Software in the Era of Extreme Heterogeneity
Workshop Report

High End Computing Interagency Working Group
and
Software Productivity, Sustainability, and Quality Interagency Working Group

Hal Finkel and Alan Sussman, Co-Chairs, HEC IWG

Ram Sriram and Sol Greenspan, Co-Chairs, SPSQ IWG

Table of Contents

Executive Summary

The NITRD High End Computing (HEC) and Software Productivity, Sustainability, and Quality (SPSQ) Interagency Working Groups (IWGs) held a joint workshop on "Software in an Era of Extreme Heterogeneity" September 22–24, 2020, to explore the advanced computing landscape and the accompanying software development challenges and opportunities for the next 5–10 years. The state of HEC technology is rapidly changing because of the need to introduce a variety of unfamiliar hardware solutions to increase performance. This increasing hardware heterogeneity is exacerbated by memory, interconnect, and file input/output performance lagging behind computational capabilities, thus leading to substantial demands on the programming environment to keep pace.

The workshop brought together thought leaders from industry, academia, and the Federal agencies over a 3-day period to look at challenges imposed by extreme heterogeneity of emerging and future computational platforms and how the software and community must evolve to respond to the challenges being placed on HEC software development and sustainment. The following lists key opportunities and challenges identified during the workshop that could better prepare the HEC community for the era of extreme heterogeneity:

- Trends in semiconductor technology and the increasing prevalence of heterogeneous system architectures are leading to an increased interest in portable, often task-oriented and data flow-oriented, programming models. Synergistically adopting these programming models could enable application portability to a more diverse set of systems featuring heterogeneous architectures.
- Artificial intelligence and machine learning (AI/ML) infuse new capabilities into both application software and programming environments, enabling software to take advantage of complex heterogeneous systems and reducing the effort required to produce reliable, robust software and tools. While significant challenges remain in curating and characterizing the many required datasets, the diverse needs of ML training and inference workloads are driving innovation in both computational architectures and data management capabilities.
- The increase in software complexity, compounded by the inherent nondeterminism in distributed, asynchronous, and approximate computing hardware, is both driving the adoption of high-productivity programming tools and languages and adding to the challenges in verifying and validating applications on state-of-the-art computing systems. Additional research could help establish methods to ensure sufficient application-level reproducibility and reliability.
- To develop, use, and support software on heterogeneous systems, it is essential to recognize the value of people with the necessary skills, to develop applicable cross-institutional career paths, and to create the proper institutional incentives to employ people with those skills. Greater exposure to the benefits and challenges of heterogeneous systems earlier in the educational process could inspire the next generation to consider careers in high-end computing.

Introduction

In an era of increasing hardware heterogeneity, there are substantial demands placed on the HEC software development community to adapt to the rapidly changing technical landscape. Software design cycles are traditionally much longer than their hardware development counterparts. The lag time between the installation of heterogeneous hardware and the implementation of new software functionality can take full advantage of the evolving hardware that has been steadily increasing as new technology has flooded the HEC space.

The joint NITRD workshop on Software in the Era of Extreme Heterogeneity brought over 100 participants from industry, academia, and Federal agencies to discuss how the community can respond to the challenges being placed on HEC software development and on long-term sustainment. The workshop participants explored the topics of software development and sustainment challenges in an era of extreme heterogeneity; productivity challenges and opportunities presented by extreme heterogeneity; workforce requirements to support software; and what is needed to reduce the human challenge of software development, evolution, and porting. The three sessions of the workshop are summarized in the following text.

Sessions I and II: Software Development, Productivity, and Sustainment Challenges of Extreme Heterogeneity

HEC software development and sustainment continue to be a challenge that prevents the community from taking advantage of existing and emerging processing capabilities. Increasing hardware heterogeneity will only exacerbate this challenge. Session I of the workshop focused on the following questions:

- How can automated software tools become more responsible for HEC software development and sustainment decisions and efforts?
- Which decisions and efforts can be automated, and which must continue to be performed by humans?
- What opportunities and challenges lie ahead in this space in the era of extreme heterogeneity?

Session II of the workshop focused on scientific computing productivity, performance, and reproducibility and on which key research challenges are likely to be highlighted in future emerging heterogeneous computing environments. Several interconnected themes were highlighted covering various aspects of application software development, system software development, and hardware development in the era of extreme heterogeneity.

Technology Trends

Process-Technology Trends

The empirical Moore's Law describing the increasing density of transistors combined with Dennard scaling produced, for many decades, progressively more sophisticated processors that ran at higher base frequencies without requiring more power.[1] This meant that processors performed calculations at faster rates, but their increasingly complex circuitry extracted higher relative efficiency from existing

[1] Hennessy, John, and David Patterson. "A new golden age for computer architecture: Domain-specific hardware/software co-design, enhanced." In *ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, 2018. https://iscaconf.org/isca2018/turing_lecture.html

applications as well. This complex circuitry, including more on-die memory, could be added at low marginal cost as a result of increasing transistor densities, which came along with decreases in per-transistor energy consumption.

By around 2006, it was becoming clear that these favorable conditions, provided by smooth advancement in the underlying manufacturing technology for integrated circuits, were giving way to a more challenging technological environment. Improved memory performance, especially the access latency to main memory, had slowed appreciably, causing a corresponding slowing of performance improvements for many important classes of algorithms. The breakdown of Dennard scaling from underlying physical factors meant that processor clock rates could no longer be increased as transistor density increased without also using more power; in addition, more active circuitry could not be added without likewise increasing the power required. The end of improvements in planar transistor density was foreseeable as the size of individual transistors approached the minute scales on which quantum mechanics dominates classical material properties.

Manifestations of Extreme Heterogeneity

As a result of the aforementioned technology trends, performance gains require increasing architectural innovation, and, especially over the last decade, this innovation has manifested in increased parallelism. Even in commodity computers, multi-core processors are now common. However, an increased prevalence of specialized processors and supporting components has also become common. Even in commodity computers, from laptops to cell phones, multi-core processors are supplemented by graphics processing units (GPUs), along with dedicated components to accelerate everything from encryption to signal processing. Because of a natural alignment of functional requirements and some successful co-design activities with the HEC community, GPUs have turned out to be adept at performing the high-performance numerical calculations needed for simulation and data analysis applications, and are now commonplace in HEC systems.

Performance in HEC systems is increasingly due to specialized, heterogeneous processors, including GPUs, Tensor Processing Units and other kinds of ML accelerators, and smart memories and networking components.[2] In many HEC systems, most of the computational and memory movement capabilities are present in their GPUs or other accelerators, and not in their general-purpose processors. Moreover, with the demonstration of wafer scale integration, chiplets,[3] and other advanced packaging and integration approaches, the demonstrated system design space is wider than ever before.[4] As a result, the processors in each node of a large computing system and the nodes themselves are becoming not only more complex but also more dissimilar from those of comparable computing systems, and the rate of change has increased considerably in recent years. On the horizon are technologies that are even more distinct from traditional processors than common present-day accelerators, including hardware for quantum computing, quantum-inspired computing such as classical simulated annealing acceleration, and neuromorphic computing.

[2] Vetter et al., "Extreme heterogeneity 2018: Productive computational science in the era of extreme heterogeneity: Report for DOE ASCR workshop on extreme heterogeneity," 2018, https://doi.org/10.2172/1473756

[3] A chiplet is one part of a processing module that makes up a larger integrated circuit like a computer processor.

[4] Semiconductor Research Corporation. "The decadal plan for semiconductors: A pivotal roadmap outlining research priorities," 2021. https://www.src.org/about/decadal-plan/

AI as a Driver for Heterogeneous Computing Architectures

Computing needs, especially training and inference of deep learning algorithms, have exploded in recent years. Deep learning is now a key driver of heterogenous computing architectures in the marketplace because the structure of deep learning algorithms can be exploited in designing hardware that is adapted for deep learning computations. Not all deep learning problems are the same; among many other relevant factors, training and inference, levels and kinds of data sparsity, and precision and reproducibility requirements differ. As a result, the space of potential architectures that are well suited for different deep learning problems is very large and is reflected in the wide array of ML accelerators in the marketplace.[5] ML algorithms are also influencing other kinds of accelerators, notably GPUs, and are on general-purpose processors, as these components are also being adjusted to better support ML and, increasingly, deep learning workloads. Functional units for tensor/matrix processing, support for data types such as bFloat16, and other now prevalent features in HEC architectures are all products of the drive toward better support of deep learning workloads. In general-purpose processors and generally programmable accelerators such as GPUs, these features are now finding uses outside of deep learning algorithms and driving further innovation in other areas.

System-Level and Node-Level Considerations

Large HPC systems are commonly composed of many compute nodes connected by a high-speed network interconnect. They have long required software design cycles to target the node-level architecture and, as a mostly separate aspect of the design, have determined how each application is distributed across the nodes. Moreover, even as node-level designs, including the incorporation of accelerators, have changed dramatically, the way that applications deal with the distributed memory aspect of these systems has remained mostly static. Each application was designed to scale to hundreds or thousands of nodes with communication between these nodes accomplished using the message passing interface (MPI), which is a standardized interface[6] supported universally. Data from file systems has been commonly accessed via MPI, and the lower level details have often been ignored. All of this is changing now. Accelerators have made each node extremely powerful, while the speed of the networks has not increased as quickly; furthermore, the networks have increasingly incorporated specialized logic to accelerate particular functions. Applications have become more complicated, and, because interesting problems are rarely uniform, load balancing across the different nodes has become more challenging. This load balancing must occur noting that the movement of data, relatively speaking, is increasingly expensive. Moreover, an analogous set of considerations applies to accessing data from, and storing data to, long-term, multi-level storage subsystems.

Software Complexity

Over the past few decades, the complexity of the software ecosystem has grown considerably, and large, modern software projects can easily have tens of millions of lines of code that generate hundreds of external dependencies. In general, the need for improved software is outstripping our ability to write and verify it, let alone to maintain the software that already exists as the underlying systems evolve. Transitioning to emerging modern programming languages and adopting continuous integration and high coverage testing methodologies, code versioning, and other current industry best practices will help, but the projected development-capacity deficit is lessened only slightly by the projected impacts of these productivity enhancements. The need to target a diverse set of heterogeneous architectures

[5] Reuther, Albert, Peter Michaleas, Michael Jones, Vijay Gadepally, Siddharth Samsi, and Jeremy Kepner. "Survey and benchmarking of machine learning accelerators." In 2019 IEEE High Performance Extreme Computing Conference (HPEC), pp. 1–9. *IEEE*, 2019. https://ieeexplore.ieee.org/abstract/document/8916327/

[6] MPI is standardized by the MPI Forum: https://www.mpi-forum.org/

makes software improvement even more challenging, in part because the space and complexity of programming models are similarly growing. Directions being pursued include the following:

- Abstraction layers and standards have become key tools in addressing this challenge, although designing abstraction layers that will likely be useful for future hardware and systems, in addition to current ones, requires a considerable amount of educated guesswork.
- Use of containers has also become a key tool in managing complexity and dependencies, but interoperability between different packaged environments then becomes its own challenge.
- Domain-specific languages, autotuning, and AI-driven programming tools are increasingly viewed as potential solutions; however, integration, training, and sustainability all present hindrances that need to be overcome.

Future Directions

Heterogeneity and Performance Portability in Programming

Computers have always been composed of a myriad of components, but what defines heterogeneity in the current context is the need for ordinary application programmers to understand many different components in terms of both functional and performance characteristics and to use that understanding when creating and configuring their programs. This involves writing code to exploit different functional units on multiple identical cores, exploit different functional units on multiple cores of different types, and manage moving memory between different parts of the system. Programming, an activity that is already complex, is made even more complex by these heterogeneity considerations. It is important to note that developers include domain scientists, computational scientists, software engineers, and others involved in developing and sustaining HEC software. This broad spectrum of developers approaches these complex programming challenges with a wide range of skill levels and educational backgrounds.[7] In decades past, an HEC application might be expected to run on successive generations of only one kind of architecture, often from only one vendor, but this is no longer the general expectation in the current technological environment. Developers, notably including domain scientists, often need to deal directly with hardware changes by altering algorithms and their implementation strategies. Accordingly, the key challenge is not in writing source code that can be used effectively on one such heterogeneous system but to instead write code that can be used effectively across a multitude of different systems and, with little effort, update the code to account for new systems and upgrades to existing systems. This kind of performance portability is hard to achieve because of a multitude of challenges, including the following:

- Heterogeneous system components, with different functional and performance characteristics, naturally have different programming interfaces. Writing a single implementation that can make use of these different interfaces effectively is often difficult.
- Predicting the performance of even a single kind of system is challenging because of the inherent complexity of modern computing components. The interactions between individually hard-to-predict systems are likewise difficult to predict, making it even more challenging to predict the heterogeneous HEC system behavior.
- Many system components, in addition to the algorithms used by software, have random aspects to their behavior by design or because of unpredictable factors from the outside world, or both.

[7] Hannay, Jo Erskine, Carolyn MacLeod, Janice Singer, Hans Petter Langtangen, Dietmar Pfahl, and Greg Wilson. "How do scientists develop and use scientific software?" In *2009 ICSE Workshop on Software Engineering for Computational Science and Engineering*, pp. 1–8. IEEE, 2009. https://software-carpentry.org/files/bib/secse-survey-2009.pdf

- It is often the case that achieving good performance on different kinds of hardware requires software using different kinds of algorithms. For example, some algorithms perform less synchronization and more computational work compared to other algorithms, and different points in this tradeoff space yield the best time to solution on different kinds of hardware.
- The methods of implementing an algorithm need to change to achieve good performance on different kinds of hardware. Currently, these methods can involve changes that are difficult to localize to small portions of the source code, including different layouts and placements of data in memory and different ways of splitting up tasks to run in parallel.
- The compilers used to translate source code into executable applications have a limited ability to optimize an application for different kinds of hardware because of the underlying difficulty of the problem, an inability to make choices that will be optimal for different sizes and kinds of data and parameters used by the application, and constraints imposed by the semantics of the programming languages used by the source code.
- The tradeoffs between maintaining different implementations of software components for different kinds of systems and maintaining a single implementation designed to run on many different kinds of systems are difficult with each choice having costs and risks that are difficult to estimate.
- Cloud computing providers, which are also adopting heterogeneous architectures for many HEC offerings, may be expanding access to, and broadening the user base of, HEC systems. However, supporting cloud platforms, which often use a mixture of both open and proprietary interfaces, adds yet another layer of complexity to the development and deployment of HEC software.

As a result of these challenges, even basic, widely accepted metrics for performance portability are difficult to define. Nevertheless, an important goal suggested during discussions is enabling domain scientists to achieve 80 percent of the theoretically achievable performance for the chosen class of algorithms across a set of hardware architectures of interest with no more than 20 percent of the effort currently required, and possibly not necessitating an HPC expert.

Challenges in Defining Correctness

A key challenge in moving applications between different kinds of heterogeneous systems is ensuring the correctness of those applications across the large space of available systems. In many cases, even defining what "correct" means is a challenge. In some cases, bitwise reproducibility[8] is a reasonable definition of correctness. However, even for traditional computing systems, efficient exploitation of parallelism often requires accepting some variance in correct answers. For example, if partial answers computed by parallel tasks are added up in the order in which the tasks complete their work, the final sum may differ between different runs of the application because of the limited numerical precision used when performing the computations. Generally, these differences are usually, but not always, small compared to the inherent uncertainties in the quantities being computed, and care in verification and validation are necessary. One of the ways in which modern computing systems deliver enhanced performance, however, is by providing enhanced computational speed for computations performed with lower numerical precision. Figuring out where it is acceptable to exploit this tradeoff without compromising correctness is a key challenge in programming modern HEC systems. However, the increased need to exploit parallelism, both across and within compute nodes, also leads to correctness challenges. These challenges are due not only to the aforementioned effects on operations such as the summing of partial results, but also to the idea that correctly programming parallel algorithms is challenging and often requires specialized skills. Looking forward, as systems incorporate probabilistic or

[8] Bitwise reproducibility means that every bit of the answer computed for identical problems is the same when that answer is subsequently recomputed, including when recomputed on different systems.

quantum computing technologies, understanding how to define correct execution is expected to become even more challenging.

Role of Abstraction in Accommodating Heterogeneity

As heterogeneous systems continue to proliferate and software is enhanced to work on this increasing variety of HEC systems, the software tends to become more complex. This complexity causes the software to be more difficult and costly to maintain and further develop, in part because it makes the software more challenging for its programmers to understand. As new developers are brought onto the development team for an application, and as other developers leave, the task of figuring out how and why the software works the way that it does, given the available documentation and the source code itself, becomes increasingly difficult and error prone. To mitigate this tendency toward complexity, abstraction is widely viewed as a key best practice. While figuring out how to break a complex piece of software into smaller components is itself often difficult, decomposing software so that it has layers of components, each with an abstract interface, allows the development of each component without a detailed understanding of the internal optimizations performed by the implementation of the layers below. As hardware, algorithmic techniques, and requirements evolve, these kinds of layered abstractions allow software changes to be localized to individual components instead of proliferating across the entire application's source code. While this general software engineering technique has long been a general best practice in software development, many widely used libraries, frameworks, and standards have now emerged to make this layered abstraction technique applicable to HEC development. Mathematical representation of computations is useful as a highest level of abstraction, but addressing performance requirements also requires representing aspects of the available implementation approaches as well. Abstraction libraries, frameworks, and programming environments implementing standards can also assist developers by providing useful correctness and performance feedback using the high-level information that use of those standard facilities implies. Modern HEC applications are generally composed from multiple classes of algorithms operating on multiple kinds of data structures, so composability of libraries specializing in certain kinds of algorithmic and data structures is an important requirement. Unfortunately, composability of software libraries, especially those making use of parallel execution, remains a significant challenge.

High-Productivity Programming Languages Uses

In addition to libraries, frameworks, and standards for traditional high-performance languages such as C++ and Fortran, high-productivity languages such as Python are now commonly included in HEC application development.[9] In part, this inclusion is the result of the increasing popularity of interactive programming environments, including web-based computational notebooks that can integrate with a huge ecosystem of available libraries for ML and other kinds of data analysis. Use of high-productivity languages brings reductions in the time to create new algorithmic implementations and add new functionality; however, even with the ability to invoke optimized subroutines authored in lower level languages, the integration into HEC applications both contributes to beneficial layered abstraction and creates additional technical challenges.

[9] Evans, Thomas M., Andrew Siegel, Erik W. Draeger, Jack Deslippe, Marianne M. Francois, Timothy C. Germann, William E. Hart, and Daniel F. Martin. "A survey of software implementations used by application codes in the Exascale Computing Project." *The International Journal of High Performance Computing Applications* (2021). https://journals.sagepub.com/doi/abs/10.1177/10943420211028940

High-productivity languages are often not designed to handle large amounts of data, exploit significant amounts of parallelism, or make it easy to predict the performance of its built-in constructs. More recently designed high-productivity languages, such as Julia[10] or Chapel,[11] may be better suited for HEC applications but currently lack the extensive ecosystem of the software libraries of Python and other more mature languages. Nevertheless, ML toolkits such as TensorFlow and PyTorch, which are generally used via their Python interfaces, contain sophisticated specialized compilers to provide high performance on heterogeneous systems despite the limitations of Python.[12] Moreover, the underlying computational abstractions provided by these toolkits have been built upon by frameworks for probabilistic programming, generalized tensor programming, and other classes of HEC computations, thus providing examples of effective hardware abstraction layers for heterogeneous systems with high-productivity interfaces.

In some cases, high productivity has de facto standard library interfaces for capabilities for which there are no standardized interfaces at the C/C++/Fortran level. For example, Basic Linear Algebra Subprograms (BLAS)[13] enjoy wide adoption, but there is not a corresponding widely adopted interface for sparse linear algebra or graph operations, while the NumPy/SciPy library pair[14] provides widely used dense and sparse linear algebra interfaces for Python. Accordingly, to most effectively optimize modern HEC applications, the HEC computing ecosystem may benefit from programming environments that represent both high- and low-level code in a unified fashion.

Novel Programming Models

Traditional methods of programming for HEC systems involve various variants of bulk synchronous parallel (BSP) programming methodologies. The MPI, version 1.0, which was released in 1994 and still represents the most widely used parallel execution interface on HEC systems, is a standardized interface designed around the BSP paradigm. While newer MPI versions have added one-sided interfaces and other capabilities to support less structured asynchronous programming, recent years have seen significant adoption of so-called MPI+X programming models. In part, this trend is driven by a need to take advantage of on-node shared memory in order to most efficiently exploit multi-core parallelism. OpenMP®, a standardized model[15] layered in C, C++, and Fortran, is the most common model paired with MPI for this purpose. The MPI+X trend has accelerated, driven by the need to support computational accelerators, especially GPUs, because exploiting the parallelism exposed by GPUs generally requires a more structured set of programming methods and techniques that support more threads of execution than even MPI provides. As applications have increased in complexity, the BSP model has seemed less applicable, with task-oriented and dataflow-oriented models increasing in popularity. Following are descriptions of both models:

- *Task-oriented models* divide a computation into tasks that can be run in parallel, constrained by declared execution dependencies, by a high-performance scheduling algorithm.
- *Dataflow-oriented models* are similar to task-oriented models, but, in addition to the decomposition of the computation into tasks, the motion of data between the tasks is also placed within the purview of the scheduling and management system.

[10] The Julia Programming Language: https://julialang.org/

[11] Chapel: Productive Parallel Programming: https://chapel-lang.org/

[12] Dai, Wei, and Daniel Berleant. "Benchmarking contemporary deep learning hardware and frameworks: A survey of qualitative metrics." In *2019 IEEE First International Conference on Cognitive Machine Intelligence (CogMI)*, pp. 148–155. IEEE, 2019. https://ieeexplore.ieee.org/abstract/document/8998988/

[13] BLAS (Basic Linear Algebra Subprograms): http://www.netlib.org/blas/

[14] NumPy: https://numpy.org/; SciPy: https://scipy.org/

[15] OpenMP is standardized by the OpenMP ARB (Architecture Review Boards): https://www.openmp.org/

OpenMP has long supported task-oriented programming, but many systems now provide capabilities for task-oriented or dataflow-oriented programming both across large distributed memory systems and on GPUs and other kinds of accelerators. Dataflow-oriented computing on large scales is also increasing in popularity for large-scale data science from the support of frameworks such as Apache Spark™,[16] and for ML from the support of frameworks such as TensorFlow and PyTorch. At a low level, dataflow-oriented models are also common for programming field programmable gate arrays, coarse-grained reconfigurable architectures, and other spatial computational architectures. While not commonly used for computational accelerators in HEC systems, spatial architectures may become common, especially given the relative popularity of such architectural designs for ML accelerators. For general HEC application development, such models are not yet mature but do represent an active area of research and hold promise as a way of programming HEC systems at a higher level of abstraction than the current common methods.

Data Management Considerations

Modern scientific applications tend to both produce and consume huge amounts of data. The large size required for datasets for ML training is a well-known challenge across many industries, and scientific ML often requires huge datasets; however, large data challenges extend to many other use cases as well. The amount of data produced by scientific experiments has increased exponentially in recent decades. Even the initial conditions used for physical simulations have grown in storage size, and the amount of data that modern scientific applications can produce is enormous. Determining how to load and manage the necessary data in memory is often difficult, especially given the multiple levels of modern memory and storage hierarchies, which include random access memory, persistent flash-style memory, solid state drives, and burst buffers.[17] Storing all of the interesting states from physical simulations and data analysis procedures has become impractical, leading to more *in situ* reductions and recomputation; nevertheless, applications are often bottlenecked by write bandwidth and storage capacities. Compounding this challenge is the need to save more data to combat system resilience issues.

Opportunities for Artificial Intelligence

AI presents opportunities to improve nearly all aspects of HEC system design and use. AI methods could improve the low-level design of computer architectures and could be embedded into the hardware itself to provide improved heuristics for dynamic, adaptive features such as branch prediction and inter-device data movement. AI technologies can potentially assist with collecting and interpreting profiling data on application performance. AI methods can be directly applied to challenges in compiler optimization, especially because the impact of many potential code transformations depends on the data being processed and other application parameters. Going further, the mapping of computational tasks to different kinds of computational accelerators and the tuning of those workloads to execute most efficiently on the chosen hardware represents a significant area of opportunity for AI technology. One of the potentially most impactful opportunities for AI lies in automating significant aspects of the overall software lifecycle. Beyond tuning, automating the creation, testing, documenting, integrating, and evolving software are all actively researched AI opportunities. While program synthesis and repair, as the areas are commonly called in academic literature, are major focuses of the research community, significant challenges remain. Areas of challenge include, but are not limited to, the following:

[16] Apache Spark: Unified engine for large-scale data analytics: https://spark.apache.org/

[17] Koo, Donghun, Jaehwan Lee, Jialin Liu, Eun-Kyu Byun, Jae-Hyuck Kwak, Glenn K. Lockwood, Soonwook Hwang, Katie Antypas, Kesheng Wu, and Hyeonsang Eom. "An empirical study of I/O separation for burst buffers in HPC systems." *Journal of Parallel and Distributed Computing* 148 (2021): 96–108. https://www.sciencedirect.com/science/article/pii/S0743731520303907

- AI methods producing implementations that are correct should be verified, especially because real applications often have complex correctness requirements and because generating or changing code is often not a task that can be localized to changing only a small number of subroutines.
- The data required for supervised learning over the space of programs often seems impractically high, coverage of programming language constructs and algorithmic techniques is uneven, and the source code for programs often lacks information on the decision-making process that produced it. Additionally, unsupervised and semi-supervised learning is challenged by the large number of potential programs combined with the difficulty of automatically generating correct and interesting programs for use by AI training procedures. Even if these challenges are overcome, the process of generating code is expected to be an interactive and iterative process, for the same reasons that traditional development processes iterate to explore potential solutions and adapt to changing requirements.

Benchmarks and Proxy Applications

Benchmarks and proxy applications have played a key role in the HEC ecosystem, allowing for the usage-informed design and characterization of HEC computing systems. While several recent efforts, such as those behind MLPerf,[18] the SPEC ACCEL® benchmark suite,[19] and the Exascale Computing Project (ECP) proxy application suite,[20] have begun to address relevant needs for modern HEC systems, significant gaps remain. Traditional benchmarks and proxy applications represent only a single implementation strategy, but given the tremendous diversity of present and future HEC hardware, there is value in representing different ways of exploiting parallelism, data locality, data movement, and variable precision. New benchmarks and proxy applications can span the space from microbenchmarks representing small parts of algorithms to representations of complex, end-to-end workflows. Moreover, future systems may also expose to users or software developers different resilience properties, allowing for tradeoffs between performance and the visibility of hardware errors to be explicitly exposed. A more robust set of benchmarks and proxy applications can help address challenges in curating data for training AI-based methods, in training HPC professionals, in exploring the design spaces available for future hardware, and in characterizing HEC systems.

Challenges in Training and Education

Training computer science students in the techniques and underlying principles relevant to heterogeneous computing is challenging, not only because compilers, system software, and hardware have been increasing in complexity, but also because the increasing diversity of hardware also requires a broad knowledge of different kinds of computing, memory, networking, and storage technologies. Abstraction layers can greatly assist with programming this complex array of systems in practice; however, from the perspective of computer science education, these add yet more critical topics to an already challenging curriculum. Educating non-computer science majors in programming modern HEC systems is likewise challenging; while the technical depth required is not the same as for computer science students, the software and hardware ecosystem has been evolving quickly. Additionally, given the significant degree to which lower level hardware properties are exposed to application developers, mature curricula are nearly impossible to develop. Sophisticated abstraction layers and tools, along with robust interdisciplinary collaboration, are viewed as essential, given these educational challenges. Developing complex software, especially as part of relatively large teams, requires a number of skills, including knowing how to do the following:

[18] MLPerf benchmark suites, produced by MLCommons: https://mlcommons.org/

[19] Standard Performance Evaluation Corporation (SPEC) ACCEL benchmark suite: https://www.spec.org/accel/

[20] Exascale Computing Project (ECP) Proxy Apps Suite: https://proxyapps.exascaleproject.org/ecp-proxy-apps-suite/

- Derive data layout, placement, and locality combined with task assignment and scheduling on GPUs and other accelerators.
- Design and develop portable, composable, modular, and resilient software for distributed systems with proper layering and abstraction.
- Structure software to provide the desired levels of accuracy, uncertainty quantification, and reproducibility when integrated into larger workflows.
- Gather requirements from users and other stakeholders, and reflect those requirements in usability-maximizing designs.
- Make use of prototypes and exploratory development with well-defined goals, and transition to production-focused development at appropriate stages in a project.
- Productively develop and maintain software throughout its entire lifecycle in accordance with best practices for agile project management.
- Follow best practices for software development, including following coding conventions, providing informative documentation, and developing other techniques that maximize maintainability.
- Use modern software engineering practices and tools for testing, continuous integration, static checking, version control, code review, and other tasks that enhance productivity and software reliability.
- Select programming languages, development tools, external libraries, and other dependencies, considering technical suitability, combined with project maturity, support availability, licensing structure, and other non-technical considerations.

Session III: Workforce and Tools

Session III of the workshop focused on how rapid changes in computer architecture toward heterogeneity are exacerbating the already challenging task of supporting software on high-end systems. The session concentrated on questions of the human challenges of software development, evolution, and porting and on how the requirements of preparing people to develop, effectively use, and support software in the era of extreme heterogeneity can be eased.

Resources To Increase the Value of Cyberinfrastructure

It has been demonstrated that, when institutions invest in cyberinfrastructure (CI) (including HEC resources), the return on that investment can be substantial. The investment includes facilities, but it also includes the people required to staff and to provide services needed for those facilities and for those who use the facilities.[21] The value that CI brings to the larger scientific enterprise should be better understood and supported from both producer and consumer perspectives. The session focused on CI professionals, such as research software engineers,[22] data librarians, and system administrators, who play important roles in the research and education enterprise. Vital issues included the following:

- Career paths for CI professionals, particularly those in academia, should be established and better defined by the institutions relying on these skilled individuals.
- Creation of effective pipelines from industry to academia, and vice versa, needs attention. Training and retraining programs that cross industry and academia could be extremely valuable in enlarging the CI workforce and in providing new and interesting opportunities for both existing staff and students. Proposals for Federal grants could require including a development plan for CI

[21] "CI Workforce Development Workshop 2020." https://www.rcac.purdue.edu/ciworkforce2020/

[22] "U.S. Research Software Engineer Association." https://us-rse.org

professionals as is currently done for postdoctoral scientists at some agencies. Curricula changes should occur at universities in all CI technologies, but software engineering with minors and certificate programs in CI are particularly valuable pathways in this context.

- Attracting a larger and more diverse workforce, especially members of underrepresented groups, by recruiting students from smaller universities and community colleges has the potential to enlarge the talent pool of available CI professionals. Workshops, tutorials, webinars, and bootcamps could play an important role in preparing a literate and capable CI workforce.
- More incentives to encourage people to pursue a career in CI technologies are critical.

Details of these vital issues are covered in detail in the following subsections.

Incentive Systems To Produce Sustainable, High-Quality Scientific Software

CI professionals play an essential role in producing high-quality, reproducible scientific results, but there is no real viable career path through academia and many other research institutions to ensure that these individuals have permanent positions with growth potential.[23] A major challenge is how that culture can be changed. The problem is exacerbated for institutions with many small research groups. A potential solution is to incorporate software and data CI professionals into computational science teams, requiring funding for CI professionals into grant proposals and creation of institutional funding sources.

Another potential solution is to engage industry to create a pipeline for CI professional staff in both directions between research institutions and industry, with an appropriate incentive structure in each direction. Creating partnerships between academia and Government institutes/labs to share best practices on producing high-quality software and data products can also help address recruiting and retention challenges.[24] Incentives could attract students to learn how to produce reliable software that generates reproducible computational results. The use of such incentives, however, often requires tradeoffs between productivity and educational opportunities for graduate and undergraduate students.

Possible solutions include minor and certificate programs and specifically interdisciplinary curricula, stressing how software and data products are a path to scientific discovery. Researchers should focus on attracting a larger, more diverse pool of talent for CI professional positions by modifying social and technical practices to make computational science more accessible and attractive to a wider range of people with different skill sets and educational backgrounds. Finally, research on developing and using software and data properly in research, sometimes called research software science, is needed to leverage the skills of cognitive and social science researchers to develop a quantitative understanding of the best practices that should be incentivized.

Artificial Intelligence and Machine Learning for Productivity

The potential impact of AI and ML on improving productivity is significant. AI/ML ideas are affecting many fields of science and technology, both in data-intensive computing and in more traditional areas of simulations. These ideas impact the design of software, compilers, and many other tools used by computer and data science and engineering researchers and professionals. A major issue determining how these technologies can reduce the effort of people in producing reliable, robust software and tools.

[23] "Transforming science through infrastructure: NSF's blueprint for a national cyberinfrastructure ecosystem for science and engineering in the 21st century: Blueprint for cyberinfrastructure learning and workforce development." https://www.nsf.gov/cise/oac/vision/blueprint-2019/CI-LWD.pdf

[24] Exemplary efforts include Better Scientific Software (BSSw) (https://bssw.io/) and Interoperable Design of Extreme-scale Application Software (IDEAS) (https://ideas-productivity.org/).

One key point is the need to collect data systematically from systems, applications, networks, etc., to train ML models and prioritize data collection in evaluating project proposals at Federal agencies.

The entire human workflow should be examined for opportunities to apply AI and ML. This includes applications, recommender systems, experimental design, simulations, and chatbots. For simulations, AI and ML may be used for many purposes, including reducing the number of simulations run, reducing simulation time, extracting more information from computations already being run, and automating scientific workflow tasks. Another approach for recruiting talent into the CI workforce is to capitalize on the high profile of AI and ML today among students and CI professionals by emphasizing the new and exciting role that they are playing in HEC. One barrier to adoption of AI and ML is the training required to integrate these technologies into standard computational science methodologies. Investigation into how industry and research lab advances in AI and ML tools can be adapted to HEC needs, where performance and scale matter, is needed.

List of Abbreviations and Acronyms

Item	Spell-out
AI	artificial intelligence
BLAS	Basic Linear Algebra Subprograms
BSP	bulk synchronous parallel
CI	cyberinfrastructure
DoD	Department of Defense
DOE/SC	Department of Energy Office of Science
ECP	Exascale Computing Project
GPU	graphics processing unit
HEC	High End Computing
HPC	high-performance computing
IWG	Interagency Working Group
ML	machine learning
MPI	message passing interface
NCO	National Coordination Office
NIST	National Institute of Standards and Technology
NITRD	Networking and Information Technology Research and Development
NOAA	National Oceanic and Atmospheric Administration
NRL	Naval Research Laboratory
NSF	National Science Foundation
SPSQ	Software Productivity, Sustainability, and Quality

About the Authors

The NITRD Program is the Nation's primary source of federally funded work on pioneering information technologies in computing, networking, and software. The NITRD Subcommittee of the National Science and Technology Council's Committee on Science and Technology Enterprise guides the multiagency NITRD Program in its work to provide the R&D foundations for ensuring continued U.S. technological leadership and that meets the Nation's advanced IT needs. The National Coordination Office (NCO) supports the NITRD Subcommittee and the IWGs and teams that report to it. The NITRD Subcommittee's Co-Chairs are Kamie Roberts, NCO Director, and Margaret Martonosi, Assistant Director of the NSF Directorate for Computer and Information Science and Engineering. More information about NITRD is available online at https://www.nitrd.gov/.

The NITRD Program's HEC IWG coordinates Federal R&D that extends the U.S. leadership in advanced computing and enables transformative research to support the Nation's economic competitiveness, security, and leadership in science and engineering. More information about the HEC IWG is available online at https://www.nitrd.gov/coordination-areas/high-end-computing/.

The SPSQ IWG gathers information from software experts to ensure that government investment in SPSQ R&D results in the software development technology that is essential to U.S. innovation, emerging technology leadership, security, and prosperity. More information about the SPSQ IWG is available online at https://www.nitrd.gov/coordination-areas/SPSQ/.

Acknowledgments

The NITRD HEC and SPSQ IWGs gratefully acknowledge the workshop planning committee members Alan Sussman (NSF), Sonia Sachs (DOE/SC), Robinson Pino (DOE/SC), Jim Kirby (NRL), Jake Fries (NCO NITRD), Leslie Hart (NOAA), Frances Hill (DoD), Walid Keyrouz (NIST), Ji Hyun Lee (NITRD NCO), and Barry Schneider (NIST); workshop advisors William Gropp (University of Illinois), Jeffrey Vetter (Oak Ridge National Laboratory), Saman Amarasinghe (Massachusetts Institute of Technology), and Kaylan Kumaran (Argonne National Laboratory); and the workshop participants whose valuable insights and contributions to the workshop resulted in this report.