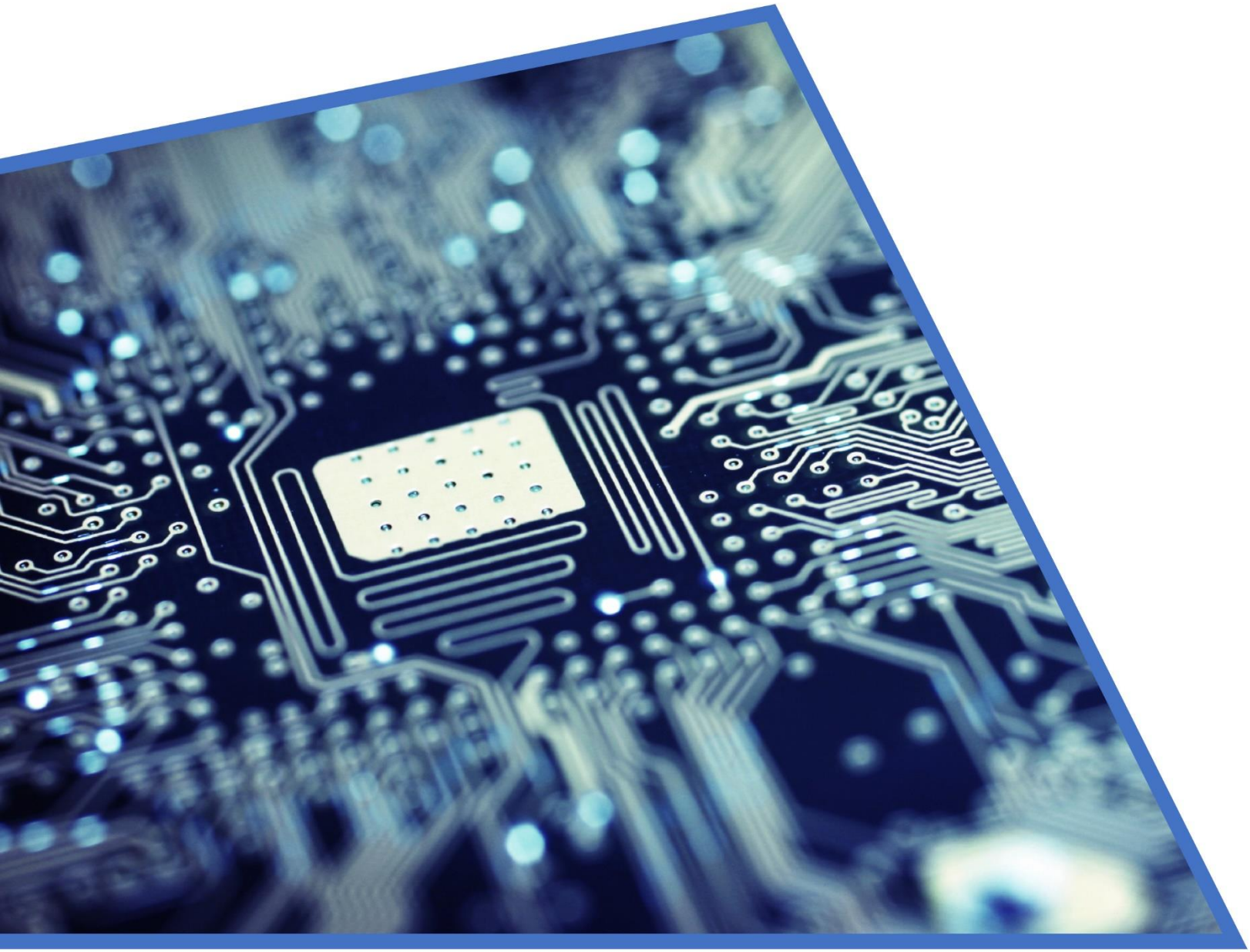
Extreme Heterogeneity 2018

PRODUCTIVE COMPUTATIONAL SCIENCE IN THE ERA OF EXTREME HETEROGENEITY



Report for

DOE ASCR Basic Research Needs Workshop on Extreme Heterogeneity

January 23–25, 2018

The artwork on the cover is used courtesy of Los Alamos National Laboratory.

Extreme Heterogeneity 2018:
Productive Computational Science in the Era of Extreme Heterogeneity

Report for
DOE ASCR Basic Research Needs Workshop on Extreme Heterogeneity

January 23–25, 2018

J.S. Vetter, R. Brightwell, M. Gokhale, P. McCormick, R. Ross, J. Shalf, K. Antypas, D. Donofrio, A. Dubey, T. Humble, C. Schuman, B. Van Essen, S. Yoo, A. Aiken, D. Bernholdt, S. Byna, K. Cameron, F. Cappello, B. Chapman, A. Chien, M. Hall, R. Hartman-Baker, Z. Lan, M. Lang, J. Leidel, S. Li, R. Lucas, J. Mellor-Crummey, P. Peltz, Jr., T. Peterka, M. Strout, and J. Wilke, "Extreme Heterogeneity 2018: DOE ASCR Basic Research Needs Workshop on Extreme Heterogeneity," US Department of Energy, Office of Science, Advanced Scientific Computing Research, 2018, doi:10.2172/1473756.

Workshop Organizers

Chair:	Jeffrey S. Vetter	Oak Ridge National Laboratory
ASCR Computer Science Lead:	Lucy Nowell	DOE Office of Advanced Scientific Computing Research
Organizing Committee:	Katie Antypas	Lawrence Berkeley National Laboratory
	Ron Brightwell	Sandia National Laboratories
	David Donofrio	Lawrence Berkeley National Laboratory
	Maya Gokhale	Lawrence Livermore National Laboratory
	Travis Humble	Oak Ridge National Laboratory
	Pat McCormick	Los Alamos National Laboratory
	Rob Ross	Argonne National Laboratory
	Catherine Schuman	Oak Ridge National Laboratory
	John Shalf	Lawrence Berkeley National Laboratory
	Brian Van Essen	Lawrence Livermore National Laboratory
	Shinjae Yoo	Brookhaven National Laboratory
Workshop Support:	Deneise Terry	Oak Ridge Associated Universities
	Jody Crisp	Oak Ridge Associated Universities

Breakout Group Leads

Programming Environments: Abstractions, Models, and Languages	Alex Aiken, Stanford Pat McCormick, Los Alamos National Laboratory
Data Management and I/O	Rob Ross, Argonne National Laboratory Suren Byna, Lawrence Livermore National Laboratory
Data Analytics and Workflows	Thomas Peterka, Argonne National Laboratory Shinjae Yoo, Brookhaven National Laboratory
OS/RM: global, composition, workflow	Ron Brightwell, Sandia National Laboratories
Software Development Methodologies	Sherry Li, Lawrence Berkeley National Laboratory David Bernholdt, Oak Ridge National Laboratory
Crosscut: Modeling and Simulation	Andrew Chien, Argonne National Laboratory David Donofrio, Lawrence Berkeley National Laboratory Maya Gokhale, Lawrence Livermore National Laboratory Zhiling Lan, Illinois Institute of Technology John Leidel, Tactical Computing Laboratories Jeremiah Wilke, Sandia National Laboratories
Programming Environments: Compilers, Libraries, and Runtimes	Barbara Chapman, Brookhaven National Laboratory Michelle Strout, University of Arizona
System Management, Admin, Job Scheduling	Paul Peltz Jr., Los Alamos National Laboratory Rebecca Hartman-Baker, Lawrence Berkeley National Laboratory/ National Energy Research Scientific Computing Center
Crosscut: productivity, composition, interoperability	Robert Lucas, University of Southern California/ISI
OS/RM: local, prog env support	Michael Lang, Los Alamos National Laboratory
Crosscut: Portability, code reuse, performance portability	Sherry Li, Lawrence Berkeley National Laboratory Anshu Dubey, Argonne National Laboratory
Programming Environments: Debugging and Correctness, autotuning, specialization	Mary Hall, University of Utah John Mellor-Crummey, Rice University
Crosscut: resilience, power	Franck Cappello, Argonne National Laboratory Kirk Cameron, Virginia Tech University

Contents

List of Figures

Abbreviations, Acronyms, and Initialisms

1D	one-dimensional
ARO	autonomous resource optimization
API	application programming interface
ASCR	Office of Advanced Scientific Computing Research
AW	autonomous workflow
CMOS	Complementary metal–oxide–semiconductor
CPU	central processing unit
CUDA	compute unified device architecture
DOE	Department of energy
DRAM	dynamic RAM
DSL	domain-specific language
ECP	Exascale Computing Project
EH	extreme homogeneity, extremely homogeneous
FCGCS	fast coarse-grain context swapping
FPGA	field-programmable gate array
FSC	Factual Status Document
GPU	graphics processing unit
GPGPU	general purpose graphics processing unit
HPC	high performance computing
I/O	in/out
IT	information technology
MD	moecular dynamics
ModSim	modeling and simulation
MRAM	magnetic RAM
NUMA	nonuniform memory access
OLCF	Oak Ridge Leadership Computing Facility
OS	operating system
PCI	peripheral component interconnect
PRD	Priority Research Direction
RAM	random access memory
RRAM	resistive RAM
SCM	storage class memory
SMT	simultaneous multithreading

Executive Summary

Providing a software environment that can overcome the complexities of changes in how future supercomputers will be designed plays a key role in improving the nation's rate of scientific discovery and innovation. In the past three decades, advances in computer technology have allowed the performance and functionality of processors to double every 2 years. This trend, known as Moore's Law, has enabled both computational and experimental science to leverage the so far unending growth of the broad computing industry with very little change to the supporting software environment. But as computer chip manufacturing techniques reach the limits of the atomic scale, this era of predictable improvements is ending. This shift will have a significant impact on the design of high-performance computers, as well as the established software infrastructure required to effectively utilize the nation's Leadership Computing Facilities.

Computer vendors are pursuing systems built from combinations of different types of processors to improve capabilities, boost performance, and meet energy efficiency goals. Some of the most current supercomputers do not rely on a single type of processor but instead have added computational accelerators to meet the growing demands of increasingly complex computational workloads. According to studies from the US Department of Energy (DOE) Office of Science Advanced Scientific Computing Research program, several types of special-purpose accelerated processing units are currently under development and will play a huge role in the future of computer architectures. It is also likely that these processors will be augmented with diverse types of memory and data storage capabilities. These significant changes are driven by extreme growth in the data-centric machine learning and artificial intelligence marketplaces that far exceed the revenues represented by high-performance computing for computational and experimental science. In the 2025–2030 time frame, external economic drivers and design diversity will result in systems built from a custom aggregation of components; and the difficulty and complexity of developing scientific software will increase. This fundamental change in computer architecture design has been deemed the era of "extreme heterogeneity."

The 2018 Basic Research Needs Workshop on Extreme Heterogeneity identified five Priority Research Directions for realizing the capabilities needed to address the challenges posed in this era of rapid technological change. They are

1. Maintaining and Improving Programmer Productivity

    - Flexible, expressive, programming models and languages
    - Intelligent, domain-aware compilers and software development tools
    - Composition of disparate software component content

2. Managing System Resources Intelligently

    - Automated methods using introspection and machine learning
    - Optimize for performance, energy efficiency, and availability

3. Modeling and Predicting Performance

    - Evaluate impact of potential system designs and application mappings
    - Model-automated optimization of applications

4. Enabling Reproducible Science Despite Diverse Processors and Non-Determinism

    - Methods for validation on non-deterministic architectures
    - Detection and mitigation of pervasive faults and errors

5. Facilitating Data Management, Analytics, and Workflows

    - Mapping a science workflow to heterogeneous hardware and software services
    - Adapting workflows and services through machine learning approaches

This report captures and expands the outcomes of this workshop. In the context of extreme heterogeneity, it defines basic research needs and opportunities in computer science research to develop smart and trainable operating and runtime systems, programming environments, and predictive tools that will make future systems easier to adapt to scientists' computing needs and easier for facilities to deploy securely.

1. Introduction

Fundamental trends in computer architecture predict that nearly all aspects of future high-performance computing (HPC) architectures will have many more diverse components than past systems, leading toward a period of extreme heterogeneity. The HPC community has already seen evidence of this trend [1, 2, 7, 9, 12, 15, 70]. In 2009, the drive to deploy more energy-efficient computers led the Oak Ridge Leadership Computing Facility (OLCF) to propose a system upgrade, composed of CPUs (central processing units) coupled with GPUs (graphics processing units), that firmly established an era of heterogeneous HPC across the Department of Energy (DOE). Continuing this trend, both recently deployed CORAL systems, Summit and Sierra—selected by OLCF and Lawrence Livermore National Laboratory, respectively—are composed of CPUs coupled with multiple GPUs and three types of memory. This trend is also manifested in the most recent TOP500 list: heterogeneous accelerators are used in 110 TOP500 systems and the majority of the TOP10 systems. Furthermore, a recent analysis of vendors' current architectural roadmaps is consistent with the extreme homogeneity that the Advanced Scientific Computing Research (ASCR) program is seeing in its computing upgrades. It indicates that future computers will be more complex and will be composed of a variety of processing units and specialized accelerators supported by open interconnects and deep memory hierarchies. Looking forward, we expect even more diverse accelerators for paradigms like machine learning, neuromorphic computing, and quantum computing.

Several characteristics define what we have termed *extreme heterogeneity*, each foreshadowed by technological trends (see Section 2) that represent significant challenges and have negative impacts on achieving productive computational science. In January 2018, ASCR convened a Basic Research Needs Workshop on Extreme Heterogeneity in HPC. The purpose of this workshop was to identify Priority Research Directions (PRDs) for ASCR in providing a smart software stack that includes techniques such as deep learning to make sure that future computers composed of a variety of complex processors, new interconnects, and deep memory hierarchies can be used productively by a broad community of scientists. Achieving this productivity is critical to both maintaining and growing DOE's leadership in scientific applications and improving the nation's overall rate of scientific discovery and innovation.

This report captures the outcomes of this workshop. In the context of extreme heterogeneity, it defines basic research needs and opportunities in computer science research to develop smart and trainable operating and runtime systems, programming environments, and predictive tools that enable future systems easier to adapt to scientists' computing needs and easier for facilities to deploy.

In this regard, significant computer science challenges remain as barriers to developing this smart software stack. The primary aim of the workshop was to identify the new algorithms and software tools needed from basic research in computer science to enable ASCR's supercomputing facilities to support future scientific and technological advances in addressing the DOE Office of Science's grand challenge problems. The resulting research directions were identified by spanning existing and next-generation system architectures, considering novel technologies that are likely in the post–Moore's Law era, and the promising tools and techniques that will be essential to the efficient and productive utilization of such architectures.

1.1 Workshop Organization

The workshop structure and content emanated from the initial topics outlined in the charge letter (see Appendix A) and the Summary Report from a June 2017 Summit on Extreme Heterogeneity. The organizing committee prepared a Factual Status Document (FSD) to provide participants with a shared understanding of the current state of affairs in several areas:

1. Programming Environments
2. Operating System and Resource Management
3. Tools for Systems Management and Administration
4. Productivity and Performance Metrics and Tools
5. Software Development Methodologies
6. Data Management and I/O
7. Data Analytics and In Situ Workflow Management
8. Modeling and Simulation

In parallel with developing the FSD, a call for position papers was issued that received more than 100 responses. These papers provided additional input for the FSD and broader insight into the thinking of the computer science and HPC communities about the topics to be addressed during the workshop. This call also provided a basis for broadening the list of potential workshop invitees. The final participants were selected by the Organizing Committee by direct invitation and through a *call for position papers* (see Appendix E), which was widely circulated throughout the community. The final list of registered workshop participants is contained in Appendix D.

Logistically, the workshop was organized into a series of plenary speakers, breakout groups, and de-briefings (see Appendix C). The breakout topic areas (see Figure 1) were organized according to the expected interest and attendance. For each breakout group, a moderator and at least one scribe were selected to create a document of shared notes during each session. Section 3 captures the distilled findings from these breakout group discussions.

1.1.1 Forced Transformation to a Virtual Workshop

Originally, the workshop was planned as a face-to-face workshop in Washington, DC. However, because of a last-minute shutdown of the federal government three days before the workshop, the organizing committee transformed the face-to-face workshop into an online, virtual workshop. The agenda therefore was adjusted to allow participation across time zones. In less than 24 hours, the organizers arranged mailing lists, video conferencing facilities, and Google Docs for the plenaries and all breakout groups. The virtual meeting format was very effective and increased participation, but it did not provide an opportunity to build a new community with spontaneous discussions to foster connections among new researchers.

BOG ID	Breakout Group Topic	Tues PM	Wed PM-1	Wed PM-2	Thurs AM	Thurs PM	Relevant OC Member
1	Programming Environments: Abstractions, Models, and Languages	Aiken; McCormick			McCormick		McCormick
2	Data Management and I/O	Ross; Byna				Ross	Ross
3	Data Analytics and Workflows	Tom P.; Yoo		Christine S.; Bethel			Yoo
4	OS/RM: Global, Composition, Workflow		Brightwell				Brightwell
5	Software Development Methodologies		Li; Bernholdt				Li
6	Crosscut: Modeling and Simulation		Chien; Donofrio; Leidel		Wilke; Lan; Gokhale		Gokhale
7	Programming Environments: Compilers, Libraries, and Runtimes			Strout; Chapman			McCormick
8	System Management, Admin, Job Scheduling			Peltz; Hartman-Baker			Antypass
9	Crosscut: Productivity, Composition, Interoperability			Lucas			Humble
10	OS/RM: Local, Prog Env, Support				Lang		Brightwell
11	Crosscut: Portability, Code Reuse, Performance Portability				Dubey; Li		Humble
12	Programming Environments: Debugging and Correctness, Autotuning, Specialization					Hall; Mellor-Crummey	McCormick
13	Crosscut: Resilience, Power					Cappello; Cameron	Shalf

Figure 1. Breakout Group grid.

1.2 Priority Research Directions

In a closing session of the workshop and during the following months, the organizing committee and breakout session representatives merged, collated, and prioritized the findings of the workshop, culminating in five PRDs:

1. Maintaining and Improving Programmer Productivity

    - Flexible, expressive, programming models and languages
    - Intelligent, domain-aware compilers and software development tools
    - Composition of disparate software component content

2. Managing System Resources Intelligently

    - Automated methods using introspection and machine learning
    - Optimize for performance, energy efficiency, and availability

3. Modeling and Predicting Performance

    - Evaluate impact of potential system designs and application mappings
    - Model-automated optimization of applications

4. Enabling Reproducible Science Despite Diverse Processors and Non-Determinism

    - Methods for validation on non-deterministic architectures
    - Detection and mitigation of pervasive faults and errors

5. Facilitating Data Management, Analytics, and Workflows

    - Mapping a science workflow to heterogeneous hardware and software services
    - Adapting workflows and services through machine learning approaches

1.3 Report Outline

The remainder of this report is organized as follows. First, Section 2 outlines the motivating technical trends and respective consequences that lead to *extreme heterogeneity* in computing and HPC. Next, Section 3 captures the prioritized findings and discussions of the workshop breakout groups. Then, Section 4 presents the distilled PRDs. Finally, Section 5 concludes. Additionally, six appendices capture the (a) charge letter, (b) organization committee, (c) agenda, (d) participants, (e) call for position papers, and (f) related activities.

2. Background and Motivating Trends

Society has come to depend on the rapid, predictable, and affordable scaling of computing performance for consumer electronics; the rise of "big data" and data centers (e.g., Google, Facebook, Amazon, Microsoft); leadership computing for scientific discovery; and national security. This scaling is due to Moore's Law and Dennard scaling. Moore's law is a techno-economic model that has predicted that the information technology (IT) industry would nearly double the performance and functionality of digital electronics roughly every 2 years within a fixed cost, power, and area [46]. This expectation has led to a relatively stable ecosystem (e.g., electronic design automation tools, compilers, simulators, emulators) built around general-purpose processor technologies such as the x86, ARM, and Power instruction set architectures. However, within a decade, the technological underpinnings for the process Gordon Moore described will come to an end as lithography drills down to atomic scale [23, 36, 38, 40, 68]. At that point, it will be feasible to create lithographically produced devices with characteristic dimensions in the 3–5 nm range. This range corresponds to a dozen or fewer silicon atoms across critical device features and will therefore be a practical limit for controlling charge in a classical sense. The classical technological driver that has underpinned Moore's law for the past 50 years is already failing and is anticipated to flatten by 2025. Absent a new transistor technology to replace CMOS (complementary metal-oxide semiconductor), the primary opportunity for continued performance improvement for digital electronics and HPC is to make more effective use of transistors through more efficient architectures, architecture specialization manifested in the form of heterogeneous computing, and programming systems that better control data movement than those available today.

In anticipation of this challenge to traditional technology scaling, DOE produced a preliminary evaluation and report on technology options for computing beyond Moore's Law [60] that identifies more than a dozen emerging novel technologies that might become components of future supercomputers. In addition, a wide variety of potential computer architectures are under consideration, including the likelihood that vendors will use multiple novel technologies in a single heterogeneous system. We already see signs of this in the planned system architectures for the DOE CORAL and CORAL2 acquisitions. Recent communications with vendors through the 2017 Exascale Computing Project (ECP) request-for-information responses suggest that exascale platforms and beyond will become increasingly heterogeneous environments. These long-term trends in the underlying hardware technology (driven by the physics) are creating daunting challenges to maintaining the productivity and continued performance scaling of HPC codes on future systems. Furthermore, the entire computing community have recognized these trends and have reacted with various initiatives (see Appendix E:).

We begin with a discussion of motivating trends and consequences to understand how computer architects are reacting to those trends. These trends are

- The end of exponential clock frequency scaling (*which leads to exponentially increasing parallelism*)
- The end of lithographic scaling as the primary driver for technology improvements (*which leads to the emergence of increasingly heterogeneous hardware accelerators to continue performance improvements*)
- Moving data operands that cost more than the compute operation performed on them (*which causes our programming models to no longer reflect the actual costs for critical resources*)
- The increasing diversity of emerging memory and storage technologies (*which upends common storage and memory paradigms after many decades of relative stability*)

- Aggressive power management and resource contention resulting in heterogeneous execution rates (*which results in performance heterogeneity that is antithetical to current programming models)*

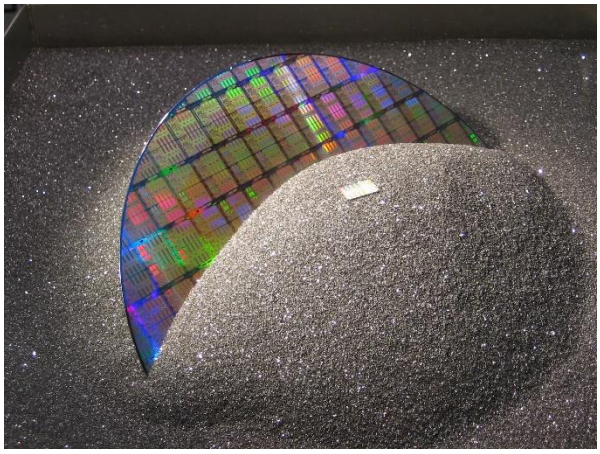
- Increasingly diverse user requirements (*which result in a rapid move toward complex multidisciplinary workflows*)

2.1 Exponentially Increasing Parallelism

Trend: End of exponential clock frequency scaling. This trend has been covered in detail by the DOE ECP, but we recount it here for clarity so that it is clearly differentiated from the tapering of Moore's law (which captures a different phenomenon). The year 2004 marked the proximal end of Dennard scaling (which had enabled decades of exponential improvements in clock frequencies) because chip manufacturers could no longer reduce voltages at historical rates [39]. Other gains in energy efficiency were still possible; for example, smaller transistors with lower capacitance consume less energy, but those gains would be dwarfed by leakage currents. The inability to further reduce voltages meant, however, that clock rates could no longer be increased within the same power budget. With the end of voltage scaling, single-processing core performance no longer improved with each generation, but performance could be improved by exponentially increasing parallelism. This result upends many assumptions that underpin the software environment.

Consequence: Exponentially increasing parallelism. This trend is an important development in that programmers outside the small cadre of those with experience in parallel computing must now contend with the challenge of making codes run effectively in parallel. Parallelism has become everyone's problem and will require deep rethinking of the commercial software and algorithm infrastructure. This increase in parallelism is one of the central pillars of the ECP, but there are other emerging changes relevant to increased parallelism that are well beyond the scope of ECP.

2.2 Heterogeneous Hardware Acceleration



Trend: End of lithography scaling. As if the end of Dennard scaling were not difficult enough, we are witnessing a tapering of the classical source of technology scaling improvements commonly known as Moore's law, which is expected to flatten within the next decade (soon after delivery of the first generation of exascale systems). Moore's law is not really a law but more of an "observation" that grew from a techno-economic model first postulated in 1965 by Gordon Moore (cofounder and chairman emeritus of Intel Corporation). The observation was deceptively simple: the nascent semiconductor industry was finding a way to double the number of transistors on an integrated circuit every 2 years, thereby doubling the processing power, efficiency, and functionality of computers and other digital devices every 2 years. Based on improvements in the pace of adding transistors, Moore subsequently updated his law to "every 18 months."

Within a decade, however, the technological underpinnings for the process Gordon Moore described will come to an end as several laws of nature come into play. Basically, the spacing of circuits on an integrated circuit is reaching the scale of individual atoms, which constitutes a fundamental limit to

further improvements. By 2025 it will be feasible to create devices with characteristic dimensions of 3–5 nm. This range corresponds to a dozen or fewer silicon atoms across critical device features and will therefore be a practical limit for controlling electric charge in a classical sense. Thus, the classical technological driver that has underpinned Moore's law for the past 50 years is already failing and is anticipated to flatten by 2025, as shown in Figure 2.

Consequence: Many forms of heterogeneous accelerators (no longer just GPU accelerators). The tapering of Moore's law improvements in transistor density and performance will have a more profound effect on the programming environment. Absent any miraculous new transistor or other device to enable continued technology scaling, the only tool left to a computer architect for extracting continued performance improvements is to use transistors more efficiently by specializing the architecture to the target scientific problem(s), as projected in Figure 4. Overall, There is strong consensus that the tapering of Moore's law will lead to a broader range of accelerators or specialization technologies than we have seen in the past three decades.

Industry is already moving forward with diverse acceleration in the artificial intelligence (AI) and machine learning markets (e.g., Google TPU [31], Nervana's AI architecture [35], Facebook's "Big Sur" [24]) and other forms of compute-in-network acceleration for mega-data centers (e.g., Microsoft's field-programmable gate array [FPGA] Configurable Cloud [8], Project Catapult for FPGA-accelerated search [54]). There have also been demonstrated successes in creating science-targeted accelerators—such as D.E. Shaw's Anton, which accelerates molecular dynamics (MD) simulations nearly 180× over contemporary HPC systems [62], and the GRAPE series of specialized accelerators for cosmology and MD [48]. Even more radical architectures are being conceived for the next 5 years. These options include coarse grain reconfigurable architectures, such as Xilinx Project Everest, and analog computing, such as Mythic's deep learning chip that uses flash transistors to capture weights of a neural net. A recent International Symposium on Computer Architecture workshop on the future of computing research beyond 2030 (*http://arch2030.cs.washington.edu/*) concluded that heterogeneity and diversity of architecture are nearly inevitable given current architecture trends. This trend toward extreme homogeneity (EH) in compute resources is already well under way.

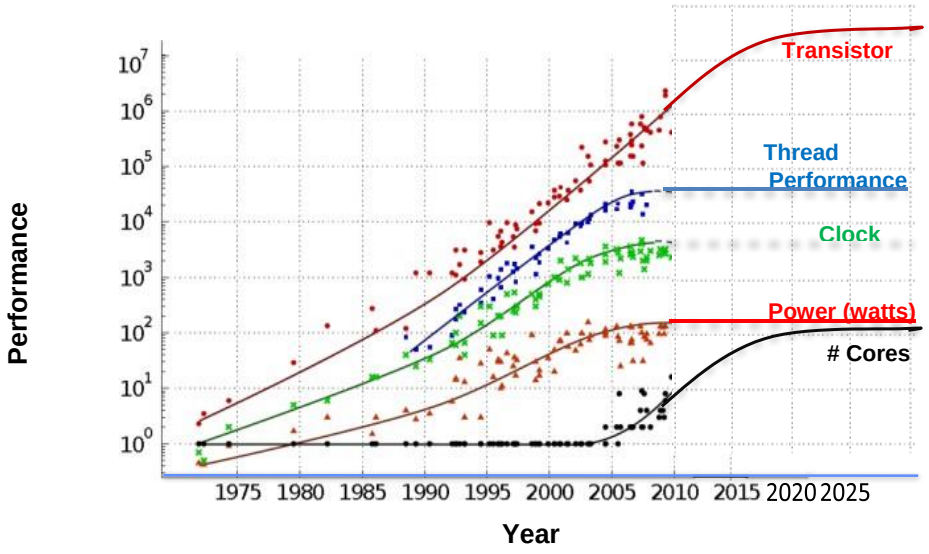


Figure 2. As we approach atomic scale, the benefits of lithography improvements to the performance and energy efficiency of silicon digital electronics are expected to taper. *Original figure courtesy of Kunle Olukotun, Lance Hammond, Herb Sutter, and Burton Smith. Figure extrapolations extended in 2016 by J. Shalf*

Apart from the adoption of GPGPU (general purpose graphics processing unit) technology, scientific computing has largely been absent from this revolution and runs the risk of being left behind if it does not accelerate its assimilation of these technologies. Furthermore, adoption of GPGPUs has already created enormous challenges for programmer productivity in the scientific community, and we have made only incremental progress toward performance-portable programming environments. If our programming paradigm is already challenged by a single kind of accelerator (the GPGPU), then we are especially ill prepared for a future with many kinds of heterogeneous accelerators. This will require a comprehensive rethinking of the software stack to develop more productive programming environments for the future.

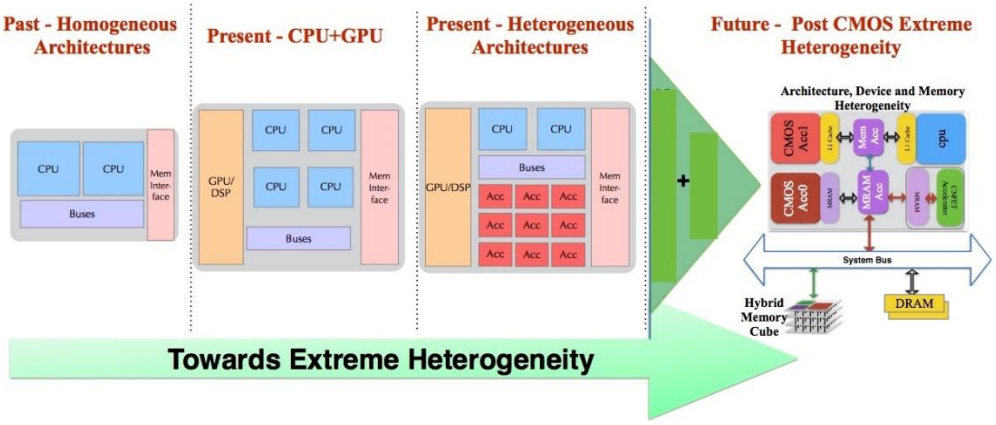


Figure 3. The consequence of Moore's law tapering is an increased reliance on heterogeneous hardware acceleration to continue performance improvements. *Original figure (a) courtesy of Dilip Vasudevan.*

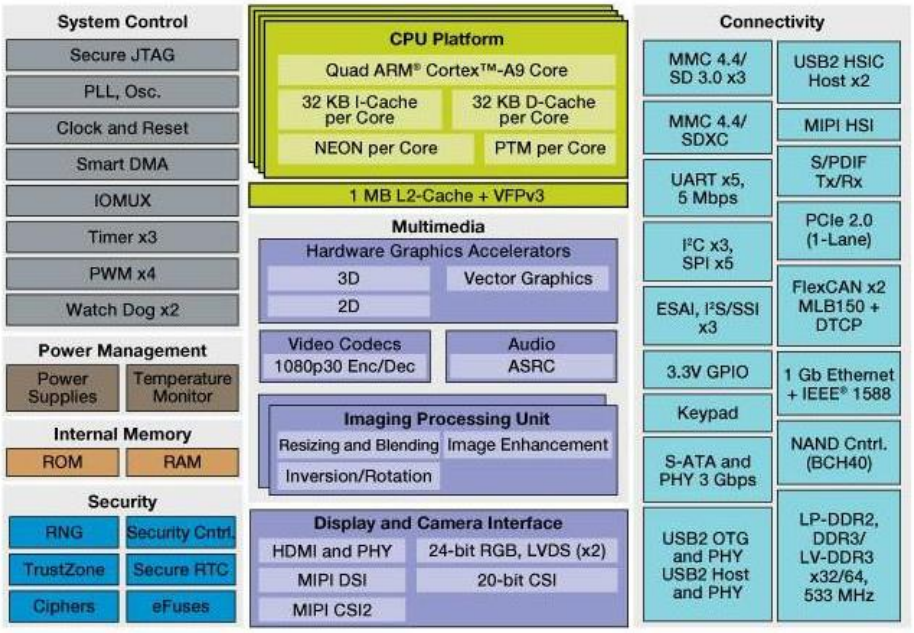


Figure 4. Example of contemporary cell phone heterogeneous processor. During the workshop, Bob Colwell (former Intel architect and Defense Advanced Research Projects Agency Microsystem Technology Office director) presented the diagram of accelerators on a typical cell phone system on chip on the bottom to make the point that this heterogeneity trend is already under way.

2.3 Programming Models No Longer Reflect Actual Costs for Critical Resources

Trend: Moving data operands costs more than the compute operation performed on them. Since the loss of Dennard scaling, a new technology scaling regime has emerged. According to the laws of electrical resistance and capacitance, the intrinsic energy efficiency of a fixed-length wire does not improve appreciably as it shrinks in size with Moore's law improvements in lithography [42, 43], as shown in Figure 5. In contrast, the power consumption of transistors continues to decrease as their gate size (and hence capacitance) decreases. Since the energy efficiency of transistors is improving as sizes shrink, and the energy efficiency of wires is not improving, we have come to a point where the energy needed to move data exceeds the energy used to perform the operation on those data, as shown in Figure 5. This leads to EH in the cost of accessing data because the costs to move data are strongly distance dependent. Furthermore, although computational performance has continued to increase, the number of pins per chip has not tended to improve at similar rates [21]. This leads to bandwidth contention, which leads to additional performance heterogeneity.

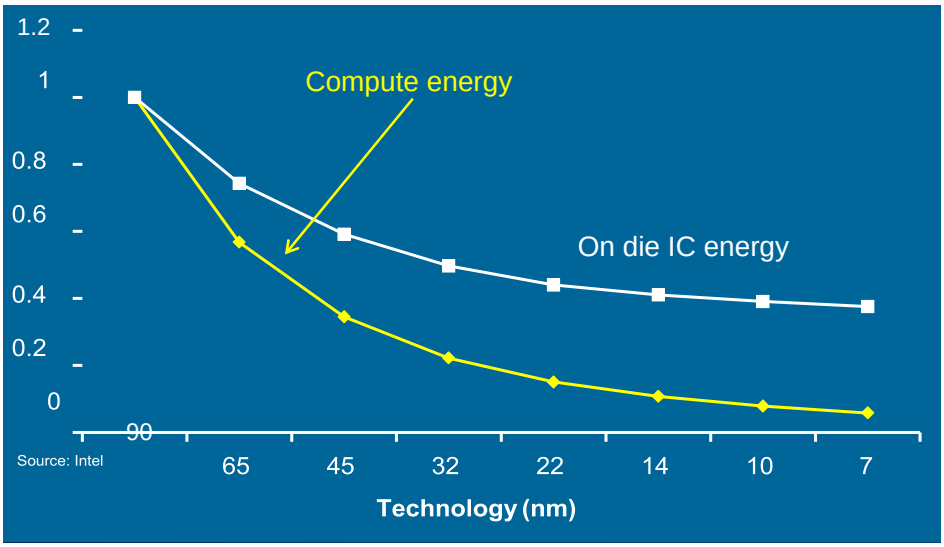
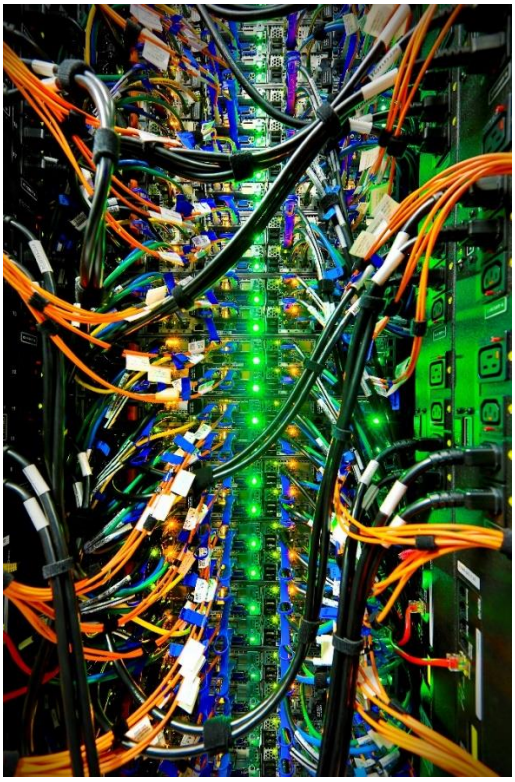


Figure 5. The energy efficiency of compute operations continues to improve at a slow pace, but the energy needed in terms of picojoules per bit per millimeter of data movement is not improving with each technology generation. The result is that data movement now consumes more on-chip energy than the compute operations performed on the data after it is moved. *Original figure courtesy of Shekhar Borkar (formerly at Intel, now at Qualcomm).*

Consequence: More heterogeneity in data movement and nonuniform memory access (NUMA) effects. Data locality and bandwidth constraints have long been concerns for application development on supercomputers, but recent architecture trends have exacerbated these challenges to the point that they can no longer be accommodated with existing methods such as loop blocking or compiler techniques. Future performance and energy efficiency improvements will require more-fundamental changes to hardware architectures. The most significant consequence of this assertion is the impact on scientific applications that run on current HPC systems, many of which codify years of scientific domain knowledge and refinements for contemporary computer systems. To adapt to computing architectures beyond 2025, developers must be able to reason about new hardware and determine what programming models and algorithms will provide the best blend of performance and energy efficiency into the future. Even our theory of complexity for numerical methods is based on counting the number of floating point operations, which fails to account for the order of complexity of compulsory data movement required by the algorithm.

Ultimately, our theories about algorithmic complexity are out of step with the underlying physics and cost model for modern computation. Future systems will express more levels of hierarchy than we are accustomed to in our existing programming models. Not only are there more levels of hierarchy, but it is also likely that the topology of communication will become important to optimize. Programmers are already facing NUMA performance challenges within the node, but future systems will see *increasing* NUMA effects between cores within an individual chip die in the future [11, 22, 47, 58, 63, 72, 74]. It will become important to optimize for the topology of communication; but current programming models do not express information needed for such optimizations, and current scheduling systems and runtimes are not well equipped to exploit such information were it available. Overall, our current programming methodologies are ill-equipped to accommodate changes to the underlying abstract machine model, which would break our current programming systems. This document outlines the current state of the art in data locality management in modern programming systems [13,19] and identifies numerous opportunities to greatly improve automation in these areas [65,66].



2.4 Storage and Memory Paradigms Are Being Upended after Decades of Relative Stability

Trend: Emerging memory technologies and a stall in disk performance improvements lead to disruptive changes in the storage environment. Tightly coupled with the issues discussed in Section 2.3 is the issue that new memory technologies are making their debuts in manufactured chips [67]. Prime examples are resistive RAM (RRAM), improved flash, and magnetic RAM (MRAM). These memories not only offer lower energy for some accesses but also are nonvolatile. This capability promises to enable architectural solutions to problems such as dark silicon, which might necessitate powering down sections of the chip. In addition, RRAM and MRAM, combined with other mature memory technologies such as STT-RAM, challenge the traditional view of memory hierarchy wherein DRAM is the last level of main memory. Emerging memory packaging methods enable processing-in-memory, which can mitigate the cost of data movement but brings many challenges in terms of complex and heterogeneous memory hierarchies.

A challenge is that these memories have diverse energy and latency costs to read and write, as well as manufacturing cost per bit and wear-out issues, making it increasingly important to select the right type of memory for the proper level of the memory hierarchy and application workload.

Consequence: Disruptive changes to memory and storage paradigms that cannot be hidden by current programming interfaces. Disruptive advances in these byte-addressable storage class memories (SCMs) blur the familiar dichotomy of low-latency, low-capacity volatile main memory and extremely high-latency, block-oriented storage. As the density of SCM devices has increased and wear characteristics (the number of times a location can be rewritten before it wears out) have

improved, it becomes feasible to include SCM in the memory hierarchy as either extended memory, persistent object store, or intermediate cache for traditional storage tiers. The unique characteristics of SCMs offer new opportunities but fully exploiting their capabilities will likely require disruptive innovation in the entire stack [45]—including hardware (e.g., enhancements to the instruction set to flush processor state to persistent memory or region-based coherency schemes for scalable management of tens of terabytes of memory), system software (operating system [OS] file system redesign to streamline low-latency storage), runtime libraries, and even application level.

We need to study how each of these technologies affects architecture and programming models, and thus at which level of the memory hierarchy each technology is best suited. This analysis should include a study of how unique capabilities of new memories (such as nonvolatility) can be used to seamlessly power down portions of future large-scale chips or to avoid frequent access to external storage to facilitate big-data applications, as well as our traditional simulation/modeling workloads.

2.5 Performance Heterogeneity

Trend: Heterogeneous execution rates from contention and aggressive power management. Emerging adaptive algorithms, low-voltage operation, and clock-speed throttling challenge current assumptions of uniformity [49, 59]. Because the most energy-efficient operation is the one you do not perform, there is increased interest in using sparse, adaptive, and irregular algorithms to apply computation only where it is required and to reduce memory requirements. Even for systems with homogeneous computation on homogeneous cores, new fine-grained power management makes homogeneous cores look heterogeneous. For example, thermal throttling on leading-edge server chips such as the Knights Landing and Haswell processors enables the core to opportunistically sprint to a higher clock frequency until it becomes too hot; but the implementation cannot guarantee a deterministic clock rate because chips heat at different rates. In the future, nonuniformities in process technology and for ultralow-power logic will create nonuniform operating characteristics for cores on a chip multiprocessor [32]. Fault resilience will also introduce inhomogeneity in execution rates, as even hardware error correction is not instantaneous; and software-based resilience will introduce even larger performance heterogeneity. Finally, adaptive routing in system-scale interconnects—which enables the popular dragonfly interconnect topology capable of rapidly shifting bandwidth from bisection to point-to-point bandwidth—also has a consequence of creating much variability in message arrival times as the routing dynamically adapts to ephemeral contention events [73]. Essentially, even homogeneous hardware will look increasingly heterogeneous in future technology generations.

Consequence: Extreme variability and heterogeneity in execution rates. We have evolved a parallel computing infrastructure that is optimized for bulk-synchronous execution models that implicitly assume that every processing element is identical and operates at the same performance. The chip's thermal condition is a function of all the running cores. Whatever is running on the other cores may directly impact the performance of a given core. In principle, you can manage those other cores so as to reserve the thermal margin needed to let a given core run at top speed, but then those other cores are slower than advertised. The core message is that (1) increasingly in the future, one must manage power in real time in conjunction with the performance requirements to hit the desired throughput; and (2) our tools, techniques, and methods for executing this combined power/performance real-time control task are

rudimentary and nowhere close to what will be needed. We can no longer depend on performance homogeneity, which presents an existential challenge to bulk-synchronous execution models. This limitation has renewed interest in alternative execution models that are better able to accommodate extreme performance heterogeneity [5,6,14,26,49,50].

2.6 Complex and heterogeneous workflows

Trend: User Requirements Are Moving Rapidly toward Complex Multidisciplinary Workflows. From 2015 to 2017, ASCR facilities commissioned a series of exascale requirements workshops for each of the six DOE Office of Science offices [28]. The goal of the workshops was to gather user requirements for the broad exascale ecosystem, which included not only compute and storage requirements but also software; programming model; workflow; algorithmic; and data transfer, movement, analysis, curation, and sharing requirements. A cross-cut report was generated that highlighted the common requirements across all offices. The report describes the increasingly complex and heterogeneous nature of workflows as scientists' computational tasks vary widely in computational complexity and have a growing range of workflow needs. The requirements reviews highlighted a need to support heterogeneous workflows and manage heterogeneous systems.

Workflows in both simulation and analysis are becoming more complex, and they need to be accommodated on HPC systems. Workflow requirements have evolved to include scalable data processing, data analysis, machine learning, and discrete algorithms. Multiscale and multiphysics simulations have become increasingly crucial for reducing and understanding the large-scale data that will be produced by leading-edge supercomputing systems. This complexity is often related to needs for data movement, including over wide-area networks.

Consequence: Next-generation systems with heterogeneous elements will need to accommodate complex workflows. These user requirements serve as both a challenge and an opportunity. Heterogeneous architectures unlock the possibility of mapping different parts of a workflow onto elements most appropriate for that application component. However, complexity will need to be managed for this possibility to be achieved. Scheduling and allocation policies are needed to support workflow needs, especially data analysis requirements at experimental facilities: real-time, pseudo-real time, co-scheduling, variable job requirements, and allocations based on other resources such as disk and memory. In general, the requirements reviews highlighted the need for improved developer productivity and the ability to package the artifacts of DOE research for broad reuse. There is a recognized need for community planning, policies, processes to improve software quality, interoperability, testing, and deployment within the DOE HPC ecosystem.

2.7 Bottom Line

If we, as a community, ignore these important trends and do nothing, DOE's mission applications, which serve as an important leg of scientific discovery (along with theory, experimentation, and data analytics), will be consigned to a future with *no* further improvements in computing performance or to unsustainable software maintenance costs where each application begins afresh on each new architecture.

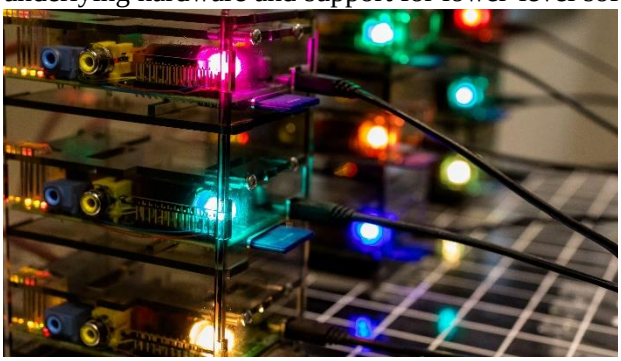
3. Workshop Findings

This section summarizes the findings of the workshop discussions, collated around the various breakout topics given in Figure 1.

3.1 Programming Environments

Programming environments play a critical role in successfully enabling the effective and efficient utilization of any computing resource for the design, implementation, evaluation, and optimization of mission-critical applications and the supporting layers of the software stack (e.g., data analytics, machine learning, math libraries, tools, runtime systems and workflow systems). In HPC, where performance and scalability are critical, the programming environment must provide a delicate balance of abstraction and efficiency.

In considering the impact of EH, these capabilities all serve to bridge the gap between the nuances of the underlying hardware and support for lower-level software infrastructure to provide application developers with abstractions that allow for the expression of algorithms, data structures, and the mapping of both data and computations to the set of available resources on a given platform. Importantly, the design and implementation choices for these portions of the environment have a direct and lasting impact on flexibility and overall, long-term, developer productivity.



The overall environment consists of several components that cover many different topic areas that have a broad impact on flexibility, expressiveness, and productivity. This scope impacts not only application developers but also the entire breadth of the software stack. Although it is common practice for the individual components of the programming environment to be independent research topics, the complexity and diversity of the challenges introduced by EH require that they be considered from the perspective of forming an integrated capability. This goal becomes increasingly challenging in considering aspects ranging across achieving cross-platform performance portability, the composition of diverse software components (e.g., workflows, machine learning, data-centric computing, and numerical and scientific libraries), support for dynamic application and system behaviors, the reasoning of program execution, and validation and reproducibility of results.

In addition to the diversity of hardware components introduced by EH, it is crucial to recognize the impact a growing diversity of developers and application domains has in shaping the capabilities of the programming environment. In addition to traditional workloads, data-centric applications from experimental and observational science activities—as well as hybrid applications that include traditional applications augmented with aspects of machine learning and artificial intelligence—present further diversification, complexity, and new challenges for the programming environment.

The remainder of this section discusses key challenges in meeting application requirements by (1) providing better programmability, (2) separating the implementation of applications from their mapping onto system resources (e.g., memory and processors), (3) dealing with the specific impacts introduced by data movement in an EH environment, and (4) understanding the impact EH has on the correctness of scientific results. Each of these topics plays a pivotal role in allowing the programming environment to assist scientists in reducing the amount of time needed to make a verifiable discovery in the face of the diverse system infrastructure introduced by EH.

3.1.1 Finding 1—Programmability

Definition. Programming systems provide abstractions for expressing algorithms, computations, and the mechanisms for mapping such computations to architectures. This research direction encapsulates the need to simplify programming and enable performance portability (write once, and high performance everywhere). Compared with today's approaches, this implies much more sophisticated implementations:

- Higher-level, orthogonal, and layered programming abstractions (descriptive and prescriptive)
- New compiler technology, domain-specific languages (DSLs), autotuning hardware, and software mapping mechanisms (see Section 3.1.2)
- Standardization of interfaces (both hardware and software) to support the hierarchical abstractions

Rationale. Our programming mechanisms must change to maintain, let alone improve, the current time-to-discovery. In EH systems, the number of different technologies exceeds the ability of individuals to master them. Thus, the need for programming environments that simplify the expression of algorithms while still being able to generate high-performance code for diverse execution engines is paramount. Various scientific domains need cooperating descriptive and prescriptive abstractions that work for a wide variety of algorithms, computations, and disciplines needed to support scientific discovery and that can effectively utilize the diverse range of EH hardware. Programming abstractions and mechanisms are required for accommodating significant scientific and hardware diversity, as well as enabling interdisciplinary development, the coupling of algorithms that cross scales and domains (e.g., ocean-to-atmosphere), and the full range of EH architectures. Ideally, all this needs to be accomplished while performance portability is provided.

Additional Discussion. Some related themes emerged during the workshop.

- Across a broad number of domains, an increasing number of scientists are using high-productivity programming languages such as Python, R, and Matlab. At the same time, their computational needs are rapidly growing, and HPC resources are becoming critical to evolving their algorithms and producing and evaluating larger and larger data sets.
- Descriptive abstractions provide a potential mechanism for the programmer to express concepts about the algorithm and data structures such as symmetry, data access patterns, and convergence properties. Such approaches could have a significant benefit over prescriptive approaches that hamper platform and performance portability.
- Programmers would prefer to have the system automatically map the computation and data to the execution environment; however, programming abstractions need to be available when automation fails and human intervention is necessary.

- Prescriptive, orthogonal programming abstractions for scheduling and memory and storage mapping should utilize the tools and mechanisms discussed below. A shared infrastructure is particularly critical for considering the complexities that arise in composing applications, or complete workflows, from multiple, independent software packages.

3.1.2 Finding 2—Mapping

Definition. Because of the diversity and complexity of EH architectures, achieving reasonable programmer productivity demands that the mapping of software to high-performance, architecture-specific implementations be automated as much as is feasible. Assume we start with a programming interface that separates program specification from details of how it is scheduled on the target architecture (see Section 3.1.1). Then, this research area defines the various approaches for deriving and realizing the mapping across a range of architectures:

- Performance introspection to evaluate performance during execution
- Autotuning, performance models, and machine learning to analyze the results of performance introspection and automatically derive or determine how to improve the mapping
- Both hardware and software mechanisms that dynamically improve executing software using results of this analysis in the execution context in which the application is executing
- Collecting and sharing performance data across applications to improve this mapping process over time.

Rationale. Today, achieving high performance across a variety of supercomputer architectures requires fairly low-level software that must be tailored to individual architectures. Even when using portable node-level programming models such as OpenMP, it is extremely unlikely that code will be performance-portable across fundamentally different platforms such as CPUs and GPUs. For this reason, it is common for application developers to write distinct code for different architectures. As architectural diversity grows for EH architectures, and complexity grows, this approach will become infeasible. At the same time, recent research in machine learning and related disciplines has developed scalable algorithms that analyze patterns in data to draw conclusions and guide actions. If such techniques can be applied to performance data, we can automate performance analysis and optimization decision processes to automate this architecture-specific mapping. Underlying mechanisms to support these techniques efficiently are needed to make such an approach practical.

Additional Discussion. The workshop discussions highlighted additional technical details in support of architecture-specific mapping:

- For efficiency, hardware support will be required for measuring and attributing resource utilization, critical paths, inefficiencies, and power states and for reducing performance data.
- A question arises as to how far automation can go. How much does the programmer need to express, and how much of the mapping can be derived automatically? Perhaps automated mapping is a critical step toward software synthesis.
- Beyond the technical challenges, this research area will make fundamental changes to the way applications are developed, built, executed, and deployed; so widespread adoption by the communities and facilities is required to realize their potential.

- To successfully address the challenges surrounding mapping will require a cross-cutting focus within the software stack that ranges from the OS to low- and high-level runtime systems and languages and programming model design and implementation. In addition, the pros and cons of particular approaches can be modeled in advance of hardware availability via modeling and simulation techniques. There are numerous open research questions about how these layers should interact and what level of the software stack is responsible for both making and controlling mapping suggestions and final decisions.

3.1.3 Finding 3—Data Centricity

Definition. Application developers need a set of higher-level programming abstractions to describe data locality on the emerging computing ecosystems. These programming model abstractions can expose crucial information about data locality to the optimizing software stack to enable performance-portable code. This requires research to explore new concepts that enable data locality to be managed more efficiently and productively, and to enable us to combine the best of these concepts to develop a comprehensive approach to expressing and managing data locality for future HPC systems.

Rationale. The conventional wisdom of the last two decades is built on the premise that computing is the most expensive component of computing systems, so all programming systems are built around the notion that computing is of primary importance. And yet the cost of data movement has become a dominant factor for the energy efficiency and performance of HPC systems, as discussed in Section 2.3, as computing is now cheap and ubiquitous. Nevertheless, contemporary programming environments offer few abstractions for managing data locality. Programming models remain steadfastly compute-centric, requiring that data move to where the computation has been placed, as opposed to a data-centric model in which compute is executed in situ where the data is located (thereby minimizing data movement). Even more fundamentally, numerical algorithm designers continue to evaluate algorithmic complexity based on computational complexity while entirely ignoring the data-movement cost or complexity scaling factors.

Absent data-centric programming facilities, application programmers and algorithm developers must continue to explicitly manage data locality using labor-intensive methods such as manually rewriting loop-nests and manually reorganizing data layouts to indirectly manipulate the memory hierarchy to *do the right thing*. Such optimizations can be implemented through intricate compiler transformations, but compilers often lack the necessary information to efficiently perform those transformations because of limited language semantics and weak internal performance models to guide optimization choices (this issue is a PRD that is covered in more detail in Section 3.4.2). These changes overturn fundamental assumptions that underpin current programming systems and portend a move from a computation-centric paradigm to a more data-centric paradigm for expressing algorithms.

Additional Discussion. The group captured a few additional points during the discussion.

- The strong relationship between EH systems with accelerators (see Section 2.2) and data-centricity was also discussed. When GPU accelerators first arrived, the kernels computed on the GPUs showed superior performance, but the performance advantage could be quickly degraded by the cost of moving data from the host to the accelerator over a PCI bus. If we extend that lesson to anticipated EH systems, almost all operations may well be performed by multiple accelerators, which will compound the data movement and scheduling challenges identified in the early days of GPU accelerators.

- Data movement to and from accelerators is far more expensive than computing, and accelerators will generally have very limited memory that must be treated as a scarce resource. Hardware accelerator efficiency is undercut by data movement costs and suboptimal placement. Better integrated performance models (see Section 3.4.2) are needed to enable proper resource allocation choices to be automated as described in Section 3.1.2. EH programs should not hide the overall structure of how a program uses its data; this needs to be made more explicit for the underlying implementation to exploit. This issue cuts across the entire software stack, from programming abstractions (this section), mapping (Section 3.1.2), and compiler optimizations, down to runtime systems (Section 3.3) and memory protection/security mechanisms in the OS/R section (Section 3.3).

- A common theme across most of the programming concepts discussed in the workshop was the need to specify locality information as part of a program's data declarations (as an integrated part of the type-system) so that all loops in the code can be modified appropriately by the runtime system without user intervention, rather than the current approach in which the organization of the program's computations or loop structures is needed to manage data locality (e.g., loop blocking and #pragma's). In current compilers, such information is not expressible or is destroyed by the early phases of compilation so that the runtime system is unable to exploit that information in any meaningful way. For example, an array could be described as distributed to a set of memories as part of its declaration rather than annotating each individual loop nest or array operation with such information (making loop transformations automatable by the runtime system), whereas current models require programmers to manually change the loop iteration space to achieve this same result. This approach permits a programmer to switch between distinct locality options in a data-centric way, allowing the compiler and/or runtime to propagate those choices down to the computations. A data-centric approach to programming models could minimize the amount of code that must be modified as architectures and program requirements change.

3.1.4 Finding 4—Correctness

Definition. "Correctness," broadly speaking, refers to our ability to predict the behavior of the software/hardware systems that we construct. In the HPC context, correctness encompasses not only functionality (how do we know that the system will produce the correct answer) but also performance (how do we know that the system will run in an acceptable period with the resources available).

Rationale. The correctness of the systems we build is central to DOE's science mission—if we cannot trust that the results we compute are correct, then the value of the solution is degraded. More complex systems are more difficult to verify, and that presents real challenges as we transition to tomorrow's EH systems. At the same time, there are significant opportunities to take advantage of technology to formally establish correctness in future EH systems.

Additional Discussion. The group captured a few additional points during the discussion.

As DOE contemplates the costs of software development for EH systems, it is clear that tools based on formal reasoning (such as full formal verification, automated analysis of programs to reason about more targeted properties, and program synthesis) could have a tremendous impact not only on the quality of the software produced but also eventually on reducing the cost of both writing and maintaining that software. However, some aspects of correctness are very important in the DOE context and have not been well

addressed in the verification and program analysis communities. Chief among these is reasoning about floating point. Although, very recently, a small community of researchers has emerged who are rigorously studying the verification of practical floating-point codes, there is still much work to be done to fully understand how we can effectively reason about floating point computations. Robust techniques for formally reasoning about floating point would also open the door to new ways of programming much higher-performance systems. Very few programmers understand how to write custom and robust floating-point code, and so the accepted method of floating point programming today is to rely on expert-written libraries that produce incredibly accurate results using the full precision of the underlying hardware, which is typically 64 bits. However, in most applications, this is overkill, as the results are not needed to 64 bits of precision; and typically, the input data is also not known to nearly that many significant digits. Thus, if it were possible to reliably reason about the required accuracy of intermediate computations, many of those computations could be replaced by much less precise and much faster versions that still produce final answers with no overall loss of accuracy. Our lack of understanding of how to support automated reasoning about floating point in realistic programs means that we cannot do this currently and must rely on simply over-computing every intermediate result to maximum precision to assure ourselves that the result is correct. Accelerators such as FPGAs, which are becoming widely available in heterogeneous platforms, heighten even more the potential benefit of being able to harness reduced precision computations.

Floating point is but one example where improved formal reasoning could have a significant impact on DOE software development practices. Another area that has not received focused attention for verification is reasoning about distributed, parallel systems, which is at the heart of all scalable simulation and data analysis codes and is often the source of the correctness issues that are difficult to debug at scale. Other examples include understanding and exploiting weak memory models and specialized accelerators. Finally, formal reasoning about performance and resource usage has yet to receive sustained research attention, and any progress in this topic could clearly have a significant impact.

On the side of challenges, by far the dominant method today of "verifying" software is testing; but one need only consider the number of bugs discovered in well-tested code to realize that testing, at least as currently practiced, is mostly good at finding problems that occur relatively often but is quite inefficient at finding fewer common problems. A second observation is that testing is good at identifying that there is a problem (e.g., the program crashes or returns the wrong answer) but is often not particularly helpful for finding the actual root cause that must be fixed. Specifically, the technology we have today for debugging remains heavily oriented toward single-thread, single-core debuggers that were developed decades ago. While progress has been made, the productivity of debugging at scale on parallel machines is much, much lower than it is for sequential programs; indeed, a large portion of the development effort of codes intended to run at extreme scales is spent in debugging both performance and functionality issues. The complexity of locating the root cause for performance issues usually scales with the complexity of the underlying hardware, as the nature of performance debugging is such that the structure of the hardware cannot be abstracted away.

There is an opportunity to use formal methods to improve this situation. Formal methods (i.e., formally proving properties of programs using rigorous formal logic) has made steady progress for decades. In recent years, the technology has reached the point that increasingly complex databases, compilers, and OSs have been formally verified to have important correctness properties (e.g., that a compiler faithfully translates a source program into assembly code). These guarantees are extremely strong: for every possible input, the program is guaranteed to produce an output satisfying the formally proven property. From the HPC point of view, the case for using formal methods is even stronger, as testing and debugging is extremely expensive at scale and will only become more so in the future. At some point, formal methods may well become not just the highest assurance but the least expensive method for building reliable systems.

Correctness in the style of fully verified systems is one goal, and there are related research technologies and other ways of using verification that are less all-encompassing. Tremendous progress has been made on fully automated approaches that target specific well-known correctness properties (e.g., absence of data races). There is also great potential in program synthesis, which turns the verification problem on its head: instead of writing code and asking a system to prove that it satisfies a property, programmers write down the properties they want a program to have and a synthesis tool finds a program that has those properties. Program synthesis has already proven effective in automatically generating tricky bits of code in several important domains. More systematic approaches to testing, perhaps integrated with verification techniques, are also a potentially promising research direction.

We also discussed correctness for more exotic future accelerators such as quantum and neuromorphic computing devices. Here, only the outlines of what the hardware may provide can be discerned, so it is difficult to speak with precision about what the distinct verification/correctness issues will be. But even the notion of what it means to be correct will need some changes for systems that work on such different basic principles. As more and more diverse kinds of accelerators appear in future machines, it was observed that it will be necessary to have the ability to construct reasoning systems compositionally, so that the reasoning module for a new accelerator can be combined with other independently written modules that model other parts of the system to give an overall formal model.

3.2 Software Development, Sustainability, and Productivity

Future heterogeneous architectures will undoubtedly cause disruptive changes in the entire software stack for supporting applications that require high performance. The software landscape has very broad scope, ranging from general-purpose libraries to application-specific and hardware-specific tools. The developers and the end users of the software stack often come from diverse science and engineering domains and have substantially different training and knowledge. Additionally, a software life cycle can be an order of magnitude longer than that of hardware. Several codes have been in existence for decades, whereas the typical lifetime of a platform is 3 to 4 years. For all of these reasons, it is imperative to organize community efforts to establish a set of well-defined and implementable software methodologies and policies.

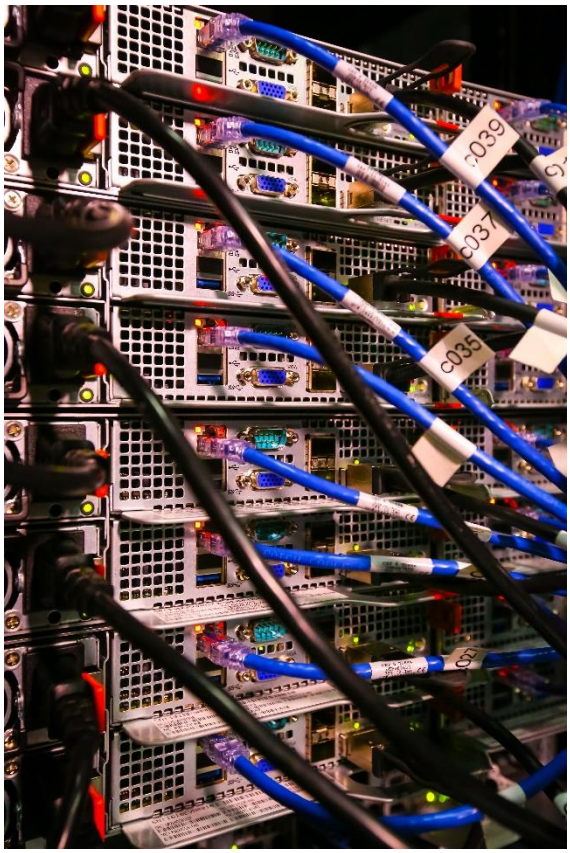
Along with emerging EH architectures, there is simultaneous growth in the complexity and heterogeneity of methods and solvers used in applications as scientific understanding grows and fidelity of models rises. This combination makes building a robust, sustainable software ecosystem to support multiphysics and multiscale simulations and couplings a daunting task [33]. Separation of concerns and layering of complexity become critically important and require designs of hierarchical encapsulation and abstraction. Additionally, for productivity, it is necessary to support interfaces that are independent of choices of algorithms and data structures so that application developers need not know the nuances of the underlying architectures. Furthermore, attention must be given to developing methodologies for evolving existing software to new programming models, which, given the longevity and complexity of such software, is contradictory.

Productivity can be considered an overall measure of the quality of the process of achieving desired results. In the context of science on HPC systems, productivity includes software productivity, along with 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 scientific exploration and discovery. During the past several years, there has been growing work on methodologies and tools to increase developer productivity and software sustainability for computational science and engineering [17, 30]. In particular, the IDEAS project [18] in the United States, and the Software Sustainability Institute [25] in the UK, have focused on the productivity and sustainability of research software.

The computational reproducibility of scientific experiments performed in heterogeneous computational environments is both critical for the credibility of scientific results and difficult to achieve, as numerous factors impact the goals of reproducibility. Against the backdrop of complicated architectures with large heterogeneity of devices, complex memory hierarchies, and communication patterns, teams face an increasing need to establish credibility by producing reproducible science. Definitions of reproducibility vary largely and depend on context, with several, sometimes complementary, often orthogonal, axes: (1) numerical reproducibility to validate code correctness, (2) scientific replicability of results to validate discoveries, and (3) performance reproducibility during time or resource constraint executions. All of these are relevant to scientific computing. Reproducibility goals and methods to attain these goals will vary depending on scientific applications and the communities around them. In some cases, reproducibility goals are implicit, with little formally expressed consensus as to what is considered reproducibility and what methods are used to achieve these goals.

3.2.1 Finding 5—Novel Design and Development Methodologies

Definition. As has been true for the past six decades, those who design and develop scientific software use tools and methodologies to facilitate their work and to maximize their productivity. Programming languages, higher-level abstractions, libraries, and run-time tools for debugging and performance analysis are all part of these infrastructures. Increasingly, they are brought together in integrated development environments. Historically, each new generation of HCP systems has tended to be more complicated than its predecessor, and this trend is further exacerbated by the increasing sophistication of the scientific software under development. The EH systems expected to be deployed in the next decade will further aggravate this problem, threatening to undermine the productivity of scientific software developers, and hence the pace of scientific discovery. Research into new tools for the design and development of scientific software is needed to enable effective use of these future systems.



Rationale. Software development environments reflect the underlying architecture of the computers they are used with; and for the last two decades what has been relatively stable—a communicating sequential processes execution model—has been applied to an exponentially growing number of virtual memory-based microprocessors. This relatively stable model has allowed the scientific computing community to accumulate a large body of software whose value greatly exceeds that of any of the ephemeral machines on which it runs.

The EH expected to appear beyond exascale threatens the scientific community in two ways. First, today's existing codes may have trouble porting to or effectively using future systems that are fundamentally different from current architectures. Second, new software, be it an entirely new application or an augmentation of an existing one, will be increasingly difficult to write. Current familiar programming languages and libraries were themselves designed and created for the existing model and may either not be available or not effectively exploit future, more heterogeneous systems (e.g., the necessity to extend C to program GPUs with CUDA or OpenCL).

Additional Discussion. The scientific computing community has faced such transitions in the past and has largely overcome them. Sometimes, they are adaptations to changes in architecture as the community moves to exploit new technology, such as the adoption of scalable systems based on commodity components. Sometimes, the transition is exploitation of new software technology, such as object-oriented programming. Such changes often take many years to effect. In both the cases mentioned, there was roughly a decade of research and experimentation before the community embraced the transition. Once that decision was made, it took many more years to transform existing codes or replace them. In the post-exascale timeframe, that pace of adaptation will no longer suffice without significant negative impact on the pace of scientific discovery. If a decade of research is again required to develop the technology necessary to create agile scientific software that can thrive on EH systems, then we need to begin immediately.

Many potential research directions could be explored to make EH machines approachable by a broad range of computational scientists. One approach is a separation of concerns, whereby the programmers are provided with abstractions that reflect their scientific domains, and the mapping of those constructs to specific systems is handled separately. DSLs are an example of this, but much common technology will need to be invented to enable the cost-effective creation of many such environments. Another approach is to try to automate many of the tedious, labor-intensive tasks involved in the porting of existing code from one machine to another, such as the automatic performance tuning work ASCR has supported. Recent advances in machine learning might provide a foundation for such research.

An even more ambitious project would be to try to exploit advances in machine learning technology to enable the automation of software development itself. One can imagine a future in which scientists formulate algorithms and then leave their realization to an automated system. Perhaps there would be interaction, in cases when the automaton encounters a construct it had never seen before and human creativity is required to resolve the mapping to a novel feature in a new machine.

3.2.2 Finding 6—Composability and Interoperability

Definition. In EH computing environments, we anticipate that composition will play a central role in software development. In the "vertical" sense, it will include the layering of application programming interfaces (APIs) and functionalities from application codes, libraries, and runtimes that translate the programmer's intent to execution in the various environments offered by the system. Likewise, in the "horizontal" sense, it will involve assembling various software "components" to form applications that must compose correctly and efficiently. Different parts of applications may be particularly well- or ill-suited to different execution environments offered by the EH system, which may motivate componentization strategies. Components may require very different implementations to work well in different execution environments (in ways that cannot be managed more transparently to the programmer), but they must be completely interoperable or interchangeable from the viewpoint of the programmer assembling components into the application. Likewise, components that interact with one another across the different execution environments must interoperate seamlessly, even with very different data representations and other factors.

Issues of composability and interoperability are understood today, for relatively homogeneous systems, at only a modest level; and available tools are very limited, especially in the HPC context. There are rich opportunities for research in this area, in both concepts and tools, with the goal of being able to systematically reason about models (in terms of performance and other characteristics) and execute composite applications, as well as facilitate the design and development of software with the desired characteristics of composability and interoperability.

Rationale. Modularity is a well-recognized best practice in software development and is already widely used, in varying degrees, within the HPC community and elsewhere. We cast our software as a collection of modules and compose them into an application. The emergence of EH systems will motivate even higher levels of modularity. For example, the same operation in an application might be feasible on a variety of different execution environments available in EH systems but require significantly different implementation strategies to obtain the desired performance. Similarly, different parts of an application may perform better in different execution environments; so, applications will likely involve the composition of multiple expressions of the same functionality targeting different execution environments, as well as the movement of data between execution environments with dramatically different data representations or constraints. Even if high-level programming environments provide common ways of coding these components at the user level, as the code is mapped onto an EH system, dealing with the issues of composition and interoperability is merely pushed down to lower levels of the software stack—perhaps not visible to users but certainly visible to the developers of those layers. Moreover, from one EH system to the next, different types of execution environments may be available, making the portability (much less the *performance* portability) of applications even more complex and placing even more demands on the ability to compose interoperable software components effectively.

The expected complexity of EH environments will make it very challenging to analyze and reason about the complex composite applications. Thus, beyond the basic need to be able to understand and deal with composite systems in a "manual" way, we believe that significant levels of automation will be required to make practical the development and use of large-scale high-performance software applications on such systems.

Additional Discussion. Discussions during the workshop produced a few examples of research issues related to composition and interoperability that we felt would be relevant to this PRD, although this list should in no way be considered limiting.

- Extracting and analyzing information about performance and resource utilization from composite applications, perhaps with help from AI or machine learning technologies to build models from the data

- Rich expression of constraints and capabilities of components, and effective orchestration of runtime behavior of composites (on both the software and hardware sides)

- Automatic reasoning in support of effective compositions, including issues of execution unit capabilities and performance characteristics, software requirements (including performance and resource needs), data movement, conversion of representations, and other issues

- The mathematics and computer science of mapping data between the diverse data representations offered by EH systems

- Understanding the behavior of the various execution units and interconnections, perhaps with the assistance of AI and machine learning techniques, and using the information to better inform and guide developers as to both "local" and composite performance and other characteristics, perhaps through abstract machine models and other approaches

- The opportunity to co-design memory and interconnect systems to facilitate composite applications and their development

- Capture and exploitation of provenance and other meta-information about data and execution patterns to allow adaptation in compilers, runtimes, and other parts of the system in the face of both changing applications requirements and changing execution environments

- Design and development of low-level APIs that enhance interoperability across diverse hardware

3.2.3 Finding 7—Evolving Existing Software to New Programming Models

Definition. Orthogonal to composability in software is the expression of program semantics within the components. In many scientific codes, these are the "kernels" that express the mathematics and logic of the solvers. To deploy new programming models, most of the kernels will need to evolve to those models. Research opportunities exist in devising tools than can automate the transition of code sections to new programming models, and tools that can verify semantic equivalence between original and transitioned codes. Note that these transformation tools would have a different purpose from those that are used for optimization. They would in effect be eliminating the need to manually rewrite software in a different high-level language.

Rationale. A lack of methodology and tools to affect the transition to new programming models has traditionally played a significant role in preventing their adoption by scientific software. The time needed to prepare a large code for a new platform paradigm using a new programming model is prohibitive if done manually. The code transformation tools that exist still require annotations or non-trivial rewriting of code chunks. This applies to DSLs as well as tools such as Kokkos. Adoption of new programming models will remain limited unless tools are developed to automate significant portions of the necessary code rewriting. Similarly, a lack of tools that can verify the semantic equivalence of rewritten code has largely been responsible for the persistence of "legacy" code sections even in otherwise well-designed and well-maintained codes. Together, the transformation and equivalence verification tools can in turn transform how scientific software is developed and maintained in future.

Additional Discussion. One of the topics that came up during the workshop was the expressibility of general purpose programming languages. The following points were raised.

- Does the full range of expressibility of general-purpose programming languages make portability harder with EH? It is possible that the same computation could be expressed differently to make targeted code generation easier.

- Can one express the constraining semantics through DSLs?

- There is a need for a broader collaboration between language and compiler teams and application teams to devise ways of optimizing expressibility so that portability is not compromised.

3.2.4 Finding 8—Reproducibility

Definition. Traditionally, reproducibility means getting the bitwise identical answer when the same program is run more than once but with possibly different hardware resources or scheduling of operations. We consider this "complete reproducibility." Numerical reproducibility, predicated on enforcing IEEE 754-2008 arithmetic for floating point instructions, presents multiple challenges in a parallel environment, because the roundoff errors make floating point arithmetic non-associative. Thus, bitwise reproducibility would require putting severe constraints on the order of execution for instructions and operations with distributed data structures, which is usually too costly to be practical at scale. Therefore, some scientific applications achieve scientific replicability when key (but not all) scientific quantities are reproduced

within error bounds. We consider this "partial reproducibility," which often suffices to ensure correctness of scientific simulations and discovery. For debugging purposes, some parallel codes implement a mode to enforce bitwise reproducibility at a much lower speed (e.g., with the same order of operations) but use the fastest mode in production runs. In addition, performance reproducibility is desirable in many situations. It can be guaranteed only when the program is compiled with the same compiler and compiler flags and run in the same execution environment.

Rationale. Although bitwise reproducibility is the best way to provide a guarantee of program correctness, many applications can be satisfied with different levels of relaxed requirements without compromising the scientific results. For example, complete reproducibility is important for climate modeling applications in which a scientific code is developed by multiple, distributed teams, such as the Energy Exascale Earth System Model (E3SM, previously known as ACME). Versions of the codes must provide bitwise-identical results when observational input data is used to ensure code stabilization during a code development phase. When the code is considered stable, this constraint can be relaxed.

Partial reproducibility is often sufficient for classical MD codes, molecular ab initio calculations, and materials codes. Although scientific replicability is considered achieved if the results of multiple runs fall within some error bounds, these tests are not currently performed in codes such as Gromacs or other common classical MD kernels. In thermodynamic integration, the length of a simulation is a factor hampering replicability, with a proposed solution of running many short simulations instead [3]. In quantum chemistry (NWChem) and, more recently, materials, exercises in comparing code to achieve replicable results are considered satisfactory if the same results are obtained at 10 decimal places. Numerical reproducibility can apply to the density functional theory of atomic calculation, but numerical integration into molecules does not produce reproducible results. While some target precision levels are desired, the costs to performance and scalability are high enough that ensuring these levels of precision would completely negate the benefits of parallelism.

If simulations are fundamentally stochastic and have an additional contribution to the random variations in their runtimes, complete reproducibility will be difficult to achieve and is best viewed as a multiple-level capability characterized by different trade-offs and costs. Performance fluctuations will exist for the same codes on different machines. Resource utilization on shared systems will also impact performance.

Additional Discussion. Ensuring the correctness and reproducibility of numerical programs in an environment where different accelerators behave very differently is extremely challenging. Several research directions need to be pursued.

- We shall develop new mathematical algorithms that deliver reproducible numerical results in the IEEE floating point semantics, regardless of the execution order of the operations. For example, the new algorithms on reproducible summation in ReproBLAS [10] make it possible for all the linear algebra libraries to be completely reproducible when run on the same machine (even with varying numbers of processors), same compiler, and same flags. Much more algorithm research is needed to expand this capability beyond linear algebra or under different execution environments with different compilers, and so on.

- We shall define different levels of reproducibility that are appropriate for specific applications and study the trade-offs between accuracy and performance. The goal is to achieve correct scientific results even in the heterogeneous environments that incur significant non-determinism in program execution. To this end, we need better mathematical foundations for what constitutes correctness.

- We shall develop new AI and machine learning methods to help derive reliable error bars and bounds for different algorithms and applications, especially for those that have historically relied on bitwise reproducibility. Thus, we need only to provide less demanding partial reproducibility.

- We shall develop new performance analysis tools that capture performance- and reproducibility-related information, including provenance, metadata about the runtime environment, and runtime characteristics of workflows with many components. The tools must integrate the data obtained from each device on the heterogeneous systems, such as low-level provenance details (e.g., clock speeds) for each device used during runtime.

- We need to understand the impact of higher levels of abstraction and many possible back-end targets on a testing regime. How can we be confident we are thoroughly testing a piece of code across the many different final instantiations it might experience as it is translated onto different platforms?

3.3 Operating and Runtime Systems

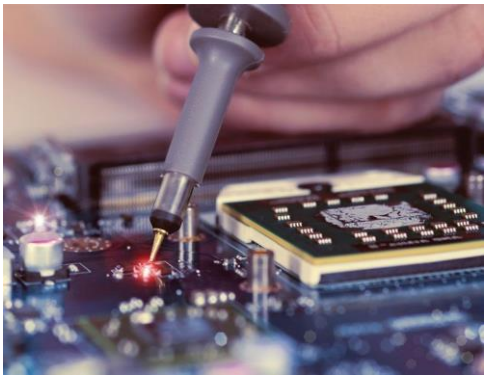
The OS provides the fundamental mechanisms to allocate and manage hardware resources provided to applications. For a large parallel computing system, the OS can be thought of as a single, persistent, privileged, distributed application that owns all of the hardware resources. Individual components of the OS may exist as system software on the individual computing elements, as well as the software that coordinates these components to provide system-wide resource allocation and management services. The OS is responsible for several fundamental capabilities, including isolating applications, resource sharing, protection, and access control.

Several factors influence the design and capabilities of the OS, including the underlying hardware; the functionality that must be provided to support applications, libraries, run-time systems, and tools; and the system usage model. Many of the challenges of EH either directly or indirectly impact the individual node-level OS as well as the overall system-level OS. As components become more diverse and specialized, it becomes increasingly complex for a traditional general-purpose, monolithic OS to manage hardware efficiently. Many of the existing approaches to scheduling computing elements and managing memory resources will need to be enhanced to manage this complexity efficiently. Similarly, as applications become more complex, and more sophisticated workflows are developed, the services provided by the system-level OS will also need to evolve to enable new approaches to coordinating resources and making the most effective use of the system.

The field of computer architecture is about to experience a shift very similar to the one that took place in the 1960s when architectures first began incorporating virtual memory. Move-in, move-out semantics to shift data from one memory space to the next were common in the 1950s. Manual move semantics were neither scalable nor portable; virtual memory was adopted despite a slight performance disadvantage. For the past 30 years, accelerators have been treated as separate memory spaces, quietly reintroducing the move-in, move-out semantics that so many others had fought to remove many years earlier. Industry trends are clear: the memory fabric is changing toward one that incorporates accelerators into the virtual memory system. What is not clear, however, is the role of the OS in managing this multi-architecture and

multi-device address space. The additional complexities of managing multiple memory devices, along with optimizing data placement for multiple classes of execution units, are beyond the scope of most current OSs.

Current systems typically have a single accelerator class, often a GPGPU or FPGA. The ratio of performance gain to portability cost for single-accelerator-class systems has been (for the most part) favorable; however, this ratio is rapidly changing, in ways much like the changes from the 1950s that pushed the industry toward adopting a virtual memory abstraction. We expect architectural advancements that will allow accelerator devices to be virtualized and more easily shared between users and processes. From an integration perspective, it is expected that advancements will soon allow (and performance trends will favor) tighter integration of multiple heterogeneous components on high-performance data paths, such as a memory bus, and enable them to share an address space with all the other devices on a node (and potentially off the node through more integrated system-level interconnects). These trends, along with the technology trends mentioned in Section 1.1.2, will likely shape the technology path toward one in which heterogeneous accelerators are at multiple locations within a system-on-chip and at multiple places within the memory and storage hierarchy. This integration, combined with heterogeneity, will strain the relationship of the OS and the runtime with the hardware. Co-design of new interfaces between hardware and software to determine the best boundary between accelerator and OS should be a research focus.



The convergence of commodity server and HPC processor cores over the past two decades has led to many compromises. Context swapping and scheduling from the operating system is one such compromise. The introduction of heterogeneous memory hierarchies with access times 10 times faster than standard storage media (but still far slower than DRAM-based memory) is driving computer architecture in two directions: (1) high-thread-count simultaneous multithreading (SMT) and (2) fast coarse-grain context swapping (FCGCS). The most likely direction for future architectures is toward a mix of both SMT and FCGCS, with FCGCS being the dominant factor in hiding nonvolatile access latencies. Given that current OS context swapping and scheduling methodologies take on the order of thousands of cycles, hardware acceleration is the clear path to enabling faster scheduling, swapping, and dispatch. This is another area in which co-design is clearly needed to find the right balance. A unique opportunity for HPC, which would enable unbridled performance portability for next-generation systems, is unifying the notion of thread context in a way that would allow portable transference of execution to accelerator devices. Enabling architectures to take on some of the notions that are currently ensconced deep within the OS would enable a world in which multi-vendor systems and even multi-IP systems-on-chip could exist to power future systems. The dominant usage model of today's HPC systems is characterized by a queue of requests for fixed-size allocations of nodes dedicated to a specific job. This batch-scheduled, space-shared usage model has many limitations, including the inability to dynamically expand or contract the number of nodes allocated to a job in response to application performance or overall system utilization. The burden is on the application developer to determine a priori the exact size of the allocation needed for each input data set based on an increasingly complex set of factors. As individual nodes become more heterogeneous, system architectures become more complex, and applications evolve into a set of integrated workflow components, the system-level OS will require new capabilities to support the demands of more sophisticated usage models.

The remainder of this section details important challenges in (1) the design of the OS and the associated runtime system to better support emerging EH hardware components, (2) decentralized resource management strategies that enable more efficient and effective coordination of potentially dedicated

and/or more intelligent devices, and (3) approaches to autonomous resource optimization that can be used to reduce complexity through introspection and dynamic adaptation. These challenges are naturally hierarchical, but we expect that the research required to address the first one will support and inform the needs identified by the second, and vice-versa.

3.3.1 Finding 9—Design of Operating and Runtime Systems

Definition. Current OSs are designed to integrate accelerators and special-purpose devices as isolated single-user devices attached over a relatively slow data path. We expect architectural advancements that will allow these devices to be virtualized and more easily shared between users and processes. We also anticipate advancements that will allow these components to be more tightly integrated on a high-performance data path, such as the memory bus, and be able to share a uniform view of the virtual address space with all the other devices on a node.

Rationale. Trends in architecture toward more tightly integrated systems blur the traditional control plane boundaries normally associated with an OS and runtime. Future systems will likely need to shift the balance of implementation responsibility for resource allocation, protection, and task dispatch between the OS, runtime, and hardware to better support EH systems and integrate accelerators as first-class compute devices versus just accelerators. The OS likely will become less involved in direct implementation and more of an administrative control plane that interacts with a robust hardware abstraction of underlying implementations for task dispatch, memory allocation, data migration, and scheduling. The runtime probably needs to evolve to pass the right kind of information to the hardware and runtime to enable lower-level interfaces to make appropriate dynamic decisions. The role of hardware likely must be increased, but tempered with appropriate open interfaces, to achieve the required efficiencies for making maximum use of an accelerator-rich architecture. Co-design of the architecture, runtime, OS, and application software should be pursued to maximize the efficiency, performance, and program portability of future systems.

Additional Discussion. The workshop discussion produced several possible research directions. While the narrative of Section 2.3.1 covers them with broad strokes, the specific issues below were expressed with high frequency and therefore were pulled out and ordered with respect to perceived importance.

- Rethinking virtual memory and translation for something that is more flexible for future accelerator classes as well as multiple memory and storage types is likely necessary. Industry trends toward heterogeneous memory systems and heterogeneous accelerator classes, as well as emergence of and awareness of so-called dark bandwidth, make this a critical focus of future architecture, OS, and runtime research and co-design.

- Co-design for accelerator offload methodologies is needed; at what level do the instruction set architecture, accelerator, compiler, runtime, and OS interact, and where are the control plane boundaries?

- Virtualization and compartmentalization of accelerators to optimize for goals such as security, efficiency, and overall performance is a critical requirement of accelerator-rich systems composed of multiple memory environments. How the OS interacts with future hardware interfaces for these environments will constrain and inform future OS design; what are the potential future hardware interfaces to accelerators and memory systems?

- What levels of system virtualization (software stack, runtime, OS, architecture, micro-architecture) are needed for future systems to maximize their ability to maintain flexibility in heterogeneous accelerator composition, maximize portability, and minimize single accelerator lock-in?

- Currently, accelerators interact with the host to handle exceptions and errors through software driver layers and exception handlers. Given future systems with many potential accelerators integrated with multiple memory/storage devices, standardizing the exception model is critical to ensuring software portability.

- To simplify offload to accelerators for runtime environments and management of offload from the OS's control plane, co-design of future OSs and runtime environments along with the hardware abstraction will be necessary to achieve the same level of software portability achieved by current systems.

3.3.2 Finding 10—Decentralized Resource Management

Definition. Resource management decisions are typically handled in a centralized fashion, under the assumption that costs and capabilities are homogeneous and constant. We expect hardware to continue to become more dynamic and adaptive in response to factors such as power/energy, so resource management policies and strategies must be enhanced to embrace and exploit these changes.

Rationale. Hardware resources are becoming inherently adaptive, making it increasingly complex to understand and evaluate optimal execution and utilization. New scalable methods of coordinating resources must be developed that allow policy decisions and mechanisms to co-exist throughout the system.

In addition to a global scheduler—common on contemporary HPC systems—a hierarchy of nested schedulers should be provided at the job allocation level and the node level. These schedulers would coordinate resource allocation and utilization in a top-down fashion, where the upper levels set the policy targets while the lower levels determine the mechanisms to implement those targets. Such schedulers should be aware of the disparate hardware present in the system, considering the inherent trade-offs of using various devices (e.g., allowing a job to use GPUs could provide a considerable speedup, but at a significant increase of the power envelope).

Resource management needs to be continuously adaptive to match the dynamically changing resource requirements of a job, as well as the changing performance and availability of the resources themselves. Programming system abstractions should be integrated with resource management, providing information such as the overall application progress, the critical path, and preferred resource management approaches (e.g., whether in power-constrained scenarios the workload prefers to under-utilize resources on the nodes or run on fewer but fully utilized nodes). A suitably adaptive runtime system could dynamically shrink a job, vacating some of the resources if so requested. Performance monitoring needs to be integrated into such a system as well, providing information on actual utilization of resources to verify that the allocated resources are used effectively, thus enabling performance introspection within the system.

Additional Discussion. Several prior research projects have advocated for such a dynamic and integrated resource management approach. They managed to demonstrate working prototypes, but in limited testing scenarios. Future projects should strive for an overall approach that can gain mainstream acceptance in production environments. Continuing research and development is needed to achieve a critical mass of integrated programming models, runtime systems, performance monitoring systems, and system management components.

Resource management also needs to consider the increasing complexity of application work-loads, which may consist of multiple loosely coupled components that could be of different provenance, coded in different programming languages and using different programming models, with widely different and potentially conflicting resource requirements. A resource arbitration interface is needed, wherein application components and the resource manager could negotiate and coordinate on turning the disparate individual requirements into a cohesive set that can be satisfied on a node-wide or even job- or system-wide basis.

3.3.3 Finding 11—Autonomous Resource Optimization

Definition. Autonomous resource optimization (ARO) refers to managing system resources without explicit direction from the application: while ARO may use provided information in the form of hints, such information is optional. The motivation behind such systems is usually to reduce the burden placed upon applications to efficiently use an increasingly complex architecture landscape. Instead of a prescriptive approach that relies on the developer's efforts, AROs automatically and dynamically manage resources via runtime system and OS assistance. Moving the arduous bookkeeping task of managing the resources needed by millions of threads of execution to the system software shifts responsibilities from humans to computers, which are much more adapt at performing enormous quantities of simple calculations quickly.

Rationale. Expecting the user or programmer to explicitly map an application onto an ever-expanding range of architectural resources is becoming untenable (see the mapping issue in Section 3.1.2). To optimize performance and scalability today, users must explicitly place threads and processes on cores and must explicitly divide up memory resources and manage locality. Problems result when developers encounter new hardware components or capabilities; instead of concentrating on the motivating science question, the developer must spend time dealing with sufficiently realizing the present machine's potential without confining the application's portability to other dissimilar systems. It is our belief that the responsibility for efficient use of resources must shift from the user to the system software. All levels of resource management should benefit from sophisticated and intelligent approaches to optimize the selection of resources to application needs. Moreover, the dynamic capabilities of ARO systems suggest important unrealized advantages for resiliency and desirable functionality like resource discovery and rolling software updates.

Additional Discussion. While ARO systems have very promising HPC potential, currently such systems are in their infancy. This immaturity may be due to several factors, including the pervasiveness and familiarity of existing systems that have few or insufficient ARO capabilities, such as the Linux OS and common HPC language runtime systems (e.g., C, C++, and FORTRAN). As a result, vital capabilities such as data placement and data movement within complex memory architectures are not well understood, much less efficiently managed throughout the lifetime of the data item. Hardware diversity makes performance portability in existing system software extremely challenging. Workshop discussions for ARO included identifying desirable goals as well as categorizing known issues. Among those points are the following:

- How should we address the abstract interface to the hardware and memory subsystem? Is it possible to handle complex architectures without placing additional requirements on software developers? What requirements will dynamic and autonomous system software place on application development and languages?

- How should we address a dynamically changing environment? We need more automated methods using machine learning to optimize the performance, energy efficiency, and availability of resources for integrated application workflows.

- How should we address complications that arise at the global OS–local OS boundary? We need more sophisticated usage models beyond batch-scheduled, spaced-shared nodes, which add significant complexity to the management of system resources. How and when does a cross-boundary interaction occur?

- How should we address performance portability? How can we allow the developer to specify policies to be applied without also specifying the portability-reducing mechanisms to be applied?

- How do we ensure that innovative ARO technologies developed for HPC are adopted by the widest supporting community possible? content

3.4 Modeling and Simulation

Modeling and simulation (ModSim) of existing and proposed computer architectures and systems have been longstanding pillars of the ASCR portfolio [20]. In supporting these research directions, ASCR recognized the need to meet performance, energy-efficiency, and resilience requirements of systems and applications at all scales—from embedded to exascale—recognizing their broad impacts to the larger computational science community in a range of research areas, including those affecting national security and domain sciences.

As we look at beyond–Moore's law computing, architectures exhibiting EH and specialization appear to offer promise to the seemingly conflicting goals of energy reduction and high performance. Given the potential diversity of heterogeneous components, there is an even greater need for tools that provide quantitative evaluation of the performance, energy, and reliability of science applications mapped onto novel architectures. Such tools are critical to design space exploration of specialized hardware building blocks and their interactions. The increasingly prominent role of ModSim in the era of EH was recognized by workshop participants even before the workshop began. A pre-workshop poll surveying attendees for their preferred breakout sessions revealed substantial interest, resulting in an additional ModSim breakout session being included in the workshop. The two ModSim sessions with 39 participants were high-energy and highly interactive and resulted in a distillation of research directions that will enable quantitative evaluation of EH architectures at the function unit, node, and system levels. Further, ModSim will play a major role in the design and evaluation of programming models/languages and runtime systems to be created for EH systems.

Status and Recent Advances. Numerous tools have been developed over decades, resulting in a rich assortment of mature modeling and simulation tools for traditional compute node components (i.e., CPU cores, caches, external memory, and networks). For example, in the CPU simulation space, the gem5 [4] simulator is widely used to study CPU cores, cache hierarchy, and on-chip communication, while Zsim [57] introduces optimizations to make possible fast simulation of hundreds of x86 cores. Sniper [16] trades off between accuracy and speed to facilitate co-design of multicore CPUs and applications. Evaluation of heterogeneous coprocessors with application-specific data paths and function units is enabled by the Aladdin extension to gem5 [61]. Highly used memory simulators include DRAMSim2 [56] to simulate DDR memory systems, 3-dimensional stacked memory simulators (e.g., HMCSim [37] and CasHMC [29]), and nonvolatile memory (e.g., NVMain [52]). With the emergence of byte-addressable nonvolatile memory, simulators such as CODES [73] and Supersim [41] are designed to be flexible event-driven, cycle-accurate, flit-level interconnection network simulation frameworks. They are being used in the Pathforward program to explore extreme-scale network topologies, routing algorithms, and router microarchitecture. The Structural Simulation Toolkit [55] is an open-source parallel simulation framework with a modular plug-in design to facilitate scalable simulation of CPU, memory, and networks.

Emulation is playing an increasingly important role in speeding up architectural exploration. While most hardware emulation tools are kept proprietary to vendors, such as Cadence's Palladium system, recent open efforts have emerged. Examples include the FireSim and the Logic in Memory Emulator (LiME) [27], which is an open-source FPGA emulator for design space exploration of novel accelerators closely coupled to CPUs.

Meanwhile, given the size of the EH design space, analytical modeling tools provide early opportunities to assess performance and other requirements, such as power, even before the simulators are constructed. Two examples include the Roofline model [69] and the Aspen toolkit [64]. These analytical tools typically provide extremely fast evaluation at the cost of detail, which can be especially useful in the early design phases of new architectures.

Addressing Extreme Heterogeneity. EH brings significant challenges to ModSim, as well as opportunities to advance the state of the art in ModSim tools. However, the participants recognized that ModSim tools for EH must span a much wider range of requirements for flexibility, accuracy, and fast evaluation than earlier tools. The number of architectural options on the horizon is daunting and demands composability and modularity among the ModSim tools themselves. There is a need to quantitatively assess new architectural features at a wide range of granularities. Architectural features are proposed as fine-grained enhancements within a CPU or GPU microarchitecture, as co-processors to traditional CPUs; memory/storage hierarchy innovations, including processing near or in memory; and processing in transit. Neuromorphic simulators—including Compass [53], NeMo [51], and MNSIM [71]—are vital for verifying hardware performance, testing new potential hardware changes, and hardware co-design. Most neuromorphic hardware has not been widely deployed or distributed, so accurate and efficient software simulators are key for identifying whether a neuromorphic system is useful for scientific applications.

As important as the need to accurately capture the characteristics of proposed architectural features, is the need to assess characteristics of the proposed interconnect linking heterogeneous components. Accurate quantitative analysis through ModSim is needed to guide development of not only the hardware itself but also, as important, the supporting software and applications. The tools should be able to measure and characterize performance improvements and slowdowns to assess to what extent applications can benefit from proposed heterogeneous resources, and how application refactoring or modification affects overall performance. The tools should give insight into reasons for unexpected performance. This requires system-level ModSim concurrently at the microarchitecture, memory interface, and network interconnect levels, posing a scalability challenge to ModSim tools—especially when "applications" consist of entire

workflows. And while accurately capturing the functions and their interconnection with respect to correctness and performance is the primary driver, the thermal and power characteristics of heterogeneous systems are also needed from the microarchitecture up to the system level.

Tools must be applied/enhanced/developed to simulate and emulate emerging and envisioned complex CPUs that incorporate wide variability in function units, including floating point units of reduced precision, mixed precision integer units, and special-purpose intellectual property blocks. It is recognized that many-core architectures may require new cache coherence protocols that balance application needs for cache coherency with scalability drivers as the number and type of function units on-chip and off-chip increases. Interaction of cache hierarchy and function units is subtle and complex, increasing the algorithm development challenges of CPU architecture simulators. The complex memory/storage hierarchy is expected to include multiple types of memory—on-chip (scratchpad), on-package, on-node, and disaggregated, such as high-bandwidth vs. high-capacity memories—requiring detailed and faithful ModSim of memory controller designs and a spectrum of memory latencies, bandwidths. and capacities [34, 45]. The inclusion of persistent memory with a variety of possibly asymmetric latencies and wear characteristics—for example, phase change memory, ReRAM, MRAM, and 3-dimensional NAND—will also require new memory simulator modules.

In highly integrated heterogeneous systems, the overhead of task dispatch and management of heterogeneous processing hardware near memory must also be modeled. It is important to capture the impact on the whole hardware/software system of introducing novel resources in nontraditional parts of the memory/storage hierarchy. ModSim tools must faithfully model, both functionally and in a timing-accurate model, the communication protocols that use load/store, put, get, and atomic operations, especially with respect to the impact of proposed fine-grained communication primitives on whole application performance.

At the discrete component level, new challenges are found in modeling heterogeneous interconnects combining photonics with electronic components. At the full system level, there is a need for scalable assessment of potential workflows that include disaggregated storage and distributed datasets.

3.4.1 Finding 12—Methodology and Tools for Accurate Modeling of Extreme Heterogeneity

Definition. New models and modeling methodologies will need to be developed or current ones adapted to an increasingly diverse and complex architectural landscape. The space under investigation is multidimensional, encompassing disparate technologies including new accelerators, memories, and storage; network concepts and beyond–Moore's law technologies; the triad of performance, power, and reliability; applications and algorithms designed for heterogeneity; and static and dynamic modeling.

Rationale. The current state of the art in modeling methodologies was developed in response to the architectures and application structures typically found in today's HPC landscape. However, as we move toward EH and beyond Moore's law, many of the assumptions built into these methods will no longer hold true. The group identified a broad set of crosscutting challenges including

- **Scope and scale:** Coverage of new technologies (e.g., device, quantum, neuromorphic, analogous computing), new models of computation, and different scales from microseconds (power) through workflows (days).

- **Integrated Modeling:** Development of methodologies and tools for integrated ModSim of EH components and power/performance/reliability/thermal.

- **Application Modeling:** Ability to capture and model realistic application workloads that use very diverse compute and storage technologies.

- **Accuracy and Cost Trade-offs:** Different levels of accuracy, depending on use cases, for similar or better accuracy and level of detail for integrated simulations through new methodologies, reduction of model generation and execution time, automation, and simplification of validation.

- **Dynamic Modeling:** Methodologies for dynamic modeling: assembling models on the fly for runtime/OS/compiler introspection.

- **Interoperability:** Supporting interoperability via composition of workload, architecture, performance, and physical (e.g., power) models.

Ultimately, the ModSim breakout participants identified potential research approaches and research directions:

- Methodologies and tools to be researched and encapsulated in tools for ModSim of new architectures and applications

- Investigation of new stochastic methods, machine learning, and other approximation techniques for model building and cost reduction

- Targeted prototyping to assist in whole-system validation; techniques from model correlation and verification

- New model reduction techniques

- End-to-end performance prediction techniques that allow accurate assessment of diverse technologies on real workloads using realistic programming paradigms

Additional Discussion. The group agreed that there would be a transformative impact on all aspects from hardware to workflow design—specifically, predictive and accurate design space exploration, optimization and design guide of new systems (memories, I/O, storage, processors, BML architectures, and application workloads. Also, this research direction is urgent for the successful transition to EH in that many other areas could benefit from useful ModSim artifacts. EH exhibits many degrees of freedom in the way workloads are mapped to system resources for optimality. ModSim is the key to achieving productive computing by many metrics. The identified research directions will enable comprehensive design-space exploration and lead to the ability to optimize execution of apps on systems dynamically. Metrics for success include cost reduction in architecture, application mapping, and programming model/language and runtime development; improved accuracy; ease of generation; and coverage of a wide spectrum of architectures and application workloads.

3.4.2 Finding 13—Cohesive Integration of Modeling and Simulation Infrastructure with Programming Environment and Runtime System

Definition. ModSim is well suited to helping solve programming and runtime system challenges (see Sections 3.1.2 and 3.3.3). ModSim will be critical in the developmental phase of these new solutions, facilitating co-design of programming environments with new devices and systems. ModSim will also be critical in the production phase, providing dynamic cost models that navigate compilers and runtime systems in the multidimensional EH space.

Rationale. While EH hardware will be required to satisfy performance/energy requirements, programming environments must evolve for these systems to be useful. Programming models geared toward EH (Section 3.1.1) and data centricity (Section 3.1.3) must emphasize performance portability across many target architectures to avoid a proliferation of device-specific programming systems. In this regard, several important challenges emerge. These challenges include the following.

- Interoperability: How can we make ModSim tools useful and accessible in the design of programming environments? How can we integrate ModSim tools into the compilers or runtime system supporting a programming environment?

- Efficiency: Can we develop approximate models and manage accuracy/cost trade-offs within ModSim tools to provide sufficiently accurate metrics within cost bounds so that ModSim is useful in informing compiler/runtime decisions?

- ModSim validation: Programming models can be tested only on speculative technologies via ModSim if architecture prototypes are not yet available. Programming models on which fundamental design decisions are based require validation and verification of ModSim tools to ensure the design decisions are valid when deployed on real systems. The term "validation" may have different interpretations in different contexts, and the ModSim community currently lacks a general rule for validating simulation models. It is worth checking a multistep workflow presented in Mubarak et al. [44].

- ModSim diversity: ModSim will explore a dramatic range of new devices from "more Moore" architectures that implement existing architectural paradigms with new devices, to disruptive non–Von Neumann architectures, including neuromorphic and quantum. The diversity of ModSim tools will grow with this diversity of proposed architectures. Can ModSim provide "performance model portability" to avoid requiring programming environment developers to understand the entire EH architecture space?

From these challenges, the group identified potential approaches. They will require interfacing with efforts that further performance portability and those that advance new directions in programming models. As we are already seeing, compilers and runtimes supporting each programming environment will carry additional responsibility, having to decide between multiple target architectures and also tuning multiple parameters within each architecture. The proposed approaches and research directions include the following:

- Management of design space complexity via approximate models: Use machine learning or synthesis models to distill complex multidimensional performance/energy behavior derived via ModSim into efficient mathematical models. Validation and verification of derived models against detailed simulation will be critical.

- Componentize ModSim software: Direct software integration (rather than workflow integration) with compilers and runtimes will be required. ModSim performance models must be usable online via well-defined APIs. Although this involves a significant engineering component, significant research and exploration is required in collaboration with Programming Environment teams to develop interfaces and componentization strategies.

- "Meta" ModSim to explore hardware acceleration of ModSim techniques: The majority of ModSim frameworks use conventional general-purpose CPUs to run simulation. Particularly for emulating performance behavior via machine learning models, ModSim can explore new devices or new

architectures for accelerating ModSim itself. This will lower its overhead and thereby increase its utility to programming environments.

Additional Discussion. The group identified cohesive integration as the second pillar of ModSim research for several reasons. Success in EH ultimately depends on programming environments enabling productive application development. Industry tools are likely to be focused on single, proprietary architecture or specific market-driven workloads. This likely leaves the burden of programming model performance portability on DOE.

The group also identified multiple metrics for success:

- Number of programming models/environments leveraging ModSim tools: Although ModSim can achieve some success in isolation, success ultimately depends on its usefulness to and adoption by programming environments.

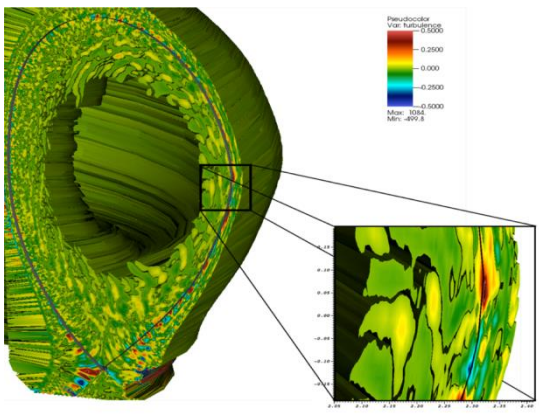
- Improvements in accuracy/efficiency of ModSim performance models: As new mathematical models and software tools become available, the speed at which "optimal" execution paths can be down-selected from a factorial design space via ModSim should be improved. Either the accuracy of models should be increased or their software overheads should be decreased to show measurable success.

- Performance/energy improvements of science and engineering codes obtained by incorporating ModSim into programming environments: When ModSim performance models are used to inform compilation or runtime decisions, we can measure the actual performance improvement relative to the "default" decision made in the absence of ModSim tools.

ModSim will have a gradual and consistent impact as new tools become available. ModSim tools can have near-term impact because ModSim precedes the full development/delivery of new hardware and applications. Adoption of tools will depend not only on the maturation of ModSim but also on the readiness of programming models/environments to incorporate them. ModSim can "lead" and be ready when the EH programming models/environment maturity level requires ModSim cost models.

3.5 Facilitating Data Management, Analytics, and Workflows

EH systems and applications bring significant new challenges to usability through an unprecedented variety and volume of data, resources, and services. Future heterogeneous workloads will feature integrated computational, extremely large, experimental, observational, streaming, real-time, and multimodal data sets. No single data ecosystem will support all these uses while fully exploiting heterogeneous architecture. EH systems will need to accommodate and integrate complex, dynamic, and disparate workflows and software stacks in an agile—not monolithic and predetermined—way and dynamically manage conflicting resource demands, balancing the workload on heterogeneous resources.



Paraphrasing from the Future of Scientific Workflows 2015 workshop report, a workflow is defined as the automation of sequencing, orchestrating, and moving data among the subset of scientists' tasks involving the collection, generation, and processing of numerical data. Workflow execution is the realization of systems that aid in the automation of these tasks, freeing the scientist from the details of the process.

With heterogeneous architectures, the challenge of productively porting and scaling the performance of data analytics algorithms, workflows, and data models across target resources will increase dramatically. Performance portability, therefore, is key to successful data science. The increased software and hardware complexity of future systems will also impede the usability and productivity of computing resources. If we do not address usability and productivity in the data ecosystem, it will be difficult, if not impossible, to achieve the needed efficiency gains on future heterogeneous systems.

To overcome these difficulties, two broad areas of research are essential:

- Mapping of science workflows to heterogeneous hardware and software services
- Adapting workflows and services through learning

Support for work in the productivity and performance portability of workflows was also identified as a need, as was research in rapid adaptive analytics to facilitate the use of EH systems by domain scientists, not just data scientists. The last two areas are discussed in the Additional Discussion subsection, following the discussion of the priority research sub-directions.

3.5.1 Finding 14—Mapping Science Workflows to Heterogeneous Hardware and Software Services

Definition. Workflow and data management decision making refer to activities related to the scheduling and placement of workflows and data on compute and data resources, and the adjustment of these plans in response to unfolding events such as new data, changes to workflow, or changes in cost or availability of resources. The group recognized that some of these challenges are similar to the challenges of those in the Operating and Runtime Systems area (see Section 3.3); but the workflow challenges exist at a higher level of abstraction, on longer timescales, for larger data set sizes, and with potentially global reach. Supporting complex workflows requires optimizing for a variety of user- and facility-level objectives that reflect goals motivated by issues ranging from scientific discovery to public policy. Optimal data management will improve the utilization of EH systems and understanding of the systems and security. Automating decision making will also ease programming and increase performance and resource efficiency (RD4.2). This in turn will allow application developers and users to focus on science goals by reducing data management and processing overhead.

Rationale. Application teams are moving toward the coupling of many activities and codes—including data ingest, simulation, data analysis, and visualization—in which each code has diverse requirements. The EH of future systems requires proper abstractions to address this challenge. Workflow execution will require distributed optimization frameworks across all layers to support a wide range of data management strategies for diverse workloads, from deep learning of simulation data on cold storage, to in situ analytics critical for code coupling, to gathering the required provenance for reproducibility. The research directions below will not only allow application teams to optimize their workflow as it is running but also will expose information to aid teams in their science across the entire data life cycle.

Data live in a variety of continually updated storage systems and heterogeneous storage devices with optimization opportunities that can change quickly. Practically, data are in very large data sets stored across—and continually moving among—geographically distributed locations, and across different storage tiers. At the same time, each analysis operation may access a different subset of the data, creating additional data access challenges.

Systems with heterogeneous elements and data coming from various sources at varying rates will need to execute workflows dynamically. Workflows will need to adapt on-the-fly to rapidly changing resource,

data, and user constraints. Such constraints may present challenges such as failing components or data logjams, but they may also include opportunities such as new resources becoming available or human intervention leading to unexpected discoveries. Irrespective of the causes, workflows will need to adapt in real time to dynamic heterogeneity.

Additional Discussion. In summary, EH systems and applications bring significant challenges to usability through an unprecedented variety and number of data, resources, and services. Research is needed to enable the discovery and mapping of science workflow requirements to appropriate data, hardware, and software services. The development of new methods of composing scientific workflows (e.g., via workflow motifs) is similarly critical to the success of future scientific endeavors, especially as it relates to moving workflows between platforms. Finally, prompt attention is needed to interfaces that facilitate hardware and service composition—"programming environments for data services." Research in this area has close ties to Programmer Productivity (Section 3.1) and should be coordinated with these efforts.

3.5.2 Finding 15—Adapting Workflows and Services Through Learning Approaches

Definition. Autonomous workflows (AWs) are mechanisms that can be used to independently automate "sequencing, orchestrating, and moving data among the subset of scientists' tasks involving the collection, generation, and processing of numerical data" with some initial human input or knowledge about workflow goals and constraints. These mechanisms can change a workflow dynamically so that it meets those goals and constraints as the workflow unfolds, as resource availability changes, and so on.

Goals in this context can be specific or general guidelines, such as minimizing or maximizing some quantity, or getting as close as possible to a specific quantity; however, the workflow could still operate even if that goal is not met: goals are soft constraints. (Hard) constraints are guidelines that specify that the workflow must operate within certain bounds: constraints are not negotiable.

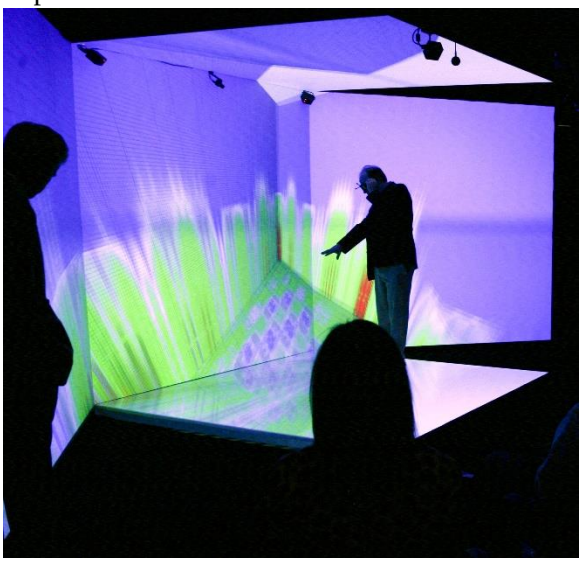
Workflow goals and constraints can be, for example, in the areas of performance, data storage, fidelity of results, time-to-solution, resource usage, resilience, provenance, energy usage, communication, and/or reproducibility. AWs will need to work with other AWs that may use a subset of the same resources. An AW can operate properly and effectively in a system and among other AWs given that

- It is possible to specify these goals and constraints easily via metrics, models, or other techniques (e.g., machine learning).
- The workflow can be modified to meet the goals and constraints—it can control the sequencing, orchestration, and moving of data, as well as the refactoring or rework of the collection, generation, and processing of data.
- It is possible to predict whether a specific workflow will meet the goals and constraints, and adapt the prediction based upon new data.
- It is possible to measure how well the workflow is meeting those goals and constraints by interpreting telemetry data in real time.
- It is possible for the AW to communicate what it knows (goals, constraints, and measurements) to other AWs and to incorporate what those AWs know into its body of knowledge.

- The inverse problem of "what workflow will have the outcome I want" within the space of possible workflows can be solved in an automated way or with minimal human intervention or decision making.

Rationale. Extracting the highest value from EH systems requires rapid, complex balancing across a wide variety of computational, storage, and networking technologies in response to a widely varying workload and significant rate of faults. Owing to the complexity of and need for timely reconfiguration, these workflows will likely need to be guided by human-specified goals and constraints for the work to be done but will need to be constructed and dynamically adapted without a human-in-the-loop or with minimal human monitoring. Thus, AW is a research area and capability that will be needed, and machine learning is a promising enabler of such a capability.

As a general theme, AWs are at the highest level, using the emerging EH capabilities and research developed in other areas mentioned in this report. AWs serve to automate the translation of the work to be done into concrete and adaptive workflows that meet human goals and constraints. Without the implementation of and research on the other capabilities, the mechanisms to create AWs cannot be implemented.



AWs involve creating a higher-level abstraction that uses the abstractions provided by OSs and runtime systems to carry out dynamic discovery of resources. AW mechanisms are users of the future programming system abstractions that will enable adaptive execution and autonomous resource optimization. AW mechanisms will interface and query these capabilities to ensure they are meeting the goals and constraints specified for the workflow by the human user.

AWs will use the future performance modeling and prediction methodologies developed in Section 3.4 to aid in reasoning about the trade-offs in the use of heterogeneous components. AWs will use "cost models" produced by future performance modeling and simulation infrastructures, and knowledge of the current and future availability of resources provided by telemetry data. They will use fault and error detection capabilities as well as the capability to check for reproducibility. A workflow system is ineffective if it is unaware of faults and errors, or if it is not reproducible to some degree.

Finally, there is a tie-in with future analytics providing an AW with the information it needs in all the areas mentioned previously (resource usage, performance prediction, reproducibility, and fault detection). Rapid adaptive analytics could provide AWs with additional nimbleness so that the analytics within workflows can be reformulated and are not bound to a specific set of resources.

At another level, several challenges exist at a lower, infrastructure level, which, when solved, would provide a major foundation to build solutions for these challenges. Methods of formally describing resources, workflows, goals, and constraints must be developed for the approach to be viable. Research into gathering telemetric data, at the system and workflow levels, is needed to understand how to best instrument these systems, what data is most valuable for enabling autonomy (e.g., via machine learning), and how to collect this data in a scalable manner. Additional research is needed in the application of online learning to adapt computation and data organization to available resources, emerging results, and new telemetry. Furthermore, research is needed in transfer learning to enable the application of the

knowledge learned from optimizing one workflow on one EH system to other types of workflows and EH systems. Research in this area has close ties to Managing Resources Intelligently (Section 3.3) and Modeling and Simulation (Section 3.4) and should be coordinated with activities in this area.

Additional Discussion. During the workshop, two additional topics were identified as promising directions for future research, although they were not selected as priorities in the short term.

Facilitating Workflow Productivity, Portability, and Performance. Workflow productivity, portability, and performance refer to properties that define time-to-solution for developers constructing these workflows, as well as measures of the "quality" of the result. In other words, how easily and rapidly a science team can develop or port a workflow to a new platform, and how well the resulting implementation makes use of the platform and its resources. Advances are key to executing science workflows and understanding their results, enabling efficient execution on diverse platforms, ensuring scalability of high-level descriptions of analytics workflows, and increasing user productivity and system utilization. This topic is in fact a superset of the two PRDs discussed earlier in this section. Additional research challenges in this area include

- Understanding the trade-offs between productivity, portability, and performance metrics for scientific workflows
- Raising the level of abstraction in workflow descriptions to hide implementations and platforms
- Modeling user interactions with workflows (e.g., human in the loop)

Rapid Adaptive Analytics. Rapid adaptive analytics refers to the ability to create advanced data analytics that are tuned to a given scientific problem and are specified via high-level goals and a description of the input data types. Such a system can compose multiple state-of-the-art machine learning and deep learning methods to allow the rapid development of complex, tailored analytics. The availability of rapid adaptive analytics means that without data modeling, algorithmic understanding, EH knowledge, and supporting data analytics staff, domain scientists can perform the data analysis task in less time, with similar or better quality of data analysis, than with supporting analytics staff. This topic also brings machine and deep learning to bear on problems of significance to DOE. Research challenges facing the development of successful rapid adaptive analytics include

- Mapping high-level goals to quantifiable optimization objective functions
- Managing and transforming data representations across algorithms
- Understanding and modeling error bounds and error propagation
- Understanding and modeling trade-offs among data models, algorithms, and EH hardware capabilities

4. Priority Research Directions

At the end of the workshop and during the subsequent months, the organizing committee and breakout group leads collated, distilled, and prioritized the workshop findings into the following PRDs for extreme heterogeneity in HPC:

1. Maintaining and Improving Programmer Productivity

    - Flexible, expressive, programming models and languages
    - Intelligent, domain-aware compilers and software development tools
    - Composition of disparate software component content

2. Managing System Resources Intelligently

    - Automated methods using introspection and machine learning
    - Optimize for performance, energy efficiency, and availability

3. Modeling and Predicting Performance

    - Evaluate the impact of potential system designs and application mappings
    - Model-automated optimization of applications

4. Enabling Reproducible Science Despite Diverse Processors and Non-Determinism

    - Methods for validation on non-deterministic architectures
    - Detection and mitigation of pervasive faults and errors

5. Facilitating Data Management, Analytics, and Workflows

    - Mapping science workflows to heterogeneous hardware and software services
    - Adapting workflows and services through machine learning approaches

This section presents these PRDs and their respective subtopics in more detail.

4.1 PRD 1—Maintaining and Improving Programmer Productivity

Key Questions

In an era of an increasingly diverse and complex computing environment, what advances in programming models, environments, and tools are required to improve the productivity of a broad range of scientific software developers?

The very high levels of effort and expertise required to develop verifiable, high-performance scientific software limit the ability to achieve key scientific milestones and transformational discoveries. Diversity and heterogeneity in the design of supercomputer architectures will increase the difficulty of these efforts and directly impact the overall time-to-solution. New programming environments, methodologies, and tools, combined with new approaches for effectively utilizing rapidly emerging capabilities such as machine learning and AI, will be essential for reducing complexity and boosting productivity. This will be critical to maintaining and establishing competitive advantages across any number of mission critical areas of study.

PRD 1.1. Programmability. Programming systems provide abstractions for expressing algorithms and computations and the mechanisms for mapping these computations to architectures. This PRD encapsulates the need to simplify programming and enable performance portability (write once and execute with high performance everywhere). This goal implies more sophisticated implementations: (1) higher-level, orthogonal, and layered programming abstractions (descriptive and prescriptive); (2) new compiler technology, DSLs, hardware autotuning and software mapping mechanisms; and (3) standardization of interfaces (both hardware and software) to support the hierarchical abstractions.

PRD 1.2. Mapping. Because of the diversity and complexity of EH machines, achieving reasonable programmer productivity demands that the mapping of software to specific features of high-performance EH architectures be automated as much as practical. Assume we start with a programming interface that separates program specification from details of how it is scheduled on the target architecture. Then, this research area defines the various approaches for deriving and realizing the mapping across a range of architectures, including (1) employing performance introspection to evaluate performance during execution; (2) using autotuning, performance models, and machine learning to analyze the results of performance introspection and automatically derive or determine how to improve the mapping; (3) developing both hardware and software mechanisms that dynamically improve the execution of software using results of this dynamic analysis in the context in which the application is executing; and (4) collecting and sharing performance data across applications to improve this mapping operation over time.

PRD 1.3. Data Centricity. The cost of data movement has become a dominant factor for the energy efficiency and performance of HPC systems. And yet, contemporary programming environments offer few abstractions for managing data locality. Programming models remain steadfastly compute-centric, requiring that data move to the location of the scheduled computation, as opposed to a data-centric model in which the compute operation is executed in situ where the data is located (thereby minimizing data movement). Lacking data-centric programming facilities, application programmers and algorithm developers must continue to explicitly manage data locality using tedious methods such as manually rewriting loop-nests and reorganizing data layouts to indirectly manipulate the memory hierarchy to *do the right thing*. Some of these optimizations can be implemented through intricate compiler transformations, but compilers are often constrained by lack of the necessary information to efficiently perform those transformations because the programming model provides only compute-centric semantics. Application developers need a set of higher-level programming abstractions to describe data locality on the emerging computing architectures. These programming model abstractions can expose crucial information about data locality to the optimizing software stack to enable performance-portable code. This requires research to explore new concepts that enable data locality to be managed efficiently and productively while enabling users to combine the best of these concepts to develop a comprehensive approach to expressing and managing data locality for future HPC systems.

PRD 1.4. Novel Design and Development Methodologies. As has been true for the past six decades, those who design and develop scientific software use tools and methodologies to facilitate their work and maximize their productivity. Programming languages, higher-level abstractions, libraries, and run-time tools for debugging and performance analysis are all part of these infrastructures. Increasingly, they are brought together in integrated development environments. Historically, each new generation of HPC systems has tended to be more complicated than the last, and this is further compounded by the increasing sophistication of the scientific software under development. The EH systems that are expected to be deployed in the next decade will further exacerbate this problem, threatening to undermine the productivity of scientific software developers, and hence the pace of discovery. Research into new tools for the design and development of scientific software is needed to enable effective use of these future systems, while striving to minimize the software changes required for each new system.

PRD 1.5. Composability and Interoperability. In EH computing environments, we anticipate that composition will play a central role in software development. In the "vertical" sense, this activity will layer APIs and functionalities from application code, libraries, and runtimes that translate the programmer's intent into execution in the various environments offered by the system. In the "horizontal" sense, various software "components" will be assembled to form applications. These separate components of applications will have a range of suitability for different architectures and execution environments offered by the EH system, which may motivate componentization strategies. In turn, components may require very different implementations to work well in different execution environments (in ways that cannot be managed more transparently to the programmer), but they must be completely interoperable or interchangeable from the viewpoint of the programmer assembling components into the application. Likewise, components that interact with one another across the different execution environments must interoperate seamlessly, even with very different data representations and other factors.

4.2 PRD 2—Managing Resources Intelligently

Key Questions

Can AI and machine learning be effectively incorporated into system software to coordinate and control a large and diverse set of computing resources?

Manually attempting to coordinate, reason about, and schedule the placement of data; which types of processors to select for certain calculations; and when computations will occur will become intractable as the complexity, diversity and scale of HPC systems grows alongside increasingly challenging scientific missions. Optimized resource management decisions must be made at a pace, scale, and level of complexity that exceeds human ability. Furthermore, today's system software is not designed to manage rapid changes in resource scheduling and workloads that cutting-edge science and EH systems will demand. Infusing AI, especially machine learning, into system software capabilities provides an opportunity for improving and automating system use, increasing overall productivity, and accelerating scientific discovery and innovation.

PRD 2.1. Design of Operating and Runtime Systems. In EH systems, current operating and runtime systems (ORTSs) are designed to integrate accelerators and special-purpose devices as isolated single-user devices attached over a relatively slow data path. We expect architectural advancements that will allow these devices to be virtualized and more easily shared between users, applications, and processes. We also anticipate advancements that will allow these components to be more tightly integrated on a high-performance data path, such as the memory bus, and to be able to share a uniform view of the virtual address space with all the other devices on a node. Additionally, early EH systems offered only one accelerator, but future EH systems will have multiple accelerators connected with different data paths. The ORTSs must support seamless scalability of these devices and be able to share a uniform view of the virtual address space with all the other devices on a node.

PRD 2.2. Decentralized Resource Management. Resource management decisions are typically handled in a centralized way, if costs and capabilities are homogeneous and constant. However, in EH systems, we expect performance responses of architectural components to be more complex, as they will vary in response to both their designed capabilities and their dynamic and adaptive response to factors such as power, thermal, and contention. Since intelligent resource management capability is being embedded into hardware devices, the system software must adapt to and react to these changes as well. As the number of entities making resource decisions increases, it becomes infeasible to serialize distribution of resources through a single point, such as a single scheduler inside the OS. Resource management policies and strategies at the node and system levels must be enhanced to embrace and exploit these changes.

PRD 2.3. Autonomous Resource Optimization. ARO refers to managing system resources without explicit direction of the resources by the application. To optimize performance and scalability today, users must explicitly place threads and processes on cores and must explicitly divide up memory resources and manage locality. While current techniques may utilize provided information in the form of hints, such information is assumed to be optional. Expecting the user or programmer to explicitly map applications onto the resources of the EH system is becoming untenable. Hence, the motivation behind such ARO systems is usually to reduce the burden placed upon applications to efficiently utilize increasingly complex EH architectures. AROs automatically and dynamically manage resources via runtime system and OS mechanisms and policies. By moving the arduous accounting task of managing the resources needed by millions of threads of execution to system software, we shift responsibilities from applications (e.g., software developers) to computers, which are much more adept at quantitative learning and decision making.

4.3 PRD 3—Modeling and Predicting Performance

Key Questions

Can advanced modeling and simulation capabilities be used to predict (online and offline) the performance characteristics of applications running on emerging hardware technologies and provide insight into the design of future systems?

With a growing number of options for processors, accelerators, networks and memories, optimally configuring a high-performance supercomputer for a wide range of scientific domains becomes overwhelmingly complex. New, intelligent modeling and simulation capabilities that facilitate evaluation of application behavior for novel hardware components would provide important guidance to software developers, programming environments, and intelligent system software, as well as enable potential system designs to be evaluated for their suitability for science and mission needs. This would also allow DOE to consider customized systems that can be tailored to mission-critical needs, thus improving productivity and maximizing the return on investment.

PRD 3.1. Methodology and Tools for Accurate Modeling of Extreme Heterogeneity. The increasing diversity and complexity of EH architectures will force the development and adoption of new performance prediction methodologies and methodologies. Procurements, programming, and optimization are becoming increasingly complex as the EH design space adds technologies, including new accelerators, memories and storage, networks, and beyond–Moore's law components like quantum and neuromorphic computing. These new technological options will make it increasingly difficult to predict performance, power, and reliability; map applications and algorithms to EH architectures; and understand how to optimize applications once these systems are deployed in the field.

PRD 3.2. Cohesive Integration of Modeling and Simulation Infrastructure with Programming Models and Environments. Performance prediction is critical to solving imminent challenges in programming and runtime systems. These predictive tools will be critical in a development phase of algorithm design, will inform the co-design of programming environments for these new devices and systems, and will help runtime systems adapt both to changes in the executing system and to the application's dynamic resource demands. Moreover, the tools will be critical in the production phase, providing cost models to help compilers and programming systems navigate the vast decision space in multidimensional EH architectures.

4.4 PRD 4—Enabling Reproducible Science Despite Diverse Processors and Nondeterminism

Key Questions

What novel methods and techniques are needed to support productive, reliable and verifiable scientific findings in the face of architectural diversity and variability in future systems?

The ability to validate scientific outcomes is essential, but it becomes increasingly difficult as the number and heterogeneity of hardware components grow and computations become increasingly non-determinate and asynchronous. Effectively leveraging the increasing diversity of accelerated processors will require different approaches to computation that have the potential to impact the precision and thus exact reproducibility of numerical calculations. New methods, algorithms, and supporting software infrastructure are needed to enable developers to better reason about, evaluate, and limit the impact of these uncertainties inherent in the variability among diverse hardware components and workloads.

PRD 4.1. Correctness. Correctness, broadly speaking, refers to our ability to predict and validate the behavior of the software and hardware systems we construct. In the HPC context, correctness encompasses not only functionality (how do we know that the system will produce the correct answer) but also performance (how do we know that the system will run in an acceptable period with the resources available). EH architectures further complicate this area by adding non-determinism, variable execution paths, and additional interactions between EH system components, such as memory consistency protocols.

PRD 4.2. Reproducibility. Traditionally, reproducibility means getting the bitwise identical answer when running the same program repeatedly, even when considering different hardware resources or scheduling. We consider this "complete reproducibility." Numerical reproducibility, which is predicated on enforcing IEEE 754-2008 arithmetic for floating point instructions, presents multiple challenges in a parallel environment, because the roundoff errors make floating point arithmetic non-associative. Thus, bitwise reproducibility would require imposing severe constraints on the order of execution for operations with distributed data structures and is usually too costly to be practical at scale. Therefore, some scientific applications achieve scientific replicability when key (but not all) scientific quantities are reproduced within measurable error bounds. We consider this "partial reproducibility," which often suffices to ensure correctness of scientific simulations and discovery. For debugging purposes, some parallel codes implement a mode to enforce bitwise reproducibility at a much lower performance (e.g., with the same order of operations), but they remove this constraint for production runs. In addition, performance reproducibility is desirable in many situations. This can be guaranteed only when the program is built with the same development toolchain and configuration and run in the same execution environment.

4.5 PRD 5—Facilitating Data Management, Analytics, and Workflows

Key Questions

What software infrastructure and tools will be necessary to achieve usable and productive scientific workflow across multiple, different and increasingly complex computing environments?

A scientific campaign relies on multiple simulations and/or experiments, which must be coordinated across an increasingly complex array of scientific instruments, distributed data resources and large geographically distributed teams. The system software environment must improve to facilitate this process across a range and combination of different computing and experimental facilities, from finding and scheduling the available resources to composing

and executing the complete workflow for a broad set

of scientific domains. New and innovative tools will be needed for tracking the scientific process from formulation of a hypothesis to final discovery, spanning both the dynamic and static selection of appropriate computing and data storage resources, analyzing the resulting data, and cataloging experimental results and findings.

PRD 5.1. Mapping Science Workflows to Heterogeneous Hardware and Software Services. EH systems and applications using these systems bring significant challenges to usability through unprecedented variety in the number of data, resources, and services. Paraphrasing from the Future of Scientific Workflows 2015 workshop report, a workflow is defined as the automation of sequencing, orchestrating, and moving data among the subset of scientists' tasks involving the collection, generation, and processing of numerical data. Workflow execution is the realization of systems that aid in the automation of these tasks, freeing the scientist from the details of the process. The participants of this PRD subdivided the topic into two subtopic RDs. This RD focuses on aspects related to scheduling and placement of workflows and data on compute and data resources, and the adjustment of these plans in response to unfolding events such as new data, changes to workflow, or changes in cost or availability of resources.

PRD 5.2. Adapting Workflows and Services Through Learning Approaches. AWs are mechanisms that can be used to independently automate "sequencing, orchestrating, and moving data among the subset of scientists' tasks involving the collection, generation, and processing of numerical data" with some initial human input or knowledge about overall workflow goals and constraints and can then schedule a workflow dynamically so that it meets those goals and constraints. Goals in this context can be specific or general guidelines like minimizing power, maximizing performance. However, the workflow could still operate and degrade gracefully, if it cannot meet those goals and constraints. Constraints are critical to satisfy and are not negotiable.

5. Summary

The fundamental trends in computer architecture predict that nearly all aspects of future HPC architectures will have many more diverse components than past HPC architectures, leading toward a period of EH, including paradigms like machine learning and neuromorphic and quantum computing. In January 2018, ASCR convened a Workshop on Extreme Heterogeneity in HPC. The purpose of this workshop was to identify the PRDs for ASCR in providing a smart software stack that includes techniques, such as deep learning, to make future computers composed of a variety of complex processors, new interconnects, and deep memory hierarchies, which can be used productively by a broad community of computational scientists and help to preserve investments in DOE software and applications. In this regard, significant computer science challenges remain as barriers to efforts to develop a smart software stack that will help increase the usability and programmability of future systems and that will also increase the productivity of the computational scientists. The primary aim for the workshop was to identify the new algorithms and software tools needed from basic research in computer science to enable ASCR's supercomputing facilities to support future scientific and technological advances on the DOE Office of Science's grand challenge problems. In the context of EH, the workshop participants defined basic research needs and opportunities in computer science research to develop smart and trainable operating and runtime systems, prediction techniques, and programming environments that will make future systems easier to tailor to scientists' computing needs and easier for facilities to securely deploy. After the workshop, the organizing committee and breakout representatives merged, collated, and distilled responses along five major areas: (1) programming environments; (2) software development, sustainability, and productivity; (3) operating system and resource management; (4) data management, analytics, and workflows; and (5) modeling and simulation.

Ultimately, the following PRDs were identified by the participants of the Extreme Heterogeneity workshop.

1. Maintaining and Improving Programmer Productivity
    - Flexible, expressive, programming models and languages
    - Intelligent, domain-aware compilers and software development tools
    - Composition of disparate software component content
2. Managing System Resources Intelligently
    - Automated methods using introspection and machine learning
    - Optimize for performance, energy efficiency, and availability
3. Modeling and Predicting Performance
    - Evaluate the impact of potential system designs and application mappings
    - Model-automated optimization of applications
4. Enabling Reproducible Science Despite Diverse Processors and Non-Determinism
    - Methods for validation on non-deterministic architectures
    - Detection and mitigation of pervasive faults and errors
5. Facilitating Data Management, Analytics, and Workflows
    - Mapping science workflows to heterogeneous hardware and software services
    - Adapting workflows and services through machine learning approaches

6. References

[1] A. Almgren, P. DeMar, J. Vetter, K. Riley *et al.*, "Advanced Scientific Computing Research Exascale Requirements Review: An Office Of Science review sponsored by Advanced Scientific Computing Research, September 27–29, 2016, Rockville, Maryland," Argonne National Laboratory (ANL), Argonne, IL (United States). Argonne Leadership Computing Facility, Tech. Rep., 2017. [Online]. Available: https://www.osti.gov/servlets/purl/1375638

[2] A. Arkin, D. Bader, R. Coffey, K. Antypas *et al.*, "Biological and Environmental Research Exascale Requirements Review: An Office of Science review sponsored jointly by Advanced Scientific Computing Research and Biological and Environmental Research, March 28–31, 2016, Rockville, Maryland," US Department of Energy, Washington, DC (United States). Advanced Scientific Computing Research (SC-21), Tech. Rep., 2016. [Online]. Available: https://www.osti.gov/servlets/purl/1375720

[3] A. P. Bhati, S. Wan, D. W. Wright, and P. V. Coveney, "Rapid, accurate, precise, and reliable relative free energy prediction using ensemble based thermodynamic integration," *Journal of Chemical Theory and Computation*, vol. 13, no. (1), pp. 210–222, 2016.

[4] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt *et al.*, "The Gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011. [Online]. Available: http://doi.acm.org/10.1145/2024716.2024718

[5] G. Bosilca, A. Bouteiller, A. Danalis, M. Faverge *et al.*, "Flexible Development of Dense Linear Algebra Algorithms on Massively Parallel Architectures with DPLASMA," in *IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW)*, May 2011, pp. 1432–1441.

[6] G. Bosilca, A. Bouteiller, A. Danalis, M. Faverge, T. He´rault, and J. J. Dongarra, "PaRSEC: Exploiting Heterogeneity to Enhance Scalability," *Computing in Science and Engineering*, vol. 15, no. 6, pp. 36–45, 2013. [Online]. Available: http://dx.doi.org/10.1109/MCSE.2013.98

[7] J. Carlson, M. Savage, R. Gerber, K. Antypas *et al.*, "Nuclear Physics Exascale Requirements Review: An Office of Science review sponsored jointly by Advanced Scientific Computing Research and Nuclear Physics, June 15–17, 2016, Gaithersburg, Maryland," US Department of Energy, Washington, DC (United States). Advanced Scientific Computing Research and Nuclear Physics, Tech. Rep., 2017. [Online]. Available: https://www.osti.gov/servlets/purl/1369223https://www.osti.gov/biblio/1369223/

[8] A. Caulfield, E. Chung, A. Putnam, H. Angepat *et al.*, "A cloud-scale acceleration architecture," in *Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, Oct. 2016. [Online]. Available: https://www.microsoft.com/en-us/research/publication/configurable-cloud-acceleration/

[9] C.-S. Chang, M. Greenwald, K. Riley, K. Antypas *et al.*, "Fusion Energy Sciences Exascale Requirements Review: An Office of Science review sponsored jointly by Advanced Scientific Computing Research and Fusion Energy Sciences, January 27–29, 2016, Gaithersburg, Maryland," USDOE Office of Science (SC), Washington, DC (United States). Offices of Advanced Scientific Computing Research and Fusion Energy Sciences, Tech. Rep., 2017. [Online]. Available: https://www.osti.gov/servlets/purl/1375639https://www.osti.gov/biblio/1375639/

[10] J. Demmel, P. Ahrens, and H. D. Nguyen, "Efficient reproducible floating point summation and BLAS," Univ. of California at Berkeley, Tech. Rep. UCB/EECS-2016-121, 2016. [Online]. Available: https://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-121.html

[11] M. Deveci, S. Rajamanickam, V. J. Leung, K. Pedretti, S. L. Olivier, D. P. Bunde, U. V. Catalyurek, and K. Devine, "Exploiting geometric partitioning in task mapping for parallel computers," in IPDPS, PHOENIX (Arizona) USA, May 2014.

[12] R. Gerber, J. Hack, K. Riley, K. Antypas et al., "Crosscut Report: Exascale Requirements Reviews, March 9–10, 2017–Tysons Corner, Virginia: An Office of Science Review sponsored by Advanced Scientific Computing Research, Basic Energy Sciences, Biological and Environmental Research, Fusion Energy Sciences, High Energy Physics, Nuclear Physics," Oak Ridge National Lab. (ORNL), Oak Ridge, TN (United States); Argonne National Lab. (ANL), Argonne, IL (United States); Lawrence Berkeley National Lab. (LBNL), Berkeley, CA (United States), Tech. Rep., 2018. [Online]. Available: https://www.osti.gov/servlets/purl/1417653

[13] B. Goglin, "Managing the Topology of Heterogeneous Cluster Nodes with Hardware Locality (hwloc)," in Proceedings of 2014 International Conference on High Performance Computing & Simulation (HPCS 2014), Bologna, Italy, Jul. 2014. [Online]. Available: http://hal.inria.fr/ hal-00985096

[14] B. Goglin, J. Hursey, and J. M. Squyres, "netloc: Towards a Comprehensive View of the HPC System Topology," in Proceedings of the Fifth International Workshop on Parallel Software Tools and Tool Infrastructures (PSTI 2014), held in conjunction with ICPP-2014, Minneapolis, MN, Sep. 2014. [Online]. Available: http://hal.inria.fr/hal-01010599

[15] S. Habib, R. Roser, R. Gerber, K. Antypas et al., "High Energy Physics Exascale Requirements Review: An Office of Science review sponsored jointly by Advanced Scientific Computing Research and High Energy Physics, June 10–12, 2015, Bethesda, Maryland," US DOE Office of Science (SC) (United States), Tech. Rep., 2016. [Online]. Available: https://www.osti.gov/servlets/purl/1341722https: //www.osti.gov/biblio/1341722/

[16] W. Heirman, A. Isaev, and I. Hur, "Sniper: Simulation-based instruction-level statistics for optimizing software on future architectures," in Proceedings of the 3rd International Conference on Exascale Applications and Software, ser. EASC '15. Edinburgh, Scotland, UK: University of Edinburgh, 2015, pp. 29–31. [Online]. Available: http://dl.acm.org/citation.cfm?id=2820083.2820089

[17] A. M. Heroux, G. Allen et al., "Computational Science and Engineering Software Sustainability and Productivity (CSESSP) Challenges Workshop Report," Sep. 2016, available via https://www.nitrd.gov/PUBS/CSESSPWorkshopReport.pdf. [Online]. Available: NetworkingandInformationTechnologyResearchandDevelopment(NITRD)Program

[18] M. Heroux, L. C. McInnes, D. Moulton, and et al. D. Bernholdt, H. Johansen, "Interoperable Design of Extreme-scale Application Software (IDEAS): Advancing Software Productivity for Extreme-scale Science, Progress Report." [Online]. Available: https://ideas-productivity.org/

[19] T. Hoefler, R. Rabenseifner, H. Ritzdorf, B. R. de Supinski, R. Thakur, and J. L. Traeff, "The Scalable Process Topology Interface of MPI 2.2," Concurrency and Computation: Practice and Experience, vol. 23, no. 4, pp. 293–310, Aug. 2010.

[20] A. Hoisie, A. Srivastava et al., "Workshop on Modeling and Simulation of Systems and Applications," DOE Advanced Scientific Computing Research, Tech. Rep., 2014, https://science.energy.gov//media/ascr/pdf/program-documents/docs/ModSim 2014 Report FINAL.pdf

[21] M. Horowitz, C.-K. K. Yang, and S. Sidiropoulos, "High-speed electrical signaling: Overview and limitations," *IEEE Micro*, vol. 18, pp. 12–24, 1998.

[22] J. Hu and R. Marculescu, “Energy-aware mapping for tile-based NoC architectures under performance constraints,” in *Proceedings of the 2003 Asia and South Pacific Design Automation Conference*, ser. ASP-DAC ’03, 2003, pp. 233–239.

[23] A. Huang, “Moore’s aw is dying (and that could be good),” *Spectrum, IEEE*, vol. 52, no. 4, pp. 43–47, 2015.

[24] F. Inc., “Introducing big basin: Facebook’s next-generation AI hardware,” 2017. [Online]. Available: https://code.facebook.com/posts/1835166200089399/ introducing-big-basin-our-next-generation-ai-hardware/

[25] S. S. Institute, ”https://www.software.ac.uk/”.

[26] Intel Corporation, “Threading Building Blocks,” https://www.threadingbuildingblocks.org/.

[27] A. Jain, S. Lloyd, and M. Gokhale, “LiME: An open source memory system emulation platform,” in *Open Source Computing Workshop: OpenSuCo-SC17*, ser. SC17: The International Conference for High Performance Computing, Networking, Storage and Analysis, 2017.

[28] R. G. James J. Hack, Katherine Riley, “Crosscut Report: Exascale Requirements Reviews,” DOE Advanced Scientific Computing Research, Tech. Rep., 2017. [Online]. Available: http://exascaleage.org/

[29] D. I. Jeon and K. S. Chung, “CasHMC: A cycle-accurate simulator for hybrid memory cube,” *IEEE Computer Architecture Letters*, vol. 16, no. 1, pp. 10–13, Jan. 2017.

[30] H. Johansen, L. C. McInnes, D. Bernholdt, J. Carver, M. Heroux, R. Hornung, P. Jones, B. Lucas, A. Siegel, and T. Ndousse-Fetter, “Software Productivity for Extreme-Scale Science,” 2014, report on DOE Workshop, January 13-14, 2014, available via http://www.orau.gov/swproductivity2014/SoftwareProductivityWorkshopReport2014.pdf.

[31] N. P. Jouppi, C. Young, N. Patil, D. Patterson *et al.*, “In-datacenter performance analysis of a tensor processing unit,” in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ser. ISCA ’17. New York, NY, USA: ACM, 2017, pp. 1–12. [Online]. Available: http://doi.acm.org/10.1145/3079856.3080246

[32] H. Kaul, M. Anders, S. Hsu, A. Agarwal, R. Krishnamurthy, and S. Borkar, “Near-threshold voltage (NTV) design: Opportunities and challenges,” in *Proceedings of the 49th Annual Design Automation Conference*, ser. DAC ’12. New York, NY, USA: ACM, 2012, pp. 1153–1158. [Online]. Available: http://doi.acm.org/10.1145/2228360.2228572

[33] D. E. Keyes, L. C. McInnes, C. Woodward, W. Gropp *et al.*, “Multiphysics simulations: Challenges and opportunities,” *International Journal of High Performance Computing Applications*, vol. 27, no. 1, pp. 4–83, Feb. 2013, special issue, available via https://dx.doi.org/10.1177/1094342012468181.

[34] M. H. Kryder and K. C. Soo, “After hard drives: What comes next?” *IEEE Transactions on Magnetics*, vol. 45, no. 10, pp. 3406–3413, 2009.

[35] A. Lavin, “Fast algorithms for convolutional neural networks,” *CoRR*, vol. abs/1509.09308, 2015. [Online]. Available: http://arxiv.org/abs/1509.09308

[36] M. Law and R. Colwell, “The chip design game at the end of Moore’s law,” *Hot Chips*, no. August, pp. Key Note–Key Note, 2013.

[37] J. D. Leidel and Y. Chen, “HMC-sim-2.0: A simulation platform for exploring custom memory cube operations,” *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 621–630, 2016.

[38] C. Mack, “The multiple lives of Moore’s law,” *Spectrum, IEEE*, vol. 52, no. 4, pp. 31–31, 2015.

[39] J. A. Mandelman, R. H. Dennard, G. B. Bronner, J. K. DeBrosse, R. Divakaruni, Y. Li, and C. J. Radens, "Challenges and future directions for the scaling of dynamic random-access memory (DRAM)," *IBM Journal of Research and Development*, vol. 46, no. 2.3, pp. 187–212, 2002.

[40] I. L. Markov, "Limits on fundamental limits to computation," *Nature*, vol. 512, no. 7513, pp. 147–154, 2014.

[41] N. McDonald, "Simersim: A flexible event-driven cycle-accurate network simulator," 2017, https://github.com/HewlettPackard/supersim.

[42] D. A. B. Miller and H. M. Ozaktas, "Limit to the bit-rate capacity of electrical interconnects from the aspect ratio of the system architecture," *J. Parallel Distrib. Comput.*, vol. 41, no. 1, pp. 42–52, 1997.

[43] D. A. B. Miller, "Rationale and challenges for optical interconnects to electronic chips," in *Proc. IEEE*, 2000, pp. 728–749.

[44] R. B. R. Misbah Mubarak, Nikhil Jain, Jens Domke, Noah Wolfe, Caitlin Ross, Kelvin Li, Abhinav Bhatele, Christopher D. Carothers, Kwan-Liu Ma, "Toward reliable validation of HPC network simulation models," in *Proceedings of the 2017 Winter Simulation Conference*, 2017.

[45] S. Mittal and J. S. Vetter, "A survey of software techniques for using non-volatile memories for storage and main memory systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 5, pp. 1537–1550, 2016.

[46] G. E. Moore, "Cramming more components onto integrated circuits," *Electronics*, vol. 38, no. 8, pp. 1–4, 1965.

[47] S. Murali and G. D. Micheli, "Bandwidth-constrained mapping of cores onto NoC architectures," in *Proceedings of the Conference on Design, Automation and Test in Europe – Volume 2*, ser. DATE '04, 2004.

[48] T. Narumi, Y. Ohno, N. Okimoto, T. Koishi *et al.*, "A 55 TFLOPS simulation of amyloid-forming peptides from yeast prion sup35 with the special-purpose computer system MDGRAPE-3," in *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, ser. SC '06. New York, NY, USA: ACM, 2006. [Online]. Available: http://doi.acm.org/10.1145/1188455.1188506

[49] OCR Developers, "Open Community Runtime," https://01.org/open-community-runtime.

[50] S. L. Olivier, A. K. Porterfield, K. B. Wheeler, M. Spiegel, and J. F. Prins, "OpenMP task scheduling strategies for multicore NUMA systems," *International Journal of High Performance Computing Applications*, vol. 26, no. 2, pp. 110–124, May 2012.

[51] M. Plagge, C. D. Carothers, and E. Gonsiorowski, "Nemo: A massively parallel discrete-event simulation model for neuromorphic architectures," in *Proceedings of the 2016 annual ACM Conference on SIGSIM Principles of Advanced Discrete Simulation*. ACM, 2016, pp. 233–244.

[52] M. Poremba, T. Zhang, and Y. Xie, "NVMain 2.0: A user-friendly memory simulator to model (non-)volatile memory systems," *IEEE Computer Architecture Letters*, vol. 14, no. 2, pp. 140–143, Jul. 2015.

[53] R. Preissl, T. M. Wong, P. Datta, M. Flickner, R. Singh, S. K. Esser, W. P. Risk, H. D. Simon, and D. S. Modha, "Compass: A scalable simulator for an architecture for cognitive computing," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society Press, 2012, p. 54.

[54] A. Putnam, "FPGAs in the datacenter: Combining the worlds of hardware and software development," in *Proceedings of the on Great Lakes Symposium on VLSI 2017*, ser. GLSVLSI '17. New York, NY, USA: ACM, 2017, pp. 5–5. [Online]. Available: http://doi.acm.org/10.1145/3060403.3066860

[55] A. F. Rodrigues, K. S. Hemmert, B. W. Barrett, C. Kersey *et al.*, "The structural simulation toolkit," *SIGMETRICS Perform. Eval. Rev.*, vol. 38, no. 4, pp. 37–42, Mar. 2011. [Online]. Available: http://doi.acm.org/10.1145/1964218.1964225

[56] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "DRAMSim2: A cycle accurate memory system simulator," in *IEEE Computer Architecture Letters*. IEEE Computer Society, 2011.

[57] D. Sanchez and C. Kozyrakis, "ZSim: Fast and accurate microarchitectural simulation of thousand-core systems," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ser. ISCA '13. New York, NY, USA: ACM, 2013, pp. 475–486. [Online]. Available: http://doi.acm.org/10.1145/2485922.2485963

[58] D. Sanchez, G. Michelogiannakis, and C. Kozyrakis, "An analysis of interconnection networks for large scale chip-multiprocessors," *ACM Transactions on Architecture and Code Optimization*, vol. 7, no. 1, pp. 4:1–4:28, 2010.

[59] J. Shalf, S. S. Dosanjh, and J. Morrison, "Exascale computing technology challenges," in *International Meeting on High Performance Computing for Computational Science*, ser. Lecture Notes in Computer Science, vol. 6449. Springer, 2010, pp. 1–25.

[60] J. M. Shalf and R. Leland, "Computing beyond Moore's law," *IEEE Computer*, vol. 48, no. 12, pp. 14–23, 2015. [Online]. Available: https://doi.org/10.1109/MC.2015.374

[61] Y. S. Shao, S. L. Xi, V. Srinvisan, G.-Y. Wei, and D. Brooks, "Co-designing accelerators and SOC interfaces using Gem5-Aladdin," *International Symposium on Microarchitecture (MICRO)*, Jun. 2016.

[62] D. E. Shaw, J. P. Grossman, J. A. Bank, B. Batson *et al.*, "Anton 2: Raising the bar for performance and programmability in a special-purpose molecular dynamics supercomputer," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '14. Piscataway, NJ, USA: IEEE Press, 2014, pp. 41–53. [Online]. Available: https://doi.org/10.1109/SC.2014.9

[63] Z. Shi and A. Burns, "Schedulability analysis and task mapping for real-time on-chip communication," *Real-Time Syst.*, vol. 46, no. 3, pp. 360–385, Dec. 2010.

[64] K. L. Spafford and J. S. Vetter, "Aspen: A domain specific language for performance modeling," in *SC12: International Conference for High Performance Computing, Networking, Storage and Analysis*, Salt Lake City, 2012, pp. 1–11.

[65] D. Unat, A. Dubey, T. Hoefler, J. Shalf *et al.*, "Trends in data locality abstractions for HPC systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 10, pp. 3007–3020, 2017. [Online]. Available: https://doi.org/10.1109/TPDS.2017.2703149

[66] D. Unat, J. Shalf, T. Hoefler, A. Dubey, and T. Schulthess, 2014. [Online]. Available: http://www.padalworkshop.org/

[67] J. S. Vetter and S. Mittal, "Opportunities for nonvolatile memory systems in extreme-scale high performance computing," *Computing in Science and Engineering*, vol. 17, no. 2, pp. 73–82, 2015.

[68] M. M. Waldrop, "The chips are down for Moore's law," *Nature News*, vol. 530, no. 7589, p. 144, 2016.

[69] S. Williams, A. Waterman, and D. Patterson, "Roofline: an insightful visual performance model for multicore architectures," *Communications of the ACM*, vol. 52, no. 4, pp. 65–76, 2009.

[70] T. Windus, M. Banda, T. Devereaux, J. White *et al.*, "Basic Energy Sciences Exascale Requirements Review. An Office of Science review sponsored jointly by Advanced Scientific Computing Research and Basic Energy Sciences, November 3–5, 2015, Rockville, Maryland," US Department of Energy,

Washington, DC (United States). Advanced Scientific Computing Research and Basic Energy Sciences, Tech. Rep., 2017. [Online]. Available: https://www.osti.gov/servlets/purl/1341721

[71] L. Xia, B. Li, T. Tang, P. Gu, P.-Y. Chen, S. Yu, Y. Cao, Y. Wang, Y. Xie, and H. Yang, "MNSIM: Simulation platform for memristor-based neuromorphic computing system," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2017.

[72] Y. Yan, J. Zhao, Y. Guo, and V. Sarkar, "Hierarchical place trees: A portable abstraction for task parallelism and data movement," in *Proceedings of the 22nd International Workshop on Languages and Compilers for Parallel Computing*, Oct. 2009.

[73] X. Yang, J. Jenkins, M. Mubarak, R. B. Ross, and Z. Lan, "Watch out for the bully! Job interference study on dragonfly network," *SC16: International Conference for High Performance Computing, Networking, Storage and Analysis*, vol. 00, pp. 750–760, 2016.

[74] W. Zhou, Y. Zhang, and Z. Mao, "An application specific NoC mapping for optimized delay," in *Design and Test of Integrated Systems in Nanoscale Technology, 2006. DTIS 2006. International Conference on*, Sep. 2006, pp. 184–188.

Appendix A: Charge Letter

The charge letter is appended below.

Department of Energy
Office of Science
Washington, DC 20585

September 28, 2017

Dr. Jeffrey Vetter
ORNL Distinguished R&D Staff Scientist
Future Technologies Group Leader
Oak Ridge National Laboratory
One Bethel Valley Road
Bldg. 5100, MS-6173
Oak Ridge, TN 37831-6173

Jeff,

Thank you for agreeing to be the overall chair for the ASCR workshop focused on the Basic Research Needs for Extreme Heterogeneity. Per our discussions, the workshop will be held on January 23-25, 2018, at the Gaithersburg Marriott Washingtonian Center in Gaithersburg, MD (north of Washington DC). This email confirms ASCR's invitation for you to lead this important ASCR activity.

The workshop will follow the model used by SC's Basic Energy Sciences program with their Basic Research Needs (BRN) workshops. As you know, critical to ASCR's success are the in-person meeting of a broad group of participants from the community and the development of a report that outlines the priority research directions, as identified by the participants, in providing a smart software stack that makes future computers composed of a variety of complex processors and accelerators, new interconnects and deep memory hierarchies productive for future science. The BRN workshops are typically 2.5 days. On the last morning, the panel leads present the priority research directions identified by their panel to the entire group. The afternoon of the third day is reserved for writing by the chairs, panel leads, and other writers who may have been selected by the group.

The draft charge for the workshop is:

> The purpose of this workshop is to identify the priority research directions for ASCR in providing a smart software stack that includes artificial intelligence techniques to make future computers composed of a variety of complex processors, new interconnects and deep memory hierarchies easily used by a broad community of computational scientists. In 2009, the drive to deploy more energy efficient computers led the Oak Ridge Leadership Computing Facility (OLCF) to propose a system upgrade, composed of CPUs coupled with GPUs, which ushered in a new era of heterogeneous high performance computing. The

planned Summit upgrade at the OLCF is composed of one CPU coupled to three GPUs and has at least three types of memory. A recent analysis of vendors' current architectural roadmaps is consistent with the increasing heterogeneity ASCR is seeing in its computing upgrades and indicate that future computers will be more complex and composed of a variety of processing units and accelerators supported by open interconnects and deep memory hierarchies, in other words extremely heterogeneous.

Scientifically, past DOE investments in applied mathematics and computer science basic research and in programs like SciDAC have broadened the community of computational scientists using HPC as one tool to address their grand challenge problems. Nevertheless, significant computer science challenges remain as barriers to efforts s to develop a smart software stack that will help increase the usability and programmability of future systems and that will also increase the productivity of the computational scientists. The primary aim for the workshop is to identify the new algorithms and software tools needed from basic research in computer science to enable ASCR 's supercomputing facilities to support future scientific and technological advances on SC program' s grand challenge problems. ASCR's grand challenges and the resulting priority basic research directions should be identified by spanning existing and next generation computer architectures, including novel technologies that may be developed in the "Post-Moore's Law era" and the promising tools and techniques that are essential to effective utilization. The workshop and subsequent report should define basic research needs and opportunities in computer science research to develop smart and trainable operating and runtime systems, execution models, and programming environments that will make future systems easier to tailor to scientists' computing needs and for facilities to securely deploy.

The chair and co-chairs are responsible for leading the entire workshop planning process. The overall tasks are listed below in approximate chronological order. We will schedule regular conference calls among the chair, co-chairs, and DOE to start the planning process beginning next week.

- Develop the high-level workshop structure, including deciding on the number and focus of the panels. Based on the meeting venue, we can have up to 3 panels.
- Based on the panel topics, identify possible plenary topics and speakers,
- Work with DOE to identify panel leads, and then work with the panel leads to identify the workshop participants, including a plan to engage a broad range of DOE Lab personnel, academics and industry representative. Ideally, this plan will provide for inclusion of people who have not participated in ASCR's workshops before. This is a time consuming process that we should begin as soon as possible in order to get the meeting on people's calendars.
- As soon as possible, coordinate preparation of a background document on the status of the field that would be distributed to participants ahead of the workshop. DOE program managers from ASCR will participate in preparing this document.
- During the workshop, synthesize the panels' ideas, guide the identification and definition of priority research directions, and coordinate an oral report to the full workshop at the closing session.

- Critically, coordinate and integrate the topical narratives provided by the panel leads and other identified writers into a final report As much of the writing as possible is to be completed during the workshop, but follow-up writing is almost always required. ASCR will support a technical editor to help finalize the document.

The goal is to have a final report within 3 months after the workshop in order to maximize the report's impact on programmatic planning.

We really appreciate your willingness to lead this essential planning activity for ASCR.

Barbara Helland
Associate Director
Advanced Scientific Computing Research
Office of Science

Appendix B: Workshop Agenda

The final workshop agenda is appended below. Note that when the workshop was transformed to a virtual online workshop (see Appendix B and Section 1.1), the agenda was adapted to accommodate several time zones and other factors.

ASCR Computer Science Workshop on Extreme Heterogeneity, January 23–25

Revised Virtual Agenda

Revised January 25, 2018, 14:43 EST

All times are Eastern Standard Time

Tuesday, January 23, 2018	
10:00 – 10:15	Introductions: Lucy Nowell and Jeffrey Vetter
10:15 – 10:35	**Welcome and ASCR Update** Barbara Helland, Director, Advanced Scientific Computing Research
10:35– 11:05	View from ASCR Research Division – Steve Lee, Acting Division Director
11:05 – 11:35	Invited Plenary Talk: IEEE Rebooting Computing – Tom Conte
11:35– 11:45	Break
11:45 – 12:15	Invited Talk: Architectural Trends and System Design Issues – Bob Colwell
12:15– 1:30	FSD Introduction to Extreme Heterogeneity – Jeffrey Vetter, John Shalf, and Maya Gokhale + FSD section owners
1:45 – 2:30	Break for lunch
2:30 – 3:15	Invited Talk: Report on the ASCAC Future of Computing Study – Maya Gokhale
3:15 – 4:30	Panel on Issues Raised by Extreme Heterogeneity –Moderator Ron Brightwell Usability, Understandability and Programmability – Salman Habib Operating and Runtime Systems – Ron Brightwell Data Analytics – Wes Bethel EH Workflow Management – Ewa Deelman Open Q&A
5:00 – 5:10	Break
5:10 – 7:00	Switch to BOG mode Breakout 1: • BOG 1: Programming Environments, Models, and Language – Moderators Alex Aiken and Pat McCormick • BOG 2: Data Management and I/O – Moderators Rob Ross and Suren Byna • BOG 3: Data Analytics and Workflows – Tom Peterka and S. J. Yoo
7:00	Adjourn for the day

Wednesday, January 24, 2018	
Session Chair: John Shalf	
10:00 – 10:30	Invited Plenary Talk: Memory Systems and I/O - Bruce Jacob
10:30 – 11:00	Invited Plenary Talk: Beyond CMOS Workshops - Neena Imam
11:00 – 11:30	Exascale Computing Project Computer Science R&D for Heterogeneity – Mike Heroux Session Chair: Maya Gokhale
11:30 - 12:00	Invited Plenary Talk: Quantum Computing in ASCR – Ceren Susut
12:00 – 12:10	Break
12:10 – 12:40	Q&A/Discussion with morning speakers
12:40 - 1:10	Invited Plenary Talk – ASCR Scientific Machine Learning Workshop
1:10 - 1:45	Break for lunch
1:45 – 2:30	Report Back from Breakout Session #1 and Discussion – Moderator: Jeffrey Vetter
2:30 - 4:30	Switch to BOG mode Breakout 2: • BOG 4: Operating Systems and Resource Management – Moderator Ron Brightwell • BOG 5: Software Development Methodologies – Moderator Sherry Li • BOG 6: Modeling and Simulation for Hardware Characterization – Moderators Andrew Chien and David Donofrio
4:30 - 4:40	Break
4:40 – 6:40	Breakout 3: • BOG 7: Programming Environments: Compilers, Libraries and Runtimes – Moderators Michelle Strout and Barbara Chapman • BOG 3: Data Analytics and Workflows – Moderator Christine Sweeney • BOG 8: System Management, Administration and Job Scheduling – Moderators Paul Peltz and Rebecca Hartman-Baker • BOG 9: Crosscut: Productivity, Composability, and Interoperability – Moderator Bob Lucas
6:40	Adjourn for the Day

Thursday, January 25, 2018	
10:00 – 11:35	Report Back from Wednesday Breakouts –Moderator: Katie Schuman
11:35 – 11:40	Break
11:40 – 1:30	Switch to BOG mode Breakout 4: • BOG 1: Programming Environments: Abstractions, Models and Languages – Moderator Pat McCormick • BOG 6: Crosscut: Modeling and Simulation – Moderators Jeremy Wilke and Zhiling Lan • BOG 10: Operating Systems and Resource Management: Locality and Programming Environment Support – Moderator Mike Lang • BOG 11: Crosscut: Portability, Code Reuse and Performance Portability – Moderator Anshu Dubey
1:30 – 2:00	Break for lunch
2:00 – 3:50	Breakout 5: • BOG 2: Data Management and I/O – Moderator Rob Ross • BOG 12: Programming Environments: Debugging, Autotuning, and • Specialization – Moderators Mary Hall and John Mellor-Crummey • BOG 13: Crosscut: Resilience and Power Management – Moderators Franck Cappello and Kirk Cameron
3:50– 4:00	Break
4:00- 4:30	Report backs from those technical areas that have not presented • BOG 11 • BOG 12 • BOG 13
4:30	Closing Session – Lucy Nowell and Jeffrey Vetter Next steps and timeline
4:45	Adjourn
4:50	OC and BOG representatives move to writing session

Appendix C: Workshop Participants

The following is a list of all registered participants for the face-to-face workshop in Washington, DC. However, as discussed in Appendix B and Section 1.1, the workshop was quickly transformed into a virtual online workshop, making it difficult to measure the full level of participation. For example, many organizations, like the DOE labs, allowed open participation for staff in scheduled conference rooms, which probably increased participation greatly. The list represents a *lower bound* on the number of participants.

Workshop Participants

First Name	Last Name	Affiliation
Gagan	Agrawal	Ohio State University
Alex	Aiken	Stanford University/Stanford Linear Accelerator Center
Esameldin	Aly	University of Kansas
Scott	Baden	Lawrence Berkeley National Laboratory
Ray	Bair	Argonne National Laboratory
Pavan	Balaji	Argonne National Laboratory
Kevin	Barker	Pacific Northwest National Laboratory
Jonathan	Beard	Arm Inc.
Peter	Beckman	Argonne National Laboratory
John	Bell	Lawrence Berkeley National Laboratory
David	Bernholdt	Oak Ridge National Laboratory
Wes	Bethel	Lawrence Berkeley National Laboratory
Jay	Billings	Oak Ridge National Laboratory
George	Biros	University of Texas, Austin
Laura	Biven	US Department of Energy
Shekhar	Borkar	Qualcomm
Ron	Brightwell	Sandia National Laboratories
Anastasiia	Butko	Lawrence Berkeley National Laboratory
Suren	Byna	Lawrence Livermore National Laboratory
Kirk	Cameron	Virginia Tech University
Franck	Cappello	Argonne National Laboratory
William	Carlson	IDA Center for Computing Sciences
Richard	Carlson	US Department of Energy
Jeronimo	Castrillon	The Technische Universität Dresden
Barbara	Chapman	Brookhaven National Laboratory
Andrew	Chien	Argonne National Laboratory
Alok	Choudhary	Northwestern University
Edmond	Chow	Georgia Institute of Technology
Almadena	Chtchelkanova	National Science Foundation
Robert	Colwell	Lawrence Berkeley National Laboratory
Tom	Conte	Georgia Institute of Technology
Jeanine	Cook	Sandia National Laboratories
Claire	Cramer	US Department of Energy
Candace	Culhane	Los Alamos National Laboratory
John	Daly	US Department of Defense

First Name	Last Name	Affiliation
Tina	Declerck	Lawrence Berkeley National Laboratory/National Energy Research Scientific Computing Center
Ewa	Deelman	University of Southern California
Ronald	DeMara	University of Central Florida
David	Donofrio	Lawrence Berkeley National Laboratory
Anshu	Dubey	Argonne National Laboratory
Christian	Engelmann	Oak Ridge National Laboratory
Mattan	Erez	University of Texas, Austin
Robert	Falgout	Lawrence Livermore National Laboratory
Nicola	Ferrier	Argonne National Laboratory
Michael	Garland	NVIDIA
Balazs	Gerofi	RIKEN Advanced Institute For Computational Science
Andreas	Gerstlauer	University of Texas, Austin
Maya	Gokhale	Lawrence Livermore National Laboratory
Gary	Grider	Los Alamos National Laboratory
Salman	Habib	Argonne National Laboratory
Mary	Hall	University of Utah
William	Harrod	US Department of Energy
Rebecca	Hartman-Baker	Lawrence Berkeley National Laboratory/ National Energy Research Scientific Computing Center
Michael	Heroux	Sandia National Laboratories
Thuc	Hoang	The National Nuclear Security Administration
Adolfy	Hoisie	Brookhaven National Laboratory
Travis	Humble	Oak Ridge National Laboratory
Wen-mei	Hwu	University of Illinois at Urbana-Champaign
Costin	Iancu	Lawrence Berkeley National Laboratory
Khaled	Ibrahim	Lawrence Berkeley National Laboratory
Neena	Imam	Oak Ridge National Laboratory
Latchesar	Ionkov	Los Alamos National Laboratory
Kamil	Iskra	Argonne National Laboratory
Bruce	Jacob	University of Maryland
Douglas	Jacobsen	Lawrence Berkeley National Laboratory/National Energy Research Scientific Computing Center
Terry	Jones	Oak Ridge National Laboratory
Jim	Kahle	IBM
Laurence	Kaplan	Cray Inc.
Ian	Karlin	Lawrence Livermore National Laboratory
Scott	Klasky	Oak Ridge National Laboratory
Brian	Kocoloski	Washington University in St. Louis
Sriram	Krishnamoorthy	Pacific Northwest National Laboratory
Zhiling	Lan	Illinois Institute of Technology
Michael	Lang	Los Alamos National Laboratory
Randall	Laviolette	US Department of Energy
Cara	Leckey	National Aeronautics and Space Administration
Seyong	Lee	Oak Ridge National Laboratory
Steven	Lee	US Department of Energy

First Name	Last Name	Affiliation
John	Leidel	Tactical Computing Laboratories
Richard	Lethin	Reservoir Labs, Inc.
Vitus	Leung	Sandia National Laboratories
Hai	Li	Duke University
Lingda	Li	Brookhaven National Laboratory
Sherry	Li	Lawrence Berkeley National Laboratory
Meifeng	Lin	Brookhaven National Laboratory
Gerald	Lofstead	Sandia National Laboratories
Robert	Lucas	University of Southern California/ISI
Andrew	Lumsdaine	Pacific Northwest National Laboratory
Abid	Malik	Brookhaven National Laboratory
Carlos	Maltzahn	University of California, Santa Cruz
Andres	Marquez	Pacific Northwest National Laboratory
Satoshi	Matsuoka	Tokyo Institute of Technology
Sonia	McCarthy	US Department of Energy
Patrick	McCormick	Los Alamos National Laboratory
Lois	McInnes	Argonne National Laboratory
John	Mellor-Crummey	Rice University
Tiffany	Mintz	Oak Ridge National Laboratory
Shirley	Moore	Oak Ridge National Laboratory
Kenneth	Moreland	Sandia National Laboratories
David	Mountain	US Department of Defense
Lucy	Nowell	US Department of Energy
Eun Jung	Park	Los Alamos National Laboratory
Valerio	Pascucci	University of Utah/Scientific Computing and Imaging Institute
Steve	Pawlowski	Micron
Kevin	Pedretti	Sandia National Laboratories
Paul	Peltz Jr.	Los Alamos National Laboratory
Thomas	Peterka	Argonne National Laboratory
Robinson	Pino	US Department of Energy
Raphael	Pooser	Oak Ridge National Laboratory
Line	Pouchard	Brookhaven National Laboratory
Lavanya	Ramakrishnan	Lawrence Berkeley National Laboratory
David	Richards	Lawrence Livermore National Laboratory
Katherine	Riley	Argonne National Laboratory
Robert	Roser	Fermi National Accelerator Laboratory
Robert	Ross	Argonne National Laboratory
Christopher	Rossbach	University of Texas, Austin
Barry	Roundtree	Lawrence Livermore National Laboratory
Florin	Rusu	University of California, Merced
Catherine	Schuman	Oak Ridge National Laboratory
John	Shalf	Lawrence Berkeley National Laboratory
Sameer	Shende	University of Oregon
Mark	Sims	US Department of Defense
Shuaiwen	Song	Pacific Northwest National Laboratory

First Name	Last Name	Affiliation
Julie	Stambaugh	US Department of Energy
Thomas	Sterling	Indiana University
Michelle	Strout	University of Arizona
Ceren	Susut	US Department of Energy
Christine	Sweeney	Los Alamos National Laboratory
Keita	Teranishi	Sandia National Laboratories
Angie	Thevenot	US Department of Energy
Stanimire	Tomov	University of Tennessee, Knoxville
Antonino	Tumeo	Pacific Northwest National Laboratory
Brian	Van Essen	Lawrence Livermore National Laboratory
Eric	Van Hensbergen	Arm Limited
Brian	Van Straalen	Lawrence Berkeley National Laboratory
Dilip	Vasudevan	Lawrence Berkeley National Laboratory
Jeffrey	Vetter	Oak Ridge National Laboratory
Uzi	Vishkin	University of Maryland/Institute for Advanced Computer Studies
David	Wang	Samsung Semiconductor
Noah	Watkins	University of California, Santa Cruz
Jeremiah	Wilke	Sandia National Laboratories
Samuel	Williams	Lawrence Berkeley National Laboratory
Matthew	Wolf	Oak Ridge National Laboratory
Nicholas	Wright	Lawrence Berkeley National Laboratory
John	Wu	Lawrence Berkeley National Laboratory
Sudharkar	Yalamanchili	Georgia Institute of Technology
Anthony	Yau	Department of Defense
Shinjae	Yoo	Brookhaven National Laboratory
Andrew	Younge	Sandia National Laboratories

Appendix D: Call for Position Papers: Extreme Heterogeneity

Important Dates

- December 1, 2017: Position paper submission deadline
- December 7, 2017: Contact authors will be notified of selection for workshop
- January 23–25, 2018: Workshop dates
- Workshop URL: http://orau.gov/exheterogeneity2018/papers.htm
- Submission URL: https://easychair.org/conferences/?conf=weh2018

Motivation

On behalf of the Advanced Scientific Computing Research (ASCR) program in the US Department of Energy (DOE) Office of Science and ASCR Program Manager Dr. Lucy Nowell, we are organizing a Workshop on Extreme Heterogeneity (EH). EH is the result of using multiple types of processors, accelerators, and memory/storage in a single computing platform that must support a variety of application workflows to meet the needs of increasingly diverse scientific domains. Extremely heterogeneous supercomputers will be acquired by the ASCR-supported computing facilities as we reach the end of Moore's Law while still facing rapidly increasing computational and data-intensive demands. The ASCR Computer Science research focus for this workshop is on system software and software development tools and environments for supercomputers that will be delivered for operational use from 2025 to 2040.

The purpose of the EH workshop is to more clearly define the challenges that EH presents to the software stack and the scientific programming environment and to identify related computer science priority research directions that are essential to making extremely heterogeneous systems useful, usable, efficient, and secure for science applications and DOE mission requirements.

The workshop aims to identify and prioritize research directions by analyzing existing and next-generation computer architectures. The workshop will target post-exascale architectures including novel technologies that might be developed in the "Post-Moore's Law era" and promising tools and techniques, such as advanced analytics and machine learning, that might enable efficient and productive use of such architectures. Participants will also discuss options to leverage methods developed by industry, such as approaches to improved developer productivity for Big Data.

We anticipate participation by personnel from universities, industry, and DOE national laboratories. The workshop will feature a variety of plenary talks and multiple breakout sessions, with every invitee expected to participate actively in discussion of potential research directions.

Workshop participants will produce a report that will define basic needs and opportunities in Computer Science research to develop smart and trainable operating and runtime systems, execution models, and programming environments that will make future systems easier to tailor to the computing needs of scientists and for supercomputing facilities to securely deploy.

Invitation

We invite community input in the form of two-page position papers that identify and discuss key challenges posed for supercomputing operating and runtime systems, programming models, and software development environments for scientific computing by the trend towards increasing heterogeneity in supercomputer architectures and workflows. In addition to providing an avenue for identifying workshop participants, the position papers will be used to shape the workshop agenda, identify panelists, and contribute to the workshop report. Position papers should not describe the authors' current or planned research, nor should they recommend solutions or narrowly focused research topics. Rather, they should aim to improve the community's shared understanding of the problem space and help to stimulate discussion.

Submission Guidelines

Position papers should describe a fundamental Computer Science research direction that addresses key challenges associated with extreme heterogeneity.

- Key Challenges: Which challenge(s) related to EH does this paper address?
- Research Direction: What is the promising research direction for this topic?
- State of the Art: Across the community, identify state of the art for this research direction.

This description should be followed by an assessment of potential research directions based on the following dimensions:

- Maturity: Are there existing methods or Computer Science Program research directions that address the challenge(s) and that show promise for scientific computing? What are the indicators that a given method or approach will address the identified challenges? If there are not existing methods or research approaches to meeting the challenge, can you suggest ways to gain new insight into the problem space?
- Timeliness: Why now? What breakthrough or change makes progress possible now?
- Uniqueness: Is the identified challenge unique to scientific applications of supercomputing, whether for simulation or data-intensive science? What makes it so?
- Novelty: To what extent is existing method or approach unique to extreme heterogeneity in supercomputing for scientific research? Is the approach being addressed by other research programs? By the private sector? How is this approach different from existing approaches or solutions? Why should it be of interest to the ASCR Computer Science program?

Each position paper must be no more than two pages, including figures and references. The paper may include any number of authors but must provide contact information for a single author who can represent the position paper at the workshop. There is no limit to the number of position papers that an individual or group can submit. Authors are strongly encouraged to follow the structure previously outlined. Papers should be submitted in PDF format at the URL listed previously.

Selection

Submissions will be reviewed by the Organizing Committee. Authors of selected submissions will be invited to participate in the workshop, which will be held on January 23–25, 2018, in Gaithersburg, Maryland. Authors are not expected to have a history of funding by the ASCR Computer Science program. Authors of selected position papers will be invited to participate in the workshop based on the overall quality of the position paper(s) and an expectation that their active participation in the workshop will stimulate constructive discussion by the workshop participants and contribute to an informative report. Unique positions that are well presented and emphasize potentially transformative research directions will be given preference.

Appendix E: Related Activities

Not surprisingly, the trends discussed in the Workshop on Extreme Heterogeneity have been recognized by the community and presented in recent reports and workshops by a number of other entities. In some cases, such as quantum computing, programs addressing these trends have already been created:

IEEE Rebooting Computing (RC) initiative and conference (link)**.** The current incarnation of RC was started by IEEE as a working group to rethink the computer, "from soup to nuts," including all aspects from device to user interface. The RC initiative works from a holistic viewpoint, taking into account evolutionary and revolutionary approaches. It sponsors RC summits and cosponsors workshops and conferences on related topics.

International Roadmap on Devices and Systems (IRDS) (link). The International Technology Roadmap for Semiconductors (ITRS) tracked the exponential lithography improvements that underpinned Moore's law for nearly 20 years but disbanded in 2015 because further technology-driven forecasts no longer made sense with the end of the technology roadmap in sight. The IRDS, which is part of the IEEE Rebooting Computing Initiative, has for all practical purposes replaced the disbanded ITRS effort by changing from a device/lithography focus to a systems and application focus.

DOE Big Ideas: Beyond Moore Electronics (BME) (link). This started as a pitch to DOE at the 2015 Big Ideas Summit for a DOE-wide initiative to create a new public–private partnership for basic/applied research to accelerate the development of energy-efficient IT beyond the end of current roadmaps as well as maintain an advanced manufacturing base in the economically critical semiconductor space. The goal of this eight-laboratory consortium is to enable low-power computing and suitably low-cost smart grid and building electronics. The scope of this endeavor ranges from materials and devices to systems, software, and advanced manufacturing.

ASCR Quantum Computing Research programs. Quantum computing (QC) is a promising early-stage technology with the potential to provide a significant impact on computing for scientific applications. The DOE ASCR program office has previously held several workshops to gather community input on scientific use cases as well as testbed prototypes for quantum computing. The results from these workshops have identified new avenues of research for creating algorithms and applications of quantum computers to scientific computing as well as a production facilities component that seeks to deploy QC capabilities for a broad user community. With a focus on demonstrating the novelty of quantum computing, these ASCR-sponsored projects hint at some of the looming issues in future computing platforms that are captured by extreme heterogeneity. Additional information is available from the workshop reports:

- ASCR Report on Quantum Computing for Science (pdf)
- ASCR Report on Quantum Computing Testbed for Science (pdf)

Alongside recent efforts from ASCR, several other government departments and agencies have invested in the development of QC theory, applications, and implementations, including multiple organizations within the Department of Defense via the Defense Advanced Research Projects Agency and the military branches, the Department of Commerce via NIST, the National Science Foundation, and Intelligence Advanced Research Projects Agency. Industrial investments have also increased significantly in recent years with an emphasis on early-stage technology development. This includes small-scale quantum processing units; small-size application demonstrations in chemistry, materials science, and high-energy physics; and initial efforts to define reusable software infrastructure. This vibrant and growing community is augmented by international efforts in the European Union, China, Canada, Australia, Japan, and several other countries.

DARPA Electronics Resurgence Initiative (ERI) (link). The DARPA Microsystems Technology Office launched a new ERI in 2017 to ensure far-reaching improvements in electronics performance well beyond the limits of traditional scaling. The scope of the program includes architectures, design, and advanced electronic materials and integration.

Disclaimer

Images

Non-captioned images used in this report are courtesy of Oak Ridge National Laboratory, Los Alamos National Laboratory, and several licensed stock photo sources.


U.S. DEPARTMENT OF
ENERGY
Office of
Science