INTER-DISCIPLINARY RESEARCH CHALLENGES IN COMPUTER SYSTEMS FOR THE 2020s

Albert Cohen[†], Xipeng Shen[‡], Josep Torrellas[*], James Tuck[‡], and Yuanyuan Zhou[◇]

Sarita Adve, Ismail Akturk, Saurabh Bagchi, Rajeev Balasubramonian, Rajkishore Barik, Micah Beck, Ras Bodik, Ali Butt, Luis Ceze, Haibo Chen, Yiran Chen, Trishul Chilimbi, Mihai Christodorescu, John Criswell, Chen Ding, Yufei Ding, Sandhya Dwarkadas, Erik Elmroth, Phil Gibbons, Xiaochen Guo, Rajesh Gupta, Gernot Heiser, Hank Hoffman, Jian Huang, Hillery Hunter, John Kim, Sam King, James Larus, Chen Liu, Shan Lu, Brandon Lucia, Saeed Maleki, Somnath Mazumdar, Iulian Neamtiu, Keshav Pingali, Paolo Rech, Michael Scott, Yan Solihin, Dawn Song, Jakub Szefer, Dan Tsafrir, Bhuvan Urgaonkar, Marilyn Wolf, Yuan Xie, Jishen Zhao, Lin Zhong, and Yuhao Zhu[+]

[†]INRIA France [‡]North Carolina State University
[*]University of Illinois Urbana-Champaign [◇]University of California, San Diego
[+]The affiliations of these authors are listed at *https://www.asplos2018.org/grandchallenges/*

September 2018

Sponsored by the National Science Foundation (NSF), USA

Executive Summary

The broad landscape of new technologies currently being explored makes the current times very exciting for computer systems research. The community is actively researching an extensive set of topics, ranging from the small (e.g., energy-independent embedded devices) to the large (e.g., brain-scale deep learning), simultaneously addressing technology discontinuities (End of Moore's Law and Energy Wall), new challenges in security and privacy, and the rise of artificial intelligence (AI).

While industry is applying some of these technologies, its efforts are necessarily focused on only a few areas, and on relatively short-term horizons. This offers academic researchers the opportunity to attack the problems with a broader and longer-term view. Further, in recent times, the computer systems community has started to pay increasing attention to non-performance measures, such as security, complexity, and power. To make progress in this multi-objective world, the composition of research teams needs to change. Teams have to become inter-disciplinary, enabling the flow of ideas across computing fields.[1]

While many research directions are interesting, this report outlines a few high-priority areas where inter-disciplinary research is likely to have a high payoff:

a) Developing the components for a usable planet-scale Internet of Things (IoT), with provably energy-efficient devices.[2] This report envisions a highly-available, geographically distributed, heterogeneous large-scale IoT system with the same efficiency, maintainability, and usability as today's data centers. This planet-scale IoT will be populated by many computationally-sophisticated IoT devices that are ultra-low power and operate energy-independently.

b) Rethinking the hardware-software security contract in the age of AI. In light of the recent security vulnerabilities, this report argues for building hardware abstractions that communicate security guarantees, and for allowing software to communicate its security and privacy requirements to the hardware. Further, security and privacy mechanisms should be integrated into the disruptive emerging technologies that support AI.

c) Making AI a truly dependable technology that is usable by all the citizens in all settings. As AI frameworks automate an increasing number of critical operations, this report argues for end-to-end dependable AI, where both the hardware and the software are understood and verified. Further, AI needs to turn from a centralized tool into a capability easily usable by all the citizens in all settings to meet an ever expanding range of needs.

d) Developing solutions to tackle extreme complexity, possibly based on formal methods. This report argues for the need to tame the explosion of system complexity and heterogeneity by creating new abstractions and complexity-management solutions. Such solutions need to be accessible to domain experts. An important step towards this goal is to scale out and extend formal methods for the real world.

This report also describes other, related research challenges.

[1] In this report, inter-disciplinary refers to cutting across multiple areas of computer and information science and engineering, rather than across engineering disciplines.

[2] This report uses the IEEE definition of IoT, which is found in https://iot.ieee.org/images/files/pdf/IEEE_IoT_Towards_Definition_Internet_of_Things_Revision1_27MAY15.pdf.

1 Introduction

It is now an exciting time in computer systems research. New technologies such as machine learning and the Internet of Things (IoT) are rapidly enabling new capabilities that were only dreamed of a few years ago. The research community is actively exploring an extensive range of topics, ranging from the small (e.g., energy-independent embedded devices) to the large (e.g., brain-scale deep learning). At the same time, technology discontinuities such as the end of Moore's Law and the inescapable Energy Wall combine with new challenges in security and privacy, and the rise of Artificial Intelligence (AI).

In this new environment, computer system researchers have a great opportunity to make lasting contributions. While it is true that, in some of these areas, IT companies are heavily invested, their focus is necessarily narrow and has relatively short-term horizons. Academic researchers, one the other hand, can afford to take a broader, longer-term view and address the hard problems.

Further, computer systems researchers have already moved away from a mindset focused exclusively on performance. They have realized that other measures, such as security, complexity, and power are as consequential as performance. All of these other measures are indeed crucial at the current technology juncture.

However, to make progress in this multi-objective world, computing systems research teams need to be inter-disciplinary. It is necessary to enable close collaboration across computing fields, allowing inter-disciplinary computer systems research to flourish. Research advances will mainly come from collaborations of researchers with various expertises, such as distributed systems, computer architecture, programming systems, security, circuits, and languages.

Against this backdrop, an NSF-sponsored community visioning workshop convened about 150 researchers of multiple computer systems areas (Section 6). The goal was to outline a few high-priority areas where inter-disciplinary research is likely to have a high payoff in the next 10 years. This report summarizes the workshop's findings. While many research directions were found interesting, the workshop outlined several high-priority inter-disciplinary research areas. They are shown in Table 1. The rest of the report describes each area in detail.

Challenge	**Research Vision**
Develop the components for a usable planet-scale IoT, with provably energy-efficient devices	A highly-available, geographically distributed, heterogeneous large-scale IoT system that has the same efficiency, maintainability, and usability as today's data centers. This planet-scale IoT includes many computationally-sophisticated IoT devices that are ultra-low power and operate energy-independently.
Develop an effective hardware-software security contract in the age of AI	Hardware abstractions that communicate security guarantees to the software, and software systems that communicate their security and privacy requirements to the hardware. Security and privacy mechanisms are integrated in the technologies that support AI.
Make AI a truly dependable technology that is usable by all the citizens in all settings	End-to-end dependable AI, where both the hardware and the software are understood and verified. AI turns from a centralized tool into a capability easily usable by all the citizens in all settings to meet an ever expanding range of needs.
Develop solutions to tackle extreme complexity, possibly based on formal methods	Tame the explosion of system complexity and heterogeneity by creating new abstractions and complexity-management solutions. Such solutions are accessible to domain experts, and are based on scaling out and extending formal methods for the real world.

Table 1: Inter-disciplinary research areas identified.

2 Internet of Things and Infrastructure

IoT devices such as vehicles and smart phones are everywhere, and becoming ever more complex. These and many other computing devices interact with increasingly sophisticated networking, embedded sensing, and actuation. The resulting evolving environment lends itself to an exciting vision where the notion of what computing systems are is changing. Specifically, built environments such as commercial buildings, transport networks, or energy and water networks are becoming the new computing systems. Eventually, a new computer system will emerge: a planet-scale, geographically distributed, heterogeneous large-scale IoT system, which has the same efficiency, maintainability, and usability as one of today's data centers. In this environment, energy efficient operation is a primary concern. This vision calls for innovations in a variety of systems technologies and ultra-low power devices.

2.1 Useable Planet-Scale Internet of Things

Enabling the full capabilities of IoT is an exciting challenge for interdisciplinary computer systems research. Individually, IoT devices have limited functionality, are energy constrained and computationally limited, and often have intermittent network connectivity. Together, however, IoT devices can build the new computer systems of the future. Helping to create such computer systems is one of the challenges.

Challenge: *Usable planet-scale IoT: develop a highly-available, geographically distributed, heterogeneous large-scale IoT-based system that has the same efficiency, maintainability, and usability as today's data centers.*

For this vision to happen, the research community needs to advance in multiple areas, learning from and leveraging past efforts such as the GRID and the World Wide Web. In addition, it should collaborate with industry and emphasize the development of standards.

Scalable models and abstractions of physical systems and computer systems. To construct a planet-scale IoT, we require understanding how its many components operate and interact with each other. For that, we need models of subsystems, such as a region's mobile infrastructure, clusters of sensors, or city-wide device infrastructure. To build such models, we can gain inspiration from work in cyber-physical systems.

Scalable distributed algorithms that can operate in disconnected or long-latency mode. We need to develop highly-scalable distributed algorithms that can operate in a three-level hierarchical system — static and mobile clients, fog computing infrastructure, and cloud computing infrastructure. The system will be of large scale and some nodes will have intermittent connectivity. The algorithms must be able to meet guarantees of safety and timeliness, either deterministically or stochastically, in this environment. Examples of algorithms are those that perform data replication across multiple sites, support queries that are partially satisfied at multiple devices, and assist for human cognition that is available partially at multiple devices.

There is much prior work that we can build upon in scalable distributed algorithms [51, 7], fault-tolerant distributed algorithms [38, 97], network protocols with intermittent connectivity [96, 85], and delay tolerant networking [44, 45].

Programming models, communication protocols, and runtimes for heterogeneous collections of systems. The IoT is composed of highly heterogeneous systems. Such heterogeneity occurs across levels (clients, fog, and cloud), and within each level (e.g., one edge device can be a laptop, another a cellular base station, and a third the cash register of a coffee shop). We need programming models, communication protocols, and runtimes that we can use to program and manage such

heterogeneous collections of systems.

Available and reliable safety-critical IoT systems using unreliable devices. IoT devices such as vehicles, virtual reality headsets, and smart digital assistants are becoming ever more complex. While IoT systems have seen significant innovations in recent years, critical issues remain in system availability and reliability. There are substantial new challenges. First, the mission-critical functions of many emerging IoT systems require unparalleled reliability and fault tolerance guarantees. One example is a connected autonomous (or even semi-autonomous) car. If the system fails to recover from failures, the consequences are potentially disastrous. Second, fault tolerance and recovery methods used in servers and traditional mission-critical systems cannot easily be adopted to new scenarios with vastly different application properties. Finally, IoT systems installed or embedded into industrial control systems or smart city networks will have much longer lifespans than the consumer electronics devices that industry is used to producing. Therefore, one critical goal for IoT infrastructure research is to enhance the reliability of IoT systems with reliable and cost-efficient data management, storage, and system recovery.

Live-updatable IoT systems. IoT systems will need to evolve with time due to various factors, such as changes in version of the application, updates to some parameters of the application (in turn brought on by changes in the requirements from the application), or changes in the physical environment in which the IoT devices are embedded. Hence, it will be important to perform live updates to the IoT system. This means that the devices must not be brought offline to upgrade them; rather, the upgrade should happen in a graceful manner while the devices are still executing the application. This will become a requirement for highly-available IoT systems, where downtime could lead to danger to human wellbeing.

There has been significant work on live-updatable systems, from the server and enterprise class systems [22, 29] to low-end wireless sensor nodes [73, 28]. Such works have developed protocols to update a distributed set of nodes, performed over wired or wireless networks, for different capabilities on the nodes (e.g., ability to dynamically link modules), and optimizing different metrics (e.g., number of wireless packets or length of time to update).

An IoT domain introduces some unique challenges to this problem. First, the update process needs to attain near perfect reliability. This is important because, even if only a few blocks of a program are received incorrectly, the operation of the node may be impaired. Second, there needs to be higher-level guarantees that mixed-mode operation (i.e., old and new versions of the software co-existing on different nodes of the network) will not lead to failures or unpredictable behavior. Third, the update protocol must be able to handle the heterogeneity of devices, which is expected in IoT systems. For example, one node may have the capability for broadcasting in an omnidirectional manner while another may be unidirectional.

New capabilities and services enabled by new types of device sensors. The popularity of smartphones has resulted in the development of sophisticated sensors, and of many applications that leverage these sensors. Examples of such sensors are accelerometers, proximity sensors, gyroscopes, and magnetometers. As these and other sensors are embedded into more and more IoT infrastructure, novel and unexpected capabilities may be provided by IoT systems. Research into applications that leverage new sensors may provide good payoffs.

Modeling reliability at multiple scales, such as device, communication network, and system. The reliability of an IoT system depends on the reliability of the devices plus the network that connects these devices. It is expected that, for most production deployments of any importance, there will be some redundancy, such that failures in a subset of devices will still allow the application to meet its requirements. To understand the failure modes, and to have some predictive ability about what conditions any element may fail in, it will be important to develop models for the reliability of the elements and then models that compose individual models.

Reliability modeling has to happen at different levels. Such levels have to include those of low-level sensors; microcontroller, memory, and storage media; low-level firmware; resource-constrained software executing on the devices; networking medium; and networking software. There is abundant literature in the reliability community on modeling at most of these levels. Some notable works focus on flash-based storage reliability [11], embedded software reliability [52], and Wi-Fi network reliability for IoT [89].

Further, reliability models have to include the effect of the environment. Environmental effects are key because many IoT devices are embedded in the environment, and thus environmental characteristics like moisture, chemical composition of soil, or corrosion, affect the device reliability.

2.2 Provably Energy-Efficient Devices

Improving the energy efficiency of our computing infrastructure is at the center of computer systems research. Hence, a second challenge relates to developing ultra-low power IoT devices and infrastructure. We refer to these devices as "provably energy-efficient" to denote that they should be designed from the ground-up for energy efficiency, using a reasoned methodology rather than ad-hoc procedures.

Challenge: *Provably energy-efficient devices: Ultra-low power and energy-independent operation of computationally-sophisticated IoT devices.*

For this vision to happen, we need aggressive research advances in multiple areas.

Robust capabilities under intermittently-available energy. We envision that an increasingly larger fraction of the devices that are embedded in deployed IoT infrastructure will have to collect from the environment the energy that they need to sense, compute, and communicate. This is because wired power is too expensive to deploy in many circumstances, and impossible to deploy in others. Batteries degrade with use, introduce maintenance issues that are not well-suited to many applications, and can be hazardous.

Environmentally-available energy sources are often very weak (e.g., radio waves, small solar panels, or vibration), and sufficient energy to operate usefully may not always be available. Systems can no longer assume that energy is a continuously-available resource but, at the same time, systems must continue to robustly support the sophisticated capabilities that applications demand. Robust, capable operation with intermittently-available energy is in itself a grand challenge because it requires rethinking systems to include efficient mechanisms to collect and store energy, and to dynamically manage and control its consumption. Intermittent operation requires new programming languages, runtime systems, architectures, and circuits that are robust to interruption, that dynamically adapt energy and power availability with time, and that provide fine-grained control of the power system [18, 62].

Formal reasoning about energy use. In an IoT system, energy is a first-class measure that is as important as or more than performance. Unfortunately, while the analysis of the performance bottlenecks in a complex system such as an IoT system is relatively easy to perform, the same is not true for an analysis of the energy consumption bottlenecks. We require the development of tools, methodologies, models, and sensor hardware to identify the most energy-consuming parts of a cluster of IoT devices, and reason about how the energy is spent in such a system.

Tradeoffs among energy efficiency, correctness, programmability, and precision. In physical systems, there is often a trade-off between the energy consumed in an operation and other characteristics of the operation. Examples of such characteristics are correctness, reliability, programmability, and precision of approximate solutions. Since energy consumption is so important in IoT systems, it is important to model and understand how energy can be traded off with these other

characteristics.

Use of control-theoretic approaches to improve the efficiency of IoT systems. IoT systems maximize their efficiency through the use of hardware or software controllers. In the general case, such controllers monitor several measures, such as power and performance, and actuate on multiple measures, such as frequency and thread scheduling. Typically, controllers use algorithms based on heuristics, produced by profiling representative applications. Unfortunately this ad-hoc approach is not robust [91]. The resulting algorithms tend to be bug-prone and hard to change. Further, as the algorithm executes, there is the danger that unexpected corner cases cause large deviations from the targets, resulting in inefficient executions.

The alternative is to design software or hardware controllers using control-theoretic approaches [63, 65, 78]. With formal approaches, designers have a systematic way to quantify the knowledge and expectations they have about the design [61, 94]. For example, they know how important each of the output targets is relative to each other. They also know the overhead of actuating on different inputs. Hence, they provide this information, and then, proven methodologies generate a controller that actuates on the most convenient and fastest inputs to safely converge to the desired output targets. This approach ensures the effective control of interdependent measures and, as a result, attains provably-optimal resource efficiency.

Rethinking the storage stack to fully leverage Non-Volatile Memory (NVM). Data-intensive IoT devices generate high pressure on storage systems. Existing flash-based storage has a high write cost and induces significant software complexity in the runtime and file systems. Emerging NVM technologies such as PCM [58, 10], ReRAM [86], and STT-MRAM [4] promise both to increase density and to lower write cost. At the same time, their byte addressability means that they can be accessed with ordinary load and store instructions, and raises the intriguing prospect of retaining pointer-rich in-memory data across application invocations and even system crashes.

A key challenge to this prospect is the need to maintain memory consistency below the level of the caches [75, 43]. There are proposals to do this in hardware [46, 56, 68], in specialized data structures [72, 15, 70], or in more general transactional [92, 17, 36, 79] or lock-based [13, 42, 41] systems.

We envision re-architecting the storage stack for IoT devices with a system-wide and cross-layer approach, taking advantage of NVM in new systems software [19, 76, 93, 26]. With carefully designed APIs [82] and hardware-software co-design, one should be able, simultaneously, to increase energy efficiency, density, and reliability; reduce application complexity and production costs; and facilitate the use of energy harvesting.

For IoT devices that work in conjunction with the cloud, much current software complexity is devoted to the management of logically shared information. Infrastructure for long-lived in-memory data may allow much of the complexity of consistency, locality management, and partitioning and reconciliation to be moved out of the application and into standard system software [88, 14].

2.3 Built-Environments as Computing Platforms

With the proliferation of computing devices, networking, embedded sensing, and actuation, societal lifelines like transportation and energy and water networks have emerged as examples of cyber-physical systems (CPS). The emerging CPS vision presents an exciting universe of societal infrastructures that are efficient, available and increasingly responsive to changing user needs and environmental conditions. These systems are also changing the notion of what underlying computing systems are. Consider, for example, a commercial building. The Heating, Ventilation and Air Conditioning (HVAC) systems are managed by the operators using a building management

system provided by commercial vendors such as Johnson Controls. More recently, buildings have seen the integration of sensors for wireless finger printing of the environment, occupancy detection, human comfort feedback, and air flow control.

Increasing use of computing devices such as desktops and servers also makes some buildings "mixed use", hosting both human and IT equipment along with their requirements for energy use. These are increasingly interdependent systems where optimization opportunities have exploited human activities such as in the SleepServer system [2]. As these attempts at large scale energy use optimizations mature, a compelling vision of the building itself as a new computing platform has emerged. Such a platform has resources, such as various sensors for environmental conditions and human feedback. It has actuators, such as variable air valves, heaters, and door locks, many of which are IOT devices themselves. Reading such devices and commanding them is typically done by control algorithms.

The entire distributed system of a commercial building can then be treated as a computing platform. It has the actions of reading from sensors (or memory locations) and performing actions as processes on a distributed computing system. Such a platform can have its own "applications", and these applications may port across buildings as programs do on computing devices.

The current attempts to standardize the abstraction of the resources for applications in buildings [6] and IoT scenarios [1] provide common representations of the physical and virtual resources in the domain. Applications can run on top of those representations instead of adapting themselves per target system. There are many technical challenges to realizing this vision.

Challenge: *Built environments as computing platforms: Large cyber-physical systems of systems as computing platforms (buildings, critical infrastructures, etc).*

For this exciting vision to become reality, many fundamental questions need to be answered. Further, models and methods need to be created that cross the boundary between computing and physical systems. For instance, how does one discover a device or its capabilities? How does one address them in a manner that they can be built into a programming environment? Is it done through structured queries or through special brokerage services that maintain a record of devices and capabilities? How does one program such a sensorized distributed system? Is it done via a collection of programs that follow a specific programming pattern or through macroprogramming facilities? What is a runtime system for such a platform? The list of questions is long because it reopens a number of foundational concepts used to build conventional computing systems.

To address these questions, we need interdisciplinary research in several areas. First, we need to expand system architecture to include sensors and actuators. Sensing and actuating are a central part of the operation in built environments. Second, we need new virtualization methods for sensors and actuators, since such devices will be used concurrently by different processes. Third, building on the previous point, it is required that built environments support multi-tenancy. This means that all of the components in such systems must support multiprogramming with different administrative domains. Fourth, researchers should strive to support application portability across these new computing platforms. Ideally, interfaces and functionally should be similar across groups of built environments, to make portability easy. Finally, we need to work toward defining what a runtime system for these platforms is, and what is its basic functionality.

All of these issues broadly intersect with the research agenda of the CPS initiative [69]. For this reason, we do not elaborate further.

3 Security and Privacy

Headlines about cyber-security have become increasingly commonplace. It has been shown that attackers can steal user social security or credit card numbers from financial companies' computer systems, grab passwords from social media sites, and loot company secrets from the cloud. Further, new side channels are periodically being discovered in various layers of computer systems. In addition, privacy has become a crucial issue as online services powered by diverse distributed systems collect and aggregate data for billions of users. The black-box aspect of current online systems introduces the challenge of understanding the provenance of data when a breach occurs. Overall, the security and privacy areas present some of the most significant research challenges for computer systems researchers today.

3.1 Rethinking the Hardware-Software Security Contract

Hardware is the new front line of the cyber-security battle. In recent years, the threat surface has widened and deepened with the trends of IoT, mobility, and the cloud. As such, it is impossible to truly secure today's computer systems without beginning from the hardware level. If hardware or firmware is compromised, the threat can evade detection and escape from most software security tools.

The Spectre [55] and Meltdown [60] attacks have demonstrated the gap between the hardware behavior and the software assumptions on the hardware behavior. The abstraction between hardware and software is specified in the instruction set architecture (ISA), which describes the behavior of each instruction as it commits. However, the hardware performs speculative execution of instructions which, even after being squashed, leave a certain footprint in the system. Hence, the traditional abstraction between hardware and software leaks [35]. This points to a major research challenge.

Challenge: *Rethinking the hardware-software security contract.*

To address this challenge, we need research advances in multiple areas.

Building hardware abstractions that communicate security guarantees. Closing the semantic gap requires building hardware abstractions that communicate the security guarantees provided by the hardware. In particular, they must specify which side channel leakage is closed and which is left open. With such primitives, software can reason about what it needs to do to provide the desired security guarantees [33].

Allowing software to communicate its security and privacy requirements to the hardware. Today, there is no mechanism available for the software to communicate its security requirements to the hardware. There is no interface that allows software to specify whether it can or cannot tolerate various types of side channel leakage. For example, some cryptographic primitives require running in constant time in order to avoid timing side channel leakage. This requirement cannot be communicated to hardware in the absence of a software-hardware security interface. Research is needed to design an interface that allows software to specify its security requirements to hardware.

Designing a Trusted Execution Environment (TEE) that specifies and eliminates side channel leakage. Currently, Intel SGX [20, 21], AMD SEV [3], and ARM TrustZone [5] provide TEEs that do not address side channel leakage. Given the reach of the recently discovered micro-architecture side channel vulnerabilities, the coverage of a TEE should be expanded to include side channel leakage. Research is needed on new mechanisms to close various side channel leakage (timing, power, branch, cache, access pattern, fault, etc.) in a way that introduces as little performance and

power overheads, and additional cost, as possible.

Mitigating side channel leakage in accelerator-based systems, including GPU, FPGA, and ASIC accelerators. Datacenters in the cloud will increasingly rely on non-CPU hardware such as GPUs, FPGAs and ASIC accelerators. Research is needed to provide TEEs and eliminate side channel leakage in the presence of such disparate hardware. It is also important to identify the best layer (e.g., hardware or software) to address a particular side channel vulnerability.

Developing security metrics, methodologies, tools and open source platforms to reason about the trade-offs between security, cost, and overheads. Research is needed to develop security metrics to measure the effectiveness of various security mitigation techniques. Methodologies and tools need to be developed to support measurements, metrics collection, risk assessment, and quantification of the trade-offs between security, cost, and various overheads such as performance, power, and area. In addition, we need to design new hardware and software systems to allow security mitigation to be deployed cheaply when a new security vulnerability is discovered post-production. More research and development efforts are needed to continue and strengthen recent initiatives on open source secure hardware (e.g., Keystone [53]).

Designing new hardware and software mechanisms for introspection of security modules. Research is needed to develop means for introspection of security mechanisms. Today, the parts of the system that are used to enforce security are typically black boxes (e.g., proprietary code running inside the system management engine on Intel processors). In the event of a security breach, there is no way to examine the state of these protection mechanisms. Attackers may find a way to break the protections, but ordinary users have no way to defend themselves. New mechanisms are needed to allow users (applications, operating system, or even other hardware modules) to observe the health of the security-critical modules, and then react in case there is any problem. Software could, for example, start to erase its data if the system management engine is broken and the computer is compromised.

3.2 Security and Privacy in the Age of AI

Two common concerns in many of the emerging AI applications such as self-driving cars, Unmanned Aerial Vehicles (AUVs), precision medicine [59], and assistive technologies for disabled people, are that they all deal with sensitive user data and directly impact the safety of human lives. Hence, the data must be kept private. Moreover, cars, drones, and implanted devices should not be altered by malicious agents. Hence, these computing devices must also be kept secure.

In addition, in many AI applications, researchers often have to ship code or use models that they do not fully understand. Code and models are shipped with relatively little understanding of how or why they work. This makes it hard to reason about attacks and information leaks, and also how the model can be deceived. Overall, for all these reasons, researchers must create strong security and privacy policies, mechanisms, and algorithms for emerging AI applications.

Challenge: *Security and privacy in the age of AI: Protecting unexplained systems.*

To address this challenge, we need research advances in multiple areas.

Integrating security and privacy mechanisms into the disruptive emerging technologies that support AI algorithms and applications. AI algorithms and applications use emerging technologies such as NVM, near-data processing, and accelerators. NVM products such as Intel and Micron's 3D-XPoint memory [67] are being shipping. Such memory devices introduce new vulnerabilities. For example, a malicious agent can simply pull out a non-volatile DIMM, walk away, and decipher the contents of the DIMM. This DIMM will contain not only persistent files, but also intermediate objects created by the application.

Given the high cost of data movement in modern systems, there is a strong push towards near-data processing paradigms. In traditional systems, the processor remains the hub of all activity and represents a localized trusted execution environment. But in a system with near-data processing, the processing power is scattered across the entire system, with parts of the computation potentially being performed on a buffer-on-board, a DIMM buffer chip, a memory chip itself, or an SSD module. The attack surface thus becomes broader.

Finally, accelerators are growing in prominence. GPUs are already commonplace, and ML accelerators like Google's TPU [47] are available to consumers. Trusted Execution Environments have only recently been introduced into commercial server-class processors. For accelerators, security and privacy have barely been considered. Therefore, much work remains in defining side channels, attack vectors, and defense strategies for these accelerators.

Developing security and privacy abstractions, specifications, and interfaces for each technology. To formally verify and reason about the security properties of any system, it is necessary to have an abstraction of the system, plus the system's specifications and interfaces. As industry and researchers are developing new disruptive technologies such as those listed above, it is important to derive detailed specifications and abstractions of the resulting components.

Anticipating and mitigating attacks before they happen. It is important to expose and eliminate underlying vulnerabilities ahead of attackers. In the age of AI, with many disruptive new technologies, research is needed in discovering new types of attacks and methods to mitigate them. Findings from such vulnerability research will serve to better protect users and systems. In addition to discovering security holes, researchers should also design effective test practices, methodologies and frameworks for emerging AI applications.

3.3 System Support for End-to-End Privacy

As more data is processed on systems owned by parties other than the data owners, and moved across interconnected systems, it becomes harder to reason about the privacy guarantees of the system. Users are most often given only one option when interacting with a remote system: either they fully trust it or fully distrust it. Systems lack application-independent capabilities for owners of the data to define, control, and audit the computing done on their data, and as well as the lifetime of their data.

These trends have led to the introduction of privacy-oriented design principles such as Cavoukian's Privacy by Design [12], and privacy-oriented regulation such as the European Union's General Data Protection Regulation (GDPR) [30]. While well intentioned, these privacy-oriented design frameworks are insufficient. We must develop systems that provide end-to-end privacy via hardware and system software capabilities to track user data and enforce user-specific policies on its use, independent of the application software.

The challenge we propose is to enhance the systems powering distributed applications that process user data. We need to enhance such systems with privacy mechanisms, techniques, and building blocks that respect user privacy. The goal is to build systems that guarantee a privacy regime that is centered around the user as owner of data, is socially responsible, and is deployed end-to-end, independently of system scale.

Challenge: *System support for end-to-end privacy.*

To address this challenge, we need the following research.

Accelerating advanced privacy-preserving techniques, including computing over encrypted data and differential privacy. There have been significant advances in cryptography and statistical analysis in support of privacy. Fully homomorphic encryption, secure multiparty compu-

tation, and differential privacy have shown the feasibility of end-to-end privacy at the algorithmic level. There is a large performance gap to be bridged for such methods to be practical, applicable, and adopted widely. The systems community must research hardware acceleration and energy-efficient solutions for privacy-preserving techniques, in order to scale them from IoT devices to cloud applications.

Developing composable mechanisms for tracking data provenance and information flow within a system and across systems. Today's complex applications often combine functionality provided by services from various vendors, resulting in highly interconnected, interdependent, distributed systems in which data privacy is hopelessly opaque end-to-end. Transparency requires that each component of a large system can be audited for privacy guarantees, and that the data flows can be similarly audited for compliance with privacy policies. We need novel system mechanisms to audit the privacy guarantees of software running on trustworthy execution environments. We also need novel system mechanisms to efficiently construct data provenance and information flow tracking histories. These mechanisms will form the basis of any privacy auditing and verification function.

Developing efficient and auditable system-wide mechanisms for users to control their data. To afford users control over their data, it is important for systems to provide the primitives that allow data management irrespective of the application processing such data. Functionality such as data masking, deletion, expiration, and access control must be efficient, auditable, and available system-wide. This also implies the need for hardware and system-software mechanisms that can enforce user-defined policies building on this data-management functionality.

Designing domain-specific privacy models to inform system designers and to serve as privacy benchmarks. While privacy policies vary significantly from domain to domain, a common set of privacy models is needed for system designers to validate their systems against. At a minimum, a privacy-preserving social network and a privacy-preserving health information system are two examples of privacy models with opposing characteristics, yet of similar complexity. Social networks are built around the idea of data sharing as default operation, where privacy controls should be applied only when sharing should be restricted in some way. Health information systems default to isolation between patient records, with infrequent sharing when really required. Both systems have a variety of parties (data owners, data users, etc.) that wish to obtain access to the data. Precise models of these and other systems will form the basis for evaluating any system design that supports privacy. In addition to capturing types of sensitive data and types of users, such models must be developed to take into account cost models and payment mechanisms to discourage the abuse of private data.

3.4 Tools, Testbeds, and an Ecosystem for Security and Privacy Research

Finally, there is a major challenge that pervades all security and privacy research.

Challenge: *Develop tools, testbeds, and a complete ecosystem for security and privacy research.*

Some of the research needed to address this challenge is as follows.

Developing metrics for security and privacy. One of the challenges in security research is how to quantify the security and the privacy of a system. What does it mean to say that a system is 95% secure? What does it mean that the privacy level of a system is 85%? Such metrics would play an essential role in the quantification of a system's performance under attack. Being able to define such metrics, even if only to a certain extent, would greatly benefit system researchers, and would provide a fair ground for cross-comparison and validation of various schemes.

Developing a methodology for specifying threats and quantifying the strength of a mitigation.

Threat analysis is key in security research. It specifies in a proactive manner how the attackers could exploit a system's weaknesses. It can also be used to guide how to design counter measures to mitigate such threats. However, such analysis is commonly done in an ad-hoc fashion. We need a systematic methodology to specify the threat models under different scenarios, as well as to quantify the effectiveness or strength of mitigation schemes.

Developing security and privacy tools, open-source testbeds, and datasets. Currently, the research community is missing well recognized, widely adopted toolsets to conduct security and privacy research. It also lacks open-source testbeds that can be accessed by the wider research community. Developing such tools and testbeds is a necessity, as they would be great enablers for advanced research in this area. Research on hardware vulnerabilities is often hampered by a lack of visibility into underlying implementations. Security researchers have therefore engaged in efforts that involve reverse-engineering and trial-and-error. Many years of such research have led to the Meltdown and Spectre attacks that have uncovered decades-old vulnerabilities in processor implementations. We must, therefore, invest in more open-source hardware platform infrastructures (e.g., RISC-V), that can help researchers discover and mitigate vulnerabilities before attackers find them. Research on privacy and security is also hampered by limited access to data. While companies have access to substantial user information, geophysical data, and health records, academia has limited access to this data, or receives severely scaled down versions of it. While such sensitive data is understandably protected, there must be a greater investment in approaches that can create versions of datasets that preserve anonymity and yet retain the scale and features of the original dataset.

4 Artificial Intelligence and Augmenting Human Abilities

Recent years have witnessed the rapid progress of Artificial Intelligence (AI) and Machine Learning (ML). Deep neural network models with many layers have demonstrated state-of-the-art accuracy in speech recognition, computer vision, robotics, information retrieval, and natural language processing. The demanding computational characteristics of deep learning has led to the emergence of many open-source and in-house frameworks including Torch, TensorFlow, Caffe, Theano, CNTK, MxNet, and TVM, and various AI accelerators. Further development is needed to make AI a truly dependable technique that can meet real-world needs in various settings. As an important pillar of modern AI, computer system support plays an essential role in ensuring the efficiency, scalability, power efficiency, responsiveness, security, productivity, and ultimately, dependability of modern AI. Innovations are needed at all levels of computer systems.

4.1 Dependable AI

There are many implementations of AI frameworks, using a variety of hardware and software support. Many such implementations make critical decisions. Hence, one of the research challenges is to ensure their dependability.

Challenge: *Dependable AI: Making AI a truly dependable technology, both in isolation and when integrated with other (non-AI) systems.*

The effort of making AI dependable is motivated by the increasing adoption of AI frameworks to automate operations in various strategic fields, such as in automotive, aerospace, and military applications. For instance, neural networks have been deployed in self-driving cars to dynamically identify and classify objects, and in unmanned aerial vehicles (UAVs) for homeland security missions. NASA's Jet Propulsion Laboratory (JPL) is working on taking advantage of AI to au-

tomate deep-space exploration. GPU-accelerated MRI analysis has been used to avoid invasive procedures such as biopsies.

Dependable AI relates with traditional software and hardware reliability, but differs in its emphasis on the end results of the AI models. An AI product on reliable software and hardware is not necessarily dependable for a particular AI task; the results from the model may still lead to catastrophic consequences in some occasions (e.g., fatal car accidents). The dependability hence requires considerations of all layers, from requirements in the application domain to AI algorithms or models, training datasets, and reliability of the software and hardware.

The complexity, heterogeneity, and rapid evolution of AI software makes it challenging to assess and ensure AI dependability. There is not a full understanding yet of how AI algorithms reach a solution, which implies a lack of understanding of how a fault could impact reaching the solution. Additionally, the need for fast (or efficient) AI has led to a proliferation of non-traditional, AI-specific hardware architectures, such as GPUs, FPGAs, Heterogeneous System on Chips (SoCs), Tensor Processing Units (TPUs), and ASICs.

AI dependability in these novel architectures suffers for several reasons. First, these platforms have high transistor counts, which match or exceed CPU transistor counts — e.g., NVIDIAs GV100 GPU includes 21.1 billion transistors, whereas an 18-core server CPU includes 5–7 billion transistors [32]. Second, these platforms have immature verification and validation processes [71], compared to the over 40 years of verification experience in CPUs. Finally, such platforms have traditionally focused more on performance than on correctness and, therefore, have little or no fault tolerance. To address this challenge, research in several directions is needed.

End-to-end dependable AI. We need a holistic, integrated approach, where both the hardware and the software are verified, first in isolation and then end-to-end. It is important to understand not only the dependability of the AI algorithms but also how a fault, or unexpected behavior, in any of the AI hardware and software layers, impacts the system they are integrated with. There are several sources of transient and permanent errors that can undermine AI reliability, including environmental perturbations, software errors, variations, and radiation-induced soft errors.

Better programming languages and models. Theorem provers and proof assistants such as Coq and Isabelle/HOL have been instrumental in creating formal specifications. They have helped produce certified compilers, translators, hypervisors, and OS kernels, and generally formally-specified and correct-by-construction software layers and software tools. We now need advances in theorem provers and specifiers that allow formal reasoning on AI data and AI algorithms. Recent years have seen some incipient efforts (e.g., Deep Neural Network (DNN) robustness [34] and formal specification and correct implementation for algorithms on stochastic computation graphs [81].) More research is needed in this direction.

Handling software faults. Incorrect results in AI software could be due to incorrect training sets (i.e., bad data), programming errors (i.e., bugs), and possibly other factors. A taxonomy of AI-specific faults is needed. Also, we need programming models that allow faults to be detected and handled — similarly to how exceptions are handled in traditional programming. Finally, we need to understand how faults propagate upwards and downwards across software and hardware layers, so we can reason about such faults and possibly contain them.

Toward debuggable AI. Current AI framework implementations offer little support for tracing bad results to their causes. For example, it is hard to trace an incorrect outcome in ML to the training sample(s) or model parts (i.e., clusters, nodes, or network weights) responsible for it. Moreover, after training with bad data, it is hard to identify the model parts contaminated by bad or adversarial samples. In addition, AI errors can be introduced during learning or inference. Overall, to achieve debuggable AI, we need to be able to collect end-to-end traces during learning and inference. The tracing must strike a precision and overhead balance. It has to be detailed

enough so that executions can be analyzed or replayed when the output is incorrect, but low-overhead enough so that tracing can be used in production and avoid performance penalties.
Developing dependable new hardware architectures. As new architectures are invented to accelerate AI applications, new failure mechanisms will emerge. For example, memristor-based accelerators exhibit a relatively high defect rate [31]. It is important to develop models for how hardware failures occur, propagate, and influence training and inference in these new architectures. Conventional failure models are unlikely to apply. In addition, after developing the models, we need to implement the hardware supports to prevent critical errors from occurring and propagating.

4.2 Ubiquitous AI (Democratizing AI)

To fully materialize the potential of modern AI, it is necessary to make the AI technology ready for adoption by the ordinary citizens for all kinds of application domains in various settings.

Challenge: *Democratizing AI: Turning AI from a centralized tool into a capability easily usable by all citizens in all settings to meet an ever expanding range of needs.*

Meeting the goal of ubiquitous AI requires solving a multitude of problems, ranging from dealing with the need for multiple scales of AI, to making AI affordable for the masses, overcoming (labeled) data sparsity and bias, enabling holistic considerations of integrated AI systems, and establishing social and ethical standards for utilizing AI-powered tools.
Dealing with multiple scales for personalized AI. The broad range of AI applications causes diverse needs for AI, in terms of both models and deployment contexts. Some applications may need very large scale models running on supercomputers, while others need tiny models operating in real time on very small IoT devices. Effectively dealing with such a large variety is a major challenge for computing systems to help democratize AI and create personalized AI-based tools. It calls for heterogeneous systems to support arbitrary precision and mixed learning algorithms, and requires rapid hardware reconfigurability without hurting programmability. The solution must be able to adaptively meet the various constraints at various scales, and effectively capitalize on model compression and quantization. It needs to enable automatic model discovery and simplification based on constraints, supporting controllable precision through automatic scaling. A gap that needs to be narrowed is the lack of principled understanding of fundamental properties in trading off accuracy for performance (time/computational complexity) of AI models. This is a major hurdle for guided exploration and efficient model scaling and adaptation of AI.
Making AI universally affordable. A major barrier for democratizing AI is the cost of AI systems, in both AI model construction and model deployment. A deep learning model typically requires expensive massively-parallel systems to train; the hardware cost and long training time are prohibitive to many users. Meanwhile, energy consumption, space, and maintenance cost of the trained models form the major concerns in the usage of the trained models. Research is needed to reduce the various types of costs to make AI affordable and accessible to everyone, ranging from AI system designers to end-users, and in every possible setting. Opportunities may exist in the use of low-end, cheap IoT devices (potentially in a distributed fashion) to assist or replace high-end, costly devices such as GPUs and TPUs. Other opportunities may exist in the creation of system support for decentralized model training (e.g., Federated Learning on edge devices) and in the design of new versatile hardware that is easily reconfigurable. The latter may eliminate the need to design specialized hardware for every new model. These optimizations are inherently multi-objective, given the multi-faceted cost associated with AI.
Embedded machine learning at uW and mW, and tradeoffs with accuracy. We are used to neu-

ral network training and inference executions that consume tens or hundreds of watt. However, many IoT devices have much lower power envelopes. If we want to use neural network inference in such environments, we need to develop machine learning algorithms that consume uW- or mW-level power. Such goal requires redesigning the machine learning algorithms and the hardware. For example, it may involve running DNNs that use single-bit data and algorithms that aggressively exploit data sparsity. Using such optimizations may have an impact on DNN accuracy.

Data and ecosystem. Modern AI depends on data. The availability of a large volume of high-quality labeled data is crucial for many AI models. But often, such datasets are either unavailable or inaccessible to general users. Research in computer systems is needed to create better infrastructure (e.g., effective crowd-sourcing) to facilitate the acquisition of more (labeled) data with improved quality control. It is also important to foster an ecosystem to promote data sharing without compromising privacy, and to help data de-biasing. In addition, we need advances in generative adversarial networks [37], novel models (e.g., CapsNet [80]), and other emerging techniques to address the data demands of AI.

Holistic consideration and deployment. In practice, AI models are rarely used alone. They are often integrated into a system that consists of other computing or non-computing components (e.g., a cyber physical system). An intelligent virtual reality system, for instance, may encompass the domains of AI, imaging, vision, graphics, video processing, and acoustics, requiring components from all these domains to operate collectively and synergistically in a single system.

To ensure the best synergy requires new holistic ways to design and build computer systems. The design decisions ultimately depend on the requirements of the application and the underlying device constraints (e.g., motion sensors, optical lens, or head-mounted displays). This underscores the need to expand the systems research scope vertically, to tie it with the application scenarios and with the hardware implementation technologies.

There are multiple open questions. One is to find the right programming and OS abstractions to enable cross-domain design, while ensuring the compliance of application requirements. Another is to create tools and infrastructure support for whole-system profiling and debugging. Yet another is to develop a scientific methodology to quantify user experience in different application scenarios, and use it to evaluate AI systems.

4.3 Brain-Scale Deep Learning

One of the reasons why deep learning outperforms traditional ML methods is the superior expressiveness of a DNN, thanks to its large number of model parameters. In general, more complex learning tasks demand larger models to capture the complexities of the tasks. For deep learning to effectively deal with more complex AI problems in the future, the scale of the DNN models is expected to grow. However, a larger model usually requires more (labeled) data to train. The amount of computations, therefore, is expected to increase super-linearly with task complexity (illustrated in Figure 1). Based on this analysis, a major challenge is how to provide effective system support for efficiently training and deploying future DNNs of an extreme scale—including DNNs with a scale like a human brain, with trillions of parameters.

Challenge: *Brain-scale deep learning: Training and deploying ultra-scale DNNs with trillions of parameters.*

To address this challenge, we need aggressive research advances in multiple areas.

Reducing and streamlining data movements. Distributed systems are necessary for dealing with an ultra-scale DNN, not only because a single node may not have enough storage to hold all the

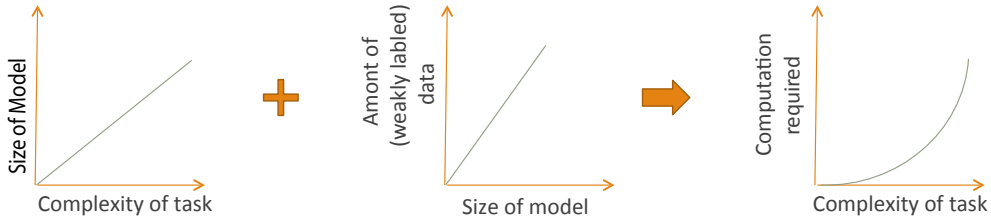

Figure 1: Superlinear relationship between the complexity of a learning task and the scale of the computation.

parameters of the DNN, but also because training a DNN of such a scale on a single node is time-wise impractical. Distributed DNN training, regardless of which distributed training algorithm is used, entails a large volume of cross-node data movement—including model parameters, error gradients, and activation maps. Efficient and timely communication is essential. Running a DNN of such a scale while meeting real-time latency constraints and minimizing total cost is especially challenging. Further, when an ensemble of DNNs need to be trained for finding the best DNN configurations, the challenge is orders of magnitude harder.

The problem calls for solutions from both system software and hardware designs. In software, promising progress has shown in model compression [40], streamlining communications [77], and direct data analytics on compressed data [95]. In hardware, in-memory and near-memory processing [16, 54] reduces data movement by moving processing elements closer to the data. Recent technological developments in monolithic 3D enable tight integration of high-density memories and high-performance processing elements. However, for in- or near-memory processing, data still needs to be moved around in the memory module. Designing many in- or near-memory processing modules that work together in a scalable fashion still requires more research. Leveraging processing capabilities throughout the system (e.g., in the network interface card) to accelerate AI applications is another promising direction.

The data movement challenge is rooted in the execution model of Von Neumann architectures. The brain does not have separate memory and processors, nor it uses instructions. New non-Von Neumann architectures are needed that mimic how the brain works, where synapses store information in a distributed manner, and information is mostly communicated among neighboring neurons. Emerging devices such as memristors exhibit neuron-like properties, which makes them promising candidates as fundamental building blocks for new architectures. The probabilistic nature of many AI workloads lends itself well to computing substrates with analog flavor and extreme parallelism. For example, there are recent results using magnetic RAM and processing in memory for accelerating inference, and progress in using quantum computers to train models. There are wide open opportunities to explore superconducting logic, and molecular processing and storage for AI.

TCO-oriented programming and run-time systems for ultra-scale AI. It is important that large-scale distributed DNN training systems be built and run cost-effectively. The ultimate goal of users of an AI system is to minimize the total cost of ownership (TCO), including both direct and indirect costs associated with the AI system. Current software systems for deep learning are oblivious to TCO, lacking the support for users to measure, model, or optimize the total cost when developing or deploying an ultra-scale AI system. An important research direction is to reconsider the design of AI/ML programming systems (programming languages, compilers, libraries, and runtimes) towards a TCO-oriented paradigm.

Automated model training scaling. An important capability of future deep learning systems is the ability to adapt the training algorithms and implementations based on the DNN scale, struc-

ture properties, data attributes, and underlying computing resources. Such a capability requires a deep understanding and quantitative modeling of the fundamental properties of scaling DNN training for performance, and the potential impact on the accuracy of training.

Co-design of training algorithm and computing systems. There is a rapid growth of hardware accelerators for AI algorithms. Popular ideas in hardware/software co-design are to design dedicated functional units for frequent operations (e.g., for activations and for linear algebra), support data type quantization, and provide custom data-types at the hardware level. These have had significant impact, but are just the tip of the iceberg. There are open possibilities in custom memory hierarchies, interconnection networks, and near-memory and storage processing. An exciting hardware/software co-design opportunity is to use ML to design better ML accelerators. For ultra-scale DNNs, it is essential to match the training algorithm with the underlying distributed system design, to streamline the communications and avoid bottlenecks.

Domain-specific compilers and whole-stack optimizations. Integrating high-level frameworks (e.g., Tensorflow, PyTorch, and MxNet) with compiler frameworks (e.g., XLA, TensorComprehensions, and TVM) enables joint high-level and low-level optimizations that would be difficult to implement otherwise. For example, fusing high-level operators (e.g., convolutions and activation functions) might enable better low-level code mapping to hardware and data locality optimizations. An important research aspect of such systems is extensibility to new ML techniques and hardware backends. Equally important is the development of domain-specific compilers that are capable of automatically generating high-quality code for any upcoming layer — targeting a wide variety of architectures/accelerators without relying on vendor-specific libraries. Unfortunately, existing frameworks heavily use library-specific implementations of layers, forgoing any benefit of cross-layer fusion that increases computational density and improves data locality. It is also necessary to use analytical models to accurately determine optimization parameters (e.g., tile sizes and unroll factors) specific to the underlying hardware.

System support for augmented and assisted AI. As Figure 1 illustrates, as the scale of deep learning models grows, the amount of labeled data needed to train a model increases quickly. Attaining a large amount of well-labeled training data is often difficult. Addressing this problem is essential for ultra-scale deep learning, due to the complexity of its models. We may need to augment deep learning with reinforcement learning or leverage generative methods (e.g., generative adversarial networks [37]) to reduce or eliminate the amount of training data needed for training the models. A direction worth pursuing is developing the system and hardware needed for effectively supporting such mixed learning models.

5 Complexity Management

Taming complexity has been a major challenge and success story throughout the history of computing systems. We identify three challenges specific to this area. Many of the underlying motivations and technological solutions are well known. However, existing approaches remain far from satisfactory, and the new demands on complexity management call for reanalyzing the problem.

5.1 Taking up the Challenges of Diversity, Scale, and Change

Heterogeneous platforms, cloud-scale systems, and cross-cutting and cross-layer concerns like security, performance, and reliability all add substantially to the complexity of designing systems.

Challenge: *Celebrating diversity, scale, and change: taming system complexity and heterogeneity, and broadening applications; looking for future-proof solutions in a dynamically evolving software and hardware environment, acknowledging that these trends are here to stay.*

Developing new mapping specifications for heterogeneous platforms at multiple scales. We need new specifications to map software components to hardware in today's heterogeneous platforms. Software abstractions go through numerous transformations and refinements, as they are being lowered to FPGA bitstreams, mapped onto transaction-oriented on-chip networks, compiled for unusual core and memory architectures, or mapped to manycore platforms [25]. These operations include a variety of trade-offs, such as those between parallelism and locality, and between re-computation and communication. They also impact data structures and algorithmic choices. Optimal choices for one level or scale may not be good or even relevant at another. Much of this need to revisit mapping specifications and tools is driven by the increases in parallelism and hardware complexity, and by both hardware and application specialization.

Creating new abstractions while avoiding leaks, inconsistencies, errors, and other hard to foresee problems. To tame complexity, we need to develop abstractions, preferably based on formal methods techniques, and using models of computing systems that are more future-proof than in the past. Indeed, abstractions have often created issues in the past. For example, abstractions of the functional behavior of processors can leak information through side-channels that may be exploited by attackers.

Another example is that abstractions are typically unable to isolate the upper layers from the presence of errors in the system. It remains a challenge to model the effect of hardware errors while formalizing an ISA or an OS interface — and such errors are typically ignored [64].

Finally, abstractions may themselves be erroneous, unclear or inconsistent. Formalization helps, and mechanized ones with proof assistants even more so, but specifications may still incorrectly model the system. The history of the C11/C++11 memory model provides some examples [90, 57, 49, 9].

Developing abstractions that work and scale in a changing environment. We need systematic methods and tools to bridge abstractions when interfaces change. We want to automate the process of porting software or architecture components from one abstraction of the underlying system to another. Such process should possibly involve the automatic inference of the system's properties, using automatic testing (such as fuzzing or the automatic generation of litmus tests), and relying on software synthesis. ML is emerging as a systematic tool in this area, and has also played an important role in the construction of compilation heuristics [50, 74, 24, 23]. When instantiating or modifying abstractions, one needs to preserve compatibility. The goal is to avoid introducing impedance mismatch or unnecessary overhead.

However, it should also be noted that automation can worsen complexity, making it harder to understand how one abstraction was converted into another, especially if something goes wrong or an unexpected result occurs. The benefits of automation must be carefully weighed against new sources of complexity it introduces.

New ways of thinking about abstractions that go beyond layers, are future-proof, and maximize reuse. The traditional layered system design has many advantages, such as allowing us to revisit concepts from one layer in another layer, or ripping apart one layer of abstraction while keeping a manageable system overall. Still, there is much to gain in broadening the interfaces. It may be helpful to facilitate the co-existence and composition of parameterized abstractions. Such approach may offer higher productivity without sacrificing efficiency. Examples of this approach include the design of type systems for dynamic languages [83], separation logic for low-level but safe programming languages [66, 48], and multi-stage programming [87, 27].

5.2 Complexity-effective Design and Implementation for Emerging Applications

One of the consequences of the diversification of applications and hardware designs is the need for more accessible complexity management — i.e., to make it easier to use complexity-effective design tools and formal methods.

Challenge: *Adoption and application of complexity management solutions. Make these solutions accessible to domain experts, automating much of the process, and facilitating reuse across domains.*

Heterogeneity and specialization can be observed at multiple levels and scales. For example, there are heterogeneous memory technologies, microprocessors, accelerators, and programming languages, all for general purpose computing. There is burgeoning support for specialized computing targeting AI and IoT with domain-specific languages and tools. In this environment, we would like to encourage research aiming to reduce the complexity faced by domain experts in need of advanced design. It should be possible for a domain expert to easily construct and validate specialized hardware and programming environment for a new application domain. Simultaneously, reuse across domains should be encouraged.

Facilitating the construction, dissemination and evolution of formally-defined abstractions. The value of formal methods and logic-based design, synthesis, and verification is widely recognized. Yet the accessibility of formal methods and logic-based programming is lagging. We need to make formal methods more accessible to college students. Also, we need to empower engineers and scientists from all areas with modern versions of these tools. Domain experts need our help to construct their own domain-specific frameworks and tools. One promising avenue for research is to automate the design and implementation of type systems and other logic-based automated reasoning about programs and systems.

Facilitating communication across communities, encouraging reuse. Different research communities typically use different abstractions, models, and approximations. The pervasiveness of heterogeneous and specialized systems requires that abstractions, models, and approximations become more portable and interoperable across communities. The scale of the systems and their rapid evolution require much more automation, reuse, and parameterization of abstractions designed by different communities. This goal is challenged by the lack of composition of abstractions, the diversity in non-functional properties and optimizations, and the difficulty to reconcile these with modular design principles. Typically, organically grown interfaces provide a more agile means for such reuse and communication. One good example is the success of the REST API for web services and storage, which permeates a rapidly growing range of services and platforms. We need more systematic means to construct and adapt such APIs.

5.3 Formal Methods for the Real World

Formal methods are one of the best tools we have to tame the growth of complexity. They also provide a level of design verification that is sorely needed, as discussed in all three earlier sections. Yet the adoption and suitability of formal methods remains sparse and insufficient. This motivates a coordinated approach across computer science and applied domains to make their use ubiquitous.

Challenge: *Scale out and extend formal methods for the real world.*

Logic-based synthesis everywhere. Verification and, to a lesser extent, testing have been at the core of the development of formal methods in the computing systems area. The most successful developments have probably taken place in the verification of complex circuits. Now, security and

safety are becoming first-class objectives in a growing number of domains. While this is an area that is ripe for formal methods, the practice of formal methods cannot be limited to the verification of systems. We need to use them to generate designs that are correct by construction.

One example of how we envision formal methods to be used is the so-called model-based design of embedded software. It involves the integration of a formal specification and a synthesizable model from a single source (sometimes called an executable specification) [39], enabling efficient code generation, verification, and testing. It is essential to scale and generalize such methods beyond safety-critical applications.

Algorithmic program synthesis is another approach. It uses declarative specifications — sketches (i.e., partial programs) — to simplify programming by delegating coding tasks to a constraint solver [84]. This approach has potential in the synthesis of formal models and type systems, the mining of information from programs, and possibly in correctness and security [8].

Unfortunately, the theory underpinning formal methods tends to lag behind practice. One major challenge is scalability: model checkers, SMT solvers, optimization research tools, theorem provers, and proof assistants need to scale more. In addition, formalization often appears to engineers as a scalability and expressiveness limitation, in the face of real world development and validation practices. This implies that the scalability and expressiveness of tools should be a focus for improvement, but also, that engineers need better training in the use of formal methods so that they will be seen as an essential ingredient in system design.

Taming abstractions with formal methods. Chapters 2, 3, and 4 discuss many properties, constraints, and optimization objectives that currently evade formal modeling. The thirst for formal reasoning in these areas should encourage a massive effort to apply formal methods across broad areas of computer science. One direction is to capitalize on the success of type systems in programming languages, and apply similar ideas to side channel defenses, security policies, and resource management for IoT.

One weakness of most abstractions is that they are not fully enforceable, in other words, there are few, if any, systematic or formal ways of ensuring an implementation adheres to all the rules of the abstraction. Making them robust to malicious attacks is a challenge in secure systems. Good areas to apply formal methods to are hardware isolation and capability management, as one can focus the formalization and verification requirements onto limited secure enclaves and specific privilege enforcement mechanisms.

Understanding the emergence and limitations of standards, and alternatives to the current practice. With rising levels of complexity, emerging and future standards should integrate more and more formal specifications. Much scientific progress has been made through industry standards. However, it is worth noting that the majority of computing system innovation does not adhere to any organized standardization process. Open source developer communities often prefer decentralized, agile development. Further, specialization pushes for optimizations across standards boundaries, breaking existing ones when higher efficiency ends up mattering more than portability. There is a need for better characterization of these ecosystems, and how to provide stronger guarantees (portability, maintainability, sustainability) to contributors and users.

Enhancing education and training. The integration of formal methods into standard tool-chains will help their dissemination and adoption. It is important to scale up the education and training initiatives, offering courses on formal methods and tools through a variety of university curricula.

6 About this Document

This report summarizes the findings in the NSF-sponsored "Workshop on Inter-disciplinary Research Challenges in Computing Systems", held in Williamsburg, Virginia on May 24-25, 2018

(https://www.asplos2018.org/grandchallenges/). This was a community visioning workshop to identify inter-disciplinary research challenges in computer systems to be addressed in the next 10 years. It was organized by Albert Cohen, Xipeng Shen, Josep Torrellas, James Tuck, and Yuanyuan Zhou. The first day of the workshop was open to everyone, and about 150 researchers attended. The second day was open only to researchers who had submitted position papers in advance. About 50 researchers participated in the discussions of the second day and helped write this report. Participants came from academia, industry, and government, representing multiple research communities, including computer architecture, programming languages, compilers, and operating systems, among others. The workshop included invited presentations and in-depth discussions. The workshop program is listed as follows.

Saturday March 24 2018 (Open to Public)
7:00 Breakfast
8:00 Introduction (Workshop organizers and Samee Khan, NSF)
8:30–10:10: Internet of Things and Infrastructure
8:30–9:00 Keynote: Marilyn Wolf (Georgia Tech)
9:00–10:10 Panel Moderators: Rajesh Gupta (UCSD) and Lin Zhong (Rice)
10:00–10:30 Break
10:30–12:15: Security and Privacy
10:30–11:00 Keynote: Dawn Song (UC Berkeley)
11:00–12:15 Panel Moderators: Sam King (UC Davis) and Yan Solihin (NC State Univ)
12:15–1:45 Lunch
1:45–3:30:Augmenting Human Abilities/AI
1:45–2:15 Keynote: Trishul Chilimbi (Amazon)
2:15–3:30 Panel Moderators: Hillery Hunter (IBM), Yuan Xie (UC Santa Barbara)
3:30–4:00 Break
4:00–5:45 Complexity Management
4:00–4:30 Keynote: Ras Bodik (University of Washington)
4:30–5:45 Panel Moderators: Gernot Heiser (University of New South Wales)
and Keshav Pingali (UT Austin)
***** **Open Session Ends** *****
6:30 Dinner (Invitation Only)
Summary points
Open mike
Sunday March 25 2018 (Invitation Only)
7:00 Breakfast
8:30–9:30 Recap of the previous day
9:30–10:45 Breakout into groups
10:45–11:15 Break (make slides)
11:15–12:15pm Plenary: Presentations from each group
12:15–1:30 Lunch
1:30–2:15 Plenary: Continuation of presentations
2:15–3:30 Breakout into groups. Incorporate feedback and initial writing
3:30–4:00 Break
4:00–5:30 Continue writing
5:30–6:15 Plenary: Status report and feedback

Acknowledgment. This material is based upon work supported by NSF under Grant No. CNS-1823068. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF.

References Cited

[1] IoT and Schema.org: Getting Started. https://iot.schema.org/docs/iot-gettingstarted.html, 2017.

[2] Yuvraj Agarwal, Stefan Savage, and Rajesh Gupta. Sleepserver: A software-only approach for reducing the energy consumption of pcs within enterprise environments. In *Proceedings of the USENIX Annual Technical Conference*, 2010.

[3] Advanced Micro Devices (AMD). AMD Secure Encrypted Virtualization. https://github.com/AMDESE/AMDSEV. 2018.

[4] Dmytro Apalkov, Alexey Khvalkovskiy, Steven Watta, Valdimir Nikitin, Xueti Tang, Daniel Lottis, Kiseok Moon, Xiao Luo, Eugene Chen, Adrian Ong, Alexander Driskill-Smith, and Mohamad Krounbi. Spin-transfer torque magnetic random access memory (STT-MRAM). In *ACM Journal on Emerging Technologies in Computing Systems (JETC)—Special issue on memory technologies*, pages 13:1–13:35, 2013.

[5] ARM. Armsecurity technology – building a secure system using trustzone technology. ARM Technical White Paperhttp://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf. 2009.

[6] Bharathan Balaji, Arka Bhattacharya, Gabriel Fierro, Jingkun Gao, Joshua Gluck, Dezhi Hong, Aslak Johansen, Jason Koh, Joern Ploennigs, Yuvraj Agarwal, et al. Brick: Metadata schema for portable smart building applications. *Applied Energy*, 2018.

[7] Stefano Basagni, Imrich Chlamtac, Violet R. Syrotiuk, and Barry A. Woodward. A distance routing effect algorithm for mobility (DREAM). In *International Conference on Mobile Computing and Networking*, 1998.

[8] Rastislav Bodík. Program synthesis: opportunities for the next decade. In *Proceedings of the 20th ACM SIGPLAN International Conference on Functional Programming, ICFP 2015, Vancouver, BC, Canada, September 1-3, 2015*, page 1, 2015.

[9] Hans-J. Boehm and Sarita V. Adve. Foundations of the c++ concurrency memory model. In *Proceedings of the 29th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '08, pages 68–78, New York, NY, USA, 2008. ACM.

[10] Geoffrey W. Burr, Matthew J. Breitwisch, Michele Franceschini, Davide Garetto, Kailash Gopalakrishnan, Bryan Jackson, Bulent Kurdi, Chung Lam, Luis A. Lastras, Alvaro Padilla, Bipin Rajendran, Simone Raoux, and Rohit S. Shenoy. Phase change memory technology. *Journal of Vacuum Science and Technology*, 28(2):223–262, 2010.

[11] Yu Cai, Erich F. Haratsch, Onur Mutlu, and Ken Mai. Threshold voltage distribution in MLC NAND flash memory: Characterization, analysis, and modeling. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2013.

[12] Ann Cavoukian. Privacy by Design. The 7 Foundational Principles. https://www.ipc.on.ca/wp-content/uploads/Resources/7foundationalprinciples.pdf, 2011.

[13] Dhruva R. Chakrabarti, Hans-J. Boehm, and Kumud Bhandari. Atlas: Leveraging locks for non-volatile memory consistency. In *Intl. Conf. on Object Oriented Programming Systems Languages & Applications (OOPSLA)*, pages 433–452, Portland, OR, 2014.

[14] DeQing Chen, Chunqiang Tang, Brandon Sanders, Sandhya Dwarkadas, and Michael L. Scott. Exploiting High-Level Coherence Information to Optimize Distributed Shared State. In *Conference on Principles and Practice of Parallel Programming*, 2003.

[15] Shimin Chen and Qin Jin. Persistent B+-trees in non-volatile main memory. *Proc. of the VLDB Endowment*, 8(7):786–797, Feb. 2015.

[16] Ping Chi, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, and Yuan Xie. PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory. In *43rd ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2016, Seoul, South Korea, June 18-22, 2016*, pages 27–39, 2016.

[17] Joel Coburn, Adrian M. Caulfield, Ameen Akel, Laura M. Grupp, Rajesh K. Gupta, Ranjit Jhala, and Steven Swanson. NV-Heaps: Making persistent objects fast and safe with next-generation, non-volatile memories. In *16th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 105–118, Newport Beach, CA, 2011.

[18] Alexei Colin, Emily Ruppel, and Brandon Lucia. A Reconfigurable Energy Storage Architecture for Energy-harvesting Devices. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2018.

[19] Jeremy Condit, Edmund B. Nightingale, Christopher Frost, Engin Ipek, Benjamin Lee, Doug Burger, and Derrick Coetzee. Better I/O through byte-addressable, persistent memory. In *22nd Symp. on Operating Systems Principles (SOSP)*, pages 133–146, Big Sky, MT, 2009.

[20] Victor Costan and Srinivas Devadas. Intel SGX explained. *IACR Cryptology ePrint Archive*, 2016:86, 2016.

[21] Victor Costan, Ilia A. Lebedev, and Srinivas Devadas. Secure processors part I: background, taxonomy for secure enclaves and Intel SGX architecture. *Foundations and Trends in Electronic Design Automation*, 11(1-2):1–248, 2017.

[22] Olivier Crameri, Nikola Knezevic, Dejan Kostic, Ricardo Bianchini, and Willy Zwaenepoel. Staged deployment in mirage, an integrated software upgrade testing and distribution system. *ACM SIGOPS Operating Systems Review*, 41, 2007.

[23] Chris Cummins, Pavlos Petoumenos, Alastair Murray, and Hugh Leather. Compiler fuzzing through deep learning. In *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2018, Amsterdam, The Netherlands, July 16-21, 2018*, pages 95–105, 2018.

[24] Chris Cummins, Pavlos Petoumenos, Zheng Wang, and Hugh Leather. End-to-end deep learning of optimization heuristics. In *26th International Conference on Parallel Architectures and Compilation Techniques, PACT 2017, Portland, OR, USA, September 9-13, 2017*, pages 219–232, 2017.

[25] Benoît Dupont de Dinechin. Kalray mppa®: Massively parallel processor array: Revisiting DSP acceleration with the kalray MPPA manycore processor. In *2015 IEEE Hot Chips 27 Symposium (HCS), Cupertino, CA, USA, August 22-25, 2015*, pages 1–27, 2015.

[26] Justin DeBrabant, Joy Arulraj, Andrew Pavlo, Michael Stonebraker, Stan Zdonik, and Subramanya Dulloor. A prolegomenon on OLTP database systems for non-volatile memory. In *Intl. Wkshp. on Accelerating Data Management Systems Using Modern Processor and Storage Architectures (ADMS@VLDB)*, pages 57–63, Hangzhou, China, 2014.

[27] Zachary DeVito, James Hegarty, Alex Aiken, Pat Hanrahan, and Jan Vitek. Terra: A multi-stage language for high-performance computing. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '13, pages 105–116, New York, NY, USA, 2013. ACM.

[28] W. Dong, Y. Liu, C. Chen, J. Bu, and C. Huang. Reprogramming Using Relocatable Code in Networked Embedded Systems. In *Proc. IEEE INFOCOM*, 2011.

[29] Tudor Dumitras and Priya Narasimhan. Why do upgrades fail and what can we do about it?: toward dependable, online upgrades in enterprise system. In *International Conference on Middleware*, 2009.

[30] European Union. European Union's General Data Protection Regulation (GDPR). `https://en.wikipedia.org/wiki/General_Data_Protection_Regulation`, 2018.

[31] B. Feinberg, U. Vengalam, N. Whitehair, S. Wang, and E. Ipek. Enabling scientific computing on memristive accelerators. In *International Symposium on Computer Architecture (ISCA)*, 2018.

[32] V. Fratin, D. Oliveira, C. Lunardi, F. Santos, G. Rodrigues, and P. Rech. Code-dependent and architecture-dependent reliability behaviors. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 13–26, June 2018.

[33] Qian Ge, Yuval Yarom, and Gernot Heiser. No security without time protection: We need a new hardware-software contract. In *Asia-Pacific Workshop on Systems (APSys)*, August 2018.

[34] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *IEEE S&P*, 2018.

[35] Daniel Genkin, Luke Valenta, and Yuval Yarom. May the fourth be with you: A microarchitectural side channel attack on several real-world applications of curve25519. In *CCS*, pages 845–858. ACM, 2017.

[36] E. R. Giles, K. Doshi, and P. Varman. Softwrap: A lightweight framework for transactional support of storage class memory. In *31st Symp. on Massive Storage Systems and Technology (MSST)*, pages 1–14, Santa Clara, CA, 2015.

[37] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pages 2672–2680, Cambridge, MA, USA, 2014. MIT Press.

[38] Indranil Gupta, Tushar D. Chandra, and German S. Goldszmidt. On scalable and efficient distributed failure detectors. In *Symposium on Principles of Distributed Computing*, 2001.

[39] Nicolas Halbwachs. *Synchronous Programming of Reactive Systems*. Springer-Verlag, Berlin, Heidelberg, 2010.

[40] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2018.

[41] Terry Ching-Hsiang Hsu, Helge Bruegner, Indrajit Roy, Kimberly Keeton, and Patrick Eugster. NVthreads: Practical persistence for multi-threaded applications. In *12th ACM European Systems Conf. (EuroSys)*, pages 468–482, Belgrade, Republic of Serbia, 2017.

[42] Joseph Izraelevitz, Terence Kelly, and Aasheesh Kolli. Failure-atomic persistent memory updates via JUSTDO logging. In *21st Intl. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 427–442, Atlanta, GA, 2016.

[43] Joseph Izraelevitz, Hammurabi Mendes, and Michael L. Scott. Linearizability of persistent memory objects under a full-system-crash failure model. In *30th Intl. Conf. on Distributed Computing (DISC)*, pages 313–327, Paris, France, 2016.

[44] Sushant Jain, Kevin Fall, and Rabin Patra. Routing in a delay tolerant network. In *Conference on applications, technologies, architectures, and protocols for computer communications (SIGCOMM '04)*, 2004.

[45] Evan Jones, Lily Li, Jakub K. Schmidtke, and Paul Ward. Practical routing in delay-tolerant networks. *IEEE Transactions on Mobile Computing*, 6, 2007.

[46] Arpit Joshi, Vijay Nagarajan, Marcelo Cintra, and Stratis Viglas. Efficient persist barriers for multicores. In *48th Intl. Symp. on Microarchitecture (MICRO)*, pages 660–671, Waikiki, HI, 2015.

[47] Norman P. Jouppi, Cliff Young, Nishant Patil, David A. 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 *ISCA*, pages 1–12. ACM, 2017.

[48] Ralf Jung, Jacques-Henri Jourdan, Robbert Krebbers, and Derek Dreyer. Rustbelt: securing the foundations of the Rust programming language. *PACMPL*, 2(POPL):66:1–66:34, 2018.

[49] Jan-Oliver Kaiser, Hoang-Hai Dang, Derek Dreyer, Ori Lahav, and Viktor Vafeiadis. Strong logic for weak memory: Reasoning about release-acquire consistency in iris (artifact). *DARTS*, 3(2):15:1–15:2, 2017.

[50] Neel Kant. Recent advances in neural program synthesis. *CoRR*, abs/1802.02353, 2018.

[51] David Karger and Matthias Ruhl. Simple efficient load balancing algorithms for peer-to-peer systems. In *Symposium on Parallelism in Algorithms and Architectures*, 2004.

[52] Gabor Karsai, Janos Sztipanovits, Akos Ledeczi, and Ted Bapty. Model-integrated development of embedded software. In *Proceedings of the IEEE*, 2003.

[53] Keystone. keystone: Open-source secure hardware enclave. `https://keystone-enclave.org`.

[54] Duckhwan Kim, Jaeha Kung, Sek M. Chai, Sudhakar Yalamanchili, and Saibal Mukhopadhyay. Neurocube: A programmable digital neuromorphic architecture with high-density 3d memory. In *43rd ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2016, Seoul, South Korea, June 18-22, 2016*, pages 380–392, 2016.

[55] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Haburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwartz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution. In *IEEE Symposium on Security and Privacy*, pages 19–37, May 2019.

[56] A. Kolli, J. Rosen, S. Diestelhorst, A. Saidi, S. Pelley, S. Liu, P. M. Chen, and T. F. Wenisch. Delegated persist ordering. In *49th Intl. Symp. on Microarchitecture (MICRO)*, pages 1–13, Taipei, Taiwan, 2016.

[57] Ori Lahav, Viktor Vafeiadis, Jeehoon Kang, Chung-Kil Hur, and Derek Dreyer. Repairing sequential consistency in c/c++11. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2017, pages 618–632, New York, NY, USA, 2017. ACM.

[58] Benjamin C. Lee, Engin Ipek, Onur Mutlu, and Doug Burger. Architecting phase change memory as a scalable DRAM alternative. In *36th Intl. Symp. on Computer Architecture (ISCA)*, pages 2–13, Austin, TX, 2009.

[59] Michael K. K. Leung, Andrew Delong, Babak Alipanahi, and Brendan J. Frey. Machine Learning in Genomic Medicine: A Review of Computational Problems and Data Sets. In *Proceedings of the IEEE*, 2016.

[60] Moritz Lipp, Michael Schwartz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown: Reading kernel memory from user space. In *USENIX Security Symposium*, August 2018.

[61] Zhijian Lu, Jason Hein, Marty Humphrey, Mircea Stan, John Lach, and Kevin Skadron. Control-theoretic Dynamic Frequency and Voltage Scaling for Multimedia Workloads. In *International Conference on Compilers, Architectures, and Synthesis for Embedded Systems (CASES)*, 2002.

[62] Brandon Lucia and Benjamin Ransford. A Simpler, Safer Programming and Execution Model for Intermittent Systems. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2015.

[63] Kai Ma, Xue Li, Ming Chen, and Xiaorui Wang. Scalable power control for many-core architectures running multi-threaded applications. In *International Symposium on Computer Architecture*, 2011.

[64] Sela Mador-Haim, Rajeev Alur, and Milo M. K. Martin. Generating litmus tests for contrasting memory consistency models. In Tayssir Touili, Byron Cook, and Paul Jackson, editors, *Computer Aided Verification*, pages 273–287, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[65] Martina Maggio, Alessandro Vittorio Papadopoulos, Antonio Filieri, and Henry Hoffmann. Automated Control of Multiple Software Goals Using Multiple Actuators. In *European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, 2017.

[66] Nicholas D. Matsakis and Felix S. Klock, II. The Rust language. In *Proceedings of the 2014 ACM SIGAda Annual Conference on High Integrity Language Technology*, HILT '14, pages 103–104, New York, NY, USA, 2014. ACM.

[67] Timothy Morgan. Intel Shows Off 3D XPoint Memory Performance. ARM Technical White Paper`https://www.nextplatform.com/2015/10/28/intel-shows-off-3d-xpoint-memory-performance/`. 2015.

[68] Sanketh Nalli, Swapnil Haria, Mark D. Hill, Michael M. Swift, Haris Volos, and Kimberly Keeton. An analysis of persistent memory use with WHISPER. In *22nd Intl. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 135–148, Xi'an, China, 2017.

[69] National Science Foundation. Cyber-Physical Systems Virtual Organization. `https://cps-vo.org`, 2018.

[70] Faisal Nawab, Joseph Izraelevitz, Terence Kelly, Charles B. Morrey, Dhruva Chakrabarti, and Michael L. Scott. Dalí: A periodically persistent hash map. In *31st Intl. Symp. on Distributed Computing*, Vienna, Austria, 2017.

[71] D. Oliveira, L. Pilla, M. Hanzich, V. Fratin, F Fernandes, C. Lunardi, J. Cela, P. Navaux, L. Carro, and P. Rech. Radiation-Induced Error Criticality in Modern HPC Parallel Accelerators. In *Proceedings of 21st IEEE Symp. on High Performance Computer Architecture (HPCA)*. ACM, 2017.

[72] Ismail Oukid, Johan Lasperas, Anisoara Nica, Thomas Willhalm, and Wolfgang Lehner. FP-Tree: A hybrid SCM-DRAM persistent and concurrent B-tree for storage class memory. In *Intl. Conf. on Management of Data (SIGMOD)*, pages 371–386, San Francisco, CA, 2016.

[73] Rajesh Krishna Panta, Saurabh Bagchi, and Samuel P. Midkiff. Efficient incremental code update for sensor networks. *ACM Transactions on Sensor Networks (TOSN)*, 7, 2011.

[74] Emilio Parisotto, Abdelrahman Mohamed, Rishabh Singh, Lihong Li, Denny Zhou, and Pushmeet Kohli. Neuro-symbolic program synthesis. In *ICLR 2017*, February 2017.

[75] Steven Pelley, Peter M. Chen, and Thomas F. Wenisch. Memory persistency. In *41st Intl. Symp. on Computer Architecuture (ISCA)*, pages 265–276, Minneapolis, MN, 2014.

[76] Steven Pelley, Thomas F. Wenisch, Brian T. Gold, and Bill Bridge. Storage management in the NVRAM era. *Proc. of the VLDB Endowment*, 7(2):121–132, Oct. 2013.

[77] Randall Pittman, Hui Guan, Xipeng Shen, Seung-Hwan Lim, and Robert M. Patton. Exploring flexible communications for streamlining DNN ensemble training pipelines. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC)*, 2018.

[78] Raghavendra Pradyumna Pothukuchi, Amin Ansari, Petros Voulgaris, and Josep Torrellas. Using Multiple Input, Multiple Output Formal Control to Maximize Resource Efficiency in Architectures. In *International Symposium on Computer Architecture*, June 2016.

[79] Andy Rudoff. Persistent memory programming. `http://pmem.io/`. Accessed: 2017-04-21.

[80] Sara Sabour, Nicholas Frosst, and Geoffrey E. Hinton. Dynamic routing between capsules. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017*, 2017.

[81] Daniel Selsam, Percy Liang, and David L. Dill. Developing bug-free machine learning systems with formal mathematics. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 3047–3056, 2017.

[82] Thomas Shull, Jian Huang, and Josep Torrellas. Defining a High-Level Programming Model for Emerging NVRAM Technologies. In *International Conference on Managed Languages and Runtimes (ManLang)*, 2018.

[83] Jeremy G. Siek and Walid Taha. Gradual typing for functional languages. In *Scheme and Functional Programming Workshop*, pages 81–92, 2006.

[84] Armando Solar-Lezama. *Program Synthesis by Sketching*. PhD thesis, Berkeley, CA, USA, 2008. AAI3353225.

[85] Thrasyvoulos Spyropoulos, Konstantinos Psounis, and Cauligi S. Raghavendra. Efficient routing in intermittently connected mobile networks: The multiple-copy case. *Transactions on Networking (ToN)*, 16, 2008.

[86] Dmitri B. Strukov, Gregory S. Snider, Duncan R. Stewart, and R. Stanley Williams. The missing memristor found. *Nature*, 453(7191):80–83, May 2008.

[87] Arvind K. Sujeeth, Kevin J. Brown, HyoukJoong Lee, Tiark Rompf, Hassan Chafi, Martin Odersky, and Kunle Olukotun. Delite: A compiler architecture for performance-oriented embedded domain-specific languages. *ACM Trans. Embedded Comput. Syst.*, 13(4s):134:1–134:25, 2014.

[88] Chunqiang Tang, DeQing Chen, Sandhya Dwarkadas, and Michael L. Scott. Efficient Distributed Shared State for Heterogeneous Machine Architectures. In *International Conference on Distributed Computing Systems (ICDCS)*, 2003.

[89] Serbulent Tozlu, Murat Senel, Wei Mao, and Abtin Keshavarzian. Wi-Fi enabled sensors for internet of things: A practical approach. *IEEE Communications Magazine*, 50, 2012.

[90] Viktor Vafeiadis, Thibaut Balabonski, Soham Chakraborty, Robin Morisset, and Francesco Zappa Nardelli. Common compiler optimisations are invalid in the C11 memory model and what we can do about it. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015*, pages 209–220, 2015.

[91] Augusto Vega, Alper Buyuktosunoglu, Heather Hanson, Pradip Bose, and Srinivasan Ramani. Crank It Up or Dial It Down: Coordinated Multiprocessor Frequency and Folding Control. In *International Conference on Microarchitecture (MICRO)*, 2013.

[92] Haris Volos, Andres Jaan Tack, and Michael M. Swift. Mnemosyne: Lightweight persistent memory. In *16th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 91–104, Newport Beach, CA, 2011.

[93] Tianzheng Wang and Ryan Johnson. Scalable logging through emerging non-volatile memory. *Proc. of the VLDB Endowment*, 7(10):865–876, June 2014.

[94] Qiang Wu, Philo Juang, Margaret Martonosi, and Douglas W. Clark. Formal Online Methods for Voltage/Frequency Control in Multiple Clock Domain Microprocessors. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2004.

[95] Feng Zhang, Jidong Zhai, Xipeng Shen, Onur Mutlu, and Wenguang Chen. Efficient document analytics on compressed data: Method, challenges, algorithms, insights. In *Proceedings of the 44th International Conference on Very Large Data Bases (VLDB)*, 2018.

[96] Zhensheng Zhang. Routing in intermittently connected mobile ad hoc networks and delay tolerant networks: Overview and challenges. *IEEE Communications Surveys & Tutorials*, 8, 2006.

[97] Ben Yanbin Zhao, John Kubiatowicz, and Anthony D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. In *Technical Report UCB/CSD-01-1141, Computer Science Division (EECS) University of California Berkeley, California 94720*, 2001.