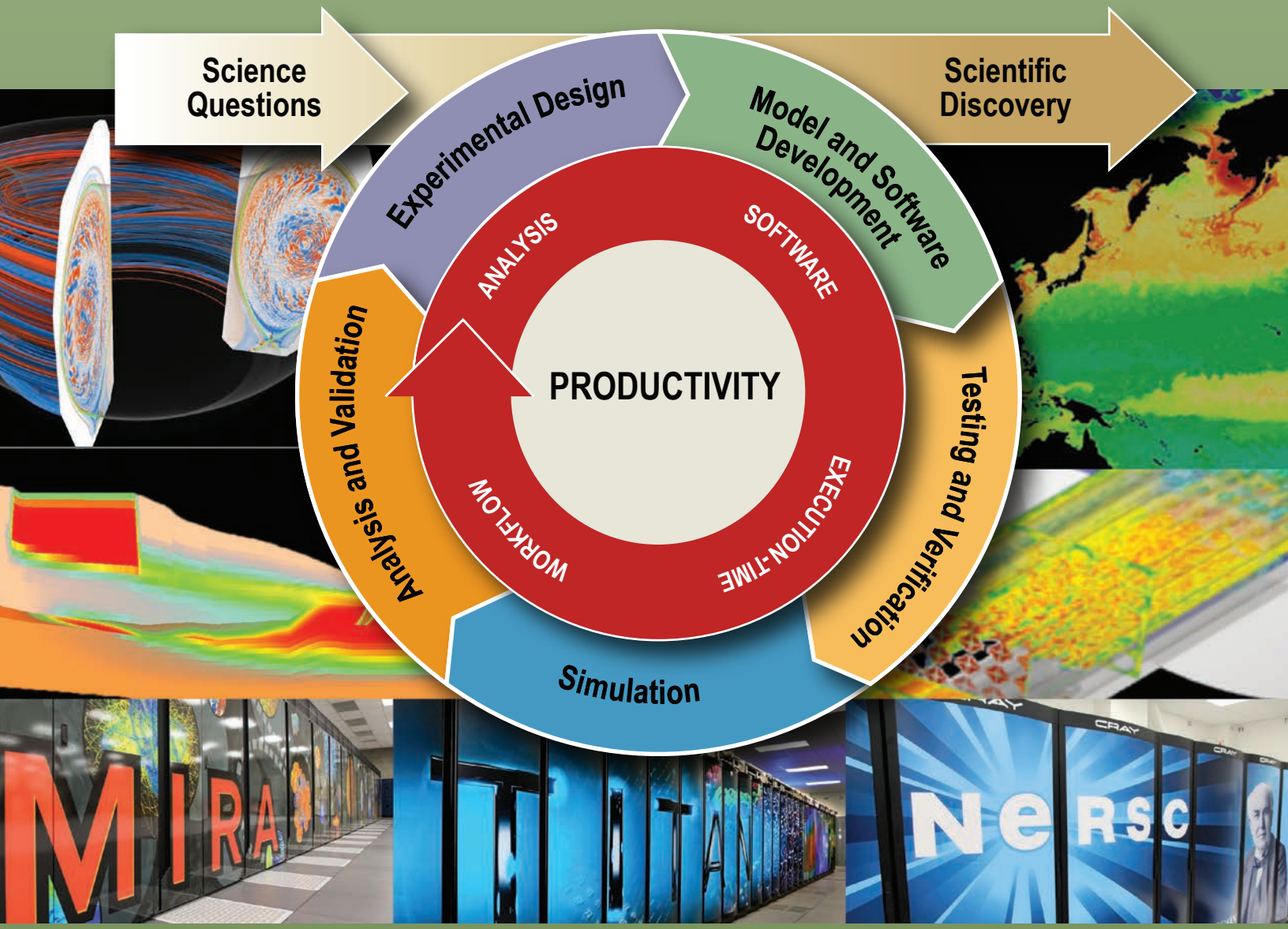
Software Productivity for Extreme-Scale Science

DOE Workshop Report
January 13-14, 2014, Rockville, MD



U.S. Department of Energy
ADVANCED SCIENTIFIC COMPUTING RESEARCH

U.S. DEPARTMENT OF ENERGY
Office of Science

Disclaimer:

Workshop on Software Productivity for Extreme-scale Science

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

January 13-14, 2014
Hilton Hotel, Rockville, MD

Workshop Organizing Committee:
Hans Johansen, Co-Chair (Lawrence Berkeley National Laboratory)
Lois Curfman McInnes, Co-Chair (Argonne National Laboratory)
David E. Bernholdt (Oak Ridge National Laboratory)
Jeffrey Carver (University of Alabama)
Michael Heroux (Sandia National Laboratories)
Richard Hornung (Lawrence Livermore National Laboratory)
Phil Jones (Los Alamos National Laboratory)
Bob Lucas (University of Southern California)
Andrew Siegel (Argonne National Laboratory)

DOE ASCR Point of Contact:
Thomas Ndousse-Fetter

Abstract

This report presents results from the DOE workshop on *Software Productivity for eXtreme-scale Science (SWP4XS)* held January 13-14, 2014, in Rockville, MD. The workshop brought together approximately fifty experts in the development of large-scale scientific applications, numerical libraries, and computer science infrastructure to determine how to address the growing crisis in software productivity caused by disruptive changes in extreme-scale computer architectures and new frontiers in modeling, simulation, and analysis of complex multiscale and multiphysics phenomena. Critical research is needed in five broad areas: (1) characterizing and measuring extreme-scale software productivity impediments and opportunities, (2) addressing the impact of extreme-scale architectures on infrastructure and applications, (3) extending software ecosystems to support extreme-scale science, (4) identifying, developing, and disseminating software productivity and engineering practices for extreme-scale computing, and (5) growing an extreme-scale software productivity community. A focused effort on improving software productivity can lead to new methods and approaches that improve the quality, decrease the cost, and reduce the development and maintenance effort for production scientific software, providing the foundation for more fully exploiting extreme-scale resources for scientific discovery.

Contents

Executive Summary

While emerging extreme-scale computing systems provide unprecedented resources for scientific discovery, DOE's scientific communities now face a **crisis in extreme-scale scientific software productivity** due to a powerful confluence of factors: ongoing disruptive changes in computer architectures and new frontiers in modeling, simulation, and analysis of complex multiscale and multiphysics phenomena. Research in mission-critical science and energy problems has become increasingly dependent on high-performance computing, but productivity in extreme-scale scientific application development has not kept pace. The principal issues are (1) a significant lag between extreme-scale hardware and algorithmic innovations and their effective use in applications, leading to poorly scaling codes; (2) obstacles in the combined use of independently developed software components; (3) lack of agile yet rigorous software engineering practices for high-performance computing; and (4) failure to consider the entire lifecycle of large scientific software efforts, leading to fragile, complicated application codes that are increasingly difficult to enhance.

Software productivity is a key aspect of overall *scientific productivity*, identified as one of the top ten exascale research challenges by a recent ASCAC subcommittee. Scientific productivity can be considered an overall measure of quality of the complete process of achieving mission-driven science results—incorporating software productivity (effort, time, and cost for software development, maintenance, and support), execution-time productivity (efficiency, time, and cost for running scientific workloads), workflow and analysis productivity, and the value of computational output in terms of scientific discovery. These pervasive and urgent problems in software productivity—as manifested by human resources in development time, machine and energy resources in runtime, and even questions about correctness of computational results—span throughout all DOE extreme-scale computational science communities and broadly impact the overall DOE research portfolio and budget. Moreover, these difficulties will grow because of the need for repeated reengineering and revision of performance-critical code during a disruptive period of rapid changes in execution models driven by the end of Dennard scaling and the looming end of Moore's Law.

In order to address these critical and timely concerns, which are far beyond current understanding, we recommend support for a *collection of focused software productivity research efforts* within the DOE Office of Advanced Scientific Computing Research. Specifically, we recommend the following activities, discussed in detail in Section 5, in order to qualitatively improve software productivity for extreme-scale science:

1. Characterize and measure extreme-scale software productivity impediments and opportunities.
2. Develop software designs that minimize the impact of ongoing computer architecture changes.
3. Characterize opportunities for enabling advanced multiphysics, multiscale, and analysis capabilities.
4. Develop composable and interoperable components and libraries.
5. Identify, develop, and disseminate knowledge of productivity tools and best practices.
6. Develop productivity partnerships throughout DOE and other agencies.
7. Establish a Software Productivity Technical Council.

Software productivity improvement is measured by the breadth of applications that can effectively utilize extreme-scale architectures; the rate at which new capabilities can be deployed; and the reduction of development and maintenance costs, including reengineering, over the lifetimes of applications, libraries, and tools. Our goal is to **fundamentally change extreme-scale scientific software infrastructure and culture through an ambitious research agenda that leads to new methods and approaches for portable, composable, and interoperable applications, libraries, and tools as foundations for scientific discovery.**

This report is based on discussions within DOE as well as contributions from over fifty experts who participated in an intensive 1.5-day workshop. The report presents an overarching vision for new research to increase extreme-scale scientific software productivity; we summarize a set of supporting goals and science drivers and identify important challenges that must be addressed to make the vision of more productive software a reality. We also discuss critical needs for crosscutting outreach and programmatic components.

1 Introduction

The peak performance of modern supercomputers continues to improve dramatically, but a growing gap exists in real productivity, as scientific application teams require more time and effort to adapt to the most advanced DOE high-performance computing (HPC) systems, characterized by myriad novel architectural features, including millions of cores, simultaneous multithreading, vectorization, core heterogeneity, unconventional memory hierarchies, and new programming models. Studies consistently point to the need for better scientific application software development processes as a key enabler for scientific advancement on extreme-scale platforms [27, 30, 31, 51–53, 63].

Large, distributed scientific teams face difficulties in fully exploiting sophisticated libraries, system software, runtime tools, and programming models in the face of rapidly changing and complex machine architectures and operating constraints; other challenges include developing, reengineering, and porting application software. Moreover, as science goals transition from qualitative to quantitative simulations that can predict experimental observations and inform high-impact policy and decision support, new frontiers of multiscale and multiphysics simulations not only need to combine multiple applications developed by different teams but also increasingly must incorporate data analytics, design, optimization, and uncertainty quantification on top of traditional forward models [19–27, 52]. No single team has the resources to address all these interdisciplinary functionalities, and new generations of computational scientists face a daunting scope of skills and knowledge as prerequisites for their own research contributions.

The concurrent trends in extreme-scale architectures and applications have created a major scientific software productivity crisis. Many application teams are unable to address the software refactoring needs and the growing feature requirements with the resources available to them. Without an increase in software productivity we will not realize the performance and feature requirements with the staffing levels available to us.

1.1 Software productivity's role in overall scientific productivity

We note that *software productivity* is a key component of overall *scientific productivity*, identified as one of the top ten exascale research challenges in a recent subcommittee report by the Advanced Scientific Computing Advisory Committee [20]. Scientific productivity can be considered an overall measure of quality of the complete process of achieving mission-driven science results. Scientific productivity includes software productivity (effort, time, and cost for software development, maintenance, and support), execution-time productivity (efficiency, time, and cost for running scientific workloads), workflow and analysis productivity (effort, time, and cost for the overall cycle of simulation and analysis), and the value of computational output in terms of grand challenge scientific discovery. Figure 1 illustrates how advances in software productivity contribute to accelerating the cycle of scientific discovery on emerging extreme-scale architectures.

In order to identify priorities in research and development to address these challenges in extreme-scale scientific software productivity, the DOE Advanced Scientific Computing Research (ASCR) program convened a workshop January 13-14, 2014, in Rockville, MD. About fifty leaders in large-scale scientific applications, numerical libraries, and computer science tools (see Appendix C) assessed the needs of computational science software on emerging architectures and considered scientific software lifecycle and infrastructure requirements for large-scale code development efforts, including potential contributions that software engineering can bring to HPC software at scale. This document distills input from workshop breakout sessions, position papers (listed in Appendix D and available at the workshop website, `http://www.orau.gov/swproductivity2014`), and ongoing conversations in the DOE community. Complementary issues in extreme-scale execution-time productivity, such as programming models, runtime systems that speculatively migrate data, workflow control and management, and automatic management of power and reliability, are the focus of another working group.

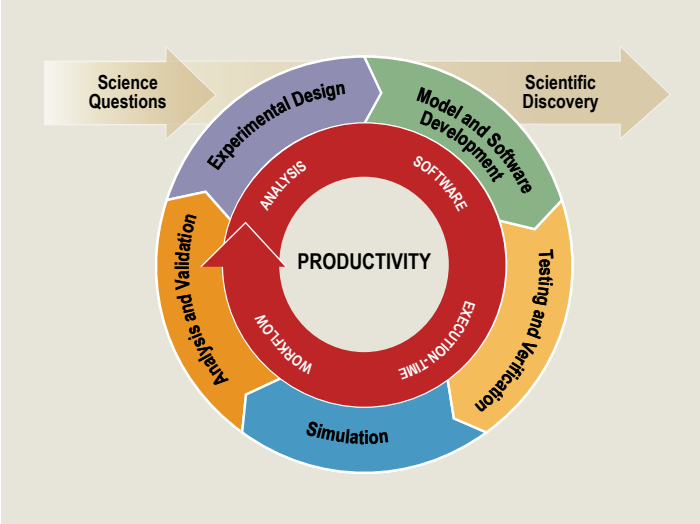


Figure 1: Increasing software productivity is a key aspect of accelerating the cycle of scientific discovery on emerging extreme-scale architectures.

1.2 Vision

Our overarching vision of software productivity for extreme-scale science is as follows:

> **To improve the quality, decrease the cost, and accelerate the delivery of production scientific applications through extensible software ecosystems of trusted, composable, interoperable libraries and tools that provide a foundation for extreme-scale scientific discovery.**

The remainder of this document delves into more details about productivity challenges in extreme-scale application software development (Section 2); identifies extreme-scale software productivity gaps (Section 3); calls out crosscutting needs and dependencies (Section 4); and highlights recommendations in research, development, outreach, and programmatic interactions to achieve our vision (Section 5). This work is essential to ensure that extreme-scale computational science, broadly recognized as an enabler of experimental and theoretical approaches, continues to thrive through vibrant and productive extreme-scale software ecosystems and communities that enable high-impact policy and decision support.

2 Productivity Challenges in Extreme-Scale Application Software Development

To provide context for understanding this software productivity crisis, we summarize extreme-scale computing trends and crosscutting software productivity challenges in the shift toward high-fidelity multiscale and multiphysics simulations. We then highlight difficulties in refactoring existing applications, developing completely new applications, and integrating simulations and data analytics; and we summarize the primary software productivity concerns of workshop participants.

Appendix A discusses these software productivity challenges from the perspective of mission-critical DOE Office of Science applications in climate (Section A.1), environmental management (Section A.2),

fusion energy sciences (Section A.3), and advanced nuclear reactors (Section A.4). These applications, developed by large, diverse, and distributed groups of domain scientists, computational scientists, applied mathematicians, and computer scientists, have been selected as representative examples of a much broader DOE simulation portfolio and confront crosscutting software productivity challenges that arise in NNSA applications as well.

2.1 Extreme-scale computing trends

While exascale platforms are not likely to appear for at least another 6-8 years and while many details of the hardware, runtime system, and programming models are still the subject of active research, we nonetheless have a relatively clear picture of the key motivating ideas and basic characteristics of exascale systems [20, 31, 53]. Application programmers can expect a programming environment that, driven largely by power constraints and budget realities, represents a sharp departure from long-established trends. Specifically, we expect to see (1) memory and memory bandwidth to FLOP/s ratios at least one order of magnitude smaller than current systems; (2) up to billion-way concurrency with multiple levels of concurrency; (3) hybrid systems with reliance on lightweight cores with support for mixed MIMD- and SIMD-type parallelism; (4) more direct control of deeper memory hierarchies, including direct control of on-node NUMA memory; (5) much higher performance penalties for global synchronization; and (6) possibly more direct programmatic control of tradeoffs in performance, power, and resilience. Indeed, some of these features have already made their way into existing leadership-class machines, for example in the appearance of high-throughput many-core coprocessors to boost single-node peak performance at relatively modest power.

The present time period in high-performance computing is arguably the most challenging in history for application developers. We have seen the elimination of sequential performance improvements, the emergence of threading and re-emergence of vectorization, and a large collection of competing parallel programming models, none of which are robust and widely available. Several many-core and accelerator-based node architectures are rapidly evolving and distinctly different in design strategies. For existing applications, the performance-critical parts of source code will need to be refactored to achieve good performance. For new applications, portable designs are just emerging. Furthermore, there is widespread concern that another disruption will occur within 10-15 years as Moore's law reaches its end.

Application productivity during the past two decades. The last major paradigm shift in HPC platforms began more than two decades ago, with the dawn of the "terascale" era, where the task of developers was to evolve existing vector codes to a distributed-memory parallel model. After a period of research in message-passing models, the Message Passing Interface (MPI) was standardized. In most applications, MPI was encapsulated and domain scientists could continue to write numerical algorithms following familiar idioms. Parallel execution models matched natural problem-partitioning strategies such as domain decomposition, and performance advancements resulted from incremental improvements to weak scaling and increases in processor speeds. This trend continued into today's "petascale" era, largely through Moore's law, breakthroughs in scalable algorithms, and slight modifications to mix moderate threading with MPI.

Thus, prior to this new era of disruptive architectural changes, continued improvement in HPC applications often equated to increased scalability, typically within the context of a familiar programming paradigm, and in parameter regimes (e.g., in terms of machine balance, network characteristics) that varied relatively little with each machine generation. This period was naturally compatible with continual application productivity in the presence of faster and faster computers, as efforts to achieve the next level of performance were typically undertaken with highly targeted or evolutionary improvements to production codes. This period also saw a tremendous increase in the complexity of applications, including an increased emphasis on simultaneous coupling of different physical models, sophisticated meshing strategies for complex geometries, and a broad range of runtime options to enable modeling different physical scenarios within a single

code base. A variety of community codes have emerged, where in many cases good software engineering practices have facilitated larger-scale software collaborations, in turn benefitting the overall scientific process [35]. Especially critical is the fact that increases in application productivity during this period were in part enabled by more robust and sophisticated application-level libraries—solvers, meshing, I/O, and so forth—that freed application developers from having to focus too much on the implementation details of common HPC approaches.

2.2 New science frontiers: Multiphysics, multiscale, and beyond

Growth in computing capabilities has historically enabled increases in the size, resolution, and physical fidelity of simulations. "Single physics" simulations are rapidly maturing, however, and in many cases we are approaching the point of diminishing scientific returns from continued refinements of problem size or resolution. Instead, attention is turning to new approaches to further improve the realism and predictive power of simulations by placing them in a larger context. Coupling of simulations across different types of physics (multiphysics) and different time and length scales (multiscale) is becoming increasingly important in improving the physical fidelity of computational models. Parameter studies and design optimization employ ensembles of simulations to extend the science beyond point solutions and into exploration of the problem space. Uncertainty quantification and sensitivity analysis studies essentially help to place "error bars" on simulation results and to understand how error bars on the inputs propagate through simulations. These approaches represent new frontiers from a computing perspective, impacting applications in new ways—both in how codes are designed and written and in how they are used.

Software for these new scientific frontiers presents a host of difficulties beyond those faced in single-physics contexts because of the compounded complexities of code interactions [52]. Perhaps the most fundamental crosscutting observation is that collaboration is unavoidable because the full scope of required functionality is broader than any single person or team can deeply understand. Moreover, because software is the practical means through which collaboration on extreme-scale algorithms and applications occurs, software productivity issues are front and center as we combine diverse code for the various phases of simulation and analysis. Scientific application teams face daunting challenges both in building application-specific infrastructure and in incorporating libraries, frameworks, and tools that are under development by many groups. Practical difficulties in collaborative research software center on the need for composable and interoperable code with support for managing complexity and change as architectures, programming models, and applications continue to advance.

2.3 Scope of software productivity challenges

Software productivity challenges arise across the full spectrum of capabilities needed to support new architectures and new science, including refactoring legacy code, developing new applications, and integrating simulations and data analytics.

2.3.1 Refactoring legacy applications

It is widely recognized in the HPC community that making efficient use of anticipated exascale platforms will in many cases require fundamental changes in the design and implementation of traditional numerical approaches. Substantial changes in machine architectures in recent years have already thrust application teams into grappling with how to most effectively use these platforms. The performance-critical parts of source code will need to be refactored to achieve good performance, primarily to address the striking increase in hardware complexity and concurrency and fundamentally new resiliency concerns that will characterize future extreme-scale systems. Dramatic changes to application design and implementation will

be necessary in order to map functional and memory requirements of applications to hardware functional units and memory systems. New physics and analysis capabilities also must be incorporated over the lifetime of application codes. In many cases, these changes must be made in legacy code bases that are not well understood and for which the software design, documentation, and testing do not meet modern expectations.

Moreover, existing applications and capabilities must be supported in order to meet scientific and mission goals during significant code transformation, further exacerbating the problem of maintaining productive development and use of application software. The problem of maintaining a high level of software development and user productivity in the extreme-scale computing era will be particularly acute for large, integrated applications, often developed over decades; the overall cost for development of these codes often corresponds to an investment of hundreds of full-time equivalent years of labor. Such codes typically combine a large code base, which provides a wide range of modeling and analysis capabilities, with dozens of independently developed libraries.

2.3.2 Developing new science applications

Because extreme-scale computing resources will enable researchers to tackle completely new science questions, out-of-the-box formulations that start from a clean slate may be needed for integrated multiphysics and multiscale modeling that includes design optimization, integrated UQ, and inverse problems. For new science applications, portable designs for emerging heterogeneous extreme-scale architectures are just emerging, and approaches will continue to evolve as programming models and architectures continue to change. Key challenges in the paradigm shift to expose, explore, and exploit multilevel parallelism include achieving performance portability while ensuring resiliency and energy awareness.

2.3.3 Integrating simulations and data analytics

Another emerging paradigm shift for extreme-scale science involves removing the barrier between traditional simulation and postprocessing of simulation data for scientific insights. Reasons for this shift are the intractability of the sheer volume of simulation data, myriad aggregated diagnostics that are too difficult to postprocess and thus are better suited for in situ diagnostics, and the potential for advanced diagnostics and analysis to guide simulations during runtime. Examples include data assimilation (incorporating experimental or other data into simulations), error calculations (using time-step or mesh refinement criteria), and simultaneous calculation of ensembles of simulation data (for sensitivity or uncertainty quantification).

The software productivity gap in this context cuts through aspects of scientific and software productivity and effective use of extreme-scale hardware resources. It also reintroduces familiar software productivity challenges: Applications must be designed to be modular, so that analysis can be done concurrently. Scalable component libraries need to be made available for offloading I/O and analysis of simulation data in situ or in transit. Tools and programming models must support data constructs appropriate for analysis while data is in-memory and streaming to file systems. In particular, we see an opportunity to establish "data brokers" as intermediaries between computational entities, data analysis tools, and databases. The functionality of these brokers exists in various applications but needs to be extracted and extended. In addition, as the software tools used in simulation and analysis are merged, environment configuration and software compatibility are important issues.

2.3.4 Concerns of application software developers

These ongoing disruptive architectural changes and pursuit of new science frontiers raise deep questions about software productivity on the trajectory to exascale computing. Many critical applications have no ability and no current plans to leverage new architectural features. Such efforts are taking a pragmatic, wait-and-see strategy with regard to future code restructuring until a path forward becomes clearer. While the

urgency of addressing these architectural changes is widely recognized, many teams have limited resources and insufficient insights about appropriate strategies to explore. These resulting delays in addressing ongoing architectural changes thus increase the pressure on development teams because the timeline to extreme-scale computations is compressed. Without a strategy and associated research to facilitate the transition of real applications to next-generation architectures, as well as the development of completely new science codes, we are likely to have mission-critical application codes capable of only petascale performance on "exascale" platforms.

Responses to a subset of questions from a pre-workshop survey provided to all workshop registrants offer insight into concerns shared across the HPC application community. Twenty-six workshop participants responded to some or all of the survey questions. Of these respondents, 24 indicated that they expect to evolve their current codes to extreme scale, rather than completely rewriting them from scratch. The respondents' planned extreme-scale parallelization approaches centered on hybrid "MPI+X" strategies, where X included OpenMP, OpenCL, OpenACC, Pthreads, Intel TBB, CUDA, accelerators, and others. One respondent captured sentiments of many participants by saying, "Everything is under consideration; we will do what is necessary." This attitude is reflected in the fact that 16 respondents said they would consider using a radically different programming model for extreme scale. We also asked survey respondents to rank the importance of various concerns in the transition to extreme-scale architectures. The top concerns were achieving and maintaining portable performance, dealing with programming model uncertainty, having robust library and tool support, and debugging at scale.

While some research teams are already incorporating aspects of productivity methodologies that are most relevant to their unique needs and microcultures, the community as a whole is only beginning to grapple with understanding and improving productivity of broader extreme-scale scientific software ecosystems.

3 Extreme-Scale Software Productivity Gaps

Considering the context of these architectural changes and new directions in extreme-scale computational science, workshop organizers distilled input from position papers, breakout sessions, and ongoing conversations in the DOE community to reach a consensus on five broad categories of software productivity gaps that require focused investment.

- **Characterizing and measuring extreme-scale software productivity impediments and opportunities (Section 3.1):** Gather, analyze, and categorize software productivity challenges as first steps to identify, measure, and track metrics for software productivity and code quality throughout project and software lifecycles.

- **Addressing the impact of extreme-scale architectures on infrastructure and applications (Section 3.2):** Transition software design and data structures to take full advantage of new features of emerging extreme-scale architectures in order to achieve portable performance as well as resilience, reproducibility, and use of high-level abstractions.

- **Extending software ecosystems to support extreme-scale science (Section 3.3):** Transform libraries, tools, and applications into robust, interoperable, and easy-to-use software ecosystems, with support for composability and library interoperability, multiphysics and multiscale coupling methodologies, and configuration and build systems, as well as testing, verification, validation, and debugging at scale.

- **Identifying, developing, and disseminating software productivity and engineering practices for extreme-scale computing. (Section 3.4).** Adapt mainstream software productivity and software

engineering methodologies in order to address the unique requirements of extreme-scale scientific computing.

- **Growing an extreme-scale software productivity community (Section 3.5):** Provide productivity training for software developers, and create forums for information exchange about software productivity issues.

Despite the breadth and complexity of these gaps, the DOE community is well positioned to exploit the unique features of mathematical and scientific software for extreme-scale software productivity research and development (R&D). The list, although not necessarily complete, represents the opportunities identified by a large, representative group from the extreme-scale computational science community. We expect topics to evolve as the community's experience deepens over time. Section 4 discusses crosscutting needs and dependencies. Section 5 summarizes our recommendations for addressing these gaps.

3.1 Characterizing and measuring extreme-scale software productivity impediments and opportunities

If we are to make significant improvements in software productivity, we must understand in greater depth than is possible in a 1.5-day workshop where we are today, how we got here, and what the impediments are to software productivity. The community has considerable experience with software development practices and tools that can both inform directions for new research and development and help lay the groundwork for the evaluation of their impact on software productivity.

3.1.1 Inventory of HPC software productivity issues and experiences

While DOE computational communities have been developing HPC software for many years, we do not have a strong tradition of self-evaluation. This situation is especially evident in areas such as software productivity, which tend to be somewhat removed from the core scientific issues driving most computational scientists.

As an initial phase of productivity research, we need to mine the community's experience with software development in order to gain a deeper understanding of the impediments to software productivity at extreme scale, and to identify and understand tools and approaches that have proven effective (or not) in addressing those issues. Every project uses *some* software development methodology, whether or not it is explicitly stated and consciously applied. Likewise, every development team encounter issues that limit their progress. Some development teams make conscious efforts to address issues, while others may have a hard time putting their finger on "the problem," much less addressing it.

Appendix B outlines in an informal fashion some of the experiences and "lessons learned" in the HPC library community. Bringing the perspective of more rigorous software engineering research to the study of both individual and communities of scientific software projects provides a way to rapidly "bootstrap" our understanding of the state of the art in software productivity and to lay the groundwork for future pilot projects and case studies, as discussed in Section 3.4.

3.1.2 Understanding how HPC scientific software is different

Developing a deeper understanding of what distinguishes HPC scientific software from other types of software with respect to development practices is a valuable related area of study. Empirically, attempts to "import" software engineering practices that are well accepted outside of scientific computing often prove unsuccessful or require significant adaptation. This circumstance is indicative of differences from the environments in which the practices were developed; however, those differences have not been well studied.

A better appreciation of the underlying differences would facilitate the tailoring of software engineering practices for use by HPC scientific software teams, as discussed in Section 3.4.

3.1.3 Metrics for software and productivity

The field of software engineering has developed many metrics to characterize and quantify different aspects of software, software quality, and software development processes. However, consensus on a concise, explicit definition of productivity has been difficult to achieve. The DARPA HPCS program (where the "P" stands for productivity) focused considerable effort on characterizing productivity [30, 65, 66]. Rather than focusing on a single overarching definition of software productivity, we propose a two-pronged approach, which relies on qualitative assessments of productivity together with widely accepted related metrics from the software engineering community.

A common phrase in product development is "better, faster, cheaper: choose any two." Intuitively, we recognize that improvements in any of the areas (better, faster, cheaper) is an improvement in the overall product development process. But at the same time, improvements in one area can have negative impacts on another. For example, as discussed in the book *Code Complete* [59], improvement in the metric of code efficiency has a negative influence on code quality metrics of correctness, reliability, integrity, adaptability, and accuracy. Thus, we will need to consider overall software productivity issues for the full range of human resources in code development time, machine and energy resources in code runtime, and even questions about scientific correctness of computational results. Moreover, we must consider the *tensions* and *balance* among these issues, with a goal of achieving overall productivity advances in the volume of "better, faster, and cheaper."

We can talk about improving software productivity if we identify changes in the way we produce software that are positive in a specific metric and do not introduce *unacceptable* degradation in other metrics. A useful analogy is corrective lenses. Although it is hard to assert that a given lens provides the *best* vision, it is easy to determine which of two candidate lenses provides *better* vision. Characterizing and managing extreme-scale software productivity can be approached in a similar way.

As such, an important part of the process of characterization of experience in HPC software productivity should be collecting and analyzing software metrics in order to gain a better understanding of their relationship to qualitative perceptions of productivity. Characterization of software productivity impediments will naturally lead to determining metrics that can be correlated with software productivity in order to determine the severity of problems, direct R&D resources where most needed, and gauge the extent to which we are making progress in the right directions. Research is needed to identify, measure, and track metrics for software productivity and code quality throughout project and software lifecycles, with the ultimate goal of improving key metrics of scientific productivity, such as how fully and efficiently applications utilize extreme-scale resources and how rapid and efficient software development is for validated scientific applications. This work requires coordination across application and library development teams as well as DOE leadership computing facilities.

3.2 Addressing the impact of extreme-scale architectures on infrastructure and applications

As discussed in Section 2, exascale platforms will require fundamental changes in the design and implementation of traditional numerical approaches. Many software productivity challenges span both scientific applications and reusable libraries that form intermediate layers of the software stack, and in this section, we use the term *application* to include libraries and other large bodies of code that need to move to extreme-scale systems.

The issues raised here are crosscutting—intertwined with already recognized R&D needs for extreme scale. Regardless of how these issues are addressed programmatically, the workshop participants were clear

on their significance to scientific productivity. We highlight these issues from the perspective of their impact on the software development process. Identified research needs include the following:

- Supporting diversity and rapid change in applications to keep pace with the changing extreme-scale hardware environment (Section 3.2.1)
- Understanding application performance (Section 3.2.2)
- Achieving performance portability across diverse and changing extreme-scale architectures (Section 3.2.3)
- Incorporating new approaches to resilience and new concepts of reproducibility into applications (Section 3.2.4)
- Introducing higher levels of abstraction into applications (Section 3.2.5)

3.2.1 Preparing for extreme-scale architectures—and continual change

The field of HPC architectures is in a state of flux, the like of which has not been seen in more than two decades. Not only will applications face a very different hardware environment from today's high-end systems, as noted in Section 2, but there may be multiple such architectures, not amenable to a single abstract machine model, for applications to target. Furthermore, it is likely to be a decade or more before the situation settles down—if it does at all. Lower-level elements of the software stack (programming models, compilers, runtime systems, operating systems, etc.) will certainly help application developers address these changes. But like the hardware, the software stack is in a state of significant flux, in large part responding to the trends in hardware architecture.

Consequently, applications running on future extreme-scale architectures, whether they are created anew or based on existing code, will face complex and changing environments. To take advantage of extreme-scale platforms, developers will have to significantly change the performance-sensitive portions of their existing code bases, or their thinking and algorithms as they create new applications. Research is needed to help application developers deal with the need for "continual" change in their applications in response to diverse, changing hardware environments, including the development of design strategies and other approaches to insulate application code from architectural details (while delivering high performance), as well as refactoring, transformation, and other tools to facilitate the rapid and efficient adaptation of code to different architectures.

3.2.2 Understanding application performance

Understanding the performance implications of existing code and determining how to obtain high performance from new hardware architectures are clearly fundamental to any effort in extreme-scale computing. The coverage, quality, and usability of this kind of information can allow developers to narrow their design choices with less of the tedious and time-consuming "brute force" performance exploration that requires producing and evaluating multiple variations of a given algorithm or code. Research is needed to facilitate the identification, extraction, and understanding of key kernels that will determine performance on future systems (which may not be the same ones that determine performance on today's systems). Also useful is an understanding of the performance of an application or phase of computation in relation to what *should be* achievable on a given platform. Research to address these issues might include the development of a minimal set of abstract machine models for emerging hardware architectures and techniques that simplify the development of performance models for applications and help automate the prediction of performance and scalability, as well as the testing and tracking of actual performance. More generally, approaches would be helpful that can provide guidance to application designers and developers about choices of algorithms and data structures, possibly including exemplars of important kernels, for different architectures or different degrees of performance portability.

3.2.3 Enabling performance portability

Performance portability offers significant benefits in terms of both development and execution-time productivity: the developer can confidently expect that a given version of an algorithm or application will deliver predictable performance across the applicable range of hardware architectures. Achieving performance portability essentially requires that the application be expressed in a form that can be effectively mapped onto the different target architectures. As the set of hardware architectures of interest becomes larger and more diverse, performance portability becomes increasingly challenging. Research is needed to identify and develop new abstractions that express the parallelism and other aspects of the computation in a way that can be mapped effectively to the breadth of emerging architectures. Researchers should be encouraged to consider approaches based on domain-specific languages as well as approaches that will work both with existing HPC languages (i.e., language extensions, directives, and libraries) and with new and emerging general-purpose programming languages that might provide a better fit for future extreme-scale systems. The ability to effectively map these modes of expression onto diverse hardware environments is the second part of achieving performance portability. Research on resource management techniques for new and emerging architectural features, and the management of resources in increasingly multilevel environments (multiple levels of parallelism, deeper memory hierarchies, etc.), especially in applications based on the composition of software modules, would be beneficial, as would approaches for dynamic adaptation or autotuning to improve the use of system resources.

3.2.4 Resilience and reproducibility

In addition to changes in the hardware architecture, resiliency is a growing concern for future systems. The overwhelming majority of current applications are written assuming that a hardware failure will occur once every few days, or even weeks. As a result, the "gold standard" of fault recovery for most scientific applications in use today is global checkpoint restart. On future extreme-scale systems, the number of system components will be several orders of magnitude larger than today's machines, and individual components may be less reliable than in the past. In an environment in which the frequency of failure may be reduced to hours, applications are likely to encounter new failures while recovering from prior failures. This situation could easily lead to paralysis of the application (inability to make forward progress) or unrecoverable failure of the application. The resilience research community is beginning to address these issues, but many of the new approaches that have been proposed are much more intrusive on the application itself than in the past, making it also a significant software productivity concern. Research is needed to complement the basic research on new methods of resilience, addressing how these new approaches can most effectively and efficiently be incorporated into existing and new applications. This might include, for example, new design strategies, resilience-oriented refactoring patterns, and transformational tools. Similarly, research will be needed to determine the relationship between new extreme-scale numerical algorithms and the algorithmic and hardware vulnerabilities, in order to understand the best ways to protect applications, as well as the best ways to factor resilience between applications themselves and libraries.

Future resilience concerns also amplify the challenges associated with reproducibility at extreme scales. Bitwise reproducibility of computational results has long been considered the gold standard. Applications that cannot reproduce their selected test cases at this level may not be trusted by users and may lead to question as to the correctness of the code. Reproducibility has long been a challenge because of the details of fixed-precision floating-point representations, which are exposed by many compiler and runtime optimizations, as well as any nondeterminism in parallel algorithms. In the extreme-scale era, these challenges will be exacerbated by the push to increasing levels of asynchrony and nondeterminism, as well as increasing use of dynamic and adaptive algorithms and runtimes, plus the increased likelihood of transient errors and silent data corruption affecting results. Further, multiscale and multiphysics applications, which are an in-

creasingly prominent part of extreme-scale computational science, are often highly sensitive to algorithmic and modeling decisions that are made dynamically based on the state of program execution.

Historically, the solution to these challenges has been primarily to serialize operations so that they can be reproduced. This approach has obvious impacts on performance and becomes increasingly untenable as the level of concurrency rises. Moreover, these techniques cannot address the variability arising from transient errors. Also, hardware vendors estimate that adding additional correctness checking capabilities to processors may increase energy consumption by as much as 20%, which will likely limit their availability and use. Research is needed to develop new definitions of, and requirements for, reproducibility in applications, as well as new approaches to testing and debugging that reduce the dependence on bitwise reproducibility. New tools and analysis methodologies will be needed to determine whether results are unacceptably different, because of a hardware or software bug, or within acceptable error bounds or margins of uncertainty for physical models to be suitably predictive.

We note that the scientific community at large is beginning to grapple with the challenges of the reproducibility of research at a higher level—essentially, whether it is possible for outsiders to reproduce the research described in a scientific paper. While this is an important and related question, this workshop and these recommendations are focused on a narrower issue—the reproducibility of a given simulation when run multiple times on the same or different hardware.

3.2.5 Higher-level abstractions

An important aspect of improving scientific productivity is raising the level of abstractions used by developers throughout the software stack. As discussed in Appendix B, many libraries and applications employ abstraction layers, where unified high-level interfaces provide easy access to multiple underlying implementations of sophisticated algorithms and data structures, thereby enabling adaptivity to the needs of particular problem instances and architectures. However, even when using such design strategies, instead of focusing on what should be the main task of introducing new functionality, developers of applications and libraries spend far too much time on the labor-intensive and ad hoc process of modifying and extending existing code in order to port to new platforms, support new data types, and improve performance. This is a long-standing challenge, and not specific to extreme-scale architectures, but the problem of dealing with ongoing disruptive architectural change brings this issue to the fore in the extreme-scale context.

In order to maximize developer productivity, calculations should be expressed in the most abstract form that is explicit enough to readily map to a computational platform. Research is needed on approaches to facilitate the design and implementation of higher-level abstractions in applications. This includes compiler technologies (code analysis, code transformations, code generation, data structure manipulation, and domain-specific languages) that can be combined with deep knowledge of algorithms and their mathematical properties to improve performance and transform code semantics for broader functionality (for example, to transform a forward PDE simulation also to incorporate optimization or uncertainty quantification).

3.3 Extending software ecosystems to support extreme-scale science

Scientific applications exist as part of a larger ecosystem of HPC software, including libraries, frameworks, compilers and runtimes, and supporting tools. While the new and different architectures of extreme-scale systems pose many challenges to software productivity (Section 3.2), which tend to be focused within individual applications or libraries, additional challenges arise across the broader HPC software ecosystem, as the capabilities of extreme-scale computers drive scientific software to new levels of complexity and sophistication. Research will be required to address key aspects of the software ecosystem at extreme scale, such as:

- Increasing use of multiphysics and multiscale coupling methodologies, as well as other ways of using unprecedented computational capabilities (Section 3.3.1)
- Growing reliance on HPC libraries for both performance and code complexity reasons, which must interoperate at every level (Section 3.3.2)
- Ability to configure and build these libraries, together with other software modules, on systems ranging from laptops to one-of-a-kind extreme-scale computers (Section 3.3.3)
- Testing, verifying, validating, and debugging large compositions of software on systems with massive, multilevel parallelism (Section 3.3.4)

3.3.1 New computational frontiers at extreme scale

The rapid growth in computational capabilities available on high-end systems is giving rise to new ways of thinking about simulations. Parameter studies and design optimizations allow the exploration of a larger problem space rather than point solutions. Uncertainty quantification (UQ) and sensitivity analysis help characterize the "error bars" around simulation results, and multiphysics and multiscale coupling provide much richer models of the physical systems under study. These new approaches not only are computationally intensive but can require new ways of thinking about the software involved.

Uncertainty quantification methods can be broadly divided into two categories, intrusive and nonintrusive. Intrusive formulations build the UQ analysis into the code itself, often requiring extensive changes to the original code base. Efficient intrusive UQ methodologies and their effective implementation are an important area of research in the UQ community today. Nonintrusive methods, also known as sampling-based methods, execute many realizations of the simulation, varying the inputs and/or parameters. Similarly, parameter studies and optimization typically involve executing many realizations of a simulation with different parameters. While it is conceptually simple and natural to implement these approaches by utilizing a "driver" external to the simulation code, it is an open research question as to whether this is the most effective formulation in general. The ability to evaluate an entire ensemble of simulations within a single invocation of the code may, in some cases, allow for better reuse of intermediate computations and may expose more, and more useful, parallelism than running the same number of completely separate instances.

But of these "new frontiers," it is multiphysics and multiscale coupling that often has the most significant impact from a software development perspective. Simulation codes involving multiphysics and multiscale coupling are inherently more complex than those that work with only one model or scale. The complexity extends throughout the software stack, from ODE integrators (where different timesteps and implicit-explicit schemes are typically needed to handle different regions, physics, and scales), to the nonlinear solver level (where nonlinear elimination may be needed to handle differing nonlinearities), to the linear solver level (where physics-based preconditioners, Schur complement preconditioners, or block preconditioners are needed to handle various problem components). Moreover, different meshes and discretizations often must be managed and coupled. The research needed in this important area is a combination of analysis, algorithms, and software [52]; software productivity issues are front and center, since all approaches require incorporating software developed by multiple groups.

Completely "problem-neutral" computer science approaches to handle software coupling have proved inadequate to the task, while more problem-specific approaches under investigation in applications and libraries indicate a possible way forward to manage the complexity and deliver efficient algorithms. Multiple groups have begun experimenting with approaches for describing a complex simulation as a composition of several pieces and decomposing a complex simulation into a series of simpler pieces. A common component in these modular ecosystems is a "data broker" that eliminates the many-to-many relation requirements inherent in a flat component system by exposing and abstracting the data transfer and transformation requirements. The data broker can greatly simplify interface development and reusability and also provide clean interfaces to databases and post-processing analysis tools.

3.3.2 Software composability and library interoperability

In order that application scientists can readily leverage the full palette of algorithms and data structures provided by HPC libraries, research is needed to ensure that multiple libraries of interest can be used together within the same application. Applications that depend on one or more of these libraries can face a difficult transition to incorporate use of additional libraries, and coupling of two applications that depend on different libraries is particularly challenging.

Composability concerns occur at multiple levels, all of which must be thoughtfully addressed in order to ensure success. In addition to the prerequisite of compatible build and configuration processes, discussed in Section 3.3.3, work is needed to ensure that applications can easily use multiple libraries side by side without conflicts between programming models and resources. Even more difficult challenges arise in software compatibility with the need to exchange or control data between different libraries. At higher levels, many of the issues have to do with the ownership and structure of the data on which the libraries act. If two libraries are expected to process the same (or closely related) application data, then they must have compatible approaches to structuring the data. And, for example, multiple libraries cannot control the creation and destruction of a given data structure. Even for libraries that do not interact directly, concerns arise regarding whether the programming models on which they rely are compatible. Can the two programming models coexist in the same application? If so, can the libraries share resources effectively enough to allow both libraries to operate and obtain the expected levels of performance? While the community already has some experience (both positive and negative) on which to build, the growing diversity of architectures and programming approaches will make it that much more challenging going forward to ensure composability while maintaining performance and portability. Research is needed to identify and catalog issues impacting composability, to identify and refine design patterns that provide composability, and to enhance the composability of key library and programming model software.

3.3.3 Configuration and build systems

The complete process of software installation—that is, downloading a package, preparing it for compilation, testing, and then installing it for use—is difficult in perhaps unappreciated ways. Because any failure of the build system can disable the entire package being installed, the build system must be designed to create actionable diagnostics—a huge task given its internal complexity and the diversity of architectures, operating systems, compilers, and libraries it must handle. Moreover, error conditions, arising from unforeseen interactions between components, far outnumber success states. These problems are multiplied when a given application relies on multiple libraries, which must not only interoperate appropriately, as described in Section 3.3.2, but must also be built in compatible ways and linked together. Supercomputer centers face the complementary challenge of supporting many applications, which may use the same libraries in different ways and may need them to be built differently for compatibility.

Although it seems mundane and might easily be assumed to have been solved long ago, configuration and building have a significant impact on software productivity. In a 2002 study, Kumfert and Epperly found the average perceived overhead due to configuration and build issues was 12%, with some teams reporting 20–30% [54]. In other words, developers and users spend approximately *one eighth* to *one quarter* of their time simply to create a working executable from code that is already written, and (as developers) try to ensure that others can do likewise. A more recent analysis of two large scientific software projects by Hochstein and Jiao found that 5–6% of the lines of code in these two applications were build-related, and 37–65% of version control commits touched at least one build-related file [46]. Further, build failures accounted for 11–47% of all failures in the automated testing regimes of the two packages.

While existing tools provide the basic capabilities needed to configure and build individual packages, research and development are needed to develop strategies and robust, production-quality tools that can

support configuration and build of complex composites of multiple, independently developed code modules and libraries on everything from a laptop or workstation (for development purposes) to one-of-a-kind high-end systems. Such tools also need to provide strong support for software development, as well as testing, verification, and validation of workflows, where it is often necessary to rebuild (small portions of) the code repeatedly with different configurations.

3.3.4 Testing, verification, validation, and debugging

Testing and V&V (verification and validation) are processes that help ensure the correctness of scientific software, while debugging is the process of isolating and correcting errors. These processes, which are critical to delivering credible scientific results, are time-consuming and challenging at any scale. However, extreme-scale trends, such as the massive increase in concurrency and widespread use of multilevel parallelism as well as the rise of multiscale and multiphysics coupling, will multiply these challenges.

Research and development are needed in many areas in order to improve the efficiency and effectiveness of testing, V&V, and debugging in extreme-scale situations. Capturing testing and V&V workflows and automating them is challenging, especially considering the prominent role of human judgment in many V&V processes. This effort will be further exacerbated by the need to rethink reproducibility at extreme scale (see Section 3.2.4). At a higher level, we need new approaches and tools to simplify and automate the creation of tests, for example, based on specifications, which are also sensitive to considerations of both code coverage (making sure that the tests exercise as many paths through the code as possible) and resource utilization (minimizing the number and costs while maximizing coverage). Research on effective integration testing, particularly that related to multiscale and multiphysics code couplings, will be required to complement the development of the coupling strategies themselves (Section 3.3.1).

3.4 Identifying, developing, and disseminating software productivity and engineering practices for extreme-scale computing

Software engineering (SE), which can be defined as "the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software," [47] is central to any effort to increase software productivity. Software engineering is, in essence, the understanding, capturing, and systematizing of practices that promote productivity and facilitate their transfer to and use by others.

With software becoming pervasive throughout our daily lives, both the academic study and the industrial practice of SE have grown tremendously during the past decade. A sizable and vigorous SE community now exists, where researchers and practitioners make available and discuss their software engineering experiences through books, papers, presentations, and other forums.

Contrary to the beliefs of some in the community, however, appropriate software engineering practices are woefully underused in large-scale scientific projects. Many of these projects do not follow traditional SE best practices, at least partially because many existing SE practices are not properly tailored for scientific environments [14]. Discussions in the *Software Engineering for Computational Science & Engineering Workshop* series [11–14], which has sought to begin to bridge between the computational science and larger software engineering communities, have highlighted a number of factors contributing to the current situation. We note that these are fundamental issues in the computational science community, and largely independent of the scale of the computer systems on which the work is done. However, while it may be possible to "get away with" some poor software engineering practices at smaller scales, addressing these issues is imperative for the broad range of code sharing and collaboration needed for next-generation extreme-scale science.

In this section we outline the following combination of R&D and outreach activities that are needed to strengthen the understanding and use of software engineering best practices in the DOE computational

science community.

- "Scalable" software engineering practices for computational science (Section 3.4.1)
- Bringing computational scientists and software engineers together to bridge the communication gap between the two communities (Section 3.4.2)
- Appropriately tailoring SE practices from the broader software industry for use in computational science (Section 3.4.3)
- Pilot projects and case studies to assist in this tailoring (Section 3.4.4)
- Documented examples of successful use of appropriate SE practices in computational science (Section 3.4.5)

These activities can also leverage broader community outreach and development efforts described in Section 3.5.

3.4.1 "Scalable" software engineering practices for computational science

One reason that scientific developers do not consistently use appropriate SE practices is the lack of appreciation of the need for and potential benefits of using those practices. Part of the problem is the slippery slope from an individual research code to its becoming central to the work of a group of researchers in a production setting.

Many scientific codes begin as small, one- or two-person efforts. As both the primary author and the primary user of the code, the developer is confident of being able to fix any problems that may arise, without using any formal software engineering practices. Moreover, the code is often considered simple and even exploratory; and developers may defer the perceived overhead of introducing SE practices until some undefined later time, when they are confident that this avenue of exploration is worth pursuing further.

Often, however, others find the code useful and want to use or modify it. Because the code was originally developed without such uses in mind, it is often difficult for someone new to begin using and modifying the code. Similarly, as a code project grows in scale, complexity, and longevity, the developer may at some point realize the need to introduce SE practices. Unfortunately, at that point, it is often too late in terms of the effort and expense. On the other hand, if the developers had used some lightweight software engineering practices from the start, the situation would not be as problematic. Science teams need to consider the longer-term benefits that could be obtained from some shorter-term costs [15], and their management and sponsors need to support these shorter-term costs.

Research is needed to develop and disseminate guidance on "scalable" software engineering best practices that recognize the challenge of the slippery slope of scientific research software development.

3.4.2 Bridging the chasm between software engineers and computational scientists

Exacerbating the lack of appreciation of the potential benefits of SE practices for some in the computational science community is a distrust born out of prior experience. In the past, well-intentioned but misguided software engineers suggested the use of software development practices that were inappropriate or too heavy for the project. These misaligned suggestions result from a lack of domain understanding on the software engineer's part. Computational scientists and software engineers often do not speak the same language. Clearly needed is a closer interaction among software engineers and computational developers to remove this communication barrier that is preventing the use of appropriate software development practices:

- Software engineers need to be willing to learn from the successes of scientific developers.
- Software engineers need to describe software engineering concepts and practices in terminology that is familiar to computational scientists.
- Scientists need to recognize a real software development need or problem before they will be motivated to consider the use of software engineering practices.

Such interactions, which should be strongly encouraged as an explicit goal of these activities, may take many forms, including broader venues such as training and workshops (Section 3.5), as well as more individualized venues such as pilot projects and case studies (Section 3.4.4).

3.4.3 Development of appropriately tailored software engineering practices

Large-scale scientific software emphasizes various types of software quality goals differently from other types of software. For example, traditional software quality goals such as reusability or maintainability are often undervalued in the development of domain applications because of the push for new scientific results at the expense of well-crafted code. As mentioned many times in the pre-workshop survey, another area in which large-scale scientific software faces important challenges is testing and V&V [16]. In some cases, scientists trust the results of their software without the level of testing that would be expected in other types of software. This situation results, at least partially, from the fact that it is not always clear how to test all parts of a complex and potentially nondeterministic code (see Section 3.3.4). These problems all suggest the need for advances in software engineering practices that are appropriately tailored for large-scale scientific codes. Concepts such as design patterns, object-oriented paradigms, and software inspections are good candidates to become part of this appropriately tailored set of practices. More research is required, however, in order to understand how these practices can be tailored to fit within the context and constraints of large-scale scientific software projects.

3.4.4 Pilot projects and case studies in software engineering for computational science

An important aspect of the development of SE practices tailored for computational science is the opportunity to put software engineers together with practicing computational scientists for extended periods to allow the study, from a software engineering research perspective, and refinement of SE practices in "real world" computational science projects. These studies might take the form of "pilot projects" or longer-term "case studies."

Pilot projects. In cases where a scientific team has a specific software development problem for which the team needs help, software engineers should be paired with the team members to deploy and evaluate specific practices designed to address that problem.

Case studies. In order to provide even more benefit to specific computational science teams and the extreme-scale scientific community as a whole, longer-term case studies are needed. In these studies, software engineering researchers would work closely with a computational science team over a period of time. Together, the team and the software engineering researchers would identify both strengths and weaknesses in the software development process. For any weakness, the software engineering researcher and the team could work together to identify and tailor software engineering practices to address that weakness. Once the new practice has been deployed, the software engineering researcher could work with the team to evaluate the effectiveness of the practice and further tailor it as necessary.

In addition to enhancing the software productivity of the target projects, both pilot projects and case studies should result in publishable software engineering research, which can also be distilled into best practices for the broader community. Such work would be a significant source of of documented examples of successful SE practices (Section 3.4.5), disseminated to the community through various forums (Section 3.5.2) to help promote awareness and adoption of software engineering best practices in the DOE computational science community.

3.4.5 Documented examples of the successful use of appropriate software engineering practices

One of the primary sociological barriers to the adoption of appropriate software engineering practices is the lack of examples. The constraints placed on scientific software and the culture within which it is developed are often quite different from those of more traditional business or IT software. These differences often lead scientists to conclude that the software engineering practices that work in the business or IT world will not work for them because "we do not look like them." In some cases this conclusion may be accurate, while in other cases it may be a way to avoid a practice they do not want to use. While many scientific software teams are effectively using appropriate software engineering practices, many of these experiences are not adequately documented. Effort is needed to capture and document these experiences in a format or venue that allows other scientists to quickly and easily access and reuse the lessons learned. This work is closely related to and could be part of efforts to inventory HPC software productivity issues and experiences (Section 3.1.1), as well as SE pilot projects and case studies (Section 3.4.4), and made available through the forums discussed in Section 3.5.2.

3.5 Growing an extreme-scale software productivity community

One of the most significant challenges to increasing software productivity among DOE computational scientists is that it is not part of the community culture. If the aforementioned R&D activities are to have an impact, we need to develop a community of software productivity *practitioners* as well as researchers by supporting complementary activities, such as training for software developers (Section 3.5.1) and creating forums to facilitate communication about software productivity (Section 3.5.2).

3.5.1 Training in software productivity

Few computational scientists receive any formal education or training in software engineering or productivity-enhancing tools and techniques. Training activities will be important for disseminating productivity practices to the HPC computational science community. In addition, because library developers have considerable experience in developing modular, reusable software components and have already made progress in developing strategies that enable portability across emerging architectures (see Appendix B), library developers need to educate application developers on the best use of library components.

Many formats and venues are possible for these training activities. Many conferences offer opportunities to present tutorials of a half-day's to a full-day's duration. ASCR computing facilities reach out to their users regularly with both in-person and distance training activities, which may range from a hour to one or more days, depending on the format and content. Longer-format courses or "summer schools" may be useful, particularly to help train early-career computational scientists (graduate students and postdocs).

From the perspective of workforce development, it may also be useful to encourage the incorporation of software engineering and productivity into educational programs that train upcoming computational scientists, though this clearly has a much longer time horizon.

3.5.2 Forums for improved information exchange

The scientific community in general encourages interactions and exchanges of information about R&D results. However, work that could benefit others in a productivity sense is often treated as a means to an end in the pursuit of new domain-science results as opposed to something interesting and useful to convey in its own right. Hence, either software developers or reviewers may discount the value of such contributions to the scientific literature. We must build a stronger community of software productivity practitioners in which the exchange of productivity-related experiences is encouraged. This can, and probably should, be done in a variety of ways. Conferences, workshops, and journals are well-recognized vehicles for scholarly

interactions, and their use for productivity purposes should be encouraged, including, as necessary, the establishment of new venues to emphasize the importance and increase the acceptance of productivity-related work. However, the timelines for these venues are relatively long; and productivity information, especially as it relates to new platforms and techniques, is more valuable the more quickly it can be disseminated to the community. Therefore, we also recommend the active development of new venues that can facilitate more rapid and timely interactions. Possible examples are webinars, wikis, mailing lists, and community forums such as the StackExchange model.

4 Crosscutting Needs and Dependencies

Because of its crosscutting nature, a major initiative in software productivity will be challenging to structure and manage. We offer these recommendations to help ensure that these investments have maximum impact:

- Coordination with all major ASCR programs, as well as with other computational science programs in both science and applied technology areas of DOE (Section 4.1)
- Active promotion of productivity-enhancing practices (Section 4.2)
- Establishment of a Technical Council to help coordinate the investment and engage with the broader community (Section 4.3)

4.1 Coordination with other R&D programs

A number of established programs within ASCR include research that is relevant to extreme-scale software productivity, including programs in co-design, applied mathematics, computer science, computational science, and SciDAC, as well as exascale architecture research and facilities planning. In many cases, however, the work may not directly address productivity perspectives or may not be mature enough for widespread use as a productivity tool. Coordination will be required in order to determine the most effective ways to extend this synergistic work into productivity contexts, as well as to coordinate with the broader HPC computational science and software productivity research communities.

As the primary developers of scientific application software in DOE, the other offices within the Office of Science, as well as DOE's applied technology offices, should be key partners in the effort to enhance software productivity. The ASCR computing facilities are positioned at the crux of the software productivity challenge and must also be important partners. In addition to providing large-scale, cutting-edge computing resources to DOE computational scientists, they also bring broad and deep staff expertise in both the hardware and applications, as well as training and other outreach activities to help users get the most out of the facilities. Moreover, partnership with NNSA is important, given that NNSA researchers have essentially the same concerns about software productivity as do ASCR and the Office of Science.

Looking beyond DOE, a number of other federal agencies that make extensive use of high-performance scientific, engineering, and technical computing, such as the Department of Defense and the National Oceanographic and Atmospheric Administration, also have similar concerns about software productivity. We therefore recommend engaging with relevant agencies to share experiences and encourage complementary investments.

4.2 Promotion of productivity-enhancing practices & tools

R&D targeting extreme-scale software productivity is not a guarantee of increased productivity among DOE computational scientists. The results of the productivity research, both in DOE and in the broader software community, along with other tools and practices identified as being productivity enhancing, must be adopted by the developers of computational science applications. However, productivity-enhancing activities often

are *investments* that pay off only over a period of time; consequently, some researchers and their sponsors may not recognize their value, and may not give them a high priority in view of the tension for achieving new science results and publications. Therefore, it is important to consider ways to change the equation for computational scientists, to encourage them to make longer-term investments. We must explicitly acknowledge that current methods of generating quality code will simply not work without significant and stable funding. We need models for sustainable software development (see, e.g., [50]) and software collaboration in research organizations.

4.3 Establishment of a Software Productivity Technical Council

DOE's needs in the area of software productivity are distinctive in that they touch upon a number of established areas of R&D within ASCR. There are also strong connections to applications, in both science and applied technology areas of DOE, as well as NNSA. Further, the leadership computing facilities and NERSC have a prominent role to play. If ASCR is to be successful in raising the levels of software productivity across DOE science and applied technology areas, it will be critical to understand, track, and engage with the researchers and managers of other research programs on an ongoing basis in order to ensure the maximum effect of the software productivity program. To this end, we recommend the creation of a Software Productivity Technical Council (SPTC), modeled after several established technical councils, with a mandate to identify requirements, capabilities, and gaps; coordinate productivity-related R&D; and facilitate the dissemination and adoption of productivity-enhancing tools and practices. The SPTC would advise and assist program managers in addressing the programmatic needs described below.

5 Recommendations

Focused work is imperative to addressing pervasive and urgent problems in extreme-scale software productivity for mission-critical computational science. Efforts to improve productivity must be explicit and deliberate, with research components to tackle issues that are far beyond current understanding. We therefore recommend the following:

> **DOE should sponsor research and development efforts specifically targeted at qualitatively improving software productivity for extreme-scale scientific applications.** The confluence of computer architecture changes and surging demands for modeling, simulation, and analysis of complex multiscale and multiphysics phenomena provide both a crisis and an opportunity to dramatically change how extreme-scale scientific applications are produced and supported within the DOE complex. Furthermore, the broader software community has realized tremendous improvements in productivity that can be leveraged directly and indirectly in our domain. A focused effort to improve productivity is essential to producing the next generations of extreme-scale applications. By fundamentally changing the extreme-scale scientific software infrastructure and culture through an ambitious research agenda, we can develop new methods and approaches for portable, composable, and interoperable applications, libraries, and tools that will serve as a foundation for computational science discovery.

Our specific recommendations include the following:

1. **Characterize and measure extreme-scale software productivity impediments and opportunities.** We must gather, analyze, and categorize software productivity impediments and opportunities in order to provide insight into fundamental problems and direction for future efforts. We must also identify, measure, and track metrics for software productivity throughout software lifecycles, with the ultimate goal of improving key metrics of scientific productivity. (Section 3.1)

2. **Develop software designs that minimize the impact of ongoing computer architecture changes.** We must determine how to achieve good performance on emerging computer architectures while at the same time provide stability for end users. This work is particularly important for libraries that insulate domain scientists from architecture-specific details. (Section 3.2)

3. **Characterize opportunities for enabling advanced multiphysics, multiscale, and analysis capabilities.** Exascale systems open up the opportunity to loosely and tightly couple multiple physics and scales, as well as to introduce parameter sensitivity studies, optimization, and uncertainty quantification. Enabling technologies can greatly improve our abilities to couple existing components and to move simulation to the ultimate goal of providing an optimal solution with error bars. (Section 3.3.1)

4. **Develop composable and interoperable components and libraries.** Research is needed to ensure that our widely used and independently developed software libraries can be readily used *in combination* in extreme-scale multiphysics and multiscale applications that require the complementary contributions of diverse teams. Furthermore, application teams can benefit by organizing their own software into reusable components so that components can be shared across related application efforts. (Section 3.3.2)

5. **Identify, develop, and disseminate knowledge of productivity tools and best practices.** Effective tools and practices provide the most tangible means to improving productivity. Specific focus on identifying the needs for productivity tools, developing and evaluating solutions to those needs, and deploying the best tools and practices will accelerate productivity across all activities, independent of other initiatives. (Sections 3.4 and 3.5)

6. **Develop productivity partnerships throughout DOE and other agencies.** Many application teams are facing the same set of challenges described in this report. Effective communication within DOE and with other federal agencies will maximize our abilities to learn and to leverage knowledge and capabilities. DOE applications already are being used by other agencies, so partnerships will improve communication and knowledge sharing. Furthermore, software productivity is intimately connected to design choices made by other system development teams, such as hardware architecture, programming models, and system software. Software productivity concerns should be taken into account as these system components are designed, developed, and deployed. (Section 4)

7. **Establish a Software Productivity Technical Council.** Although productivity is a major concern for many applications teams, the topic has received little formal or programmatic attention until very recently. As a result, there is little institutional awareness of existing productivity efforts. Some research teams are aware of productivity requirements, and some have made significant local efforts to improve productivity within their own scope of influence. The Software Productivity Technical Council (SPTC) will be a vehicle for gathering requirements and capabilities within DOE and identifying the most important gaps. The SPTC will host presentations by leading research teams, provide regular status reports to the community, and advise DOE program managers on the technical aspects of productivity-related efforts. We anticipate that the SPTC will be important in effectively communicating across the many application development efforts, especially in the early years of building a software productivity R&D effort. (Section 4)

Acknowledgments

The organizers thank Keri Cagle and Deneise Terry, ORISE, for excellent workshop support. We also thank Gail Pieper, ANL, for editing this report, and Zosia Rostomanian, LBNL, for creating the cover.

The organizers are grateful for everyone's thoughtful contributions to the workshop and position papers. We especially thank Bill Harrod, Thuc Hoang, Dorothy Koch, Randall Laviolette, and John Mandrekas, as well as plenary speaker Douglass Post, panelists (John Cary, Kevin Fall, Bill Gropp, Robert Harrison), and all report-back presenters and scribes (Aron Ahmadia, Ivan Bermajo-Moreno, Ross Bartlett, Lisa Childers, Eric Cyr, Anshu Dubey, Bill Gropp, Robert Harrison, Judy Hill, Paul Hovland, Balint Joo, Darren Kerbyson, Mike Kumbera, Mark C. Miller, David Moulton, Ron Oldfield, Lenny Oliker, Roger Pawlowski, Jack Poulson, Andrew Salinger, Tim Scheibe, Andy Terrel, Brian Van Straalen, and Sam Williams). We also thank David Moulton and Bill Tang for information about extreme-scale software productivity issues in environmental management and fusion energy sciences, respectively.

A Science Drivers

Section 2 discusses crosscutting productivity issues in developing mission-critical DOE extreme-scale scientific applications. The following four sections provide more details on science drivers in four application areas: climate (Section A.1), environmental management (Section A.2), fusion energy (Section A.3), and advanced nuclear reactors (Section A.4). The software productivity challenges encountered by these applications are representative of those in a much broader DOE simulation portfolio, including applications in fossil energy, high-energy physics, nuclear physics, chemistry, biology, biophysics, molecular sciences, and materials sciences, as well as those in NNSA.

A.1 Climate

Climate scientists are asked to provide predictive tools to inform the public and decision makers on climate change and its impacts. Climate modeling programs in the DOE Office of Science develop models "to inform the development of sustainable solutions to the Nation's energy and environmental challenges [28]." Climate models are complex multiphysics, multiscale applications. Representing all the internal variability and other features of the climate system requires several individual models of the atmosphere, ocean, ice, and land coupled to one another through exchanges of heat, mass, and momentum. Each of these models is itself a multiphysics application with a variety of subcomponents representing physical, chemical, and biological processes. In order to provide a diverse community of stakeholders with better estimates of regional/local changes and climate extremes, ensembles of high-resolution simulations are needed. Current high-resolution ($\sim$ 10 km grid spacing) configurations require hundreds of millions of core-hours on leadership-class computing resources for a small (5-member) ensemble of only 30 simulated years each. For future extreme-scale computations and the science needed to inform climate-related decisions, larger ensembles at subkilometer resolution will be desired, with associated uncertainty estimates.

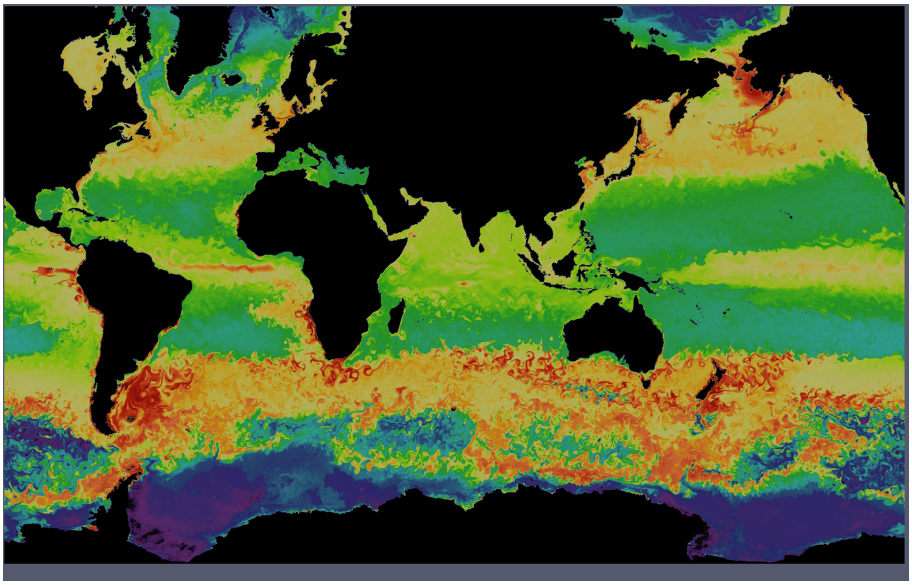


Figure 2: Chlorophyll concentration from an eddy-resolving ocean simulation that includes a biogeochemical model to explore carbon uptake and ocean acidification.

Climate scientists since the 1960s have been implementing models on the most advanced computing architectures while also preserving and advancing the fidelity of climate simulations as they have evolved.

The last major computing transition in the early 1990s required roughly five years to translate climate models from vector-based hardware to the parallel message-passing programming model that has persisted until the present. Since then, climate scientists have taken advantage of Moore's Law and the relative stability in programming models to add far more capability to climate models, couple more components, and increase resolution for more accuracy and detail. Today's climate modeling efforts involve hundreds of climate scientists, applied mathematicians, and computer scientists working on a highly complex code base of over 1.2 million lines (mostly Fortran) written over decades of development.

DOE climate scientists are now initiating a project to transition this large and complex modeling system to advanced architectures and prepare for exascale systems while maintaining and advancing the scientific capabilities. Early experience on accelerated systems has shown that even with substantial effort and extensive refactoring, only moderate factors ($\sim$ 2x) of performance improvement can be obtained [60] because of the lack of well-defined kernels and the broad mix of algorithms deployed. Like previous transitions, the path to exascale will require exploration of new algorithmic choices and a significant investment in software engineering as we revisit software design choices and develop testing to ensure quality throughout the transition. Challenges will include the following.

Characterization and measurement of software productivity and impediments. Climate modelers target a throughput of 2-10 simulated years per CPU day to maintain a reasonable workflow and achieve the required ensemble simulation results. But performance is only a small part of overall productivity. From past experience, high-resolution grand challenge simulations take five years from the start of model configuration to the publication of the first result. Because of the complexity of the system and the lack of overall productivity metrics, it is unclear how best to reduce that five-year timescale. New software productivity metrics and consistent measurement of those metrics are needed (Sec. 3.1).

Development of performance-portable climate models on diverse extreme-scale systems. A substantial effort in refactoring and algorithm development will be required to create performance-portable climate models for exascale (Sec. 3.2). A recent NRC report on climate modeling [10] pointed to the need for a community-wide high-performance climate modeling software infrastructure. Such an infrastructure would feature shared abstractions that enable performance portability, adaptability to rapid changes in underlying architecture, and ease of programming for nonexpert developers.

Climate model testing and reliability. During the transition, the model must be continuously tested to ensure trust in the model solutions. Currently, climate models are validated against the historical climate record for a number of observed variables. Only a fraction of the code is routinely unit tested, typically in the infrastructure routines where functionality is better defined and answers can be verified. Unit testing in other cases is inhibited by the lack of data or reference solutions. The complexity of a large production climate code presents too many options to completely test all combinations of potential user choices. Better testing coverage and methodologies are needed (Sec. 3.3.4), recognizing also the limitations of current bit-reproducibility tests and the challenges of resilience (Sec. 3.2.4).

Use of high-performance libraries and tools. Use of libraries and tools would help the goal of performance portability (Sec. 3.3), but use of libraries and tools within the climate community has in the past been inhibited by the fragility of software (e.g., parallel I/O), lack of long-term support, lack of language support, portability problems, or software design that requires too many modifications to the climate model in order to incorporate the library or tool.

Scalable and effective software processes. The large number of developers with varying software expertise poses significant challenges for a large, distributed climate modeling project. Current processes in the climate community utilize centralized repositories and a relatively small group of software engineers. A more distributed and scalable software process is needed but will require training in and adoption of proven software methodologies that can be customized for large, distributed model development teams (Sec. 3.4).

A.2 Environmental management

The Office of Environmental Management (EM) oversees the remediation and closure of DOE sites storing legacy waste from the development of nuclear weapons and related technologies. Although significant cleanup progress has been made, several large and technically complex sites remain, with total lifecycle cost estimates between $272 and $327 billion. The groundwater and soil contamination alone includes more than 40 million cubic meters of contaminated soil and debris and 1.7 trillion gallons of contaminated groundwater at 17 sites in 11 states [61]. At these complex sites the conservative simplifications and abstractions used in the past to estimate the fate and transport of contaminants can lead to overly conservative and costly remediation and closure scenarios. For the complex cleanup problems that remain, the highly uncertain and multiscale nature of the subsurface hydrology and geochemical transport must be treated mechanistically, requiring advanced modeling and data analysis techniques in order to inform a comprehensive approach to risk and performance assessment. To address this urgent need for transformational solutions, EM initiated the Advanced Simulation Capability for Environmental Management (ASCEM) program in 2010 [3]. The expectation of DOE-EM and the earth science community is that programs such as ASCEM can move beyond existing regulatory codes, significantly reducing reliance on conservative abstractions and simplifications through the use of advanced and emerging computational methods.

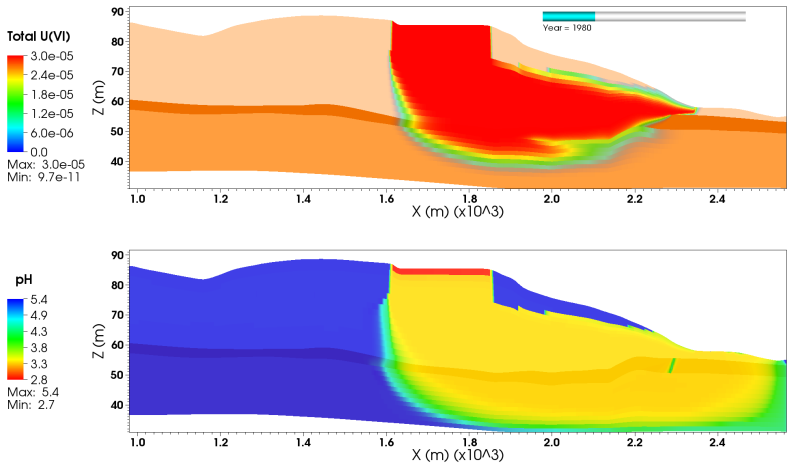


Figure 3: Simulations of complex geochemistry and hydrology at the Savannah River F-Area seepage basins capture the retardation of the uranium plume, prior to any remedial action. Sorption of uranium depends on the pH, and hence, the uranium plume (top) lags behind the acidic front of the waste water (bottom). Manipulation of pH is a key ingredient in controlling uranium migration in the future [39]. ASCEM tools under development will support simulations and analysis of the effects of ongoing and future remedial actions leading to better projections of the time and cost required to meet remedial goals.

Subsurface flow and transport are governed by a particularly challenging suite of coupled multiscale processes (including surface and subsurface flows, biogeochemical reactions, and thermal-mechanical deformations) occurring in highly heterogeneous subsurface environments with external forcing. Exacerbating the complexity is the often-limited information about the heterogeneous distribution of subsurface flow and transport properties and reaction networks. Consequently, risk and performance assessment uses a graded and iterative approach to underpin scientifically defensible decisions and strategies. This approach first establishes simplified models and then iteratively enhances geometric and process-level complexity to identify and characterize the key processes and assumptions that are needed to efficiently reach a defensible decision. ASCEM must design its approach and tools to handle this workflow, not only to be flexible and extensible,

but also to leverage an increasingly powerful and diverse set of computational resources. To achieve this goal, the ASCEM program has identified common themes found in multiscale and multiphysics applications and has assembled an interdisciplinary team to leverage advances from various DOE SC programs, ASC, and earth science. This effort is in its infancy and faces many of the challenges outlined in Section 3. At the heart of these challenges for ASCEM is the need to develop a new approach to the lifecycle of software tools in the earth sciences. In particular, development and collaboration tools continue to advance, enabling much more agile approaches that can make use of hierarchical and automated testing. Furthermore, languages and design methodologies continue to advance, improving our ability to reduce long-term maintenance and refactoring costs. These advances could enable a much more efficient approach to development and maintenance of regulatory codes, whose strict quality assurance requirements have significantly hindered adoption of new algorithms and architectures. Key software productivity challenges include the following.

Characterization and measurement of software productivity and impediments. Productivity of developers and users of subsurface flow and reactive transport simulation codes has not been measured or tracked in the past. Consequently, there is little understanding of the true cost of developing current regulatory codes. For example, despite their long history, there is no record of the impact that design and testing practices have had on the cost of adding new capabilities or supporting new architectures. It is imperative that metrics of software productivity such as these (Section 3.1.3) be developed to help generate *best practice* guidelines for both developers and users (Section 3.4.3).

Embracing of continual change to enhance productivity. Challenging science questions naturally drive the demand for higher-fidelity predictive simulations, while emerging architectures demand higher levels of parallelism in algorithms and implementation. A new, modular, hierarchical design of integrated tools that addresses directly these facets of the software lifecycle (Section 3.2.1) will greatly enhance overall productivity. Specifically, this flexible and extensible design would provide a suitable framework to encapsulate and refactor key computational kernels, as well as legacy codes and tools.

Hierarchical testing tools and techniques. A hierarchical testing framework that provides significant automation for execution, reporting, and analysis is critical to productivity in the extreme-scale settings (Section 3.3.4). Specifically, the positive impact of testing on productivity is observed in the efficiency and confidence with which components can be refactored and in the ability to demonstrate reproducibility across platforms. In addition, it enables the efficient migration of code from the research branch, through stable community code releases, to fully qualified regulatory releases (NQA-1 [48])

Support for flexible nonlinear workflows. A critical feature of EM applications is the integration of observations, modeling, and simulation in a nonlinear iterative workflow that informs the graded approach to supporting decisions for cleanup and monitoring. This nonlinear iterative workflow is increasingly found in extreme-scale-enabled high-impact science, where the need for fundamental understanding drives analysis beyond isolated high-fidelity simulations (Section 3.3). But, this workflow is inherently difficult to manage and requires new interfaces and tools to support productivity.

Productive software development with an interdisciplinary team. The complexity of both the science in EM applications and the software on extreme-scale architectures is driving the need for truly interdisciplinary development teams. To be productive, however, an interdisciplinary team requires significant training in order to build a common understanding of both the application modeling and the simulation needs, as well as modern agile software design, development, and testing practices (Sections 3.4.1 and 3.5.1).

Multiscale algorithms, frameworks, and libraries. The steady increase in computational power is driving interest in high-fidelity multiscale simulations. Multiscale methods are in their infancy, however, and so a flexible modular approach is needed to support algorithmic research on emerging architectures (Section 3.3.1). This approach should maximize code reuse and leverage existing numerical libraries.

A focused effort to address these complex software productivity challenges is needed in order to realize the potential benefits of emerging computational power.

A.3 Fusion energy sciences

Fundamentally, the fusion of light nuclides forms the basis of energy release in the universe, which can potentially be harnessed and used as a clean and sustainable supply of energy. The development of fusion energy to meet national energy needs is a complex, international endeavor, with the next magnetic-fusion plasma confinement device, ITER [49], having an estimated construction cost of $20B. It is estimated that the integrated cost of an ITER discharge, of which there will be approximately 10 per day, will approach $1M. Consequently, close collaboration between experiment and simulation is essential, with computational modeling used to guide the planning of experimental campaigns and analyze the results of each shot. Studies emphasize the imperative for whole-device modeling that integrates nonlinearly-coupled phenomena in the core plasma, edge plasma, and wall region on time and space scales required for fusion energy production [62].

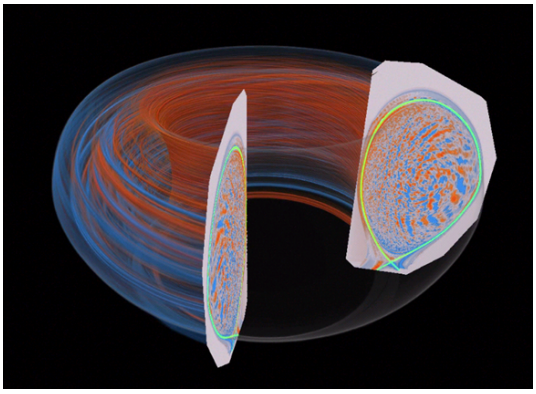


Figure 4: Flux driven turbulence (fluctuating electrostatic potential), filling the whole plasma volume in diverted DIII-D geometry. This simulation by the XGC-1 code (24M CPU hours, engaging 100K cores of the OLCF Jaguar system for 240 hours) integrates plasma dynamics in the complex edge with the core region of the DIII-D Tokamak plasma.

Because achieving high-fidelity predictive modeling on extreme-scale computing systems requires the combined contributions of diverse teams for various aspects of simulation and analysis, software productivity challenges are at the forefront of fusion simulation concerns. A sizable body of code exists, which has been carefully validated against experiment and is accepted by the community. Many of these codes were originally developed years and even decades ago and are not designed to easily expose the degrees of parallelism required to scale well on current HPC systems, much less future systems. On the other hand, many of these codes continue to be developed and their models refined even today. Many in the community tend to be leery of efforts to significantly revise such codes for future architectures, as well as efforts to incorporate them into integrated whole-device models, because of concerns about divergence of code bases and the validation challenges. On the other hand, wholesale re-creation of the capabilities of this software base in new codes that are more friendly to future architectures and to multiphysics coupling also raises concerns about validation, as well as time and expense [40] in a period where FES budgets in the DOE have been stretched very thin.

Regardless of which path forward is chosen (probably at the level of individual codes or modeling capabilities), the computational fusion community will have to make extensive use of software productivity techniques in order to deliver the needed simulation capabilities in a form ready for extreme-scale systems under tight constraints on time and budget. Key concerns are as follows.

Modernization of legacy fusion codes for new many-core computing architectures. Novel program-

ming approaches are needed in order to achieve scalability and preserve investment in large and trusted code bases. These needs span throughout both fundamental fusion applications and numerical software libraries on which they depend (Section 3.2).

Fusion simulation testing, verification, and validation. Proper cross-validation of laboratory experiments with a suite of advanced codes in regimes relevant for producing practical fusion energy will demand systematic testing across a large parameter space (Section 3.3.4).

Advanced coupling methodologies. Coupled multiphysics modeling of fusion plasmas, also referred to as whole-device modeling or integrated modeling, is in its infancy. However, experience to date suggests that such simulations pose many physical, mathematical, and computational challenges that will require extensive research (Section 3.3.1).

Integrating data analytics with simulations. We must address "big data" challenges associated with avoiding macroscopic disruptive events in fusion-grade plasmas.

Development of community-wide fusion modeling software infrastructure. Shared higher-level abstractions are needed in order to enable portability and adaptability to rapid changes in underlying architecture, as well as to ease programming for nonexpert developers (Section 3.2.5). Additional needs for a fusion community codes include build systems (Section 3.3.3), support for composability (Section 3.3.2), and integration testing (Section 3.3.4).

Improved fusion community software development practices and training. Computational fusion teams have widely varying software expertise and preferences. In the push for integrated whole-device modeling, there is a strong need for software methodologies that can more effectively scale across multiple distributed teams (Section 3.4). Also needed is outreach help to train new staff on productivity practices (Section 3.5).

A.4 Nuclear energy

The design, certification, and operation of the existing fleet of nuclear reactors depend on sophisticated computational tools that have been developed and improved continuously since the advent of the nuclear era. Reactor analysis codes aim to predict key macroscopic quantities relevant to the safe and efficient operation of reactors. These include reactivity, power distributions, temperature profiles, material integrity, and detailed isotopics both in quasi-steady-state and transient (including accident) scenarios. The underlying computational methods involve various approximate solutions to the quasi-linear Boltzmann transport equation, the Navier Stokes equations with conjugate heat transfer, and the Bateman equations for isotopics, coupled to mechanistic models of macroscopic fuel properties and transient behavior.

In the early days the computational approaches were based on significant a priori simplifications to the governing equations (e.g., point kinetics model, porous flow) to render them computationally feasible. Over time, as new algorithms have been discovered and computer power has increased, these methods have been slowly replaced by models of continuously increasing physical fidelity, such as multigroup transport, Monte Carlo methods, and large eddy simulations.

From the perspective of the end user—the analyst, designer, or licensing authority—reactor simulation tools must meet certain rigid requirements in terms of usability, time to solution, accuracy, and reproducibility. For example, the end user must be able to develop arbitrary reactor models in reasonable time (one month) without detailed knowledge of the inner workings of the code; for design scoping studies a single, fixed-point power distribution calculation must take no more than several minutes to be of practical use; estimates of sensitivities to simulation results based on the uncertainty in input data must be provided for each calculation; the ability to obtain bitwise reproducibility of simulation results is mandated by licensing authorities. Furthermore, new methods have to be rigorously validated against experimental data.

The nation has recently invested in several significant simulation programs aimed at pushing forward the state of the art in reactor simulation tools. The CASL Hub aims to integrate near-term and existing advanced

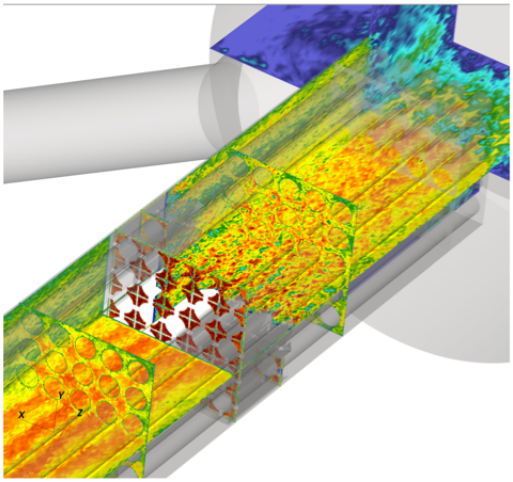


Figure 5: Large eddy simulation of turbulent coolant flow through a 217-pin nuclear reactor assembly.

simulation capabilities into an end-user tool ('virtual reactor') that can be adopted immediately by end users to tackle outstanding industry problems. The CESAR co-design center is focusing on the development and impact of next-generation methods for exascale architectures.

Faster time to solution for more detailed physics models defines only one aspect increased overall productivity for end users. Improved computing capability must co-exist with several key aspects of the simulation lifecycle. Some key examples are as follows.

Flexible software architectures. Many legacy reactor physics packages suffer from an exponential proliferation of physics subroutines—minor variants of each physics module for each physical scenario—resulting in unmanageable software suites. Part of this is the result of different numerical requirements for different parameter regimes, but more often it is the result of poorly engineered software. In order to increase productivity, software architectures are needed that enable reuse of basic physics models for a broad range of end-user applications, ranging from transient accident scenarios to steady-state power profiles at startup. (Sections 3.3 3.4).

Reproducibility for licensing. Licensing authorities depend on reactor analysis codes to adjudicate new designs. Historically, authorities have required bitwise reproducibility of results as a demonstration of code consistency. This level of constraint is likely unnecessary and infeasible on next-generation systems. The interaction between strict licensing requirements, testing, verification, and solution reproducibility on emerging systems is a key productivity question for next-generation codes (Sections 3.3.4 and 3.2.4).

Computational meshes for physics coupling. Reactor designers need to quickly evaluate concepts based on geometric modifications to existing designs. Often, re-meshing new geometries, particularly for coupled neutron-fluid problems, can be a several-month, painstaking process. Overall productivity would be greatly enhanced if this timeline could be reduced, either by developing more robust meshing methods, or by exploring innovative algorithms (particle based, cut-cell, etc.) (Section 3.3.1).

Propagation of uncertainty through coupled codes. All reactor neutronics calculations rely on large amounts of experimental data that encode the probability of a vast array of different neutron-nuclide interactions. The accuracy limits of this data are fairly well understood, but their impact on solutions to the transport equation, particularly in multiphysics calculations (e.g., transport-fluids-fuel), is much less well understood. The net effect is that overly precise meshes are employed, yielding solutions whose accuracy is

not warranted by the data accuracy. Better understanding of the propagation of uncertainty through coupled physics calculations could greatly reduce overall analysis time for future codes (Section 3.3.1).

Community frameworks for pre- and post-processing. Much of the time in the overall workflow of reactor analysis is spent in both setting up and analyzing data. While community visualization and meshing software can be leveraged for much of this work, other related issues have been less well addressed by tool designers. In-situ data analysis, analytics packages for data on complex meshes, and real-time control visualization are areas where community efforts could be profitably leveraged by the nuclear energy community (Section 3.3).

Development of software architectures that enable new algorithmic developments to be adopted in complex production codes. In the era of exascale computing existing production codes must be able to take advantage of the research of exascale methods without being completely rewritten. This work requires careful thought into how to refactor production codes to allow targeted improvements appropriate for exascale architectures (Sections 3.2 and 3.4).

B HPC libraries: Exemplars for enhancing scientific productivity

Scientific libraries have long been exemplars of rigorous software engineering efforts. The advantages of developing and using these libraries are clear from the numerous applications that have benefited from their use. Even so, we have much progress to make, both in making existing libraries better and in increasing the fraction of scientific software that is delivered in this way [42]. It is not too strong to say that the default approach for all scientific software should be to make it modular and reusable. A developer first should have to defend why new development is required (and not obtained from existing components) and then should design and develop the new functionality so it is modular and reusable, or defend why doing so is not appropriate.

B.1 Attributes of successful libraries and tools

The value proposition of DOE software libraries is that a client can obtain suitable functionality from an external source such that the combined functionality, performance, ease-of-use, and reliability outweigh the alternative of developing a proprietary equivalent. Historically, mathematical libraries for dense linear algebra have provided a clear advantage for users. Data structures are easy to standardize, performance improvements from optimal implementations are substantial, and the most robust implementations require a great deal of numerical algorithms expertise to implement correctly. LINPACK [32] and EISPACK [64] were seminal efforts, eventually replaced by BLAS [1, 29, 33, 55, 56] and LAPACK [5], which are unmitigated successes. Furthermore, these packages have sustained reliability and versatility because their interfaces have survived numerous architecture changes, new algorithms continue to be added, and each package has an extensive test suite [34] that ensures confidence in development and refactoring of optimized implementations. There are certainly use cases for which BLAS and LAPACK are not optimal (e.g., stiffness matrix computations for some finite element applications), but even in those cases an application developer has to provide a well-designed and well-implemented alternative in order to obtain better performance, and is then prohibited from accessing the rich collection of functionality that BLAS and LAPACK provide.

The advent of MPI and the relative stability of programming models during the past two decades have enabled the development of HPC libraries for parallel meshing, discretization, solvers, visualization, I/O, and other required functionality. For example, numerical packages within the FASTMath project [18], including BoxLib [8, 9], Chombo [2, 17], hypre [37, 38], MOAB [67, 68], PETSc [6, 7] SUNDIALS [45, 69], SuperLU [57, 58], and Trilinos [43, 44], and others such as PyClaw [4] and MOOSE [41], have a strong track record of scientific application impact by providing scalable performance, good and stable interfaces, a large collection of functionality, rigorous test suites, and continued growth in capabilities.

B.2 Focus on modularity and reusability

Almost any scientific application can be designed for modularity and implemented as a collection of reusable components. However, doing so does require extra effort, additional skills, and attention to details not required for single-use source code. Even so, the exercise of developing a domain model for an application, including the application's description of the equations that must be formulated and solved, has intrinsic value [36]. The application domain model can be used to define application-specific modules that may be of value to other projects and can in fact become library components themselves. Furthermore, the domain model helps clarify how an application can interface with external entities such as solvers and provides the proper abstraction layer that permits solvers to be integrated with minimal incidental coupling and optimal cohesion.

Application teams have adopted some level of modularity and reusability and have gradually moved to incorporating solvers, physics packages, and other services as external dependencies. However, much can

be gained by expanding this approach. Eventually, much of the application software that is written can be made sufficiently modular to allow a striking increase in flexibility to incorporate a diverse range of physical models and numerical algorithms.

B.3 Abstraction layers and usage strategies

Use of DOE numerical software libraries comes with nontrivial risks. The decision to adopt a capability and put its use on the critical path to success requires some risk mitigations. In software design, one proven strategy is to introduce an abstraction layer, that is, to encapsulate the external functionality in an interface where any given library is one option to providing the service, an option that can be replaced by another service provider or can be used side by side with another provider. At another level, abstraction layers are commonly used within high-performance libraries to protect applications from the details of algorithmic implementations.

The same abstraction layer that protects applications from particular library details can often be part of numerical software libraries themselves, instead of within each application code, through the use of "wrapper libraries." For example, PETSc's abstract linear solver interface supports solvers from hypre, SuperLU, Trilinos, and several other external packages. Similarly, there is support for Chaco, Parmetis, Scotch, and Zoltan within the partitioning interface. Thus, applications can have a wide variety of choices for algorithmic implementations without requiring a great deal of interface code that they must maintain.

Abstraction layers are useful risk mitigations, but they come with their own risk: too many abstraction layers. At some level, concrete functionality must be invoked, and complicated code must be written in terms of some specific interface. Furthermore, specialized abstractions can lead to repeated implementations of concrete adapters for two or more very similar abstraction layers.

Therefore, the client of a library must either create an abstraction layer to encapsulate calls to the library, or directly use the library's interface, or use some combination of these two options. The right choice is determined by factors such as the following: (i) How probable is it that the given library will be replaced by or used with another library to provide the service? (ii) How difficult is it to define an abstraction layer to encapsulate the library? (iii) Is there a performance or complexity penalty inherent in creating an abstraction layer? (iv) Is the library reliable enough to be trusted as a "must-use" component?

Use of third-party software libraries can be a tremendous productivity and capability benefit. At the same time, issues in interfacing with and depending on these libraries must be carefully considered in application design and implementation. In an ideal world, almost all scientific software would be written to be modular and reusable, but this ideal is tempered by constraints of time and skill.

B.4 Regression testing and backward compatibility

To effectively use a third-party library, application developers must be able to trust that updates to the library will maintain compatibility and avoid regressions. For library developers, these commitments are very demanding and require a level of investment in staffing, testing infrastructure, and software design that is far more costly than for single-purpose efforts. A comprehensive unit test suite, a commitment to avoid interface changes, and rapid response to regressions all require a large commitment on the part of library development teams. Likewise, application development teams must embrace a comprehensive integration testing process to ensure that updates and changes to individual components do not adversely affect application end users. Ideally, developers of an application that relies on third-party software components should have the capability to reconstruct any version of the integrated application by composing the corresponding versions of the individual libraries from which it was originally built.

References

[1] . An updated set of basic linear algebra subprograms (BLAS). *ACM Trans. Math. Softw.*, 28(2):135–151, June 2002. ISSN 0098-3500. doi:10.1145/567806.567807.

[2] M. Adams, P. Colella, D. T. Graves, J. N. Johnson, N. D. Keen, T. J. Ligocki, D. F. Martin, P. W. McCorquodale, D. Modiano, P. O. Schwartz, T. D. Sternberg, and B. V. Straalen. Chombo Software Package for AMR Applications - Design Document. Technical Report 6616E, Lawrence Berkeley National Laboratory, 2014.

[3] Advanced Simulation Capability for Environmental Management (ASCEM) program. Advanced simulation capability for environmental management (ASCEM) program. http://ascemdoe.org.

[4] A. Alghamdi, A. Ahmadia, D. I. Ketcheson, M. G. Knepley, K. T. Mandli, and L. Dalcin. PetClaw: A scalable parallel nonlinear wave propagation solver for Python. In *Proceedings of SpringSim 2011*. ACM, 2011.

[5] E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, J. J. Dongarra, J. Du Croz, S. Hammarling, A. Greenbaum, A. McKenney, and D. Sorensen. *LAPACK Users' Guide (Third Ed.)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1999. ISBN 0-89871-447-8.

[6] S. Balay, M. F. Adams, J. Brown, P. Brune, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, K. Rupp, B. F. Smith, and H. Zhang. PETSc Web page. URL http://www.mcs.anl.gov/petsc.

[7] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith. Efficient management of parallelism in object oriented numerical software libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhauser Press, 1997.

[8] J. Bell et al. BoxLib Web page. URL https://ccse.lbl.gov/BoxLib.

[9] J. Bell et al. BoxLib User's Guide. Technical report, CCSE, Lawrence Berkeley National Laboratory, 2012. https://ccse.lbl.gov/BoxLib/BoxLibUsersGuide.pdf.

[10] C. Bretherton (chair) and committee. *A National Strategy for Advancing Climate Modeling*. National Academies Press, 2012.

[11] J. C. Carver. First international workshop on software engineering for computational science & engineering. *Computing in Science & Engineering*, 11(2):7–11, 2009. ISSN 1521-9615. doi:10.1109/MCSE.2009.30.

[12] J. C. Carver. Report: The second international workshop on software engineering for CSE. *Computing in Science & Engineering*, 11(6):14–19, 2009. ISSN 1521-9615. doi:10.1109/MCSE.2009.203.

[13] J. C. Carver. Software engineering for computational science and engineering. *Computing in Science & Engineering*, 14(2):8–11, 2011. ISSN 1521-9615. doi:10.1109/MCSE.2012.31.

[14] J. C. Carver. Software engineering for computational science and engineering. *Computing in Science & Engineering*, 14(2):8–11, March 2012. ISSN 1521-9615. doi:10.1109/MCSE.2012.31.

[15] J. C. Carver and G. K. Thiruvathukal. Software engineering need not be difficult. In *Proceedings of the First Working towards Sustainable Software for Science: Practice and Experiences*, 2013. URL http://dx.doi.org/10.6084/m9.figshare.830442.

[16] J. C. Carver, R. P. Kendall, S. E. Squires, and D. E. Post. Software development environments for scientific and engineering software: A series of case studies. In *Proceedings of the 29th International Conference on Software Engineering*, ICSE '07, pages 550–559, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 0-7695-2828-7. doi:10.1109/ICSE.2007.77. URL http://dx.doi.org/10.1109/ICSE.2007.77.

[17] P. Colella et al. Chombo Web page. URL https://commons.lbl.gov/display/chombo.

[18] L. Diachin (PI). SciDAC-3 Frameworks, Algorithms, and Scalable Technologies for Mathematics (FASTMath) Institute. URL http://www.fastmath-scidac.org/.

[19] DOE ASCAC Subcommittee Report. The opportunities and challenges of exascale computing, 2010. URL http://science.energy.gov/~/media/ascr/ascac/pdf/reports/Exascale_subcommittee_report.pdf.

[20] DOE ASCAC Subcommittee Report. The top ten exascale research challenges, 2014. URL http://science.energy.gov/~/media/ascr/ascac/pdf/reports/2013/report.pdf.

[21] DOE report. Applied mathematics at the U.S. Department of Energy: Past, present and a view to the future, May 2008. URL http://science.energy.gov/~/media/ascr/pdf/program-documents/docs/Brown_report_may_08.pdf.

[22] DOE report. Scientific grand challenges: Challenges for the understanding the quantum universe and the role of computing at the extreme scale, December 2008. URL http://science.energy.gov/~/media/ascr/pdf/program-documents/docs/Hep_report.pdf.

[23] DOE report. Scientific grand challenges: Discovery in basic energy sciences: The role of computing at the extreme scale, August 2009. URL http://science.energy.gov/~/media/ascr/pdf/program-documents/docs/Bes_exascale_report.pdf.

[24] DOE report. Scientific grand challenges: Fusion energy sciences and the role of computing at the extreme scale, March 2009. URL http://science.energy.gov/~/media/ascr/pdf/program-documents/docs/Fusion_report.pdf.

[25] DOE report. Scientific grand challenges: Forefront questions in nuclear science and the role of computing at the extreme scale, January 2009. URL http://science.energy.gov/~/media/ascr/pdf/program-documents/docs/Np_report.pdf.

[26] DOE report. Scientific grand challenges for national security, October 2009. URL http://science.energy.gov/~/media/ascr/pdf/program-documents/docs/Nnsa_grand_challenges_report.pdf.

[27] DOE report. Scientific grand challenges: Cross-cutting technologies for computing at the exascale workshop, February 2010. URL http://science.energy.gov/~/media/ascr/pdf/program-documents/docs/Crosscutting_grand_challenges.pdf.

[28] DOE Strategic Plan. Climate and environmental sciences division strategic plan. Technical Report DOE/SC-0151, U.S. Department of Energy, Biological and Environmental Research, 2012. URL http://science.energy.gov/~/media/ber/pdf/CESD-StratPlan-2012.pdf.

[29] J. Dongarra, J. DuCroz, S. Hammarling, and R. Hanson. An extended set of Fortran basic linear algebra subprograms. *ACM Trans. Math. Softw.*, 14, 1988.

[30] J. Dongarra, R. Graybill, W. Harrod, R. Lucas, E. Lusk, P. Luszczek, J. McMahon, A. Snavely, J. Vetter, K. Yelick, S. Alam, R. Campbell, L. Carrington, T.-Y. Chen, O. Khalili, J. Meredith, and M. Tikir. DARPA's HPCS program: History, models, tools, languages. *Advances in Computers*, 72:1–100, 2008.

[31] J. Dongarra, P. Beckman, et al. The International Exascale Software Project Roadmap. *Int. J. High Perf. Comput. Applics.*, 25:3–60, 2011.

[32] J. J. Dongarra, J. Bunch, C. Moler, and G. Stewart. *LINPACK Users' Guide*. SIAM Pub., 1979.

[33] J. J. Dongarra, J. Du Croz, S. Hammarling, and I. Duff. A set of level 3 basic linear algebra subprograms. *ACM Trans. Math. Softw.*, 16(1):1–17, March 1990.

[34] J. J. Dongarra, J. Du Croz, S. Hammarling, and I. Duff. A set of level 3 basic linear algebra subprograms: Model implementation and test programs. *ACM Trans. Math. Softw.*, 16(1):18–28, March 1990.

[35] A. Dubey, E. Balaras, D. Q. Lamb, and C. Eder. Workshop on Building Community Codes for Effective Scientific Research on HPC Platforms, 2012. URL http://flash.uchicago.edu/cc2012.

[36] E. Evans. *Domain-Driven Design: Tackling Complexity in the Heart of Software*, volume 70. Addison Wesley, 2003. ISBN 0321125215. doi:10.1159/000067454.

[37] R. Falgout et al. hypre Web page. URL http://www.llnl.gov/CASC/hypre.

[38] R. D. Falgout, J. E. Jones, and U. M. Yang. *The Design and Implementation of hypre, a Library of Parallel High Performance Preconditioners*, volume 51, pages 267–294. Springer-Verlag, 2006.

[39] M. Freshley, S. Hubbard, I. Gorton, J. D. M. C. I. Steefel, V. Freedman, H. Wainwright, et al. Phase II Demonstration. Technical Report ASCEM-SITE-2012-01 (LANL: LA-UR 12-25496), Office of Environmental Management, United States Department of Energy, Washington, DC, 2012.

[40] FSP. Fusion Simulation Project (FSP) Web Site. http://www.pppl.gov/fsp/.

[41] D. Gaston, C. Newman, G. Hansen, and D. Lebrun-Grandié. MOOSE: A parallel computational framework for coupled systems of nonlinear equations. *Nuclear Engineering and Design*, 239:1768–1778, 2009. doi:10.1016/j.nucengdes.2009.05.021.

[42] W. D. Gropp. Exploiting existing software in libraries: Successes, failures, and reasons why. In *Proceedings of the SIAM Workshop on Object Oriented Methods for Inter-operable Scientific and Engineering Computing*, pages 21–29. SIAM, 1999.

[43] M. A. Heroux, R. Bartlett, V. Howle, R. Hoekstra, J. Hu, T. Kolda, R. Lehoucq, K. Long, R. Pawlowski, E. Phipps, A. Salinger, H. Thornquist, R. Tuminaro, J. Willenbring, A. Williams, and K. Stanley. An overview of the Trilinos project. *ACM Trans. Math. Software*, 31:397–423, September 2005. ISSN 0098-3500. doi:10.1145/1089014.1089021.

[44] M. A. Heroux et al. Trilinos Web page. URL http://trilinos.org.

[45] A. C. Hindmarsh, P. N. Brown, K. E. Grant, S. L. Lee, R. Serban, D. E. Shumaker, and C. S. Woodward. SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. *ACM Trans. Math. Software*, 31:363–396, September 2005. ISSN 0098-3500. doi:10.1145/1089014.1089020.

[46] L. Hochstein and Y. Jiao. The cost of the build tax in scientific software. In *Empirical Software Engineering and Measurement (ESEM), 2011 International Symposium on*, pages 384–387, Sept 2011. doi:10.1109/ESEM.2011.54.

[47] IEEE Standard Glossary of Software Engineering Terminology. Ieee standard glossary of software engineering terminology. *IEEE Std 610.12-1990*, pages 1–84, Dec 1990. doi:10.1109/IEEESTD.1990.101064.

[48] A. N. (including addenda 2005 and 2007). *Quality Assurance Requirements for Nuclear Facility Applications (QA)*. American Society of Mechanical Engineers, 2007.

[49] ITER Organization. ITER: The Way to New Energy. http://www.iter.org.

[50] D. Katz, G. Allen, N. C. Hong, M. Parashar, and D. Proctor. First Workshop on Sustainable Software for Science: Practice and Experiences. URL http://wssspe.researchcomputing.org.uk/wssspe1.

[51] J. Kepner (Guest Editor). High productivity computing systems and the path towards usable petascale computing, part a: User productivity challenges. *Cyberinfrastructure Technology Watch (CTWatch) Quarterly*, 2(4A), 2006.

[52] D. E. Keyes, L. C. McInnes, C. Woodward, W. 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. P. Randles, D. Reynolds, B. Rivière, 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. *International Journal of High Performance Computing Applications*, 27(1):4–83, Feb 2013. doi:10.1177/1094342012468181. Special issue.

[53] P. Kogge (Editor). Exascale computing study: Technology challenges in achieving exascale systems, 2008.

[54] G. K. Kumfert and T. G. W. Epperly. Software in the DOE: The hidden overhead of ”the build”. technical report UCRL-ID-147343, Lawrence Livermore National Laboratory, 2002. http://dx.doi.org/10.2172/15005938.

[55] C. Lawson, R. Hanson, D. Kincaid, and F. Krogh. Basic linear algebra subprograms for Fortran usage. *ACM Trans. Math. Softw.*, 5, 1979.

[56] C. Lawson, R. Hanson, D. Kincaid, and F. Krogh. Algorithm 539: Basic linear algebra subprograms for Fortran usage. *ACM Trans. Math. Softw.*, 5, 1979.

[57] X. Li, J. Demmel, et al. SuperLU Web page. URL http://crd.lbl.gov/~xiaoye/SuperLU.

[58] X. S. Li. An overview of SuperLU: Algorithms, implementation, and user interface. *ACM Trans. Math. Softw.*, 31(3):302–325, September 2005.

[59] S. McConnel. *Code Complete: A Practical Handbook of Software Construction*. Microsoft Press, 2004. second edition.

[60] M. Norman. Targeting atmospheric simulation algorithms for large, distributed-memory, gpu-accelerated computers. In D. Yuen, L. Wang, X. Chi, L. Johnson, W. Ge, and Y. Shi, editors, *GPU Solutions to Multi-scale Problems in Science and Engineering*, Lecture Notes in Earth System Sciences. Springer, 2013.

[61] Office of Environmental Management. Science and technology to reduce the life cycle cost of closure – investing in our future: Technology innovation and development for footprint reduction. Technical report, U.S. Department of Energy, Washington, DC, 2010.

[62] D. Post, D. Batchelor, R. Bramley, J. Cary, R. Cohen, P. Colella, and S. Jardin. Report of the Fusion Simulation Project Steering Committee. *J. Fusion Energy*, 23:1, 2004.

[63] S. Sachs (Editor). Tools for exascale computing: Challenges and strategies: Report of the 2011 ASCR exascale tools workshop, 2011. Office of Science, U.S. Department of Energy.

[64] B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow, Y. Ikebe, V. C. Klema, and C. B. Moler. *Matrix Eigensystem Routines – EISPACK Guide*, volume 6 of *Lecture Notes in Computer Science*. Springer–Verlag, New York, second edition, 1976.

[65] M. Snir and D. A. Bader. A framework for measuring supercomputer productivity. *IJHPCA*, 18: 417–432, 2004.

[66] T. Sterling. Productivity metrics and models for high performance computing. *IJHPCA*, 18:433–440, 2004.

[67] T. Tautges, R. Meyers, K. Merkley, C. Stimpson, and C. Ernst. MOAB: a Mesh-Oriented database. SAND2004-1592, Sandia National Laboratories, Apr. 2004.

[68] T. Tautges et al. MOAB Web page. URL http://trac.mcs.anl.gov/projects/ITAPS/wiki/MOAB.

[69] C. Woodward et al. SUNDIALS Web page. URL https://computation.llnl.gov/casc/sundials.

C SWP4XS Workshop Participants

Aron Ahmadia, U.S. Army Corps of Engineers
Ross Bartlett, Oak Ridge National Laboratory
Ivan Bermejo-Moreno, Stanford University
David Bernholdt, Oak Ridge National Laboratory
Martin Berzins, University of Utah
Jed Brown, Argonne National Laboratory
Jeff Carver, University of Alabama
John R. Cary, Tech-X Corporation
Lisa C. Childers, Argonne National Laboratory
Bill Collins, Lawrence Berkeley National Laboratory
Mike Crowley, National Renewable Energy Laboratory
Eric Cyr, Sandia National Laboratories
Pedro Diniz, USC Information Sciences Institute
Anshu Dubey, Lawrence Berkeley National Laboratory
Kevin Fall, Carnegie Mellon University
Stuart Faulk, University of Oregon
Charles Ferenbaugh, Los Alamos National Laboratory
Al Geist, Oak Ridge National Laboratory
Mike Glass, Sandia National Laboratories
Bill Gropp, University of Illinois at Urbana-Champaign
Robert J.Harrison, Stony Brook University
Mike Heroux, Sandia National Laboratories
Judy Hill, Oak Ridge National Laboratory
Thuc Hoang, U.S. Department of Energy
Rich Hornung, Lawrence Livermore National Laboratory
Paul Hovland, Argonne National Laboratory
Hans Johansen, Lawrence Berkeley National Laboratory
Phil Jones, Los Alamos National Laboratory
Renu Joseph, U.S. Department of Energy
Balint Joo, Jefferson Laboratory
Dan Katz, National Science Foundation
Darren J. Kerbyson, Pacific Northwest National Laboratory
Matt Knepley, University of Chicago
Dorothy Koch, U.S. Department of Energy
Mike Kumbera, Lawrence Livermore National Laboratory
Mike Kuperberg, U.S. Department of Energy
Alex Larzelere, U.S. Department of Energy
Randall Laviolette, U.S. Department of Energy
Patricia Lee, U.S. Department of Energy
Steven Lee, U.S. Department of Energy
John Mandrekas, U.S. Department of Energy
Justin Marble, U.S. Department of Energy
Sally McFarlane, U.S. Department of Energy
Lois Curfman McInnes, Argonne National Laboratory
John Mellor-Crummey, Rice University
Mark C. Miller, Lawrence Livermore National Laboratory

David Moulton, Los Alamos National Laboratory
Thomas Ndousse-Fetter, U.S. Department of Energy
Ron Oldfield, Sandia National Laboratories
Lenny Oliker, Lawrence Berkeley National Laboratory
Karen Pao, U.S. Department of Energy
Roger Pawlowski, Sandia National Laboratories
Doug Post, Department of Defense
Jack Poulson, Georgia Institute of Technology
Sonia Sachs, U.S. Department of Energy
Andy Salinger, Sandia National Laboratories
Tim Scheibe, Pacific Northwest National Laboratory
Andrew Siegel, Argonne National Laboratory
Thomas Sterling, Indiana University
Ceren Susut-Bennett, U.S. Department of Energy
Bill Tang, Princeton University
Andy Terrel, Continuum Analytics, Inc.
Brian Van Straalen, University of California, Berkeley
Jeffrey Vetter, Oak Ridge National Laboratory
Larry Votta, Brincos, Inc.
Sam Williams, Lawrence Berkeley National Laboratory

D SWP4XS Workshop Position Papers

The following position papers are available at the workshop website,
http://www.orau.gov/swproductivity2014/papers.htm

- Roscoe Bartlett, Michael Heroux and James Willenbring, *Agile Lifecycles for Research-driven CSE Software*
- Roscoe Bartlett *Overview of Software Challenges in CSE*
- Roscoe Bartlett *Fortran Isolates the CSE Community*
- Iván Bermejo-Moreno, *Position Paper on Testing and V&V*
- David Bernholdt, *Software as "Instrumentation" for Computational Research*
- David Bernholdt, Al Geist and Barney Maccabe, *Resilience is a Software Engineering Issue*
- Jed Brown, Matthew Knepley, and Barry Smith *Run-time Extensibility: Anything Less is Unsustainable*
- Jed Brown, *Scalable Repository Workflows*
- Jeff Candy, Phil Snyder and Vincent Chan, *Whitepaper for Extreme Scale Science Panel*
- Jeffrey Carver and George Thiruvathukal, *Applying Appropriate Software Engineering to Exascale Software Development*
- John Cary, *New Software Technology and its Relation to Application Lifecycle*
- Eric Cyr and Roger Pawlowski, *Challenges for Component-based Multiphysics PDE Codes on Multicore Architectures*
- Anshu Dubey, *Preparing Mature Codes for Generations of Heterogeneity*
- Stuart Faulk, *Effective Software Engineering Approaches to Resolving Scientific Computing's Productivity Gridlock*
- Charles Ferenbaugh, *Software Engineering Issues in Moving Legacy Codes to Future Architectures*
- Ian Foster, *Dark Software: Addressing the Productivity Challenges of Extreme-scale Science On-ramps and Off-ramps*
- Robert Harrison, *Productivity and Performance for Extreme-scale Science*
- Michael Heroux, Improving CSE Software via Rigorous Research Expectations
- Paul Hovland, Barry Smith, Marc Snir, Lois Curfman McInnes and Boyana Norris, *Exposing and Expanding Compiler Technologies to Improve Software Productivity in Developing Mathematical Libraries and Simulation Codes*
- Phil Jones, *Software Productivity Challenges in Climate Modeling*
- Michael Kumbera, *Efficient HPC Software Development*
- John Mellor-Crummey, *Improving Software Productivity for Extreme-scale Systems with DSLs*

- Mark Miller, *A Scalable Mesh and Field Data Source that is both Virtual and Tunable*
- Ron Oldfield, Nathan Fabian, Kenneth Moreland and David Rogers, *Productivity Challenges for Integrating Simulation and Analysis*
- P. Sadayappan, *Enabling the Productive Development of Efficient Partitioned-address-space Parallel Applications from Global-address-space Abstractions*
- Andrew Salinger, *Component-Based Scientific Application Development*
- Andrew Salinger, *Software Engineering Best Practices*
- Andy Terrel, Chris Kees, Aron Ahmadia, Dag Sverre Seljebotn and Ondrej Certik, *State of Scientific Software Stacks*
- Andy Terrel and Matthew Turk, *HPC Communities of Practice*
- Brian Van Straalen, *Always be Profiling*
- Brian Van Straalen, *Have a Default Implementation of Everything*
- Venkatram Vishwanath, Tom Uram, Lisa Childers, Hal Finkel, Jeff Hammond, Kalyan Kumaran, Paul Messina and Michael Papka, *Toward Improved Scientific Software Productivity on Leadership Facilities: An Argonne Leadership Computing Facility View*
- Michael Wilde, Justin Wozniak, Timothy Armstrong, Daniel Katz and Ian Foster, *Productive Composition of Extreme-scale Applications using Implicitly Parallel Dataflow*
- Samuel Williams, Brian Van Straalen and Leonid Oliker, *Productive Extreme-Scale Computing via Common Abstract Machine Models, Programming Models, and Integrated Performance Modeling*
- Justin Wozniak, Timothy Armstrong, Dan Katz, Michael Wilde and Ian Foster, *Toward Computational Experiment Management via Multi-language Applications*

DOE ASCR Workshop on
Software Productivity for eXtreme-scale Science (SWP4XS)

http://www.orau.gov/swproductivity2014/
January 13-14, 2014
Hilton Washington DC/Rockville

Monday, January 13 (8:00 am - 5:15 pm)

8:00 am	Continental Breakfast
8:20 - 8:30	Welcome and Introduction
8:30 - 8:50	DOE ASCR and Office of Science Context Thomas Ndousse-Fetter (ASCR), Dorothy Koch (BER) and John Mandrekas (FES)
8:50 - 9:30	***Scientific Software Productivity Challenges at Extreme Scale*** Hans Johansen (LBNL), Lois Curfman McInnes (ANL), Mike Heroux (SNL) and Phil Jones (LANL)
9:30 - 9:45	***Results of Pre-Workshop Survey on Extreme-Scale Software Productivity*** Jeffrey Carver (Univ of Alabama)
9:45 - 10:00	Breakout Instructions, Q&A
10:00 - 10:30	Break
10:30 - 11:00	***Transforming computational science software research for extreme-scale computing: Patterns and best practices*** Introduce Topic 1 Questions, Lightning Presentations ● Charles Ferenbaugh (LANL), *Software Engineering Issues in Moving Legacy Codes to Future Architectures* ● Ron Oldfield (SNL), with Nathan Fabian, Kenneth Moreland and David Rogers, *Productivity Challenges for Integrating Simulation and Analysis* ● Lisa Childers (ANL), with Venkatram Vishwanath, Tom Uram, Hal Finkel, Jeff Hammond, Kalyan Kumaran, Paul Messina and Michael Papka, *Toward Improved Scientific Software Productivity on Leadership Facilities: An Argonne Leadership Computing Facility View* ● Ivan Bermejo-Moreno (Stanford Univ), *Position Paper on Testing and V&V* ● Anshu Dubey (LBNL), *Preparing Mature Codes for Generations of Heterogeneity*
11:00 - 12:15	Concurrent Breakout Sessions #1
12:15 - 1:30	Working Lunch Speaker: Douglass Post (DoD High Performance Computing Modernization Program and the Carnegie Mellon Software Engineering Institute) ***Addressing Application Software Productivity Challenges for Extreme-scale Computing***
1:30 - 2:00	Outbrief 1; Q&A
2:00 - 2:30	***Bridging the gap between domain science applications and computational science software: Research and development needs*** Introduce Topic 2 Questions, Lightning Presentations ● Al Geist (ORNL), with David Bernholdt and Barney Maccabe, *Resilience is a Software Engineering Issue* ● Roger Pawlowski (SNL), with Eric Cyr, *Challenges for Component-based Multiphysics PDE Codes on Multicore Architectures*

DOE ASCR Workshop on SWP4XS (continued)

- Paul Hovland (ANL), with Barry Smith, Marc Snir, Lois Curfman McInnes and Boyana Norris, *Exposing and Expanding Compiler Technologies to Improve Software Productivity in Developing Mathematical Libraries and Simulation Codes*
- John Mellor-Crummey (Rice Univ), *Improving Software Productivity for Extreme-scale Systems with DSLs*
- Samuel Williams (LBNL), with Brian Van Straalen and Leonid Oliker, *Productive Extreme-Scale Computing via Common Abstract Machine Models, Programming Models, and Integrated Performance Modeling*

2:30 - 3:45 Concurrent Breakout Sessions #2
3:45 - 4:15 Break
4:15 - 4:45 Outbrief 2; Q&A
4:45 - 5:15 Lightning Presentations: Software engineering and community issues

- Jeffrey Carver (Univ of Alabama), *Applying Appropriate Software Engineering to Exascale Software Development*
- Andy Terrel (Continuum Analytics), with Chris Kees, Aron Ahmadia, Dag Sverre Seljebotn and Ondrej Certik, *State of Scientific Software Stacks*; also Andy Terrel and Matthew Turk, *HPC Communities of Practice*
- Andrew Salinger (SNL), *Component-Based Application Development*; also *Software Engineering Best Practices*
- David Bernholdt (ORNL), *Software as "Instrumentation" for Computational Research*

5:15 pm Wrap-up, Adjourn for the day (dinner on your own)

Tuesday, January 14 (8:00 am - 1:00 pm)

8:00 am Continental Breakfast
8:30 - 8:45 Summary of Day 1, Review Agenda for Day 2
8:45 - 9:45 Panel Discussion and Q&A
Panelists: John Cary (Tech-X Corporation)
Bill Collins (LBNL)
Kevin Fall (Software Engineering Institute, Carnegie Mellon Univ)
Bill Gropp (Univ of Illinois at Urbana-Champaign)
Robert Harrison (Stonybrook Univ and BNL)
9:45 - 10:00 Break

10:00 - 11:30 Concurrent Breakout Sessions #3:
Computational science software productivity at extreme scale: Short-term/long-term priorities

11:30 - 12:00 Break
12:00 - 12:45 Outbrief 3; Q&A
12:45 - 1:00 Workshop wrap-up, review timeline, process and assignments for report
1:00 pm Workshop adjourns (box lunches to go)