Future Directions for Parallel and Distributed Computing

SPX 2019 Workshop Report*

Scott D. Stoller[1,†], Michael Carbin[2,†], Sarita Adve[3], Kunal Agrawal[4], Guy Blelloch[5], Dan Stanzione[6], Katherine Yelick[7], and Matei Zaharia[8]

[†]Co-Chair
[1]Stony Brook University
[2]MIT
[3]University of Illinois at Urbana-Champaign
[4]Washington University in St. Louis
[5]Carnegie Mellon University
[6]University of Texas at Austin; Texas Advanced Computing Center (TACC)
[7]University of California, Berkeley; Lawrence Berkeley National Laboratory
[8]Stanford University

October 2019

Contents

*This work was supported by NSF grant CCF-1931235. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

1 Executive Summary

The drumbeat of progress in computing is the regular and timely increase in computational capability provided by traditional technology scaling trends and improved computer systems that exploit and expose computing resources. Parallel and Distributed Computing is at the center of this progress in that it aggregates multiple computational resources, such as CPU cores and machines, into a single effective and powerful system.

Over the years, parallel and distributed systems have expanded beyond traditional supercomputing applications (i.e., scientific simulation) to now constitute major computing infrastructure [7, 4] that advance Artificial Intelligence (AI) and Machine Learning (ML) [6, 22], large-scale communication networks (e.g., social media), smart cities, etc. However, the field of Parallel and Distributed Computing faces a number of existing and emerging challenges that must be overcome in order to meet the rapidly increasing computational requirements of new and larger highly distributed applications.

Existing parallel and distributed computing abstractions impose substantial complexity on the software development process, requiring significant developer expertise to extract commensurate performance from the aggregated hardware resources. An imminent concern is that the end of Dennard scaling and Moore's law will mean that future increases in computational capacity cannot be obtained simply from increases in the capability of individual cores and machines in a system.

While proposed solutions do include a transition to new computational fabrics that may offer future increases in performance, the highest-performance future computing systems will need to be *specialized*—customized in the hardware itself and the algorithms and abstractions underlying the software—to fit the exact organization and requirements of the application at hand [1, 10]. Systems incorporating a variety of specialized components form *heterogeneous* platforms that are potentially very powerful but also potentially make application development more difficult.

The pressure towards application-specific design is compounded by the expansion of computing into new environments and domains, such as digital agriculture, in which applications are highly distributed and decentralized, and involve both localized sensing computations on thousands—or millions—of low-power embedded devices and predictive simulations at landscape or global scale in massive data centers. Such applications will challenge future systems to compose and orchestrate both small and large devices alike. These applications will also push computing towards new abstractions and algorithms, such as sensing and AI/ML, that have classically been on the periphery of traditional Computer Science (CS).

These challenges have put parallel and distributed computing at a crossroads: new abstractions are needed to manage the complexity of developing systems at the scale and heterogeneity demanded by new computing opportunities. This applies at all levels of the computing stack, encompassing the many disciplines (subfields of CS) that contribute to each level: the application itself (traditionally High-Performance Computing (HPC) but increasingly a variety of other application domains as well), its algorithmic design and implementation (Theory and Algorithms), the programming languages and implementations of the application and the underlying computing platform (Programming Languages, and Systems), and the underlying hardware platform (Computer Architecture).

In addition to seeking the traditional goals of performance and scalability (latency, throughput, etc.), energy efficiency, and reliability, there is an increased demand for these abstractions to facil-

itate reasoning about correctness, security, and privacy, which have become central system goals. The methodology for building these systems may therefore require new automated analyses and correct-by-construction development methods that address these goals starting from the outset of the design process.

In sum, the next generation of parallel and distributed computing systems will be domain-specific and combine a heterogeneous mix of computational patterns, algorithms, and hardware to achieve a set of goals that go beyond traditional systems aims to meet society's needs for more scalable, energy-efficient, reliable, verifiable, and secure computing systems.

2 Recommendations

This section briefly summarizes top-level recommendations for research in parallel and distributed computing. Subsequent sections of the report provide more detail on specific research directions.

- Traditional boundaries between disciplines need to be reconsidered. Achieving the community's goals will require coordinated progress in multiple disciplines within CS, especially across hardware-oriented and software-oriented disciplines. Close collaboration with domain experts outside CS will also be essential to develop domain-specific solutions for challenging emerging applications.

- The end of Moore's Law and Dennard scaling necessitate increased research efforts to improve performance in other ways, notably by exploiting specialized hardware accelerators and considering computational platforms that trade accuracy or reliability for increased performance. New techniques are needed for domain-specific and platform-specific design that directly integrate specialized abstractions into algorithm, language, and system design.

- Developing correct large-scale parallel and distributed systems is already a daunting challenge, which will become even more difficult as the systems become ever larger and more complex at the hardware and software levels. Ease of development, and the need to assure correctness, are fundamental considerations that must be taken into account in all aspects of design and implementation. Correctness is broadly construed to include not only functional correctness but also security, privacy, and other requirements crucial for many emerging distributed applications. Emerging techniques for testing, debugging, and verification will become key technologies that enable the development of functional systems at future scales.

- Full-stack, end-to-end evaluation is critical. Innovations and breakthroughs in specific parts of the hardware/software stack may have effects and require changes throughout the system. Therefore, research activities need to go across scales, from basic proof of concept to determining how the ecosystem must change, and new techniques need to be evaluated in a full-system context using realistic applications and workloads. Evaluation must be comprehensive, considering not only the traditional system goals of performance, scalability, and reliability, but also accuracy, energy efficiency, security, privacy, and assurance.

3 Organization of the Workshop and Report

Well before the workshop, the workshop organizers asked each participant to suggest one or more ideas about future directions for parallel and distributed systems that they thought should be discussed at the workshop. From this input, the workshop organizers distilled cross-cutting themes and discussion questions. These were posted in advance of the workshop, so participants could prepare their thoughts about them. The workshop organizers also invited participants to submit proposals for lightning talks.

The workshop's morning program featured spotlight talks and selected lightning talks. The afternoon featured two breakout sessions, each followed by summary presentations from the breakout groups. The first breakout session was organized by cross-cutting theme: each breakout group focused on one of the themes. The second breakout session was organized by discipline (within CS): each breakout group focused on how one discipline could help address the cross-cutting themes and challenges. To help ensure productive discussions, each breakout group had a leader to guide the discussions and a worksheet (reproduced in Appendix C) with questions to address during the discussions and in the summary presentations.

The core of this report is organized similarly as the breakout sessions. Section 5 is organized by cross-cutting theme; Section 6, by discipline.

4 Future Trends in Applications and Technology

The demand for computing across all areas of science, engineering, entertainment, government, and commercial applications will continue to grow in spite of the increasing technology challenges to enabling that growth. Applications of computing will become more complex and distributed, and the lines between different types of applications and systems will blur as experimental and observational data are combined with simulation.

4.1 Applications

Parallel computing was once dominated by science and engineering, where it has been used to simulate physical systems, from molecular structures and fluid dynamics, to earth systems and cosmology. But the growth of social, medical, commercial and biological data sets mean that the need for parallelism is now universal across computing domains beyond the physical sciences. The use of machine learning methods will also drive the need for more performance and more parallelism; deep learning methods are some of the most computationally intensive applications on current systems, and they often operate on massive data sets or search vast state spaces. Driven by society's ambitious goals of understanding increasingly complex phenomena in natural and human-constructed systems, the demand for more cost-effective, energy-efficient, and capable computing devices and systems will grow across basic science, engineering, business, government, and entertainment.

Three broad types of applications are driving the need for extreme-scale parallel and distributed computing:

- **Machine learning.** The success of machine learning in extracting models from large data sets, whether generated or observed, is revolutionizing many applications in science, business,

government, and other domains. Applications of machine learning will continue to expand as data is collected, curated and shared in more domains. Emerging techniques for federated learning will allow machine learning methods to operate on distributed data that is not fully shared and may even be encrypted, pushing computational requirements at the edge. The amount of data and computation required to train the most successful image classification problems has grown three orders of magnitude in the last eight years.

- **Data analysis.** Data sets will continue to grow in scale, driven in part by higher resolution, higher speed, and lower cost of cameras, detectors, sequencers and other sensors, but also by the increased connectivity of individuals and institutions, making it easier to aggregate and integrate data. Government policies on open sharing of federally funded scientific projects have also contributed to data availability. The benefits of combining data across modalities is well known, whether one is placing advertisements or analyzing microbial interactions in the environment. Large data problems also occur in less obvious areas; for instance, current approaches to the opioid addiction problem include developing biosignatures for addiction across -omics (genomics, etc.), outcome, sensor, and health record data for large populations, with analysis streams scaling potentially into the zettabytes.

- **Simulation.** Scientific simulation has expanded beyond traditional HPC simulations to also include high-throughput simulation campaigns to build large sets of simulated results, such as the Materials Genome Initiative[1] and climate modeling studies. Extreme-scale multi-faceted simulations of entire ecosystems are needed to evaluate proposed environmental sustainability initiatives. Simulations may be combined with optimization to solve inverse problems in application domains including medical imaging, large scientific experiments, and distributed sensors in mobile phones. Inverse problems, which aim to calculate from a set of observations the causal factors that produced them, increase the computational needs by factors of hundreds or thousands over today's largest simulations, but open new numerical pathways to science and uncertainty quantification. Inverse problems are used to reconstruct 3D molecular structures from x-ray imaging at large experimental facilities, interpret MRI data from medical tests, and improve our understanding of the fundamental laws of physics by matching simulations to cosmological or other observations.

These types of applications are not distinct but will be increasingly blurred and combined. Domain experts from the social sciences may use natural language to interface with automatically generated software components that scrape data from multiple websites around the world, build statistical models of complex social behavior, and use those models to make predictions about changes in demographics and changing requirements for housing, medical services, and other infrastructure. Smart cities will be instrumented with sensors that detect energy use in buildings and will optimize the complex power grid of dynamic renewable energy sources to manage energy creation, storage and distribution. Smart cities will also use comprehensive models of the transportation system, built from numerous data streams, to optimize for convenience, cost and energy efficiency. Self-driving cars will collect and analyze some data in-situ, but will also share data with nearby vehicles as well as stationary servers. Data from vibrational sensors, biosensors, carbon sensors and more

[1] `https://www.mgi.gov/`

will collect data with extremely low signal-to-noise ratios and be mined to understand earthquake safety and environmental risks and to aid the design of safe buildings, pollution management, and digital agriculture. Digital agriculture will help society address the enormous challenge of feeding the planet's growing population. It will be intensely data-driven and will involve integration of heterogeneous information from several kinds of in-ground or on-ground sensors (electrochemical, mechanical, airflow, etc.) as well as sensors in drones, satellites, etc. It will use machine learning and other data analysis to control schedules for planting, watering, fertilizing, and harvesting, and (in some cases) implement those actions robotically. These systems need to be predictive of normal behavior, adapting to changes in the time of day, the season, and social patterns of use, and must also adapt to extreme events brought on by natural disasters, cyberattacks, physical attacks, and failures in the systems.

Predictions will be made with simulations that combine first-principles physical laws with observational data or models learned from data, and will use machine learning and optimization methods to automatically guide the exploration of a large state space of simulations or experiments. Laboratory facilities will be increasingly automated with robotic "cloud" laboratories for experiments, and scientists will increasingly re-cycle and re-purpose data from other research groups and other disciplines. Some of the most challenging problems will combine noisy, diverse and multi-modal data sets for new insights, requiring the data to be easily searchable and widely available, labeled with and indexed by automatically computed metadata. A student from a small college or rural high school should be able to access data from international experiments and use remote computing for new scientific discoveries.

To meet the demands imposed by more massive datasets, higher data rates, and expensive search-based methods, applications will use larger and increasingly heterogeneous collections of hardware resources and software platforms, as extreme-scale parallel and distributed applications expand beyond their traditional homes in supercomputer centers and datacenters and span from the edge to the cloud, incorporating devices ranging from weak Internet of Things (IoT) nodes to small local servers to massive datacenters, and even span multiple geodistributed datacenters. Deciding where to place and when to move data and computation in such systems is a major challenge, which work on fog computing has only started to address.

4.2 Technology

An important technology trend that, while not new, will continue to shape the the design of parallel and distributed computing systems is the end of Moore's Law and Dennard scaling, and the resulting drive to improve performance in other ways, notably by increasing parallelism at all levels and by exploiting specialized hardware accelerators: Graphics Processing Units (GPUs), Digital Signal Processors (DSPs), image signal processors, etc. These two approaches overlap: increasing numbers of hardware accelerators provide more potential for parallelism [10].

Beyond Digital Computing. With the end of Moore's Law, there are new opportunities to consider new computing platforms that may yield orders of magnitude improvements in available computing power by breaking the digital computing abstraction in which computations are presumed to produce certain values. Emerging computational models, such as quantum computing, stochastic computing [2], and neuromorphic computing [21], offer the promise of extreme-scale

computations under current computing constraints (e.g., energy or power), with the trade-off that these computational approaches can only compute results up to a given precision or up to a given probability. While each of these platforms offers new opportunities if adopted, a key general opportunity is that by embracing such approximate computations (including developing new abstractions to reason about and manipulate them) there is the promise of new computing techniques that bring new scalability to existing technologies, such as increased parallelism by allowing data races with an acceptable bounded probability [18, 17, 14, 23].

5 Cross-cutting Concerns

Future challenges and research directions within Parallel and Distributed Computing will cut across disciplines to deliver integrated hardware and software systems to address complete application solutions. A key technology driver is that the traditional performance improvements from Moore's Law have plateaued, making computer architecture the primary mechanism for continued performance improvements. This means more parallelism, more specialization, and increasingly complex hierarchies of memory, processing and networks, which will require an overarching theme of rethinking the current state of software abstractions (Software Abstractions – Section 5.1). At the same time, applications will be increasingly complex, distributed and heterogeneous, taking advantage of large networks of computational devices both within data/compute centers and embedded throughout intelligent infrastructure and the environment. Future challenges and research directions will require: (1) new models of performance and correctness (Computational Models – Section 5.2), (2) applications, algorithms and systems that take advantage of application domain knowledge (Domain-Specific Design – Section 5.3), and (3) platforms that integrate diverse hardware and software components into portable and resilient systems (Heterogeneity – Section 5.4).

5.1 Software Abstractions

Parallel and distributed software and hardware have become increasingly sophisticated. To bridge the gap between high level user interfaces and complex hardware, developers leverage modular and reusable software with many layers of abstraction involving software development kits (SDKs), middleware, application-level libraries, system-level libraries, language runtime systems, cross-language interfaces, etc. Current abstractions have been invaluable in dealing with overall system complexity and improving programmer productivity, but new research is needed to address several challenges. As the complexity of the underlying system increases and there is increased need for high level control, the fundamental question for abstraction is which features to hide and which ones to expose.

Challenges

Abstractions may be designed with different goals in mind, including performance, scalability, ease of use, flexibility, compositionality, re-usability and verifiability (assurance). These often trade off against one another, and systems that allow users to manage the details of data movement, energy consumption, and resilience may be difficult to use, whereas a system that abstracts these considerations from users and automatically manages them may incur significant time or storage

overhead and might not manage them well for some applications. Each of these requirements can be considered within a particular usage context such as an energy budget, system scale, or application workload. There is need for more research on how to design abstractions for parallel/distributed systems that meet these goals, to what extent they can be hidden and how to expose them when necessary in an elegant and usable interface.

There is also perpetual tension between general abstractions and domain-specific abstractions. For example, load balancing can be done in a generic way based on overall system load, or it can be done an application level library that takes advantage of the specific data structures and computations, e.g., distributing blocks in an adaptive mesh refinement framework can optimize for both load balancing and locality, but that specialized software is not reusable in other domains. Consider extreme points in the parallel/distributed design space, ranging from physical simulations with abundant data parallelism deployed on batch-scheduled HPC facilities to inherently distributed IoT applications that use machine learning on a dynamically changing set of heterogeneous computing nodes that includes both small edge devices and powerful cloud systems. How much convergence can we achieve in the abstractions for building these different kinds of systems and applications?

Composition of abstractions can also be challenging, especially in extreme-scale systems, due to potential performance problems resulting from interactions between the components (e.g., contention for resources). Composition of libraries involves multiple challenges including differences between process and threading models, from the flat process model of MPI (a popular standardized message-passing interface for parallel computing) to more complex dynamic threading and accelerator parallelism. Another major issue is the consistency of data structure representations across libraries, especially for sparse or hierarchical data structures where the representations are tuned to particular use cases. In current practice, this often means an application is designed to use a single library (e.g., PETSc or Trilinos) and the data structures and data distributions it supports.

Research Directions

The traditional approach to designing an API for a parallel/distributed ADT or service is to make it look as much as possible like the API for the centralized version, e.g., by requiring linearizability or similar conditions. An interesting research direction is to try to look beyond this sequential mindset and instead develop intrinsically concurrent specifications, and techniques to compose and verify them.

Abstractions designed with performance in mind traditionally focus on running time and sometimes space. An increasingly important research direction is to design abstractions with energy efficiency in mind. Such abstractions are needed in datacenters as well as mobile devices and IoT. There may be more economic incentive to increase the energy efficiency of a datacenter than to make computations faster.

The emergence of improved non-volatile random-access memory (NVRAM) calls into question the traditional storage abstractions, namely, byte-addressable transient storage and block-oriented persistent storage. Rethinking this deeply ingrained structure may be necessary to obtain the maximum benefit from this technology. Should systems still use different data representations for in-memory and non-volatile storage? This has huge implications not only for sequential programs but also for parallel and distributed programs. For the latter, research needs to reconsider the abstractions for remote storage (both transient and non-volatile) as well as local storage, starting

with the question of whether to use the same or different abstractions for them. For example, remote direct memory access (RDMA) is becoming ubiquitous in data centers; what are the right abstractions for it?

As the programming community expands at all levels of expertise, it is important to design abstractions useful for a broad range of people. How can we design abstractions useful both for the novice programmer working on applications with lax performance requirements and for expert programmers working on applications with strict performance requirements?

Risks

Design of abstractions should be driven by real users, applications, and workloads representing a range of application domains, to ensure the abstractions address real needs. On the other hand, we should strive to design abstractions that are broadly applicable, not tied too closely to specific applications. Developing benchmark suites that reflect a sufficiently broad range of realistic usage scenarios for a class of applications can help guide the research community to develop abstractions with strong potential for practical impact.

Robust implementations are crucial for adoption. Implementations should support tool APIs for debugging, performance profiling, etc.

5.2 Computational Models

Future applications will create new first-order concerns that need to be encapsulated in new computational models to support formal reasoning about these properties at the level of algorithms as well as to support translating these algorithms to implementations that run on practical computational devices yet still retain these formal properties. These applications will require fundamentally new specifications of computation that must be reflected through the entire computing stack, from algorithmic descriptions to software and hardware.

Challenges

Our standard specification of a computational model supports reasoning about computational complexity as well as functional correctness, however there are three key challenges for rethinking computational models: 1) future applications will be multi-scale, spanning everything from small devices to large datacenter-scale or supercomputer-scale resources, 2) future systems will be heterogeneous, incorporating different devices with different computational attributes (e.g., efficiency, reliability, security), and 3) applications in a massively distributed environment supporting interactive and real-time applications will bring new challenges for reasoning about timing.

There are many unsolved problems at many levels of scale. In particular, investigating models and algorithms that are expressive at many levels simultaneously are of interest. As with traditional multi-level or multi-scale computing fabrics algorithms and implementations often need to be tuned to the parameters of each level. For example, there is a large body of work in program compilation on identify the parameters of loop transformations, such as unrolling, tiling, and blocking, to optimize for the memory hierarchy of the platform at hand [3]. There has been extensive work on cache-efficient algorithms, but there are many open questions on how to design efficient algorithms with deep memory hierarchies.

The set of devices in these massively multi-scale systems will be heterogeneous and therefore have different interfaces. Integrating emerging devices into resource-aware or resource-oblivious frameworks will require the development of new computational models for these devices. A particular area of interest is in understanding upcoming NVRAM technologies. For example, with Intel's Optane NVRAM, writes are significantly more costly than reads: read bandwidth is roughly 6 to 10 times the write bandwidth; in contrast, for DRAM, that ratio is typically in the range of 1.2 to 1.7 [24]. The cost difference between reads and writes for DRAM has generally been ignored in algorithm design. The much larger cost difference for NVRAM requires design of algorithms that take this difference into account and will require new techniques.

The nature of these computational models will also be varied, with disparate models for different types of devices. For example, a device with an alternative reliability or accuracy specification may be better characterized by its worst-case error or average-case error, which may then differ from the best characterization of another device [23]. Building computational models that unify different characterizations is an open challenge.

Of the many attributes of a device to be reified in computational models (latency, throughput, energy, power, etc.), a key consideration is that many future applications will operate in real-time or interactive computing modes. In these modalities, developers will seek guarantees about processing times and response times. Different applications may require different guarantees, such as worst-case response time or average response time. The challenge is that very large and complex systems are inherently uncertain: computations may be dynamically scheduled on a variety of different devices, each with different performance properties, varied load, and varied reliability. Given the scale of these systems and the inherent uncertainty in execution time at scale, these bounds may extend to constraints on multiple points of the full response time distribution, including tail bounds and its variance [8].

Research Directions

As with multi-level memory hierarchies, there are opportunities to investigate *resource-oblivious* approaches that seek out the same style of guarantees as classic *cache-oblivious* approaches, in that a single algorithm achieves near-optimal resource utilization regardless of the parameters of the underlying resources [9]. Other directions include developing reasonably simple models that capture the key features of these hierarchies, developing efficient algorithms in these models, and showing that they lead to efficient algorithms on actual machines with such memory hierarchies. In concert, there are opportunities to improve programmability by reflecting resource specifications in emerging abstractions (languages, libraries, and hardware) and building automated tools that automatically navigate the space of algorithms and configuration choices.

There are considerable opportunities to investigate how to formally model timing and ensure timing properties of the resulting algorithm and implementation. Key considerations also include how to navigate timing as an additional dimension in the optimization space to support, for example, trading resources, efficiency, reliability, or quality-of-result to achieve specific timing objectives.

Real-time, interactive computations will also need to consider how to execute effectively under timing constraints given that many computations are large or are backed by inferences from large sources of data. For example, to maintain real-time interactivity, computational modelling may need to consider how to integrate streaming computations that enable scalable incremental updates

to computational results as the dataset, workload, or environment changes dynamically.

5.3 Domain-Specific Design

The goal of Domain-Specific Design (DSD) is to develop complete algorithms-software-hardware solutions that absolutely maximize our objectives for a given domain. This stands in contrast to the Heterogeneity topic, where the goal is to design flexible abstractions that map across platforms, while here we desire domain-specific, and potentially platform-specific, abstractions. DSD can occur at the level of hardware, library, programming language or model.

Experience suggests that new opportunities will come from mating the full system (algorithms-software-hardware) with the computational abstractions of the application at hand [16, 11, 1]. For example, consider the case of CNNs (convolutional neural networks). The agreement on shared primitives, by researchers and developers in academia and industry spanning all levels of the computing stack, allowed innovation to flourish both at the hardware level—tensor processing unit (TPUs [11]), GPU tensor cores, new CPU instructions, new CNN chips—and at the software level—a proliferation of frameworks built off common libraries, including PyTorch, Tensorflow, Horovod, etc. While such interfaces embodying the right abstractions for a specific domain are difficult to establish, the potential payoff is huge, as they can unleash innovation both above and below the interfaces.

Challenges

The most fundamental challenge with DSD is that reaching the maturity level to gain widespread adoption is enormously challenging. Perturbing any part of the ecosystem—how applications are built, the compiler, the operating system, or the hardware—tends to break many things up and down the stack. This causes an inherent reluctance to change throughout the industry. Most successes have happened with sustained investment, usually supported by another market. For instance, the GPU is now widely accepted as a valid and valued architecture in computing, but this was a result of 15 years of persistent software investment, made possible by a large market for GPUs from gaming and graphics. Similarly, development of Field Programmable Gate Arrays (FPGAs) was sustained largely by the embedded applications market, before FPGAs saw substantial adoption in other areas.

Research activities and projects therefore need to go across scales – from basic proof of concept to determining how the ecosystem must change. Often, this is too big a scope for a single academic research project. A portfolio of research must be considered to allow breakthroughs to happen at a particular point in the hardware/software stack, but then to evaluate them in a larger context and, when appropriate, carry them through to success.

Promising breakthroughs need sustained research and development to translate to the marketplace, possibly on a scale of more than a decade. Buy-in from user communities, and other communities up and down the stack is essential, and needs to feed a co-design process. For example, new hardware APIs will be useful only with buy-in from the compiler community to generate code that exploits them.

Research Directions

There are a number or promising application areas for DSD. For example, genomics has potential for significant efficiency improvements from DSD, because of the simple form of the basic data (strings over a 4-character alphabet). There is also strong incentive to use DSD in edge devices and IoT devices, in order to achieve the desired functionality with limited local resources (energy, processing power, and so on). Even at the highest end of computing, there are significant research opportunities in DSD. For a broad mix of scientific applications, the last decade has brought application improvement at about half the rate of peak performance improvement. For example, the Sustained Petascale Performance (SPP) benchmark, a mix of eleven full applications and representative large science problems used by NSF to evaluate the performance of large scale systems, was recently used to compare performance of the NSF's previous flagship supercomputer, Blue Waters (deployed in 2013) and the newest one, Frontera (deployed in 2019). While Frontera outperformed Blue Waters on SSP by a ratio of more than eight to one on a per-node basis, the peak performance of the processors between the two systems have a ratio of sixteen to one. Domain-specific designs promise to provide application speedups closer to peak performance. For example, research should investigate how architectural tweaking of "mainstream" architectures for particular research domains could provide better speedups. In some cases, a "partially reconfigurable" architecture might yield the flexibility to accelerate several domains, but limited attention has been paid to this area.

While most research on DSD is domain-specific, some research directions support adoption of DSD across many domains. In particular, research on techniques that make systems easier to specialize and extend facilitate DSD. For example, designing programming languages and compilers to be extensible facilitates adding constructs needed to support domain-specific hardware. Recent advancement in open source instruction set architecture designs [19] and open source hardware implementations [20] bring new opportunities to more easily deliver application-specific hardware solutions. Generally, research is needed to make systems easier to specialize and extend at all levels of the computing stack, from hardware on up.

5.4 Heterogeneity

Moore's law and Dennard scaling have enabled large productivity increases through general-purpose computing over several decades. This model of computing provided a fixed, long-lived hardware-software contract that enabled software and hardware designers to innovate independently above and below the contract boundary. Future generations of productivity increases, however, are likely to come from application-specialized computing, which fundamentally changes the nature of this contract [1]. Computing systems will continue to exploit parallelism, but this parallelism will come from heterogeneous, application-specialized components [10] spanning compute, memory, storage, and communication, motivating a rethinking of hardware, software, and the hardware-software interface. This rethinking will shape the design of systems comprising a single package to large distributed ensembles, deployed for applications from the edge to the cloud, with potential for orders of magnitude improvements in system capability.

Challenges

Application-specific accelerators have been designed for a long time; however, as Moore's Law fades, there are more opportunities to exploit specialization and heterogeneity. At the same time, chip design cost has exploded, with architecture, software, and validation cost at hundreds of millions of dollars [13] and long time scales for concept to deployment. An urgent need is to develop tool flows to enable developing specialized systems much faster, cheaper, and for a broader set of application domains than feasible today (e.g., for regular and irregular applications; for end-application and infrastructure specialization; and for the edge and the cloud). Underlying such research is the question of whether there is a common set of computational models that can serve as a unifying architecture to support broad domains, without the inefficiencies associated with today's general-purpose architectures.

Research Directions

A foundation for the research necessary to meet these challenges is provided by recent nascent efforts in new abstractions for heterogeneous computing, by the success of new domain specific languages coupled with advances in program synthesis, code generators, and autotuners, and by the explosion in open source hardware, software, and design tools.

Much research has focused on designing a single accelerator for a single computation. A key research direction is to investigate how to design *systems* of multiple heterogeneous compute accelerators and heterogeneous memory, storage, and communication components. Such research must focus not just on specialized computation, but also on specialized communication and data movement to enhance the capabilities of the larger system. Research is also needed in techniques to build such systems out of intellectual property components designed by independent third-party entities, such as whole microprocessors or small domain-specific DSP cores. What technologies will enable reasoning about functionality, performance, correctness, and security of the larger system built from such components?

The evolving hardware/software contract motivates a new science of hardware-software codesign. This will require research in new abstractions and interfaces at different layers of the system. Can we develop an analog of the familiar instruction-set architecture (ISA) contract to abstract heterogeneity in a way that will enable independent innovation in hardware and software? If so, what should this new contract look like, and at which system layer should it be expressed? Or must hardware and software be truly codesigned, and if so, how? What is the rich ecosystem of domain specific languages, compiler technologies, and runtime systems that can enable new applications to be developed for and quickly deployed on—and that can influence—the emerging heterogeneous hardware landscape? How should these software layers deal with functional correctness, verification, security, and performance portability in the face of an evolving heterogeneous hardware landscape and hardware-software interface?

Risks

Work requiring coordination of communities across system layers is inherently risky. However, it is clear that industry will continue to march towards heterogeneous systems as the urgency from the decline of Moore's law increases. If the research community does not act quickly enough to

create open interfaces and new hardware-software stacks based on scientific rigor, there is a danger that industry practice will evolve to bake in short-term approaches. The current small number of vertically integrated companies dominating major sectors of computing further exacerbates this risk, creating a danger of closing large segments to new players and stifling innovation. An analogy is the baking of the x86 ISA as a de facto standard. It has since taken decades for alternative, more energy-efficient RISC designs to find a footing in the market place.

6 Disciplines

Research on parallel and distributed computing has historically spanned a wide range of disciplines. Addressing research challenges in parallel and distributed computing involves domain insight for application-specific design (HPC), algorithmic implementation (Theory and Algorithms), software architecture and implementation (Programming Languages, and Systems), hardware systems (Computer Architecture) and, in a cross-cutting fashion, the Security and Privacy aspects of each of these layers.

6.1 High Performance Computing (HPC)

Traditionally, HPC has referred to the scale-up of scientific and technical computing (primarily simulation) to the largest possible scales. This makes HPC somewhat of a meta-field of computer science, relying on innovations in architecture, systems, languages, and theory and algorithms, to produce a scientific instrument.

In the modern world, HPC is often interpreted more broadly to mean the scale-up of virtually any computational process, as HPC techniques are used to scale not only simulation but data analytics and machine learning/AI Applications as well. The term "Internet scale" has gained some traction for cloud-based scaling of large collections of loosely connected, largely asynchronous processes. In this report, we do not limit HPC to scientific and technical computing, but we do mean relatively tightly coupled parallel computing—the division of a large task into multiple pieces, rather than simply aggregating small independent tasks.

Challenges

The time is ripe for innovation and discovery within HPC itself. For a long time, interest in new parallel techniques was somewhat dampened by the continued progress from Moore's Law and Dennard scaling. As this process has slowed or stopped, "wait for faster chips" is no longer a way to make progress. At the highest end of computing, there is evidence that we have delayed the full impact of this problem by simply extending the life of systems, allowing for larger individual purchases (e.g., five years of equipment budget, rather than four). Most new systems that have appeared in the Top 10 in the world (as ranked by top500.org) now cost $100M to $300M; ten years ago, they cost $50M to $100M. The average age of systems on this list has also increased. The large budget has meant larger power footprint as well; the next generation of U.S. leadership systems will consume more than 40 Megawatts. Neither the cost nor power trend is sustainable.

New application domains have meant explosive growth in HPC usage, both in scale and in number of users. These new applications also bring new challenges that require additional research.

The new applications tend to be not just pure simulation, or pure machine learning, but blends of learning, analytics, and simulation in a single workflow.

Increasing irregularity of workloads and platforms poses a challenge. Irregularity of various kinds arises from multiple sources within advanced scientific applications. For instance, in molecular electronic structure, differences in local sparsity and symmetry leads to great variation in the computational and data intensity of tasks; in many-body physics, reduced-scaling methods replace computation on dense tensors with dynamically-screened computation either on block-sparse data structures or spatial trees. These features combine to produce an irregular and often small granularity of computation as well as highly data-dependent data flows. Furthermore, as applications become more complex (e.g., due to multiple physics models being coupled), and as they seek greater concurrency, it becomes necessary to run multiple dissimilar operations in parallel. Irregularity in HPC platforms is also increasing as a result of trends in modern computer architectures, such as dynamic power management varying processor speeds, heterogeneous nodes (e.g., some with GPUs or other accelerators), and deep memory hierarchies.

Research Directions

There is a need for research on more holistic approaches to applications that blend simulation, analytics, and machine learning, in order to develop systems and tools with high performance and extreme scalability for blended applications that exhibit a broad range of resource usage behaviors that may differ across components of the application and phases of its workflow. For example, I/O systems need to be redesigned to efficiently support more diverse I/O patterns, from a few large files (in traditional simulation) to many small ones (in AI and data analytics workloads), and scheduling policies need to be reconsidered to meet real-time constraints or near-real-time constraints that arise in applications involving streaming data.

Another important research direction is to amend existing approaches to conversion of scientific theories into computational implementations to handle irregularity more robustly, dynamically balancing computational load and network load across extreme-scale HPC systems.

A promising research direction is to explore the opportunities provided by new technologies that will impact HPC, from potentially revolutionary new technologies such as quantum computing and neuromorphic computing, to more evolutionary technologies such as deeper memory hierarchies resulting from the large scale introduction of non-volatile RAM or new classes of storage.

HPC can also benefit from further research into a variety of translational opportunities from other fields. Reduced precision, which has seen a resurgence of interest due to work with deep neural networks in GPUs, can be exploited in other HPC contexts. Deep Learning can be used in many ways to improve scientific applications in HPC, from pruning the parameter search space in ensembles to potentially replacing parts of the modeling workflow.

6.2 Theory and Algorithms

A key purpose of algorithmic theory is to be able to roughly predict in a portable manner how performance scales with problem size. Historically such algorithmic theory has been immensely useful in leading algorithm designers in the right general direction before worrying about low-level optimizations. It has also been very useful in determining what approaches to avoid. Because

asymptotics are particularly relevant for very large-scale data, and because performance is often the most important goal for large-scale systems, the theory of algorithms has the potential for significant impact on the design of algorithms for such systems. However, this requires creating a fine balance in the underlying models between accuracy, simplicity and generality. It is not hard to design a simple abstract model of no relevance, and it is not hard to design a hugely complicated and accurate model relevant to a single machine and algorithm.

Challenges

One of the largest challenges for the theory of algorithms is in developing appropriate models for the many new features of modern large-scale parallel systems. These features include complicated memory hierarchies, new memory technologies (e.g. NVRAM), heterogeneity, support for remote memory access (RDMA), frequency scaling, smart memories, energy saving modes, and application-specific hardware. The new technologies also present challenges, and opportunities, in algorithm design. A second challenge involves handling fault tolerance in extreme-scale distributed systems. As systems get larger, it becomes impractical to ensure that all components are always working. The challenge is to develop a theory of reliability for large systems built from heterogeneous components subject to different kinds and rates of faults. A third challenge is to consider environments with changing resources shared by multiple users. Most existing theory on parallel algorithms assumes full control over a fixed and static collection of resources, but this is likely to change with the advent of cloud resources, faults, and more dynamic workloads. A fourth challenge is in taking key ideas relevant to sequential algorithms and applying them in the context of parallelism, and with regards to new hardware features. An example is dynamic algorithms, as discussed below under Research Directions.

For each of these challenges, we discuss some research directions below. Theory goes beyond analysis of algorithmic costs, and touches areas of security, privacy, game theory, and learnability. These areas also all play an important role in large scale parallel and distributed systems and present many challenges.

Research Directions

One important research direction is in developing new models and algorithms that maximize the benefit of emerging hardware technologies. There are many potential areas of interest within this general area. General directions include relating models, proving lower bounds, and developing new algorithmic techniques. Cross-cutting research is particularly important here, since it is essential for the models to capture the salient features of new hardware.

Another research direction is to design algorithms that take advantage of specialized operations provided by compute accelerators, which are becoming more prevalent and more diverse. A more ambitious goal is more systematic and more automated design of algorithms, so we do not need to manually design new algorithms for many combinations of different problems and different hardware.

Faults may be relatively frequent in some extreme-scale distributed systems, e.g., IoT systems with inexpensive nodes with low reliability, limited power, intermittent network connectivity, etc. Design and analysis of algorithms for systems with frequent faults is understudied. Beyond applying

generic fault-tolerance techniques such as replication and checkpointing, research should study how to design algorithms that work efficiently with frequent faults. Another research direction is to understand how to best exploit NVRAM for reliability.

Design and analysis of energy-efficient algorithms will be increasingly important, as extreme-scale systems use increasing amounts of energy. Energy usage will need to be optimized simultaneously with traditional cost metrics such as time, space, and work. From a theoretical perspective, an important research direction is to develop and solve multi-parameter models, which are notoriously hard to optimize.

Resource sharing will be increasingly important for cost-efficient and energy-efficient extreme-scale parallel and distributed computing. To date there has been significant research in various communities, each addressing different aspects of scheduling for resource sharing. On the queueing theory side, there has been significant work on arrivals of single processor jobs to be served by the system with the goal of minimizing response time. In the parallel algorithms and parallel programming communities, there has been significant work on scheduling a single task among a fixed number of processors to minimize runtime (e.g. work-stealing schedulers). In the OS community, there has been much work on schedulers for real operating systems. A research direction is to put many of these ideas together in the context of systems that have arrivals of multiple jobs, each of which is itself capable of using multiple cores/processors. Although this topic has been touched on in the systems community, there is very little theory on it.

Given the conflict between the simple models desired and needed by the theory community, and the complication of real hardware, another research direction is to develop a multi-layered approach to algorithm design and analysis with a succession of more refined models, each capturing more detail of the hardware and better predicting runtime or other resource usage.

There are also classic algorithmic and complexity problems in parallel computing that could have significant practical and theoretical consequences, including:

- Dynamic algorithms that maintain some property while a data structure is updated dynamically have been studied extensively in the context of sequential algorithms—for example, algorithms that support adding and deleting edges of a graph while maintaining its minimum spanning tree. However, there has been little work on studying algorithms that make many updates in parallel. Such bulk dynamic updates could have significant practical relevance to maintaining large data sets, such as a web graph, as they are rapidly updated on the fly. Indeed, there are already several systems that support such updates, although with no theoretical guarantees.

- Use of data structures is fundamental for design of many algorithms. Sequential algorithms can use data structures as "black boxes": as long as we know a sequential data structure's interface and operation costs, we can design and analyze a sequential algorithm that uses it in a composable fashion. For parallel algorithms that use data structures, the situation is much more complicated due to contention. A fundamental problem is to design data structures and techniques that allow for composability in both the design and analysis of parallel algorithms.

Risks

As mentioned above, a theoretical cost model can fault in two directions: it can be irrelevant to real machines, or it can be too complicated to be usable and portable. The risk of a lack of a good model is therefore the large development costs wasted on what end up being inadequate or non-portable algorithms and codes. Another risk is diverging from programming methodology. Ultimately, algorithms are programs, and if the programming models diverge from algorithmic models, programmers are unlikely to use the theoretical models, and algorithm designers will find it difficult to implement their algorithms. Another risk is in developing models that optimize for the wrong thing. For instance, if our theoretical models optimize for the worst-case behavior, but the worst case never arises in the real-world application, or the application cares only about the average cost or the tail bound, then the theoretically-best algorithm (designed to optimize for the worst case) may not be best for the application. Therefore, we must consider application needs when designing models. Cross-cutting efforts that develop theory and verify it experimentally, or use it for specific applications, will help here.

6.3 Programming Languages

The programming language community considers itself a steward of the abstractions in the system, with the overall goal of designing and managing (i.e., composing, validating, and compiling) the abstractions of the system to achieve standard objectives, such as efficiency, reliability, programmability, security, and privacy. A key hypothesis of the community is that if the programming language enables developers to program with abstractions that are well-suited to the domain and system, then the language environment (analyzer, compiler, and runtime system) can help automate the process required to deliver a system that achieves its objectives.

Challenges

Heterogeneity stands as one of the greatest emerging concerns for future programming language research. This heterogeneity takes two forms: domain heterogeneity and system heterogeneity.

Domain heterogeneity refers to the fact that future applications are themselves cross-cutting, spanning many problem domains and associated computation styles. For example, the language-level abstractions for domains that are largely simulation versus data analysis versus inference are disparate. In the case of deep learning, the *inference* process consists chiefly of tensor algebra operations executed on a single machine, while the *training* process requires computing gradients – a fundamentally different abstraction of a computation – and exchanging gradient information among a set of distributed workers.

System heterogeneity refers to the fact that future systems themselves will consist of a set of devices, each with unique computing interfaces (e.g., CPU versus GPU versus FPGA) and unique physical locations in the overall system. These devices' programming interfaces and their attributes (e.g., efficiency, reliability, security) will therefore each be unique. For example, these systems will export multi-level memory models, adding emerging NVRAM technologies and distributed compute units to the traditional memory hierarchy. Programming for future applications and systems will therefore require navigating the Cartesian product of domain abstractions and device attributes to deliver an application that meets its objectives.

Research Directions

The programming languages community must therefore 1) develop new semantic and syntactic abstractions for emerging domains, such as probabilistic modeling and programming, differentiable programming, and tensor algebra computations, 2) develop new tools and techniques for rapidly integrating new semantic and syntactic abstractions into programming systems, 3) develop new tools and techniques for composing language-level abstractions, 4) develop new semantic and syntactic abstractions to express device attributes, 5) automate device attribute characterization, such as reliability analysis, 6) develop techniques for mapping abstractions (and their compositions) to devices, and 7) develop techniques for reasoning about the efficiency, reliability, security, and privacy of the resulting systems despite extreme variation in abstractions across both domains and devices.

Risks

Adoption is the biggest risk in programming languages research. New tools and techniques that arrive as independently developed languages or libraries that exist outside of the natural momentum of widely adopted languages and libraries often see limited adoption or see adoption only at long timescales. Where the community seeks to achieve impact through broader adoption, the community must consider which of many design approaches may be appropriate for new technology: namely, design at the language level, language feature level, or library level.

Working in concert with challenges to adoption are classic challenges to developing languages and libraries that are well-informed or developed in collaboration with practitioners. Results developed within the cottage industry of small, isolated, academic efforts have the risk of being properly reasoned, but infeasible to integrate in practice.

6.4 Systems

The increasing heterogeneity of hardware platforms and the shift towards more domain-specific systems are creating new challenges at all levels of the systems stack for extreme-scale computing.

Challenges

In addition to the always-present research challenge of maximizing performance on new hardware platforms, multiple new challenges are becoming important with the recent evolution of extreme-scale systems.

First, *correctness* has become a major concern with changes in hardware platforms, because we often find that codes that ran well on previous systems are giving different results on new ones due to differences in the hardware semantics. For example, new accelerators may have subtly different behavior for floating-point applications, or increased concurrency may lead to different end results. There is currently no support in the systems stack to identify and prevent such issues, leading to potentially erroneous results and costly efforts to detect and debug them.

Second, the rise of heterogeneous hardware and domains-specific libraries means that applications are increasingly composed of many separately designed, domain-specific components. Optimizing *across* components will be increasingly important to achieve end-to-end performance.

Third, security is an understudied aspect of extreme-scale systems that will be critical in decentralized or privacy-sensitive settings such as edge computing and the public cloud. Security also goes hand-in-hand with correctness to ensure that our existing applications behave well in the presence of adversarially controlled input data, which is a topic with limited research attention so far. These applications are so challenging to monitor that even basic attacks such as buffer overflows may occur undetected.

Finally, there is a severe lack of realistic large-scale benchmarks for extreme-scale systems. Many research papers artificially scale up small workloads such as publicly available graph datasets, often creating input data with very different properties from real-world application.

Research Directions

To reliably support future machines, compilers, debuggers and testing methods will need to evolve to ensure correctness. An increased focus on correctness would also facilitate portability of extreme-scale applications across emerging hardware platforms.

To deliver performance in heterogeneous environments, research on new intermediate representations that can combine high-level operators on different devices, such as the Multi-Level Intermediate Representation (MLIR) [15] and the Heterogeneous parallel Virtual Machine (HPVM) [12], is a promising direction. However, efficient composition of domain-specific libraries needs to be considered at all levels of the software stack, from interfaces to the accelerators themselves up to library designs and compilers.

To support well-validated research, there is a need for new, well-designed benchmarks in the graph processing, high performance computing and machine learning domains.

Risks

Lack of suitable benchmarks for new extreme-scale applications could lead to a series of research works tackling the wrong problem. In addition, ignoring mainstream datacenter and machine learning systems could lead to duplicated effort or forfeit an opportunity to influence the broader hardware and software roadmap. More connections with researchers and developers in these domains are necessary. Moreover, a closer connection with commercial domains would allow research to influence and create collaboration around the top challenges. Finally, because there are multiple viable design directions for extreme-scale computing (e.g., domain-specific versus general-purpose intermediate representations and compilers), the community needs to explore and compare these approaches in parallel instead of "betting the farm" on a single approach that may fail in practice.

6.5 Architecture

At the heart of the field of computer architecture is the definition of the hardware-software interface, classically expressed as the instruction set architecture (ISA). This definition was first embraced by the IBM 360 family of machines, unleashing decades of innovation by hardware architects innovating below the ISA and software architects innovating above it. The emerging importance of increasingly application-specialized heterogeneous parallel systems and the underlying diversity of software-exposed hardware features has reopened the question of what the hardware-software interface should be. This unresolved and evolving definition of the hardware-software interface impacts the entire

discipline of computer architecture. In the words of the 2017 Turing Award laureates John L. Hennessy and David Patterson, we have entered the "golden age for computer architecture."

Challenges

Many challenges and research directions for the field of computer architecture are evident from the cross-cutting themes described earlier in this report and highlighted below. A key theme is that much recent architecture research has explored point solutions in the form of accelerators for individual computations, but there are many challenges when deploying such specialization and heterogeneity at scale for a large number of application domains with rapidly changing requirements.

Research Directions

What principles should guide architectures when considering scaling to full systems deploying multiple accelerators and application-specializations in compute, memory, communications, and storage for a large number of application domains? How should we expose the diversity of such systems to software? What principles should guide the design of the hardware so that it is amenable to easily programmable software interfaces? What models of special-purpose architectures will result in general-purpose design methodologies and tools that can quickly and cheaply transform an application (or domain or algorithm) specification to the required hardware and software stack? What are such agile design methodologies? How can we design systems out of components that come from multiple third parties, while ensuring verification, security, performance portability, functional correctness, and other desirable properties?

Although CMOS transistor scaling is reaching its end, there is an opportunity to leverage several new technologies, e.g., Resistive Random Access Memory (RRAM) and other non-volatile memories, embedded logic in memory, mixed signal devices, and 3D stacking. These technologies introduce fundamentally new capabilities and/or affect well-established design rules of thumb. They also affect upper layers of the software stack in fundamental ways, e.g., by providing new avenues for trading off computation accuracy for efficiency, and providing persistence.

Risks

The need to iterate rapidly over hardware designs in conjunction with software to respond to changing application requirements across a large number of domains presents an inherent risk. This risk can be mitigated by research on stable software abstractions to test new hardware ideas and research on evaluation tools and benchmarks.

6.6 Security and Privacy

Security and privacy is a critical concern for many extreme-scale distributed systems. The workshop identified three specific emerging domains in which they will be especially important: edge computing, confidential cloud computing, and multi-party computing.

Challenges

Each of the domains mentioned above poses distinct challenges for research in security and privacy.

Massive-scale edge-computing applications, such as sensor or video monitoring, require sophisticated computation but cannot centralize the data due to communication costs. Updating extreme-scale algorithms to work in this setting, while addressing the security and privacy issues involved, is a difficult challenge.

With clouds increasingly becoming the computing environment of choice for extreme-scale industrial applications, there are significant concerns about security. New ideas across cryptography and system/hardware design are enabling much more efficient confidential computing methods, some of which are already being adopted by cloud providers. For example, Microsoft's Azure confidential computing[2] and Google's Asylo framework for confidential cloud computing[3] build on hardware-supported Trusted Execution Environments (TEEs), such as Intel SGX. Running extreme-scale applications in these environments, and using extreme scale to accelerate confidential computing techniques, are exciting research directions.

Multi-party computation (MPC) schemes are capable of providing strong security properties in distributed computations such as queries across private datasets. In particular, they offer security guarantees that continue to hold even if some users or servers are compromised. However, known schemes for multi-party computing incur high overhead. New algorithms and system designs are needed for these schemes to scale.

Research Directions

There are two broad research directions here: (1) developing techniques to ensure security and privacy properties for extreme-scale applications deployed in decentralized distributed systems, and (2) using extreme-scale computing to accelerate confidential computing schemes in order to improve security for more applications.

Extreme-scale computing has always been concerned about locality and communication costs, but bringing it to new settings such as edge computing and multiparty computation will require system designers to consider security and privacy too. For example, consider the problem of monitoring millions of video streams to search for a behavior or perform a running computation (e.g., track a large number of objects). Video data at this scale is far too large to aggregate in a central location, so any algorithm will have to be distributed across local sites and intermediate aggregation points. Research is needed to develop such algorithms that can also gracefully handle faulty or compromised sensors and implement strong privacy controls. For example, if the data concerns private citizens, as in the case of video surveillance in public areas, it may be necessary to ensure that a user can search for a specific person only if they have an appropriate warrant. The capability to successfully deploy such extreme-scale sensing and monitoring applications could have major benefits for commercial and governmental applications as well as scientific research. Likewise, with the increasing prevalence of cloud computing, cloud technologies built on Trusted Execution Environments are offering new means to protect remote computation, but extreme-scale algorithms need to be modified to work within the constraints of these cryptographic and hardware-based isolation mechanisms.

A second research direction is to use extreme-scale computing to scale up confidential computing

[2] `https://azure.microsoft.com/en-us/solutions/confidential-compute/`

[3] `https://cloud.google.com/blog/products/gcp/introducing-asylo-an-open-source-framework-for-confidential-computing`

mechanisms, allowing them to be used for larger applications than currently feasible. Theoretical research and systems research have made great strides in the past few years to accelerate powerful cryptographic techniques such as Fully Homomorphic Encryption (FHE) and verified computation, to the point where many of these are within a factor of 10^3 in overhead compared to direct computation for important applications. While this overhead is acceptable for some smaller applications, research is needed to make these techniques scale to larger applications. Much of the underlying computation in these cryptographic schemes is parallelizable, suggesting that high-performance computing techniques may be a viable approach to further scale up these algorithms. Recent results show the promise of this approach, e.g., use of compiler techniques to optimize FHE computations [5], and acceleration of cryptographic computations by running them on GPUs or FPGAs. Further advances in this research direction could enable important computations such as medical data sharing or federated machine learning (in which mobile personal devices improve a shared machine-learning model using local information, and then send a summary of the changes to the cloud for aggregation with changes from other mobile devices) to be done securely, reducing the risk of exposure of private data.

Risks

One risk to the success of research in this direction is that the performance of techniques for decentralized and private extreme-scale computing might not improve enough to be practical for very large applications of interest. Nonetheless, existing techniques are already practical for many smaller-scale applications with real impact (e.g., voting and health datasets), and we expect this research will significantly expand the range of applications that can be supported securely, even if it does not reach the top range of scale. A second risk is that the parallel and distributed computing research community might develop techniques based on unrealistic or incomplete threat models. This risk can be mitigated by collaborating with security researchers and professionals, to ensure that the threat models and security guarantees considered are viable.

7 Closing Thoughts

The landscape of computing has changed as computation has increased in ubiquity and scale. Computation is now performed in more domains and devices than ever, and emerging applications in AI/ML and Autonomy are some of the most computationally intensive applications to date. At the center of this innovation is the field of Parallel and Distributed Computing, which provides the basic abstractions, algorithms, and systems for computing across different devices at scale. For these emerging applications to succeed, future research in the field must move forward to overcome the substantial interdisciplinary challenges faced by the Parallel and Distributed Computing community.

References

[1] Sarita Adve, Ras Bodik, and Luis Ceze. I-USHER: Interfaces to unlock the specialized hardware revolution, April 2019. Outbrief of a DARPA/ISAT study. `http://rsim.cs.illinois.edu/Talks/I-USHER.pdf`.

[2] Armin Alaghi and John P. Hayes. Survey of stochastic computing. *ACM Transactions on Embedded Computing Systems*, 12(2s), 2013.

[3] David F. Bacon, Susan L. Graham, and Oliver J. Sharp. Compiler transformations for high-performance computing. *ACM Computing Surveys*, 26(4), 1994.

[4] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, JJ Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Dale Woodford, Yasushi Saito, Christopher Taylor, Michal Szymaniak, and Ruth Wang. Spanner: Google's globally-distributed database. In *Proceedings of the USENIX Symposium on Operating Systems Design & Implementation*. USENIX, 2012.

[5] Roshan Dathathri, Olli Saarikivi, Hao Chen, Kim Laine, Kristin E. Lauter, Saeed Maleki, Madanlal Musuvathi, and Todd Mytkowicz. CHET: an optimizing compiler for fully-homomorphic neural-network inferencing. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM, 2019.

[6] Jeffrey Dean, Greg S. Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc'Aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, and Andrew Y. Ng. Large scale distributed deep networks. In *Proceedings of the Conference on Neural Information Processing Systems*, 2012.

[7] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified data processing on large clusters. In *Proceedings of the USENIX Symposium on Operating Systems Design & Implementation*. USENIX, 2004.

[8] Christina Delimitrou and Christos Kozyrakis. Amdahl's law for tail latency. *Comunications of the ACM*, 61(8), July 2018.

[9] Matteo Frigo, Charles E. Leiserson, Harald Prokop, and Sridhar Ramachandran. Cache-oblivious algorithms. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, 1999.

[10] Mark D. Hill and Vijay Janapa Reddi. Accelerator-level parallelism. arXiv preprint arxiv:1907.02064 [cs.DC], 2019. `https://arxiv.org/abs/1907.02064`.

[11] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark

Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the International Symposium on Computer Architecture*. ACM, 2017.

[12] Maria Kotsifakou, Prakalp Srivastava, Matthew D. Sinclair, Rakesh Komuravelli, Vikram Adve, and Sarita Adve. HPVM: Heterogeneous parallel virtual machine. In *Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2018.

[13] Mark Lapedus. Big trouble at 3nm, June 2018. `https://semiengineering.com/big-trouble-at-3nm/`.

[14] Sasa Misailovic, Deokhwan Kim, and Martin Rinard. Parallelizing sequential programs with statistical accuracy tests. *ACM Transactions on Embedded Computing Systems*, 12(2s), May 2013.

[15] The Multi-Level Intermediate Representation compiler infrastructure. `https://github.com/tensorflow/mlir`.

[16] Andrew Putnam, Adrian Caulfield, Eric Chung, Derek Chiou, Kypros Constantinides, John Demme, Hadi Esmaeilzadeh, Jeremy Fowers, Jan Gray, Michael Haselman, Scott Hauck, Stephen Heil, Amir Hormati, Joo-Young Kim, Sitaram Lanka, Eric Peterson, Aaron Smith, Jason Thong, Phillip Yi Xiao, Doug Burger, Jim Larus, Gopi Prashanth Gopal, and Simon Pope. A reconfigurable fabric for accelerating large-scale datacenter services. In *Proceedings of the International Symposium on Computer Architecuture*, 2014.

[17] Kaushik Rajan and Bill Thies. ALTER: Exploiting breakable dependences for parallelization. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM, 2011.

[18] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Proceedings of the Conference on Neural Information Processing Systems*, 2011.

[19] RISC-V instruction set architecture specification. `https://riscv.org/specifications/`.

[20] Open source RISC-V cores. `https://riscv.org/risc-v-cores/`.

[21] Catherine D. Schuman, Thomas E. Potok, Robert M. Patton, J. Douglas Birdwell, Mark E. Dean, Garrett S. Rose, and James S. Plank. A survey of neuromorphic computing and neural networks in hardware. arXiv preprint arxiv:1705.06963 [cs.NE], 2017. `https://arxiv.org/abs/1705.06963`.

[22] Noam Shazeer, Youlong Cheng, Niki Parmar, Dustin Tran, Ashish Vaswani, Penporn Koanantakool, Peter Hawkins, HyoukJoong Lee, Mingsheng Hong, Cliff Young, Ryan Sepassi, and Blake Hechtman. Mesh-tensorflow: Deep learning for supercomputers. In *Proceedings of the Conference on Neural Information Processing Systems*, 2018.

[23] Phillip Stanley-Marbell, Armin Alaghi, Michael Carbin, Eva Darulova, Lara Dolecek, Andreas Gerstlauer, Ghayoor Gillani, Djordje Jevdjic, Thierry Moreau, Mattia Cacciotti, Alexandros Daglis, Natalie Enright Jerger, Babak Falsafi, Sasa Misailovic, Adrian Sampson, and Damien Zufferey. Exploiting errors for efficiency: A survey from circuits to algorithms. arXiv preprint arxiv:1809.05859 [cs.AR], 2018. `https://arxiv.org/abs/1809.05859`.

[24] Alexander van Renen, Lukas Vogel, Viktor Leis, Thomas Neumann, and Alfons Kemper. Persistent memory I/O primitives. arXiv preprint arxiv:1904.01614 [cs.DB], 2019. `https://arxiv.org/abs/1904.01614`.

A Workshop Overview

The primary goal of the NSF Workshop on Future Directions for Parallel and Distributed Computing was to elicit input from the research community to identify and promote important and groundbreaking research directions in the area of extremely scalable parallel and distributed computing. The workshop also aimed to broaden participants' perspectives and provide a venue to form new collaborations. It brought together NSF Scalable Parallelism in the Extreme (SPX) Program PIs conducting research in this area and other researchers from relevant computer science disciplines who may bring novel ideas and perspectives. Participation was by invitation only.

The workshop was held on June 22, 2019, in the Phoenix Convention Center, as part of the ACM Federated Computing Research Conference (ACM FCRC 2019). The workshop featured spotlight talks and lightning talks in the morning, and two breakout sessions in the afternoon. A detailed program appears in Appendix E. On June 23, the organizers had a half-day meeting to begin work on this report.

A.1 Organizers

Chairs
Michael Carbin, MIT
Scott D. Stoller, Stony Brook University

Steering Committee
Sarita Adve, University of Illinois at Urbana-Champaign
Kunal Agrawal, Washington University in St. Louis
Guy Blelloch, Carnegie Mellon University
Dan Stanzione, University of Texas at Austin; Texas Advanced Computing Center (TACC)
Katherine Yelick, University of California, Berkeley; Lawrence Berkeley National Laboratory
Matei Zaharia, Stanford University

B Cross-Cutting Themes

The workshop included breakout discussions of the following cross-cutting themes.

B.1 Abstractions

Parallel and distributed software and hardware have become increasingly sophisticated. To bridge software and hardware, developers leverage a multi-layer software stack. This multi-layer stack has led to layers and layers of abstractions (e.g., SDK, library interfaces, boundaries between languages) that limit the programming system's understanding of applications and hence limit the system's ability to optimize their executions on the hardware platform at hand.

This motivates rethinking the boundaries between layers of the software/hardware stack, especially in light of the opportunities provided by higher-level parallel languages.

Discussion Questions

- What are the existing abstractions used in practice for parallel/distributed computing?
- Which computations are well-captured by existing practice?
- Where do contemporary abstractions fall short? For example, when do we need to violate these abstractions to achieve one of our objectives?
- What new abstractions are promising? What challenges do they solve and pose?

B.2 Computational Models

There is an opportunity to design new computational models, algorithms, and corresponding hardware systems that when mated shield users from the increasingly complicated details of parallel and distributed systems, algorithms, and programming. To what extent can we shield users (e.g., computational scientists) from those intricacies by enabling them to write high-level code that has provably good performance on hardware in practice?

Bridging the gap from high-level specifications to practical, low-level performance will require new hardware that is amenable to algorithmic modeling, general frameworks for designing algorithms for such model(s), and translators that provably map algorithms into implementations that run with performance guarantees on the hardware in question.

Discussion Questions

- What are the fundamental computational paradigms that we should be building machines to implement?
- How will we design algorithms on top of those computational models?

B.3 Heterogeneity

The computing landscape has evolved to include a wide variety of different modalities and hardware platforms. These modalities include computing in the cloud and at the edge (e.g., mobile, wearable). The hardware platforms include not only the dominant platforms in each of these modalities, but also the emerging landscape of reconfigurable computing fabrics and hardware accelerators.

How do we develop flexible abstractions that support the development of tools to automatically map computations to different modalities and hardware platforms with acceptable (but perhaps not optimal) performance in our objectives?

Discussion Questions

- What are the different computational modalities for parallel/distributed computation?
- What are the different hardware platforms among these modalities?
- What are the constraints of these modalities that dictate different solutions to parallel/distributed computation?
- What are the primary technical challenges that these constraints impose?
- What existing abstractions are used?
- What new abstractions are promising? What challenges do they solve and pose?

B.4 Domain-Specific Design (DSD)

Contemporary wisdom suggests that new opportunities will come from mating the full system (algorithms-software-hardware) with the computational abstractions of the application at hand. The goal of DSD is to develop complete algorithms-software-hardware solutions that absolutely maximize our objectives for a given domain. This stands in contrast to the Heterogeneity topic, where the goal is to design flexible abstractions that map across platforms, while here we desire domain and – potentially – platform-specific abstractions.

Discussion Questions

- What is the current practice for developing custom algorithm-software-hardware support for an application domain?
- What are the skills and roles required of the team to deliver such an application?
- What tools would that team need? Do these tools exist?
- Are existing tools usable enough to make this possible?
- What research needs to be done to create this tooling or to make existing tools easier to manage?

B.5 Decentralization and Privacy

There are modern distributed applications whose utility comes from the shared participation of many users. For example, social networks, large-scale deep learning, and shared ledgers are valuable because they align and aggregate the data and computation of many users into a productive good. At the same time, users are concerned about the security/privacy of their data in the standard regime of centralized computation. Instead, there is a demand for decentralized versions of these applications in which users retain control of their data and sensitive computations are done locally. Federated machine learning, confidential computing, edge/mobile computing and blockchains are a few examples.

Discussion Questions

- What are the challenges that decentralize pose to meeting our objectives for parallel/distributed computing?

C Breakout Group Worksheet

Each breakout group was asked to discuss the following concerns.

- **Applications and Capabilities.** What are the applications (e.g., AI, scientific simulation, social applications) and/or capabilities (e.g., brain scale AI, long-timescale simulation, billion-entity applications) that primarily motivate discussion topic?
- **Scale.** What is the scale of these capabilities? Is this a 2x increase in capability or a 100x increase?
- **Challenges.** What are the major scientific/engineering challenges that are related to this topic that we need to address to achieve these capabilities?
- **Opportunities and Emerging Technologies.** What are the major opportunities and research directions that can shed light on this topic? For example, are there new massive datasets and workloads? Are there new algorithmic frameworks? Are there new programming languages or systems? Are there new hardware designs?
- **Scope.** Which of these challenges and opportunities are wholly new? Which are evidenced with existing applications and platforms at the current scale of their capabilities?
- **Risks.** What are the main threats to success for research? How do we hedge against them?
- **Intellectual Merits.** How will investigating these research opportunities and directions advance our knowledge and understanding of the application and/or disciplines?
- **Interdisciplinary Research.** What are the needs and opportunities for interdisciplinary research?
- **Broader Impacts.** How will investigating these research opportunities and directions benefit society or advance desired societal outcomes?

D Participants

Participants included the co-chairs, the steering committee (except Guy Blelloch), Anindya Banerjee (NSF/CISE/CCF), Vipin Chaudhary (NSF/CISE/OAC), Yuanyuan Yang (NSF/CISE/CCF), and the people listed below.

Acar, Umut
Aiken, Alex
Altman, Erik
Arora, Ritu
Boehm, Hans
Campanoni, Simone
Chapman, Barbara
Chowdhury, Rezaul
Das, Reetuparna
Dinda, Peter
Ding, Chen
Eijkhout, Victor
Fineman, Jeremy
Gokhale, Maya
Gordon, Colin
Hall, Mary
Harchol-Balter, Mor
Harrison, Robert
Hill, Mark
Jha, Shantenu
Kjolstad, Fredrik
Kulkarni, Milind
Kuper, Lindsey
Lee, Hsien-Hsin Sean
Lee, I-Ting
Li, Hai
Li, Xiaosong
Liu, Xu
Liu, Zhenhua
Lu, Shan
Luchangco, Victor
Majumdar, Amit
Martonosi, Margaret
McKinley, Kathryn S.
Miller, Heather
Moss, Eliot
Musuvathi, Madan
Nagarakatte, Santosh
Norman, Michael
Palsberg, Jens
Panda, Dhabaleswar K.
Patra, Abani
Peng, Richard
Prasanna, Viktor
Ragan-Kelley, Jonathan
Ramachandran, Vijaya
Reinhardt, Steve
Sampson, Adrian
Scott, Michael
Shen, Xipeng
Shun, Julian
Sinclair, Matt
Solomonik, Edgar
Spear, Michael
Sundar, Hari
Wentzlaff, David
Yan, Gu

E Workshop Program

8:15am	Continental Breakfast
9:00am	Opening Remarks: Anindya Banerjee, Michael Carbin, and Scott Stoller
9:20am	Spotlight Talk: Hai Li (Duke), *Decentralized Machine Learning*
9:50am	Spotlight Talk: Abani Patra (U. at Buffalo), *Application Kernels from NSF Supercomputers* Lightning Talk: Victor Eijkhout, *Integrative Parallel Programming*
10:20am	Lightning Talks: Applications and Architectures Mark Hill (U. of Wisconsin - Madison), *Accelerator-level Parallelism* Yan Gu (MIT), *Write-efficient algorithms for NVRAMs* Jens Palsberg (UCLA), *Quantum Computing* Erik Altman (IBM), *The Worldwide Swarm* Maya Gokhale (LLNL), *Extreme Heterogeneity as the Norm* Robert Harrison (Stony Brook U), *Roles of Architecture, Algorithm, & Programming Model*
11:00am	Break
11:20am	Spotlight Talk: Matei Zaharia (Stanford), Optimizing DNNS with Automated Mapping and Algebraic Substitutions
11:50am	Lightning Talks: Abstractions and Algorithms Umut Acar (CMU), *Disentanglement for Efficient Parallelism* Peter Dinda (Northwestern), *Interweaving the Parallel Hardware/Software Stack* Fredrik Kjolstad (MIT), *Abstraction without Friction* Xipeng Shen (NCSU), *"Abstraction Wall", Compilers, and Machine Learning* Vivek Sarkar, Unstructured Parallelism Considered Harmful -- Using Structured Parallelism for Enhanced Software Performance and Verification Julian Shun (MIT), *Parallel Graph Analytics* Mor Harchol-Balter (CMU), *Optimal Scheduling of Parallel Jobs*
12:30pm	Lunch
1:30pm	Remarks: Michael Carbin and Scott Stoller
1:35pm	Cross-Cutting Breakout Groups Abstractions Computational Models Heterogeneity Domain-Specific Design (DSD) Decentralization and Privacy
2:45pm	Presentations from Cross-Cutting Breakout Groups
3:30pm	Break
4:00pm	Research-Area Breakout Groups Architecture HPC + Applications Programming Languages Systems Theory + Algorithms
4:55pm	Presentations from Research-Area Breakout Groups
5:25pm	Closing Remarks: Michael Carbin and Scott Stoller