MPI, or a combination of MPI and OpenMP. On current systems we already have other programming possibilities—such as directives for many integrated core co-processors, OpenACC directives, and CUDA—but these are not interoperable. Extreme-scale systems likely will make portability more challenging, given potentially very different architectures, memory hierarchies, and power considerations. While autotuners may help, higher levels of abstractions for solvers will need to be devised.

4 Solver Research Opportunities

Section 3 provides the context and motivation for and constraints on solver research. In this section we discuss the core content of the workshop: solver research opportunities. The list of topics presented here is not necessarily complete, but represents the opportunities identified by a large, representative group from the solver community.

The content is arranged in four subsections:

- Section 4.1: General classes of new algorithm development. The workshop identified the need for fundamental research on several classes of new algorithms over the next decade. Many of the near- and longer-term activities listed in this report feed into these overarching research areas that include, for example, communication reduction, energy efficiency, and fault tolerance, among others.

- Section 4.2: Near-term research addressing the needs of 100 PF computers. We expect that topics listed here can be at least partially realized within the 100 PF timeframe.

- Section 4.3: Longer-term research needed in the 100 PF through exascale timeframe. Although full realization of these efforts may go beyond the 100 PF timeframe, we believe efforts must start in earnest in order to make timely future delivery.

- Section 4.4: Needs from the external community. This subsection identifies several requirements from the greater community that are needed to facilitate the design and development of extreme-scale solvers.

A common theme in most of the topics is how to exploit emerging architectural features, by adapting existing algorithms to effectively use new architectures, by developing new techniques to cope with the architectural challenges, or by expanding the scope of the problem to increase available parallelism. Another theme is how to address the extreme environmental constraints we face, especially power and resilience. A third theme is how to use this time of "disruptive change" as an opportunity to better coordinate solver development efforts.

4.1 General Classes of New Algorithms

We begin by outlining several broad classes of algorithms identified by the workshop participants as presenting significant opportunities for fundamental research. In some instances these are new classes of algorithms emerging in response to revolutionary challenges at scale. Other classes build on and extend techniques from the dawn of computing (such as the error analysis of Wilkinson [70]) to address the challenges of today's computational landscape.

4.1.1 Communication/Synchronization-Hiding Algorithms

Data movement is widely recognized as the dominant cost for today's large-scale solvers with respect to execution time (and energy as well, which is discussed in detail in Section 4.1.5). On a single node, data moves through the memory hierarchy; and on a parallel computer, data also moves across the network. In both cases, because the cost of moving this data greatly exceeds the cost of computing with it on current HPC systems, communication is a bottleneck [55]. The hardware trends that have led to this communication bottleneck are predicted to continue (see Section 3), thus justifying an approach to algorithm development where reducing the communication cost is considered. To speak more concretely about communication costs, we use a latency-bandwidth model in which contiguous words of data are moved in messages; the communication cost of a message is the sum of a fixed latency cost, representing the overhead of a message, and a bandwidth cost, based on the number of words in the message.

One way to reduce communication cost is through *communication hiding*. A communication-hiding algorithm hides some fraction of the total communication cost by performing other useful work while waiting for data. It is often straightforward to overlap communication and computation without significantly reformulating an algorithm (e.g., by using nonblocking message-passing routines); however, this optimization promises at most a $2\times$ speedup. One can achieve much larger speedups by overlapping communication with other communication, although this approach often requires a substantial algorithmic reformulation. As an example, consider Krylov methods, which form a class of iterative solvers for sparse linear systems, where the bottleneck is often the dot product operation in the innermost loop. A Krylov method can be reformulated so that dot products and the interposed sparse matrix-vector multiply (SpMV) operations are overlapped. By pipelining this overlap, a communication-hiding Krylov method can reduce the effective cost of a dot product to the cost of a SpMV. In order to achieve this reformulation, additional computation must be performed; but this extra cost is justifiable because it is typically small compared with the communication savings. This communication-hiding approach offers potential speedups for GMRES (see, e.g., [34]). Nevertheless, while this pipelining approach appears to generalize easily to other Krylov methods, new algorithmic development is needed to generalize it to other types of computation with costly synchronizations, such as global dot products.

4.1.2 Communication/Synchronization-Reducing Algorithms

Rather than hiding communication, as described above, a *communication-reducing* algorithm actually moves less data or sends fewer messages than do conventional approaches, typically by improving data reuse. In some cases, this approach increases the computations performed (e.g., redundantly computing some quantities to avoid communicating), but the communication savings may outweigh this additional cost. In other cases, the approach increases memory requirements by storing duplicated data to avoid data movements. A communication-reducing algorithm may also move extra data in order to pack multiple messages together to reduce the overhead. Returning to the example of Krylov methods, the bottleneck is often the overhead (latency) of synchronizing the processors to perform the dot product and SpMV operations in the innermost loop. One can reduce the parallel latency costs by $O(s)$ by fusing together s Krylov iterations and performing the dot products and SpMVs with $O(1)$ synchronizations (instead of $O(s)$). Although this algorithmic formulation reduces latency, however, it increases the bandwidth and computational costs.

As is the case with communication-hiding algorithms, this tradeoff is often beneficial because the bandwidth and communication costs of dot products and SpMV kernels are typically dominated by the latency costs. Results for GMRES [62] show that this approach attains speedups in practice. Similar ideas also apply to multigrid solvers, in which repeated smoother applications are fused to reduce the number of synchronizations; such approaches have already demonstrated benefits [35].

Although communication-reducing algorithms have been developed for several linear algebra problems, such as dense LU and QR factorizations and sparse matrix-vector multiplication [6, 21, 44], much work is needed to develop communication-reducing algorithms for other classes of solvers. Moreover, the algorithmic reformulation may alter the numerical properties of the original algorithms; work is needed to ensure that the communication-reducing (as well as communication-hiding) variants are stable alternatives.

4.1.3 Mixed-Precision-Arithmetic Algorithms

Mixed-precision arithmetic has often referred to the use of a combination of single- and double-precision arithmetic in a calculation. The rationale behind the use of mixed-precision arithmetic is that single-precision arithmetic is generally faster than double-precision arithmetic. On the other hand, one usually cannot perform the entire computation using single precision, since certain parts of the calculation may be so sensitive to roundoff errors that double precision must be used. As an example, the recent work in [3] has focused on the use of mixed-precision arithmetic in dense and sparse matrix computations. For the solution of dense systems of linear equations using iterative refinement, Baboulin et al. perform the LU factorization and the subsequent triangular solutions using single-precision arithmetic. In the iterative refinement phase, the residual calculations and the updates of the solution are performed by using double-precision arithmetic in order to avoid catastrophic cancellations. Significant performance improvements over the full, double-precision implementation were reported for a parallel implementation [3].

The use of mixed precision arithmetic not only improves the time to solution but also can reduce memory requirements. As the memory available on current and future HPC systems is becoming relatively small, there is an increasing need to reformulate algorithms in order to reduce memory usage. If multiple precisions are available, then the use of lower-precision arithmetic means that less memory is required to store the data; hence, fewer bytes of data will need to be moved through the memory hierarchy or communicated between processes, thus reducing the overhead incurred. This may prove to be the primary reason for using mixed-precision arithmetic on future HPC systems.

Research is needed to identify what classes of algorithms can benefit from the use of mixed-precision arithmetic. In particular, researchers need to understand the numerical behavior of such an approach and to quantify when such an approach is appropriate. Also crucial in the presence of mixed-precision arithmetic is a solid understanding of how roundoff errors propagate through the software stack and how they may affect the accuracy achieved in application codes.

4.1.4 Fault-Tolerant and Resilient Algorithms

As the number of cores increases in HPC systems, the number of hard and soft errors also increases; see Section 3. One possibility for handling hard errors is through checkpoint/restart. The efficiency of checkpoint/restart depends on how fast the I/O system is and whether the file system can save the amount of data needed to restart a computation. A possible avenue of research is to investigate the use of localized checkpoints and asynchronous recovery to avoid global synchronous checkpoint/restart on extreme-scale systems. Another avenue is data compression. At the workshop, data compression was mentioned as a possible approach for reducing the size of the data that must be kept in memory. One benefit is to allow large problems to be solved. Another benefit of data compression is to reduce the size of the data that must be checkpointed.

For soft errors, if they are memory errors and if error-correcting code is available, then the faults may be handled by the hardware or firmware. Otherwise (e.g., when bit reversals occur in the processors), other means to recover the faults will be needed. Recent work has been focused on algorithm-based fault tolerance (ABFT). Much of the ABFT work on linear solvers is based on the use of checksum [18, 46, 59]. Research is needed to determine to what extent the idea of checksum can be applied to other numerical linear algebra solvers, as well as other classes of solvers. Also open is the question of what other techniques are available or feasible for fault recovery (see, e.g., [15]).

4.1.5 Energy-Efficient Algorithms

As alluded to in Section 3, power usage is going to be a significant challenge facing future HPC systems. Hardware design, such as incorporating low-energy accelerators, will lower the power requirement; but other means are needed to further lower the energy consumption. Algorithmic research offers one possible avenue. Many algorithms inherently have different degrees of concurrency. Sparse matrix algorithms—including those for direct methods, iterative methods, multigrid methods, and domain decomposition methods—are good examples, and they are often heavily used in the innermost loops of many large-scale scientific applications. These algorithms exhibit concurrency at the fine, medium, and coarse scales. For sparse matrix algorithms, these scales correspond, respectively, to the nonzero elements, rows/columns, and submatrices of the sparse matrix.

Execution time scalability depends on how effectively the available concurrency can be mapped to the available parallelism in the hardware, which ranges from fine (instructions or threads level), to medium (thread groups), to coarse (multicore cores with or without accelerators) scales. Hardware energy-efficiency relates to execution time speed-up, or performance efficiency. Ideally, energy should be directed where useful work can be done. The feasibility of such an approach has been demonstrated for sparse matrix computations expressed as weighted task graphs [14, 53], for which significant energy savings—with only a small degradation in performance—are achieved by using dynamic voltage and frequency scaling for CPUs and links that are not on the critical path. By utilizing attributes of the problem and by taking advantage of the power management capabilities provided by the hardware (at the CPU, link, and disk levels), it is conceivable that not only can energy be saved but speed-up can also be increased (e.g., by acceleration along the critical path).

The interaction between computations and hardware attributes represents a rich opportunity in algorithmic research for driving lightweight optimizations at execution time, including hardware power control features, acceleration modes, and low-power functional structures such as scratch-pads or in-memory/in-network computations.

4.1.6 Stochastic Algorithms

A recurring theme in the workshop was that algorithm developers and users should accept and embrace nondeterminism in computations at scale. Tolerating nondeterminism in data and operations is a useful, and likely necessary, feature to enable the fault-tolerant/resilient and asynchronous algorithms discussed above. Fundamental to the success of such solvers is analysis to identify the degree of stochasticity that individual computations and their mathematical operators can tolerate. In the face of stochasticity, analysis/metrics will need to be developed or rediscovered to define the nature of a solution and conduct error and stability analysis based on this stochasticity. Stochastic algorithms tend to be naturally flexible, so that one does only as much work as is needed to obtain a solution either within a given accuracy or with a given probability of correctness. Inevitably, this stochasticity can cause an algorithm to fail, for example because unrelaxable mathematical or physical properties and assumptions are violated. Opportunities exist for algorithms that can specify a level of stochasticity in their dependencies or recover from an unacceptable realization, or both. For example, an algorithm may benefit from reductions in global communication by using stochastic estimates of an inner product or linear solve, but these benefits will not be realized if the estimates result in a failure or significant increase in the number of iterations required.

Algorithms that employ pseudorandom numbers to reduce time to solution or solve higher-level problems also present a wealth of opportunities at the extreme scale. Monte Carlo-like approaches can increase orders of magnitude more concurrency by employing stochastic replications to resolve dynamics, propagate uncertainties, and find robust designs. Gains in performance, however, may require deeply embedding higher-level algorithms so that the computational components required throughout an algorithmic stack can be localized through amortized analysis. Other large classes of problems require stochastic approaches fundamentally different from Monte Carlo (e.g., stochastic dynamic programming [40]). In both cases, in order to perform well at scale, the higher-level optimization, calibration, and uncertainty problems, which are often an afterthought in the simulation sciences (e.g., after the physics has been gotten right), need to be moved to the forefront.

4.1.7 Algorithms with Reproducibility

Until recently, many scalable parallel applications could expect that results would be bit-wise identical from one run to another, as long as the same number of MPI processes were used. With the emergence of manycore nodes and dynamic parallel execution, this expectation cannot be maintained without prohibitive performance costs. The reason is that no guarantee exists that the order in which communication and data movement occur will be the same for repeated runs. This nondeterminism can lead to nonreproducible results, particularly if the calculation is forward unstable. This phenomenon is expected to worsen when parallelism increases.

In contrast, high-consequence decisions based on modeling and simulation, such as the certification of nuclear reactors, require some kind of reproducibility. The mathematical challenge is how to guarantee that answers which differ bitwise are simply a result of non-associative arithmetic and not some serious flaw in the algorithms or execution environment.

4.2 Near-Term Research Opportunities for 100 PF Machines

In order to provide a foundation of approaches and infrastructure that can extend toward exascale, the near-term opportunities discussed here for 100 PF machines emphasize longer-term perspectives and more generalized approaches than current work.

4.2.1 Enabling and Exploiting Mixed-Precision and Variable-Precision Arithmetic

The extreme computational environments of emerging systems demand a careful consideration of all data storage. One opportunity for improved efficiency or higher quality computations is to consider the precision of floating-point data, both in storage and computation. One obvious opportunity is to store and compute with 32-bit floating-point data. Single-precision operations are often about twice as fast as the corresponding double-precision operations; and the performance of many algorithms can be significantly enhanced, while maintaining accuracy where needed, by using an appropriate combination of 32-bit and 64-bit arithmetic. For example, using mixed-precision computations in [1] resulted in a near-2× speedup without significantly reducing the overall accuracy in the simulation of high-temperature superconductors. In contrast, 128-bit floating-point arithmetic can be used to mitigate the impact of roundoff error, for example using a high-precision accumulator for a norm or dot product, which can all but eliminate variability in answers [65].

As indicated in Section 4.1.3, many research questions must be addressed in both algorithms and data structures. One possible approach is to reformulate algorithms to find corrections rather than solutions (e.g., as in [3]). Another approach is to determine tolerable levels of precision adaptively or at execution time. In each case, the goal is to develop highly efficient solvers that operate on only as much data as necessary to obtain a result to a required precision.

4.2.2 Optimizing Data Placement and Movement

Data movement already dominates the cost of many simulations, and this trend will become more severe on emerging architectures. Many parallel algorithms already explicitly control data motion horizontally (e.g., off-node via message-passing communication), and much progress has been made to reduce vertical data movement, both through reordering of data and/or computation and through development of multistep (communication-reducing) and block algorithms. In the latter case, much of the work has been focused on specific areas of numerical computation, such as linear algebra. Other classes of algorithms may also benefit from the incorporation of communication-reducing techniques.

The energy constraints of 100 PF systems and beyond require new algorithms and data structures to address increasing vertical and horizontal hierarchies, with explicit acknowledgment that moving data is expensive whereas floating-point operations are cheap. Also important is increasing algorithmic intensity (e.g., through higher-order discretizations) and reformulating approaches to minimize poor-performing operations in favor of better performing alternatives. Related needs for appropriate metrics and tools to measure data movement are discussed in Section 4.4.

4.2.3 Circumventing Memory Bandwidth Limitations

Sparse matrix computations, which form the computational kernel of many DOE simulation codes, are severely memory bandwidth limited. In the best of circumstances one cannot achieve more than 15 to 20 percent of the processors' performance in such computations because the processors are waiting on memory loads for much of the time. Such computations arise because the paradigm for solving nonlinear PDEs is to iteratively perform a global linearization and then solve a sparse matrix linear problem. The linearization involves computing a sparse matrix approximation to a nonlinear problem, which is stored in main memory. The linear solution process requires constantly moving the sparse matrix entries from main memory into the CPU, where a small number of computations are performed using those values, before more values need to be loaded. Thus, no register or cache reuse of the sparse matrix entries occurs. In contrast, function evaluation code that encapsulates the physics of models tends to have much higher flops/load than do matrix kernels in Newton-Krylov and segregated solvers.

Research is therefore needed on techniques that require less memory bandwidth. Promising linear approaches that completely bypass sparse matrices include coefficient-matrix-free representations (see, e.g., [54]) and fast multipole methods [38]. Promising nonlinear approaches include nonlinear Schwarz (see, e.g., [13]) and full approximation scheme (also called nonlinear multigrid [9]), as well as nonlinear accelerators such as Anderson mixing [2, 68]. While preliminary work on such methods is promising [11, 47, 69], much research is still needed on algorithmic analysis and computer science techniques to make using such methods straightforward for application codes.

4.2.4 "Semi"-Asynchronous Solvers

Relaxation and reduction of synchronization are among the most important extreme-scale requirements, with a potential benefit of relieving pressure on load balancing. While eliminating periodic synchronization in solvers is likely impossible, many algorithms could be written in styles that require significantly less synchronization than occurs using current programming practices. Examples of tasks that need not be performed in synchronization with overall iterative algorithms include computing linear operators and refreshed preconditioners for the solution of successive linear systems arising in nonlinear and time-dependent problems, as well as in testing for convergence. While an algorithm progresses, such tasks could be performed asynchronously and dynamically redistributed to improve load balancing. Likewise, additive algorithmic variants (with no synchronization among components) may be favorable compared to multiplicative counterparts [51]. Relaxing synchrony could help overcome well-known scaling limitations for vector norms and dot products, used in many algorithms for convergence testing and orthogonalization.

4.2.5 Nonblocking Algorithms via Nonblocking Collectives

Nonblocking collectives are a crucial abstraction for hiding network latency. MPI-3 adds nonblocking variants of all standard collective operations, thereby creating opportunities for development of nonblocking algorithms. Nonblocking collectives enable the initiation of a collective operation followed by an overlapping, simultaneous execution of some other computation while the collective completes. If the overlapped computation is sufficiently long, we may see substantial improvements in scaling because of the inherent hiding of collective communication overhead and because any "OS jitter" that can limit scalability may also be hidden (see Section 4.1.1). A related issue is the need for better support for user-defined collective operations; currently no standard mechanism exists by which other libraries can implement nonblocking collectives that make progress.

Utilizing nonblocking collectives requires new or reformulated algorithms, but this work cannot be done without regard for numerical stability. Often formulations that expose more concurrency also tend to reduce robustness, unless care is taken in the formulation.

4.2.6 Exploiting Processor and Memory Hierarchies via Algorithmic Hierarchies

As emerging architectures incorporate deeper levels of processor and memory hierarchies, numerical algorithms that leverage such hierarchies to exploit finer-grained parallelism will provide significantly higher performance than those that do not. In other words, algorithms need inner kernels that work on data in the local cache, nested within mid-level kernels that work on data in the most local memory (associated with a subnode of the compute node), nested with higher-level kernels that use data on the node's memory, nested with parallel internode computations. Explicitly dealing with these hierarchies presents profound opportunities for leveraging the natural synergies among hierarchies in architectures, modeling, algorithms, and software—and for fundamentally rethinking problem formulations to exploit these relationships. Moreover, we can raise the level of abstraction of scientific problems, as further discussed in Section 4.3.7, thereby incorporating additional levels of algorithmic hierarchies whose parallel potential is independent of forward solves, resulting in a multiplicative speedup. An example is multilevel optimization, where lower-level discretization structures are exploited to reduce time to solution (see, e.g., [37, 63]).

4.2.7 Solver Components and New Composition Models

Because of both architectural challenges and the complexities of multiscale and multiphysics simulations expected to dominate on 100 PF systems, applications will encounter steep performance penalties for suboptimal code. Consequently, single-source solvers will be inadequate. Composable multilevel algorithms and data structures that can be automatically tuned to exploit operator-specific insight as well as memory and thread hierarchies are essential to address these issues. While preliminary work on composable data structures (see, e.g., [4]) and composable solvers (see, e.g., [10, 66]) have demonstrated the promise of such approaches, much research is needed on composable algorithms, for example, hybrid and hierarchical preconditioners that leverage operator-specific knowledge and custom implicit-explicit time integration methods that handle stiff and nonstiff system components. Furthermore, even within the category of standard iterative methods, robust versions of s-step iterative methods such as CA-GMRES [44] require a different component model for integrating preconditioners.

Additional opportunities include research on computer science techniques, such as code generation schemes and abstract data-structure-specific languages, to facilitate solver composition at higher levels of abstraction and injection into the HPC code development process.

4.2.8 Implicit-Explicit and Partitioned Time Integration

Time-marching strategies, typically used in all unsteady simulations, pose significant challenges for extreme-scale computing. While only implicit schemes can handle stiff problems and enable larger timesteps, they require the solution of a large-scale (non)linear system at each timestep; in contrast, explicit algorithms are limited to very small steps because of stability reasons, but they parallelize well and confer data locality. In many situations, however, different problem components evolve on different temporal scales. Thus, in practice, PDE discretizations often cannot be effectively integrated in time by using a single timestepping strategy. Partitioned methods overcome these difficulties by exploiting the advantages of multiple approaches and alleviating the shortcomings of using a single fixed strategy. Implicit-explicit (IMEX) methods, also known as semi-implicit schemes [17], are a class of partitioned time integration methods that generalize explicit and implicit schemes and handle stiffness. At the same time, they preserve as much data locality as possible [36] and accommodate variable redundant computations in favor of more scalable algorithms [16, 23]. Thus, by integrating the advantages of locality to facilitate scalable algorithms and implicitness to address stiff problems, partitioned methods offer an additional layer of flexibility to reach extreme-scale computing. Research needs include the development of architecture-aware schemes that facilitate dynamic and runtime partitioning strategies with reliable error and stability analysis.

4.2.9 Parallel-Time Integration

Current methods for transient simulations in production parallel applications are almost exclusively sequential in time. Regardless of how spatial domains are discretized, progress in time is accomplished by using one or more solutions from previous timesteps to compute the next. Therefore, the time to solution for a transient application is determined by how fast one timestep can be computed times the number of timesteps. Since high-fidelity simulations often require many timesteps and since explicit formulations often impose artificial timestep limitations for numerical stability, this sequentiality is a critical performance issue.

Parallel-time algorithms have been studied for many years. Some of the best known are parareal [32, 58] and spectral deferred correction [8, 27, 56]. These algorithms typically use a sequential predictor that is relatively cheap to compute, followed by parallel-in-time correction iterations. The challenge these methods face is that they generally require more computation and increase parallelism only incrementally. Even so, emerging approaches such as Krylov deferred correction [45, 48] permit accurate integration and exhibit sufficient parallelism to be promising. Furthermore, the timestep sequentiality bottleneck in current parallel applications is such a critical problem that we must continue research in this area.

4.2.10 Application-Oriented Solvers

While general-purpose solvers that provide interfaces at high levels of abstraction are essential for mainstream application scientists, solvers that exploit application-specific knowledge have already proved more robust, efficient, and scalable than general-purpose approaches for many application areas (see [28] and [49] for two of many examples). Solvers tailored to particular sets of equations, physics knowledge, and discretizations will have even greater importance in the 100 PF era and beyond. Solver libraries should provide opportunities for application scientists to customize those phases of solution deemed of highest priority for their particular simulations, while still leveraging library-provided capabilities for other aspects of work. Examples include application-specific preconditioners that can be used with library-provided Krylov solvers for scalable and efficient solution of linear systems and customized time-integration schemes that exploit application knowledge about stiff and nonstiff system components to achieve rapid time to solution, while leveraging library components for vector and matrix kernels and the solution of (non)linear systems.

4.2.11 Move to Greater Interoperability among Solver Libraries

Numerous areas of algorithmic research are essential to address the broad scope of extreme-scale opportunities, and diverse and complementary aspects of algorithms naturally are the focus of different researchers throughout the HPC community. Encapsulating the best algorithms and data structures in solver libraries is an effective approach to make these tools available to applications scientists. In fact, many applications currently employ solver libraries and application-specific frameworks that hide some of the parallelism so that application programmers need not be concerned with low-level message-passing details. These software layers will continue to be important on 100 PF and exascale machines and may help ease the transition to new systems.

Currently, many of the major DOE solver libraries have good interoperability. For example, PETSc [5] provides access to hypre [29] and ML (a Trilinos [43] package); Trilinos provides access to hypre and accepts PETSc data objects (matrices and vectors); and SuperLU [20] is an often-used component for hypre, PETSc, and Trilinos. Nevertheless, although the community has already made significant strides in interoperability among solver libraries, we believe that even stronger levels of interoperability will be needed to harness the intellectual contributions of the solver community into suites of tools that can be seamlessly employed by end users. Furthermore, migration to many-core architectures, which will require substantial refactoring of data structures and interfaces, and preparation for new classes of problems [41, 50], provides opportunities for tighter collaboration and commonality.

4.3 Longer-Term Research Opportunities for 100 PF through Exascale Computers

Although this workshop focused primarily on requirements and ideas for 100 PF computing systems—anticipated to be built using components and system software similar to what we have today—we want to ensure that any proposed activities are aligned well with future exascale systems. Topics discussed in this section are important for exascale systems and, although likely not to be fully realized in the 100 PF timeframe, require efforts today in order to build toward exascale capabilities.

For each topic, we discuss the core goals and the activities that were identified in order to realize these goals.

4.3.1 Solver Algorithms and Power Awareness and Management

One overarching concern for all future leadership-class systems is power and energy efficiency. Since solver solution times can dominate the overall execution time of an application on these large-scale systems, making algorithms "power-aware" is attractive [12, 14]. Exascale systems are likely to have hardware support for power management APIs, which are exposed to solvers through the runtime system software stack. If algorithm choice and implementation can be informed by power usage metrics, or by some useful model, we can reason about tradeoffs in performance vs. power or more easily select the most power-efficient approach from two that have similar performance.

Challenges in this pursuit include (1) characterizing power usage correlations between concepts that are meaningful to hardware designers and algorithm designers, (2) determining an interface that will allow applications and libraries to interact with the system, and (3) developing an understanding of what power management features are tunable (if any) by the user.

One near-term outcome from activities in this area would be power usage models correlated with algorithms concepts. Questions we would like addressed include the following: How does algorithm data access impact power usage? What is the relative cost difference between computation and data access? What is the relative cost of data access at each level in the memory hierarchy?

4.3.2 Selective Data and Computation Reliability

Without proper handling of the soft errors described in Section 3, long-running solvers and applications will almost certainly realize data corruption at some point during their execution.

One approach to mitigating the impact of these faults is to introduce selective reliability into our programming models, with underlying support from the runtime/operating system and hardware, which should be available in a similar way to power management by the time exascale systems emerge. By allowing a user to specify certain data and computations as more reliable than the default, we can reason about which data and computations must be reliable and which can tolerate potential soft errors. The goal of this distinction is to be resilient in the presence of soft errors while minimizing the amount of highly reliable data and computation. Such an approach may require solver and application developers to rethink how their algorithms and application codes should be designed and structured.

4.3.3 Approximate and Probabilistic Solvers

Traditionally solvers have been deterministic and exact up to some tolerance, ignoring the impact of roundoff error. One characteristic of these solvers is a strict sequentiality (within which parallelism is certainly present) that ultimately determines the potential for parallel execution. Another broad class of solvers relaxes the certitude of intermediate results and produces an answer that represents an approximation, or a family of approximate solutions. Algorithms of this class typically offer a much larger resource of parallelism and dramatically reduce the number of global synchronizations that can plague existing algorithms [39].

These approaches, however, require a more context-dependent usage and a more sophisticated integration of the solver answer into the application results. As a result, a "black-box" usage seems far away. Moreover, this class of approximate and probabilistic algorithms exhibits a large degree of parallelism but typically performs much more overall work, so the potential payoff may be seen only on our largest systems.

4.3.4 Coupled Nonlinear Solvers for Large-Scale Multiphysics

One natural approach to increasing the degree of parallelism for large-scale computation is to simultaneously solve multiple components of a coupled system that were previously solved in sequence. Such an approach can greatly improve convergence in some cases but also lead to scale resolution challenges. Often multicomponent and multiphysics simulations have very different temporal and spatial scales represented in each component such that decoupled approaches—which benefit from uniformity in scale within each component—can more easily solve a given problem in finite-precision arithmetic.

As a result, coupled approaches will require research into new nonlinear and time integration algorithms that can bridge the scale differences across components. By doing so, we can realize an increase in the amount of parallelism for coupled problems that will carry forward to all future parallel systems. Additional challenges and opportunities for multiphysics simulations at the extreme scale are discussed in [52].

4.3.5 Dynamic Load Balancing and Scheduling Millions of Tasks

The bulk synchronous SPMD computing model, represented by common usage of MPI, and MPI+X (where X is some node-local threading model such as OpenMP), rely heavily on an implicit task-to-processor mapping that is assumed to be static and one-to-one, although the X in MPI+X can support more general execution patterns. An emerging programming and execution model adopts a different strategy: manytasking. This strategy proposes a simple contract with the programmer such that, as long as the programmer can provide a sufficient number of parallel tasks that the runtime can schedule and execute simultaneously (and switch between when one task stalls on a resource request), then application performance will reach the active constraint of the algorithm being executed.

Since solvers are one of the primary computations on large-scale systems, a manytasking programming model has important implications for the design of new solver algorithms and implementations. For example, an active global address space (where subdomains are not permanently pinned to a process) and a lightweight manytasking programming and execution model (e.g., [33, 67]) could permit the scalability of data-driven task graph computations with strong implications for robust preconditioned iterative methods that are not possible today but are essential for the future.

4.3.6 Incremental Load Balancing

Complementary to the manytasking model is a repartitioning approach that addresses how we can incrementally balance workloads and keep the cost of our dependency analysis updates proportional. Presently most partitioning and domain decomposition algorithms require a complete rebuilding of data structures, factorizations, and parallel scheduling queues regardless of how minimal a repartitioning of data and work balance is.

As synchronization penalties of high end computers grow, we cannot afford global reconstruction of logically shared, physically distributed entities. If we are to adjust loads on a large-scale system in the presence of changing work demands, we need to develop new approaches such that the cost of rebalancing is proportional to the amount of change.

4.3.7 Higher Levels of Abstraction for Ensemble Computations, Uncertainty Quantification, Stochastic Models, and Design Optimization

Presently most DOE application scientists run single-point simulations—that is, a relatively small number of physical simulations with specific initial and boundary conditions. Extreme-scale computing provides the capability to replace this paradigm with the routine running of large ensemble computations for uncertainty quantification, stochastic models, and design optimization. This approach is favorable for extreme-scale computing because ensemble computations are closely related problems at the ODE level, the nonlinear solver level, and the linear solver level. Taking advantage of these relationships offers the chance for a great deal of data reuse, more data locality, and larger aggregates for communication, all of which lead to higher and more efficient utilization of emerging hardware.

It is often the case, when properly implemented, that running n related ensemble simulations has much lower execution time than n times that of a single simulation. However, at the linear solver level, most applications today use large-scale solvers in a way that is strongly biased to the solution of a single linear system of the form $Ax = b$, with little to no reuse of information that might be common to a family of such systems (even if some libraries support this reuse). Ensemble, uncertainty quantification, stochastic models, and design optimization often produce a family of related systems that have common properties, such as simultaneous multiple right-hand-sides ($AX = B$), sequential multiple right-hand-sides ($Ax_i = b_i,\ i = 1, \ldots$), or general families ($A_i x_i = b_i$), where the A_i have a strong fundamental relationship (e.g., A_1 is a perturbation of A_0). These relationships and others are currently underutilized in algorithm formulations and solver library infrastructure. Also, such optimization problems need different primitives that facilitate dealing with inequalities. As these new "metaproblems" involving systems and aggregate formulations become far more important, we will need new solver algorithms, data structures, and kernels in our libraries in order to realize optimal results [42].

4.4 Needs from the External Community

The workshop participants identified several requirements from the greater community that must be met in order for applied mathematicians to design and develop solvers that can fully exploit the capabilities of extreme-scale architectures.

Abstract machine models. An abstract machine model is required in order to develop new solver algorithms. The multicomputer model of homogeneous von Neumann machines connected via an opaque interconnection network is becoming increasingly inappropriate. New abstract machine models are needed in order to express the heterogeneity, hierarchical organization, and dynamic execution environment of contemporary and future architectures.

Programming models. Closely related to the need for abstract machine models is the need for well-defined programming models. Although the traditional message-passing programming model probably will not suffice for the 100-petaflop and exaflop machines under consideration, MPI+X—where X is a thread-based node-level programming model—seems likely to work for 100-petaflop systems. For exascale systems, however, it is unclear whether MPI+X will suffice or whether an as-yet-undetermined programming model or collection of interoperable programming models will be required.

Architectural simulators and performance modeling tools. Architectural simulators are required in order to test algorithm prototypes and to characterize both node-level performance and scaling behavior. In addition, performance modeling tools are needed to help predict the performance of algorithms and implementations on future architectures. The simulators and performance models should provide a characterization not just of execution time but also of memory footprint, peak power demands, and overall energy consumption. The simulators should also support software-controlled architectural resilience mechanisms and software-controlled memory subsystem behavior.

Performance measurement tools. Performance measurement tools are needed in order to guide the implementation of algorithms. These tools should measure all aspects of performance, including execution time, memory performance, data movement, faults, and energy consumption. The tools should provide not only graphical user interfaces but also APIs so that they can be used at execution time to guide the behavior of algorithms and enable adaptivity in solver implementations. A related capability is the ability at execution time to identify system components that may not be reliable.

Architectural features. The participants identified several architectural features as necessary for the development of scalable, high-performance solver algorithms. User management of cache, control over cache versus scratchpad memory, and control over the memory hierarchy were deemed critical to performance. Hardware support for resiliency is imperative. For the most efficient computation, however, this hardware support must be under software control. Solver developers can best determine how to respond to faults (fail, restart, ignore) and what level of reliability is required for different algorithmic phases. Hardware support for a diverse collection of nonblocking collectives could provide orders of magnitude improvements in scalability over the current generation of blocking collectives implemented in software.

Solver developers also require early information about the likely properties of future architectures. Among the characteristics deemed especially important are the ratio of compute capability to memory/network bandwidth, details on how collectives are computed, the frequency of "bit flips" during data transfer and during computation, the cache coherence model, the number of cores per node, and the topology within a node and across the system.

5 Transitioning to New Solvers

Specific and measurable pathways to transition from current solvers to those that effectively use 100 PF computing systems are needed to ensure successful development and incorporation of solvers across a range of application codes. These pathways begin with the development and optimization of algorithms on available petascale architectures that already contain heterogeneous processor layouts, while also keeping an eye toward likely performance on future systems. Initial developments can provide near-term gains for application scientists. The most successful efforts will analyze the behavior of algorithms that use these solvers on existing systems to identify issues that will prevent optimal solution behavior on future computing systems (see, e.g., [7, 30, 31]). Significant interaction across scientific expertise areas, including applied mathematicians, application scientists, computer scientists, and hardware designers, will assist with the initial analysis and the eventual revolution in solver development required to reach 100 PF machines and beyond.

5.1 Evolutionary Algorithmic Research

The initial development of algorithms involves evolutionary algorithmic research on near-term available architectures. Development of architecture-specific algorithms for existing machines, such as approaches mentioned in Section 4.2, can be optimized for a given system and provide detailed data regarding performance, robustness, and applicability to application codes. Utilizing externally provided tools identified in Section 4.4 can assist in this effort. Identification and quantification of the successes and limitations of these solvers will inform future solver efforts and provide valuable information for hardware designers, compiler vendors, and other computing capability providers. Part of this evolutionary process needs to involve the development of novel metrics, such as the use of memory motion rather than flops, that can predict how architecture-specific algorithms will perform on future computing systems. Other tools, such as portable translators to transition code to other architectures, will allow researchers to evaluate solver robustness across existing systems and minimize rewriting of code.

While identifying algorithms that are most likely to perform well on future systems using information about existing implementations is crucial, algorithms that do not perform well with current systems are still candidates for future systems. These algorithms also need metrics to predict solution capability within the changing landscape of hardware, such as memory layouts and network topology.

5.2 Transition to New Application-Library Interfaces

Although application-library interfaces for the construction of data objects (e.g., matrices and vectors) have been stable for many years, the departure from an SPMD, MPI-only programming model brings with it many new challenges when integrating solvers into applications. New application-library interfaces must support node-parallel execution and must more carefully manage data placement and storage costs. If these interfaces are not successfully redesigned and refactored, any gains we make in solver parallelism will be prematurely limited in scalability by the setup costs, which will not scale with increasing core counts per node, and by poor data locality and excessive memory use by having too many intermediate copies of data.

This transition is further complicated by the manycore transition that applications must make themselves. Most applications are presently moving or planning to move to MPI+OpenMP, which provides adequate node-level scalability and the easiest transition. Solvers can utilize OpenMP for some node-level parallel constructs, but not all, and must use something (such as pthreads, Intel Threading Building Blocks (TBB) or the emerging C++11 threading standards) in addition to or instead of OpenMP. Therefore, the node-level parallel environment must support multiple node-level execution environments. Specifically, the underlying runtime system must efficiently manage threads from two or more threading models (e.g., OpenMP and TBB), something that is not done well today.

Finally, all of this work must be done while looking forward to exascale systems and beyond, where MPI+X itself may not be sufficient.

5.3 Community Interactions to Facilitate Longer-Term Algorithmic Research

The complexity of issues that need to be addressed to extend near-term solver improvements to achieve successful simulation at the 100 PF requires interactions across many related research communities. Hardware designers and computer scientists need to inform solver developers of ongoing research and development and vice versa on topics such as application-specific network topology and memory adaptability. These communities also need to work together to develop correlations between energy cost, which is the ultimately limiting hardware design constraint, and algorithmically meaningful metrics to aid in solver development for future machines. Machine performance models could be useful to solver developers if provided with higher-order information such as reproducibility. Increased interaction between the solver and application developer communities is also crucial. On the one hand, solver developers need to know what types of operations will be executed more or less frequently in future scientific models. On the other hand, application developers need to know what types of operations must be minimized and what others will remain or become inexpensive. If applications must be reformulated to improve problem properties for solvers at the 100 PF, strong relationships among applications and solver developers will facilitate this work. Such communication among communities also will enable easier implementation of solvers that will target model equations and features that are yet to be included but are being considered as computing systems grow.

5.4 Revolutionary Algorithmic Research

Incorporating solver research results on near-term systems with knowledge from other experts in the computational science community will set the foundation for revolutionary solver development that is needed for simulations on 100 PF and beyond. Most important in the process of revolutionary solver development is to rethink the solver process—the entire solution stack, including both algorithms and programming approaches—where decisions must be informed by performance models and architectural understanding. Successful revolutionary developments in solver technology are obviously as yet unknown. However, to maximize algorithmic research breakthroughs, the community needs to (1) revisit algorithms that may not perform well on current systems, (2) engage and co-develop ideas with others in the "human stack" of computational science, and (3) demonstrate enhanced capability through effective use of evolutionary solver developments. An important aspect of this work is open source software development, as breakthroughs occur when many different ideas reinforce one another, crossing disciplines and narrow ruts of current thought. Novel ideas for extreme-scale solvers are made concrete in software; open exchange of software enables these ideas to mix and create a fertile ground for revolutionary advances. Such synergistic activities can attract the funding and talented staff necessary to support the opportunity to perform wide-open solver research.

6 Conclusions

This report describes the broad array of challenges in algorithms and software that face the solver community in the time horizon of 100 PF and beyond. These challenges highlight the importance of rethinking how solvers can exploit the architectural features of future computers. Because solver computations often dominate the overall execution time of large-scale applications, research into creating efficient, robust, resilient, and portable solvers is key to science advances of the next decade.

Moreover, solver research will provide a much-needed focal point of engagement for multidisciplinary research by applied mathematicians, computer scientists, and application scientists. In particular, the solver regime of the software stack naturally connects both to lower levels for fundamental algorithms/computer science issues and to higher levels for essential algorithms/applications synergy. Addressing the interdisciplinary solver issues for the 100 PF time horizon will help applications teams exploit the unprecedented computing capabilities of emerging architectures for scientific inquiry. Moreover, such efforts will lay the groundwork for next-generation paradigm shifts in new categories of simulations.

Acknowledgments

The organizers are grateful to the participants of the workshop, especially those who served as plenary speakers (Bill Gropp and Bob Lucas), panel participants (Victor Eijkhout, Robert Falgout, Tim Mattson, and Barry Smith), and as note-takers and slide-preparers (Karen Devine, Judith Hill, Mark Hoemmen, Richard Mills, Boyana Norris, and Todd Munson). The organizers are also grateful to Gail Pieper at Argonne for her editing of this report.

References

[1] G. Alvarez, M. S. Summers, D. E. Maxwell, M. Eisenbach, J. S. Meredith, J. M. Larkin, J. Levesque, T. A. Maier, P. R. C. Kent, E. F. D'Azevedo, and T. C. Schulthess. New algorithm to enable 400+ TFlop/s sustained performance in simulations of disorder effects in high-T_c superconductors. In *Proceedings of the ACM/IEEE Conference on Supercomputing (SC 2008)*, pages 61:1–61:10, 2008. doi:10.1109/SC.2008.5218119.

[2] D. G. Anderson. Iterative procedures for nonlinear integral equations. *J. Assoc. Comput. Mach.*, 12:547–560, 1965. doi:10.1145/321296.321305.

[3] M. Baboulin, A. Buttari, J. Dongarra, J. Kurzak, J. Langou, J. Langou, P. Luszczek, and S. Tomov. Accelerating scientific computations with mixed precision algorithms. *Comput. Phys. Commun.*, 180(12):2526–2533, 2009. doi:10.1016/j.cpc.2008.11.005.

[4] C. G. Baker, M. A. Heroux, H. C. Edwards, and A. B. Williams. A light-weight API for portable multicore programming. In *18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP 2010)*, pages 601–606. IEEE, 2010. doi:10.1109/PDP.2010.49.

[5] S. Balay, J. Brown, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.2, Argonne National Laboratory, 2011. URL `http://www.mcs.anl.gov/petsc`.

[6] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz. Minimizing communication in numerical linear algebra. *SIAM J. Mat. Anal. Appl.*, 32(3):866–901, 2012. doi:10.1137/090769156.

[7] A. Bhatele, P. Jetley, H. Gahvari, L. Wesolowski, W. D. Gropp, and L. V. Kalé. Architectural constraints to attain 1 exaflop/s for three scientific application classes. In *International Parallel and Distributed Processing Symposium (IPDPS 2011)*, pages 80–91. IEEE, 2011. doi:10.1109/IPDPS.2011.18.

[8] A. Bourlioux, A. T. Layton, and M. L. Minion. High-order multi-implicit spectral deferred correction methods for problems of reactive flow. *J. Comput. Phys.*, 189(2):651–675, Aug. 2003. doi:10.1016/S0021-9991(03)00251-1.

[9] A. Brandt. Multilevel adaptive solution to boundary value problems. *Math. Comp.*, 31:333–390, 1977. doi:10.1090/S0025-5718-1977-0431719-X.

[10] J. Brown, M. G. Knepley, D. A. May, L. C. McInnes, and B. F. Smith. Composable linear solvers for multiphysics. Preprint ANL/MCS-P2017-0112, Argonne National Laboratory, 2012. URL `http://www.mcs.anl.gov/uploads/cels/papers/2017-0112.pdf`.

[11] P. Brune, M. Knepley, B. Smith, and X. Tu. Composing scalable nonlinear solvers. Preprint ANL/MCS-P2010-0112, Argonne National Laboratory, 2012.

[12] V. Bui, B. Norris, K. Huck, L. C. McInnes, L. Li, O. Hernandez, and B. Chapman. A component infrastructure for performance and power modeling of parallel scientific applications. In *Workshop on Component-Based High Performance Computing*, pages 6:1–6:11, 2008. doi:10.1145/1456190.1456199.

[13] X.-C. Cai and D. E. Keyes. Nonlinearly preconditioned inexact Newton algorithms. *SIAM J. Sci. Comput.*, 24:183–200, 2002. doi:10.1137/S106482750037620X.

[14] G. Chen, K. Malkowski, M. Kandemir, and P. Raghavan. Reducing power with performance constraints for parallel sparse applications. In *International Parallel and Distributed Processing Symposium (IPDPS 2005, Workshop 11)*. IEEE, 2005. doi:10.1109/IPDPS.2005.378.

[15] Z. Chen. Fault tolerant iterative methods: Algorithm-based recovery without checkpointing. In *Proceedings of the 20th ACM International Symposium on High-Performance Parallel and Distributed Computing (HPDC 2011)*, 2011. URL `http://inside.mines.edu/~zchen/papers/hpdc2011.pdf`.

[16] E. Constantinescu and A. Sandu. Extrapolated implicit-explicit time stepping. *SIAM J. Sci. Comp.*, 31(6):4452–4477, 2010. doi:10.1137/080732833.

[17] M. Crouzeix. Une méthode multipas implicite-explicite pour l'approximation des équations d'évolution parabolique. *Numer. Math.*, 35:257–276, 1980. doi:10.1007/BF01396412.

[18] T. Davies, C. Karlsson, H. Liu, C. Ding, and Z. Chen. High performance LINPACK benchmark: A fault tolerant implementation without checkpointing. In *Proceedings of the International Conference on Supercomputing (ICS 2011)*, pages 162–171, 2011. doi:10.1145/1995896.1995923.

[19] N. DeBardeleben, J. Laros, J. T. Daly, S. L. Scott, C. Engelmann, and B. Harrod. High-end computing resilience: Analysis of issues facing the HEC community and path-forward for research and development. Whitepaper, Dec. 2009. URL `http://www.csm.ornl.gov/~engelman/publications/debardeleben09high-end.pdf`.

[20] J. Demmel, J. Gilbert, and X. S. Li. SuperLU users' guide. Technical Report LBNL-44289, Lawrence Berkeley National Laboratory, 2003. URL `http://crd.lbl.gov/~xiaoye/SuperLU/`.

[21] J. Demmel, J. Hoemmen, M. Mohiyuddin, and K. Yelick. Avoiding communication in sparse matrix computations. In *International Parallel and Distributed Processing Symposium (IPDPS 2008)*. IEEE, 2008. doi:10.1109/IPDPS.2008.4536305.

[22] J. Dennis, J. Edwards, K. J. Evans, O. Guba, P. Lauritzen, A. Mirin, A. St.-Cyr, M. Taylor, and P. H. Worley. A scalable spectral element dynamical core for the Community Atmosphere Model. *Int. J. High Perf. Comp. App.*, 26:74–89, 2012. doi:10.1007/978-3-642-01973-9.

[23] P. Deuflhard. Recent progress in extrapolation methods for ordinary differential equations. *SIAM Rev.*, 27(4):505–535, Dec. 1985. doi:10.1137/1027140.

[24] DOE report. Architectures and technology for extreme scale computing, December 2009. URL `http://science.energy.gov/~/media/ascr/pdf/program-documents/docs/Arch_tech_grand_challenges_report.pdf`.

[25] DOE report. Workshop on architectures I: Exascale and beyond, August 2011. URL `http://www.orau.gov/archI2011`.

[26] DOE report. Exascale programming challenges, July 2011. URL `http://science.energy.gov/~/media/ascr/pdf/program-documents/docs/ProgrammingChallengesWorkshopReport.pdf`.

[27] A. Dutt, L. Greengard, and V. Rokhlin. Spectral deferred correction methods for ordinary differential equations. *BIT Num. Math.*, 40(2):241–266, 2000. doi:10.1023/A:1022338906936.

[28] H. Elman, V. E. Howle, J. Shadid, R. Shuttleworth, and R. Tuminaro. A taxonomy and comparison of parallel block multi-level preconditioners for the incompressible Navier-Stokes equations. *J. Comput. Phys.*, 227(3):1790–1808, Jan. 2008. doi:10.1016/j.jcp.2007.09.026.

[29] R. Falgout et al. hypre users manual. Technical Report Revision 2.8.0, Lawrence Livermore National Laboratory, 2011. URL `https://computation.llnl.gov/casc/hypre/`.

[30] H. Gahvari and W. Gropp. An introductory exascale feasibility study for FFTs and multigrid. In *International Parallel and Distributed Processing Symposium (IPDPS 2010)*, pages 1–9. IEEE, 2010. doi:10.1109/IPDPS.2010.5470417.

[31] H. Gahvari, A. H. Baker, M. Schulz, U. M. Yang, K. E. Jordan, and W. Gropp. Modeling the performance of an algebraic multigrid cycle on HPC platforms. In *Proceedings of the International Conference on Supercomputing (ICS 2011)*, pages 172–181. ACM, 2011. doi:10.1145/1995896.1995924.

[32] M. J. Gander and S. Vandewalle. Analysis of the parareal time-parallel time-integration method. *SIAM J. Sci. Comput.*, 29:556–578, 2007. doi:10.1137/05064607X.

[33] G. Gao, T. Sterling, R. Stevens, M. Hereld, and W. Zhu. ParalleX: A study of a new parallel computation model. In *International Parallel and Distributed Processing Symposium (IPDPS 2007)*, pages 1–6. IEEE, 2007. doi:10.1109/IPDPS.2007.370484.

[34] P. Ghysels, T. Ashby, K. Meerbergen, and W. Vanroose. Hiding global communication latency in the GMRES algorithm on massively parallel machines. Tech. report 04.2012.1, Intel Exascience Lab, Leuven, Belgium, 2012. URL `http://twna.ua.ac.be/sites/twna.ua.ac.be/files/latency_gmres.pdf`.

[35] P. Ghysels, P. Kłosiewicz, and W. Vanroose. Improving the arithmetic intensity of multigrid with the help of polynomial smoothers. *Numer. Linear Algebra Appl.*, 19(2):253–267, 2012. doi:10.1002/nla.1808.

[36] F. Giraldo, J. Kelly, and E. Constantinescu. IMEX formulations of a 3d nonhydrostatic unified model of the atmosphere (NUMA). *In preparation*, 2012.

[37] S. Gratton, A. Sartenaer, and P. L. Toint. Recursive trust-region methods for multiscale nonlinear optimization. *SIAM J. Opt.*, 19(1):414–444, 2008. doi:10.1137/050623012.

[38] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *J. Comput. Phys.*, 73 (2):325–348, 1987. doi:10.1016/0021-9991(87)90140-9.

[39] N. Halko, P. G. Martinsson, and J. A. Tropp. Finding structure with randomness: Probabilistic algorithms for matrix decompositions, Part I: Introduction. *SIAM Rev.*, 53(2):217–288, 2011. doi:10.1137/090771806.

[40] F. Hanson. Techniques in computational stochastic dynamic programming. *Control Dynam. Sys.*, 76:103–162, 1996. doi:10.1016/S0090-5267(96)80017-X.

[41] M. A. Heroux. Software challenges for extreme scale computing: Going from petascale to exascale systems. *Int. J. High Perform. Comput. Appl.*, 23(4):437–439, Nov. 2009. doi:10.1177/1094342009347711.

[42] M. A. Heroux. Emerging architectures and UQ: Implications and opportunities. In *Proceedings of the 2011 IFIP Workshop in Uncertainty Quantification for Scientific Computing*, August 2011. URL http://math.nist.gov/IFIP-UQSC-2011/slides/Heroux.pdf.

[43] M. A. Heroux and J. M. Willenbring. Trilinos Users Guide. Technical Report SAND2003-2952, Sandia National Laboratories, 2003. URL http://trilinos.sandia.gov/.

[44] M. Hoemmen. *Communication avoiding Krylov subspace methods*. PhD thesis, Computer Science Division, U.C. Berkeley, May 2010. URL http://www.cs.berkeley.edu/~mhoemmen/pubs/thesis.pdf.

[45] J. Huang, J. Jia, and M. Minion. Arbitrary order Krylov deferred correction methods for differential algebraic equations. *J. Comput. Phys.*, 221(2):739–760, 2007. doi:10.1016/j.jcp.2006.06.040.

[46] K.-h. Huang and J. A. Abraham. Algorithm-based fault tolerance for matrix operations. *IEEE Trans. Comput.*, C-33(6):518–528, 1984. doi:10.1109/TC.1984.1676475.

[47] F.-N. Hwang and X.-C. Cai. A class of parallel two-level nonlinear Schwarz preconditioned inexact Newton algorithms. *Comp. Methods App. Mech. Eng.*, 196:1603–1611, 2007. doi:10.1016/j.cma.2006.03.019.

[48] J. Jia and J. Liu. Stable and spectrally accurate schemes for the Navier-Stokes equations. *SIAM J. Sci. Comput.*, 33(5):2421–2439, Sept. 2011. doi:10.1137/090754340.

[49] D. Kaushik, M. Smith, A. Wollaber, B. Smith, A. Siegel, and W. S. Yang. Enabling high-fidelity neutron transport simulations on petascale architectures. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis (SC 2009)*, pages 67:1–67:12, New York, NY, 2009. doi:10.1145/1654059.1654128.

[50] D. E. Keyes. Partial differential equation-based applications and solvers at extreme scale. *Int. J. High Perform. Comput. Appl.*, 23(4):437–439, Nov. 2009. doi:10.1177/1094342009347504.

[51] D. E. Keyes. Exaflop/s: The why and the how. *C.R. Acad. Sci. II B*, 339:70–77, 2011. doi:10.1016/j.crme.2010.11.002.

[52] D. E. Keyes, L. C. McInnes, C. Woodward, W. D. Gropp, E. Myra, M. Pernice, J. Bell, J. Brown, A. Clo, J. Connors, E. Constantinescu, D. Estep, K. Evans, C. Farhat, A. Hakim, G. Hammond, G. Hansen, J. Hill, T. Isaac, X. Jiao, K. Jordan, D. Kaushik, E. Kaxiras, A. Koniges, K. Lee, A. Lott, Q. Lu, J. Magerlein, R. Maxwell, M. McCourt, M. Mehl, R. Pawlowski, A. Peters, D. Reynolds, B. Riviere, U. Rüde, T. Scheibe, J. Shadid, B. Sheehan, M. Shephard, A. Siegel, B. Smith, X. Tang, C. Wilson, and B. Wohlmuth. Multiphysics Simulations: Challenges and Opportunities. Technical Report ANL/MCS-TM-321, Argonne National Laboratory, Dec. 2011. URL http://www.ipd.anl.gov/anlpubs/2012/01/72183.pdf. Report of workshop sponsored by the Institute for Computing in Science (ICiS), Park City, Utah, July 30 - August 6, 2011.

[53] H. Kimura, M. Sato, Y. Hotta, T. Boku, and D. Takahashi. Empirical study on reducing energy of parallel programs using slack reclamation by DVFS in a power-scalable high performance cluster. In *Proceedings of 2006 IEEE International Conference on Cluster Computing*, 2006. doi:/10.1109/CLUSTR.2006.311839.

[54] D. A. Knoll and D. E. Keyes. Jacobian-free Newton-Krylov methods: A survey of approaches and applications. *J. Comput. Phys.*, 193:357–397, 2004. doi:10.1016/j.jcp.2003.08.010.

[55] P. Kogge, K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, K. Hill, J. Hiller, S. Karp, S. Keckler, D. Klein, R. Lucas, M. Richards, A. Scarpelli, S. Scott, A. Snavely, T. Sterling, R. S. Williams, and K. Yelick. Exascale computing study: Technology challenges in achieving exascale systems, 2008. URL `http://www.notur.no/news/inthenews/files/exascale_final_report_100208.pdf`.

[56] A. T. Layton and M. L. Minion. Conservative multi-implicit spectral deferred correction methods for reacting gas dynamics. *J. Comput. Phys*, 194(2):697–715, Mar. 2004. doi:10.1016/j.jcp.2003.09.010.

[57] P. T. Lin and J. N. Shadid. Towards large-scale multi-socket, multicore parallel simulations: Performance of an MPI-only semiconductor device simulator. *J. Comput. Phys.*, 229(19): 6804–6818, Sept. 2010. doi:10.1016/j.jcp.2010.05.023.

[58] J. L. Lions, Y. Maday, and G. Turinici. A "parareal" in time discretization of PDE's. *C.R. Acad. Sci. I-Math.*, 332(7):661–668, 2001. doi:10.1016/S0764-4442(00)01793-6.

[59] F. T. Luk and H. Park. Fault-tolerant matrix triangularizations on systolic arrays. *IEEE Trans. Comput.*, 37(11):1434–1438, 1988. doi:10.1109/12.8712.

[60] P. Maris, J. P. Vary, P. Navrátil, W. E. Ormand, H. Nam, and D. J. Dean. Origin of the anomalous long lifetime of ^{14}C. *Phys. Rev. Lett.*, 106:202502, May 2011. doi:10.1103/PhysRevLett.106.202502.

[61] S. Mitra, T. Karnik, and N. Seifert. Logic soft errors in sub-65nm technologies design and CAD challenges. In *Proceedings 42nd Design Automation Conference*, pages 2–4. IEEE, 2005. doi:10.1109/DAC.2005.193762.

[62] M. Mohiyuddin, M. Hoemmen, J. Demmel, and K. Yelick. Minimizing communication in sparse matrix solvers. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, page 36. ACM, 2009. doi:10.1137/090769156.

[63] S. Nash. A multigrid approach to discretized optimization problems. *Optim. Methods Softw.*, 14(1/2):99–116, 2000. doi:10.1080/10556780008805795.

[64] Neale, R. B. et al. Description of the community atmosphere model (CAM 5.0). *NCAR Tech. Note*, TN-486+STR, 2010. URL `http://www.cesm.ucar.edu/models/cesm1.0/cam/docs/description/cam5_desc.pdf`.

[65] R. W. Robey, J. M. Robey, and R. Aulwes. In search of numerical consistency in parallel programming in search of numerical consistency in parallel programming. *Parallel Comput.*, 37(Feb.):217–229, 2011. doi:10.1016/j.parco.2011.02.009.

[66] B. Smith, L. C. McInnes, E. Constantinescu, M. Adams, S. Balay, J. Brown, M. Knepley, and H. Zhang. PETSc's software strategy for the design space of composable extreme-scale solvers. Preprint ANL/MCS-P2059-0312, Argonne National Laboratory, 2012. URL `http://www.mcs.anl.gov/uploads/cels/papers/P2059-0312.pdf`.

[67] SWARM. SWARM (SWift Adaptive Runtime Machine). `http://www.etinternational.com/index.php/products/swarmbeta/`.

[68] H. F. Walker and P. Ni. Anderson acceleration for fixed-point iterations. *SIAM J. Numer. Anal.*, 49(4):1715–1735, 2011. doi:10.1137/10078356X.

[69] H. F. Walker, C. S. Woodward, and U. M. Yang. An accelerated fixed-point iteration for solution of variably saturated flow. In *XVIII International Conference on Computational Methods in Water Resources (CMWR 2010)*, 2010. URL http://users.wpi.edu/~walker/Papers/var_sat_flow,CMWR_2010,CIMNE,2010.pdf.

[70] J. H. Wilkinson. *Rounding Errors in Algebraic Processes.* National Physical Laboratory Notes on Applied Science No. 32, 1963.

[71] M. Wolfe. Compilers and more: Programming at exascale (Part I); Expose, express, exploit (Part II); Exascale programming requirements (Part III). *HPCWire*, March 8, March 28, and April 14, 2011. URL http://www.hpcwire.com/hpcwire/2011-03-08/.

Workshop Participants

James Ang, Sandia National Laboratories
Wolfgang Bangerth, Texas A&M University
John Bell, Lawrence Berkeley National Laboratory
David Brown, Lawrence Berkeley National Laboratory
Xiao-Chuan Cai, University of Colorado Boulder
Edmond Chow, Georgia Tech University
Jonathan Cohen, NVIDIA
Ed D'Azevedo, Oak Ridge National Laboratory
Karen Devine, Sandia National Laboratories
Lori Diachin, Lawrence Livermore National Laboratory
Jack Dongarra, University of Tennessee
Milo Dorr, Lawrence Livermore National Laboratory
Sudip Dosanjh, Sandia National Laboratories
Victor Eijkhout, University of Texas at Austin
Howard Elman, University of Maryland
Katherine Evans, Oak Ridge National Laboratory
Robert Falgout, Lawrence Livermore National Laboratory
George Fann, Oak Ridge National Laboratory
Paul Fischer, Argonne National Laboratory
Al Geist, Oak Ridge National Laboratory
William Gropp, University of Illinois at Urbana-Champaign
Ray Grout, National Renewable Energy Laboratory
Anshul Gupta, IBM
Michael Heroux, Sandia National Laboratories
Judith Hill, Oak Ridge National Laboratory
Rob Hoekstra, Sandia National Laboratories
Mark Hoemmen, Sandia National Laboratories
Paul Hovland, Argonne National Laboratory
Victoria Howle, Texas Tech University
Moe Khaleel, Pacific Northwest National Laboratory
Matthew Knepley, University of Chicago
Nick Knight, University of California, Berkeley
Sherry Li, Lawrence Berkeley National Laboratory
Robert Lucas, University of Southern California
Barney MacCabe, Oak Ridge National Laboratory
Osni Marques, Lawrence Berkeley National Laboratory
Tim Mattson, Intel
Lois Curfman McInnes, Argonne National Laboratory
Richard Mills, Oak Ridge National Laboratory
Todd Munson, Argonne National Laboratory
Esmond Ng, Lawrence Berkeley National Laboratory
Boyana Norris, Argonne National Laboratory
Alex Pothen, Purdue University
Martin Schulz, Lawrence Livermore National Laboratory
Barry Smith, Argonne National Laboratory
Marc Snir, Argonne National Laboratory

Pieter Swart, Los Alamos National Laboratory
Keita Teranishi, Cray
Heidi Thornquist, Sandia National Laboratories
Ray Tuminaro, Sandia National Laboratories
Homer Walker, Worcester Polytechnic Institute
Stefan Wild, Argonne National Laboratory
Sam Williams, Lawrence Berkeley National Laboratory
Carol Woodward, Lawrence Livermore National Laboratory

Workshop Agenda

	DOE Workshop on Extreme-Scale Solvers: Preparing for Future Architectures ***March 8-9, 2012*** ***American Geophysical Union Washington, DC***		
Day 1			
8:00	Continental Breakfast		
8:20	ASCR Welcome and Introduction	ASCR	Room A
8:30	Charge to Workshop Participants	Organizing Committee	
8:45	Keynote Briefing I	Bob Lucas	
9:15	Keynote Briefing II	Bill Gropp	
9:45	Q&A		
10:00	Break		
10:30	Concurrent Breakout Sessions #1: Architectural Considerations		Rooms A, B, C
12:00	Lunch (on your own)		
13:30	Outbrief #1; Q&A		Room A
14:00	Concurrent Breakout Sessions #2: Algorithmic Research and Development Needs		Rooms A, B, C
15:30	Break		
16:00	Outbrief #2; Q&A		Room A
16:30	Adjourn for the day (dinner on your own)		
Day 2			
8:00	Continental Breakfast		
8:30	Panel Discussion & Q&A	Moderator: Mike Heroux Panelists: TBA	Room A
9:45	Break		
10:00	Concurrent Breakout Sessions #3: Transition Strategy; Gaps		Room A, B, C
11:30	Break		
12:00	Outbrief #3; Q&A		Room A
13:00	Workshop Adjourns		