Workshop on Advancing Computer Architecture Research (ACAR-1)

Failure is not an Option: Popular Parallel Programming

Organizers: Josep Torrellas (University of Illinois) and Mark Oskin (University of Washington).

Steering Committee: Chita Das (NSF and Pennsylvania State University), William Harrod (DARPA), Mark Hill (University of Wisconsin), James Larus (Microsoft Research), Margaret Martonosi (Princeton University), Jose Moreira (IBM Research), and Kunle Olukotun (Stanford University).

Written by: Josep Torrellas, Mark Oskin, Sarita Adve, George Almasi, Luis Ceze, Almadena Chtchelkanova, Chita Das, Bill Feiereisen, William Harrod, Mark Hill, Jon Hiller, Sampath Kannan, Krishna Kant, Christos Kozyrakis, James Larus, Richard Murphy, Onur Mutlu, Satish Narayanasamy, Kunle Olukotun, Yale Patt, Anand Sivasubramaniam, Kevin Skadron, Karin Strauss, Steven Swanson, and Dean Tullsen.

Funded by the Computing Research Association's (CRA) Computing Community Consortium (CCC) as a "visioning exercise" meant to promote forward thinking in computing research and then bring these ideas to a funded program.

Held on February 21-23, 2010 in San Diego, California
Contact: torrella@illinois.edu; oskin@cs.washington.edu
Websites: http://www.cra.org/ccc/acar.php; http://iacoma.cs.uiuc.edu/acar1

August 2010

Contents

1 Executive Summary

The arrival of the ubiquitous multi-core is a game-changing event for the computing industry. Much of the industry today is relying on parallel processing becoming main-stream --- although most software is still single-threaded and past experience consistently shows that parallelization is a difficult task. For the industry to make progress at historical rates, the next few years will need to witness significant changes in the software and hardware of our computing platforms. In particular, from a computer architecture perspective, multi/many-cores will have to evolve to enable and support high-productivity parallel software development and execution.

This workshop brought together computer architecture researchers from academia, industry, and national laboratories, and program managers from funding agencies to examine how computer architecture can help enable ubiquitous parallel computing. The main goal was to identify the key computer architecture research challenges in devising the programmable parallel computing platforms of years 2020-2025, and to articulate an agenda and roadmap to address these challenges. The resulting research directions had to have broad community support, be key for funding agencies to fund, be forward looking rather than incremental, and lead to a deep understanding of our field.

The attendees identified four main computer architecture research thrusts. They are (1) data centers and large-scale systems, (2) architectures to enhance programmability, (3) hardware-software co-design and asymmetry, and (4) domain specific languages. For each thrust, the report outlines the research efforts recommended.

1. Data centers are emerging as a critical component of our IT infrastructure. In particular, they form the backbone of cloud computing, which will likely provide the utility computing necessary to revolutionize experimental computing techniques. Data centers will be used by billions of cost-conscious users and run applications written by many developers. In this research thrust, our ambitious long-term goals are to reduce the cost of data-center infrastructure to 1 Watt and $1 per month for the typical user, and to enable individual programs to efficiently scale from a single-node system with tens of users to a full data-center deployment with millions of nodes and billions of users.

2. The continued performance scaling of computer systems now requires extensive software and hardware changes to exploit parallelism. Attaining a high-performance, correct 1000-core chip that is also highly programmable is a major challenge. In this research thrust of architectures for programmability, our long-term goal is three-fold: (1) to ensure that programming for parallel architectures is as easy as it is now for sequential architectures; (2) to maintain Moore's Law for performance --- namely, to double the speed-up every 2 years; and (3) to eliminate concurrency bugs.

3. Specialization of the hardware and system-software layers eliminates inefficiencies and overheads that come with the flexibility of general-purpose systems. It is possible to obtain orders of magnitude improvements in performance, performance per Watt, and performance per dollar. In the past, specialization has usually been limited to a small number of high-volume consumer applications. In this research thrust, we want to develop technologies necessary to deliver "turn-key" specialized computing systems quickly and economically. Specifically, our long-term goal is to design specialized architectures that deliver up to 10,000x speed-up for particular applications for less than $10,000. The ultimate goal is a fully-automated generation of application-specific hardware for each program.

4. Domain specific languages (DSLs) are designed to express a particular class of applications efficiently. In the context of parallel computing, a DSL provides a set of semantics that are expressive and natural for developers of such a class of applications, yet describe the computation at a sufficiently high enough level that it is easy to exploit parallelism. In this research thrust, our long-term goals are to create DSL infrastructure that makes it easy to develop new DSLs, and to attain 5-10 commercially-available DSLs in the next 10-15 years. These DSLs should have widespread use. They should be as successful as SQL for their particular application domains.

To make progress in all of these challenges, our universities must offer a strong educational program in parallel computing. We suggest making low-resistance changes to the curriculum to embed parallel thinking, including augmenting existing courses with simple additions where they make sense. We should work with NSF to impart urgency in the need to teach parallel practices throughout the curriculum.

It is crucial to initiate a broad discussion between industry and academia on programming models, programming languages, programmer productivity, applications, and the computer architectures to support them all. To make the academia/industry relationship robust, academics should invite industry participants to panels, workshops, and other discussion forums on parallel computing. Academics should also prepare short courses on parallel programming and computing for professionals in industry. Industry, in turn, should provide direction and input into the critical problems it faces, provide access to experimental data, and provide funding for academic research.

Government funding for computer architecture research should be organized along larger, more ambitious projects that cover multiple layers of the computing stack. It should also involve a concerted, complementary effort by multiple funding agencies, focusing on the research thrusts identified here.

The next steps involve working with our professional colleagues to publicize this report among funding agencies, industry, academic circles, and the broad computer science and engineering community.

2 Introduction

2.1 Background

The arrival of the ubiquitous multi-core has caused an upheaval in the computing industry. Hardware vendors can no longer produce microprocessors that make yesterday's software exponentially faster. Instead, the industry is betting on radical changes in software and hardware for its success. The fate of much of the IT industry rests on the success of main-streaming parallel (or concurrent) computing.

The switch to parallel hardware began, from the consumer's perspective, around 2004. Since then, the number of cores per chip has kept increasing. Similar to the switch from 32 to 64-bit computing, hardware and software changes are uncoordinated. Consumer-oriented computers with eight cores are easily available for less than $1,000. Yet, just as a vast array of systems still execute 32-bit operating systems on 64-bit capable hardware, most software today remains single-threaded. Those applications that do take advantage of multi-core architectures do so only tepidly --- with limited threads and little hope of scaling to the kind of parallel resources that hardware vendors will be capable of providing in the near future. Against this disappointing background, domain-specific systems such as GPUs, and novel concurrency models such as cloud computing flourish, but they do not use many of the bread-and-butter multi-core designs currently planned.

2.2 Strategic Importance

The Information Technology (IT) sector has been a leading driver of economic growth in the modern world. As recent downturns have shown, a significant drop in the IT sector has widespread ramifications for all areas of our economy. At the center of the IT growth and innovation is the ability for hardware to provide the core building block of the industry, the processor, in a form that is exponentially more efficient --- either faster or lower power --- each year. Historically, the computing industry has used this exponential efficiency gain to provide the world with ever richer software environments, more powerful portable devices, and faster communication links. Multi-core changes this. These efficiencies are no longer applied to products automatically. Instead, software developers are expected to be an integral component in the exponential efficiency gains we hope to see going forward. It is for this reason that popularizing parallel programming is of strategic importance to the IT industry and nation at large.

2.3 The Opportunity

As with all great challenges, there comes a great opportunity: if software can be successfully adapted to execute on multi-core devices, then new levels of performance and efficiencies can be obtained. If one task can be parallelized across multiple cores, those cores can finish that task sooner or, for the same total execution time, with significantly less energy.

In practice, many applications can deliver new capabilities if one can provide orders of magnitude improvement in performance, performance per unit of power, or performance per unit of cost. This is true in all types of computer systems, from embedded and mobile systems to supercomputers. Examples in the mobile space include speech recognition, language translation, data analysis or situational awareness. For example, these could benefit firefighters, police, doctors, or soldiers in real-time field work. In the desktop, notebook, or workstation space, some examples of applications include video processing, rich user interfaces or virtual reality interfaces, and interactive problem solving and data analysis. As local storage capacities grow, the types of important problems that can be solved in a personal computing environment also grow. For example, the entire human genome fits in approximately 4 GB, allowing a single workstation to perform significant computations. Finally, at data-center or super-computer scale, problems that can benefit from the efficiencies of multi-cores include computational fluid dynamics, drug discovery, simulation and modeling (e.g., weather, geology, and tsunami prediction), data mining, machine learning, and graph analytics.

2.4 Limited Industry Experience and Deficient Educational Systems

The fact that parallel programming has never been widespread in the IT industry complicates the ability to solve this challenge. Parallel programming has traditionally been a niche --- successful in a few important domains such as

scientific computing, graphics computations, and server workloads. There are lessons to be learned from these successes. Specifically, parallel programming is possible with relatively small code bases written by a handful of domain experts (such as in scientific computing); where the interactions between threads are limited (such as in many server workloads); or where the programming model is sequential in nature, with parallelism occurring under the hood (such as in graphics). Unfortunately, these key enablers of a small set of expert programmers, of limited communication, and of sequential abstractions, do not generalize to all applications.

A second, almost chicken-and-egg problem exists with parallel computing. Few university professors are competent to teach concepts in parallel computing. Professors with specialties in machine learning or graphics have expert knowledge in those domains --- not in the craft of parallel computing. For this reason, when they create projects and homework assignments for students in classes on those subjects, there is no expectation that students will design a parallel solution. Hence, our universities do not produce undergraduate computer science majors with much expertise in parallel computing. This means that employers cannot hire the type of engineer they need to build products that successfully utilize multi-core devices.

2.5 Talent Abundance and Funding Scarcity

There is an outdated sense in computer science, and hence in the levers of computer science funding, that computer architecture is a solved problem. The fact that the multi-core was unleashed in 2004 without any software co-design consideration is ample proof that this is not the case. If anything, computer architecture is undergoing a period of profound revolution like when the single-chip microprocessor was invented or the von-Neumann model itself was conceived. Far from being a solved problem, computer architecture stands at an inflection point. With the right talent, funding, and will, it is a field poised to bring about lasting revolutionary changes to the hardware/software interface.

Of the necessary ingredients for progress, the talent component is abundant: the last five years have seen a great increase in the number of new faculty members who are actively doing research in computer architecture. Our community now includes a large number of assistant professors, many of them overflowing with new ideas and plans. The funding component is lacking. Specifically, the general trend in the last few years at NSF has been only very modest funding increases (or perhaps no increases) for computer architecture, and the large majority of funded projects are small and very modest in ambition. DARPA has recently started an initiative on Extreme Scale computing, but its size is small. DOE focuses its limited efforts in computer architecture on Petascale/Exascale systems, which relate to a relatively small fraction of our community. Finally, other funding agencies have little interest in funding computer architecture research.

Against this backdrop, industry (e.g., Intel and Microsoft) has funded centers that focus on the whole computing stack for parallelism. This is a measure of the importance of the problem and the perceived lack of support from the funding agencies. In reality, solving the challenges facing the IT industry will likely need placing funding bets on as many researchers and worthy topics as possible.

3 Workshop Objectives

The Computing Research Association's (CRA) Computing Community Consortium (CCC) convened a workshop of computer architecture researchers and funding agencies to address the challenge of attaining popular parallel programming. The goal of the workshop was two-fold:

1. Articulate an agenda and roadmap for computer architecture research to devise the programmable parallel computing platforms of years 2020-2025. The recommended research directions must meet the following tests: (a) have broad community support as (some of) the most important research questions to address; (b) be key for academic researchers to explore and national agencies to fund; (c) lead to significant changes in and/or deep understanding of our field; (d) be forward looking and not incremental extensions of the status quo; (d) be critical to solve for the health of our IT industry.

2. Create excitement within the computer architecture community and help build new bridges and collaborations.

The workshop had to be inclusive of enough voices in the research community that the results are legitimately viewed as coming from the community, speaking with a unified voice. Moreover, the workshop should include as many junior researchers in our community as possible to ensure continuous leadership.

A call for position papers was issued and publicized widely. Appendix A shows the Call for Position Papers. The co-organizers and the Steering Committee selected a total of 16 position papers among them. The workshop took place on February 21-23, 2010 in San Diego, California, and 25 individuals attended. Appendix B lists the attendees and Appendix C shows the schedule of the workshop. The workshop had keynotes from George Almasi (IBM Research) and James Larus (Microsoft Research). It also had a panel of funding agency directors chaired by Sampath Kannan (NSF). In the morning of the first day, the authors of the selected position papers presented them. The rest of the workshop consisted of breakout sessions (where the attendees broke into small working groups) and plenary sessions (where the findings of the groups were presented and critiqued).

The workshop converged into and fleshed out four research thrusts, namely (1) data centers and large scale systems, (2) architectures to enhance programmability, (3) hardware-software co-design and asymmetry, and (4) domain specific languages. In addition, there were discussions on educational issues, industry collaboration, and funding collaboration.

The rest of this report documents the findings. Appendix D shows the slides of the working groups, on which this report is based.

4 Recommendations for Computer Architecture Research Thrusts

We identify four key computer architecture research thrusts. They are (1) data centers and large scale systems, (2) architectures to enhance programmability, (3) hardware-software co-design and asymmetry, and (4) domain specific languages. We present each in turn.

4.1 Data Centers and Large Scale Systems

4.1.1 *Problem Statement and Goals*

Data Centers (DCs) are emerging as a critical component of our IT infrastructure. These large-scale systems provide the computation power and storage necessary for online services ranging from webmail and search, to social networking and customer-relationship management tools. Moreover, DCs form the backbone of cloud computing, which is viewed as the prominent way provide the utility computing necessary to revolutionize experimental techniques for biology, neuroscience, drug discovery, and other scientific disciplines.

DCs are in many ways similar to traditional supercomputers, which support large-scale scientific experiments. Both include tens of thousands of nodes, implement distributed memory and storage schemes, and involve hierarchical networks. However, as both the enterprise and the scientific community strive toward utility computing, a new set of requirements is emerging. DCs will now be used by billions of users, either directly through services like social networks or indirectly through services like bioinformatics. The number of data center application developers will also be significantly higher, including developers of online services and scientists using analytical techniques. Moreover, the vision of utility computing requires low cost, for both capital and operational expenses.

We offer two grand challenges or goals for research in enterprise DCs and next generation supercomputers: (1) Reduce the cost of DC infrastructure to 1 Watt and $1 per month for the typical user; and (2) Enable individual programs to efficiently scale from a single-node system with tens of users to a full DC deployment with millions of nodes and billions of users.

The first challenge comes from the need for low cost, utility computing. It is an aggressive goal that requires cooperative research in hardware architecture, system software, and resource management. If we put all aspects of our lives and all scientific experiments on DCs using current approaches, we will probably be off by factors of 100x to 1,000x from the goal of 1 Watt and $1 per month per user.

The second challenge stems from the need for scalable capabilities and ease-of-use. It requires significant improvements to the hardware infrastructure, programming models, and system software used in DCs.

4.1.2 *Research Thrust Description*

To address these challenges, we propose several research vectors that systematically revisit all aspects of a DC. The key is to view and optimize the DC in the same way we optimize an individual computer, with energy efficiency as the crosscutting theme. The five, interacting research vectors are: (1) Chip architecture --- what new features are needed for chips used in DCs? (2) Node architecture --- what is the most efficient node organization for DCs? (3) Memory and storage --- what are the hardware and software mechanisms for DC-scale memory and storage hierarchies? (4) Operating and runtime systems --- what is the management layer for the whole data center? and (5) Energy-efficient DC design --- a crosscutting theme across all layers.

4.1.2.1 Chip Architecture for Data Centers

Our vision is to enable many-core chips that are efficient building blocks for DCs. The key research question we aim to answer is how do we design a many-core chip from the ground up to enable a mini supercomputer/DC on a chip?

There are several key topics to research to enable this vision. First, we need to enable mechanisms for flexible on-chip resource management. This includes on-chip support for isolation and privacy, to eliminate interference between multiple tasks or virtual machines. It also includes prioritization and partitioning mechanisms to enable flexible allocation of on-chip resources (e.g., cores, caches, interconnects, memory, and bandwidth) among threads or applications. Second, we need to enable mechanisms for flexible scaling down of on-chip and memory resources. This is essential for the scalability and energy proportionality of the data center. It can be attained with flexible modes for energy/performance levels, possibly with configurable or heterogeneous designs, and for different on-chip resources. Third, we need to enable fast messaging and synchronization support for both on- and off-chip communication, as well as mechanisms for safe and efficient remote memory and storage access. Fourth, we need to provide mechanisms for end-to-end monitoring of application performance. This is essential for understanding, tuning, and optimizing applications running on the large-scale system. Fifth, we need to enable efficient execution of dynamic languages. Finally, we need to provide chip-level mechanisms for manageability, reliability, and upgradability to enable easier management of the entire DC. For each of these topics, we need to consider the hardware/software interface, virtualization mechanisms, and practical hardware implementations.

4.1.2.2 Node Architecture for Data Centers

Current DC nodes are a direct evolution of personal computers. As such, they are not optimal for the parameters of large-scale DCs. Therefore, a rethinking of the node is necessary to build an optimal DC.

A number of questions need to be researched to enable efficient node architectures. One is how to compose the chips and memory into a flexible node. The node can be homogeneous with simple cores, homogeneous with complex cores, or heterogeneous with a mix of cores of varying degrees of complexity and energy/performance levels. The node can include accelerators or, instead, acceleration can be performed at the chip level. The memory system and interconnect should be designed to enable trade-offs between throughput and quality of service. Different design choices have implications for the programming languages and the operating and runtime systems of the DC. It is therefore important to investigate the node design in conjunction with the design of such software.

Another key research question is memory and communication system organization at the node level. We need to understand how to organize node-level memory and storage among different chips (e.g., shared or partitioned). We also need to research what are the best mechanisms to enable efficient sharing and resource management. Most existing systems solely use commodity DRAM, which has high idle power consumption. The role of solid state disks and emerging non-volatile technologies need to be researched to enable large improvements in energy efficiency. It is also important to re-think the communication substrate at the node level to enable efficient and high-bandwidth communication and synchronization. An important avenue of investigation is using new technologies for communication, such as photonics.

4.1.2.3 Memory and Storage Hierarchy

The memory and storage hierarchy for DCs must provide software developers with fast, high-bandwidth access to peta-bytes of data, hiding its distributed implementation nature and all its consequences (i.e., latency and bandwidth bottlenecks, synchronization, and consistency issues).

There are several key topics to research to enable this vision, including hardware and software issues at the node and global level. To reduce latency, a significant percentage of DC services store data in DRAM. This creates a need to revisit the balance of processing and memory within each node. Current and new solid-state memory technologies must also be utilized to alleviate the challenges due to the cost or volatility of DRAM. At the system level, we need to consider locality optimizations that will hide latency. Depending on the application characteristics, DC-level prefetching and caching mechanisms may be possible. Alternatively, we can employ runtime techniques that move computation to data. In any case, locality optimizations must interact well with isolation, availability, and energy management considerations.

4.1.2.4 Operating and Runtime Systems

The operating and runtime systems of the DC play a critical role in managing the distributed resources to ensure the right provisioning to the right activity at the right time. This software stack has to necessarily scale to thousands of nodes, handling multiple levels of parallelism --- from tightly knit cores on a chip to multiple sockets in a node and to the hundreds of nodes connected by a loosely coupled network --- and ensuring that the computational, communication and storage resources are allocated to meet the different needs. These needs include (1) response time and/or throughput requirements of the applications, (2) availability requirements (e.g., five 9's of availability), (3) security and privacy assurances to insulate the set of running applications, and (4) compliance for regulation and auditability.

There are several dimensions to the research that is needed for the development of this software stack, namely scalability, end-to-end monitoring, and multi-tenancy. Scalability is of paramount concern to accommodate millions of nodes under performance, availability and power vagaries. The assumptions about parallelism possible at a node level do not hold under the loosely-coupled distributed environment of a DC, mandating dynamic adaptation to errors, performance, and resource availability.

To build a software stack that makes dynamic adaptation decisions, we need end-to-end monitoring of resource usage, performance, and availability. The software stack should also accommodate graceful degradation and scale-down based on resource availability.

Finally, multi-tenancy is an inherent feature of these systems, and accommodating the different applications and users temporally and spatially mandates extensive support for isolation from the performance, security and failure perspectives. This not just a software problem, but includes the entire stack from the silicon to the runtime system.

4.1.2.5 Energy-Efficient Data Center Design

Energy efficiency is a crosscutting theme across all the layers of a DC. By optimizing hardware and software within and across nodes, we can improve energy efficiency in terms of requests/sec/Watt by a factor of 100—1,000x or more over what conventional technology scaling can provide. The research highlighted above applied to commodity chips for compute, memory, and storage can improve energy efficiency by at least a factor of 10--100x. The improvements come from optimizing the balance and organization of chips in the system, the type of chip used for each task, and how tasks are moved close to data or vice versa. They also come from static and dynamic compilation techniques that reduce the software bloat of online services, which directly contributes to energy overheads. If we use chips optimized for data centers, we can expect another factor of 10--100x in energy efficiency. These improvements stem primarily from eliminating the overhead of complex software protocols for messaging, remote access, or isolation, reducing the overhead of checks for dynamic languages, and providing lower-power active modes for storage and network components that can be exploited by the runtime to achieve energy proportionality.

Energy proportionality is important in order to avoid waste during periods of low utilization. The active portion of the DC should be scaled down to match the lower utilization. While such scale-down approaches are already popular

for stateless computations, they are currently impossible for most services because each node stores gigabytes to terabytes of state. To achieve state-aware scale-down, we need new low-power active modes for all DC resources (including on-chip resources, chips, and memory/storage resources) and runtime techniques that can manage scale-down without compromising quality of service or availability.

Beyond energy efficiency, environmental sustainability is also a noteworthy goal for DCs. Current DC hardware is replaced every three years. We should build data center chips and memory/storage systems such that the hardware continues to be useful for a long time --- even in the presence of high failure and wear-out rates. We should (re)use materials that are environmentally friendly.

4.1.3 *Why is this Research Transformative*

Application requirements are growing at a very rapid pace, both in terms of problem scale and resolution. Evolutionary technology scaling is woefully inadequate to meet these requirements, since (1) voltages are not likely to be reduced much further, (2) as a result, it is no longer possible to power all the transistors on the chip, and (3) wires and interconnect do not scale with feature size reduction. Instead, these requirements can only be met with disruptive innovations at all levels of hardware and software. The research proposed here will enable DCs and supercomputers that are 1,000x more energy efficient and higher performance than what is possible with evolutionary technology scaling and known energy-efficient design techniques. This will enable new applications that require major improvements in energy efficiency and compute power.

The growing size of DCs introduces several new challenges related to large scale infrastructure management and operational costs associated with power delivery and distribution systems, cooling, space, and asset management. Only innovative architectural solutions can help address these issues, reducing power consumption and real estate footprint, doing more with less, and extending the lifetime of the infrastructure. In addition, integrated management at all levels can significantly lower the management and administration costs of DCs, enhance their availability and lower the bar on enabling widespread usage across a diverse user base.

4.1.4 *What Are the Deliverables*

The deliverables are DCs and supercomputers that, for the same level of performance, are 1,000x more energy efficient than what evolutionary technology scaling can provide, measured in requests/sec/Watt.

4.1.5 *Research Disciplines Involved*

The research proposed requires comprehensive, whole-system research. It includes the areas of computer architecture (chip design, platform architecture, and memory and storage systems), systems software, parallel programming, and new technologies.

4.1.6 *Risk to Industry and Society If Not Pursued*

The inability to solve the more complex and larger problems that are critical to the well-being of mankind --- such drug discovery and climate change adaptation --- will have a large negative impact on society. Even if one could solve larger problems by relying on evolutionary technology scaling, it will not be a cost-effective and scalable option, since the cost will increase significantly faster than economically desirable.

4.1.7 *Benefits of Success to Industry and Society*

The well-being of mankind critically depends on being able to solve key problems in the medical and physical sciences. The research proposed, by delivering more cost-effective DCs and supercomputers, can help us solve these problems.

4.1.8 *Why Not Let Industry Do It*

Industry is typically bound by concerns for backward compatibility. To make faster progress, we need research that breaks backward compatibility barriers and is not bound by possible short-term product concerns. Even if industry

performed some of this research internally, it might not make it available widely outside its organization. This would have a limiting effect on the advancement of science and technology.

4.1.9 *Likelihood of Success*

The proposed vision is very ambitious and requires advances in many areas, including chip design, platform architecture, memory and storage systems, systems software, parallel programming, and new technologies. We are confident that there will be rapid progress in all of these areas. We also hope that there will be breakthroughs in device technologies and materials that will significantly reduce energy consumption and will prove to be useful building blocks. Fortunately, the community is highly aware of the need to improve the energy efficiency of IT components.

4.2 Architectures to Enhance Programmability

4.2.1 *Problem Statement and Goals*

Historically, the computer industry has delivered sustained performance increases without asking programmers for significant software changes. However, continued performance scaling now requires extensive software and hardware changes to exploit parallelism. It is now much harder to get programmability, performance and correctness. This research thrust addresses the challenge of attaining a highly-programmable, high-performance, and correct 1000-core chip. Our long-term, ambitious goals are: (1) to ensure that programming for parallel architectures is as easy as it is now for sequential architectures; (2) to maintain Moore's Law for performance --- namely, to double the speedup every 2 years; and (3) to eliminate concurrency bugs.

4.2.2 *Research Thrust Description*

We have identified the following research vectors: programming model, correctness, introspection, scalable memory, and communication fabric, and resource management. We now detail them.

4.2.2.1 Programming Model

The von Neumann programming model supports advances of sequential computer architectures and programming languages. No such consensus facilitates progress for parallel architectures and languages. Instead, many current architectures and languages more or less expose multiple von Neumann processors, which burdens programmers and constrains architects.

Our vision of future parallel computers is that we should co-evolve programming models and architectures to facilitate the rapid development of correct, high-performance programs, i.e., enhance "programmability." The long term challenge is to determine the best ways to abstract locality, communication, and coordination so that programmers understand the "big picture" without being unduly burdened by details. We must co-develop several programming models and architectures as either an end or as a means to a possible consensus on "the" programming model for parallel architectures.

We should start by developing architectures for existing programming models --- data parallel, task parallel, functional, and sequential --- and co-evolving them. In particular, we should look for methods to express locality, communication (e.g., producer-consumer or pipelined), and coordination. We should ask how language restrictions (data-race-free, memory safety, and language features to be developed) can simplify hardware, improve performance, and reduce power. We should work with colleagues to develop new programming models and their corresponding architectures. We should study how to model hardware asymmetries (e.g., CPU vs. GPU). We should explore the role of explicit and implicit speculation. We should facilitate systems that allow the rapid development of correct programs and then enable a successive refinement of performance and resource use.

4.2.2.2 Correctness

Correctness should be treated as a first class citizen. Radical advances in architectures and programming models are required to increase the chance of having a correct program execution.

One of the grand challenges is to design a computer system where concurrency bugs do not affect correctness. One possible approach to achieve this can be a programming model and a runtime system that guarantee sequential semantics to a parallel program execution without sacrificing efficiency. Is non-determinism essential for performance? In addition, the same techniques that address software correctness can be used to ensure hardware correctness in spite of hardware design bugs, transient and hard faults.

In the near term, we should develop a wide range of software correctness checkers, debugging and testing tools. Idle cores could be used to improve the efficiency of these tools. Beyond support for a few watch-point registers, modern processors provide little to no support for testing and debugging tools. Advances in hardware primitives (e.g., bloom filters and hashes) could significantly help us develop efficient tools, and perhaps allow us to efficiently check the correctness of production runs. We should promote inter-disciplinary research that brings in ideas from various domains such as machine learning to help build efficient correctness checkers that leverage those hardware primitives. Such efficient correctness checkers could help us proactively stop the program on a bug or even skip incorrect parts of an execution.

4.2.2.3 Introspection

Given the complexity and programmability challenges of future systems, we should develop introspection mechanisms to enable continuous adaptation for performance, correct behavior, and energy efficiency in future systems. We envision a scenario where the hardware provides a set of detailed, precise and pervasive data collection mechanisms at a low cost. The information collected by these mechanisms will percolate up to the right levels of the system stack for automatic adaptation, fault management, and analysis. To support this vision, we need two main research avenues: (1) designing the mechanisms and their interface; and (2) using them to dynamically adapt system behavior.

The longer term goals of this research are to develop flexible and efficient ways to collect information about system behavior, from low-level architectural events to higher-level algorithmic events. The longer term goals also include how to use this information for performance, reliability, and energy efficiency. Interesting avenues of exploration here include support for machine learning to digest all of the information collected from the lower levels, and support for efficient online analysis of real-time information.

The shorter-term goals involve understanding the design space of introspection support and start exploring it. For example: When should collection be turned on/off? How is the information communicated up to the runtime system? How does the runtime system inform the hardware mechanisms of higher-level application properties for more targeted event collection? What information do we need to collect to allow the system to be constantly aware of what resource is in the critical path? What is the best way to visualize that information to understand complex systems?

4.2.2.4 Scalable Memory and Communication Fabric

We envision a highly-scalable memory and communication fabric for a 1000-core chip that provides performance, scalability, power efficiency, and flexibility. The memory and communication fabric is likely to be the bottleneck in many of the parallel applications in the time frame considered.

The long term research topic in this area is the design of the memory hierarchy and interconnection network for a 1000-core chip, including the cache coherence protocol, memory consistency issues, synchronization, and the support for task management. Such a design should provide high performance, be scalable to a 1000 cores, be power efficient in its operation, and provide flexibility to adapt the hardware to the actual needs of the programmer or application. Adaptation is challenging because there are different types of programmers, such as the one for whom safety is paramount, the one who wants high performance, and the expert one who needs no safety nets. The first

type of programmer --- which may include beginner programmers or those coding for mission-critical operations --- may require full cache-coherence support, a conservative memory consistency model, and various debugging checks turned on. The second one may relax some of these issues. The final one may want a completely programmer-managed data coherence, a relaxed memory consistency model, and no debugging checks.

There are several topics to be covered on our way to that vision. One of them is an analysis of the real needs of the different types of programmers. Another one is a re-structuring of the memory hierarchy and communication fabric where a program only pays the performance overhead and the power consumption of the resources that it uses. This "pay only what you use" approach requires lean structures for the memory hierarchy and network. Finally, the memory hierarchy and network should be designed for energy proportionality.

4.2.2.5 Resource Management

Compared to existing systems, emerging systems will contain far more resources, more tunable and adaptable resources, increased heterogeneity, and the potential for far more inter-application interaction. Expecting naïve users to manage these aspects is unworkable, and involving the programmer heavily in them only increases the difficulty of parallel programming. However, the programmer should be able to express goals (e.g., performance or energy efficiency) and access primitives that allow him/her to reach those goals. We should strive to provide highly flexible and informed management and allocation of resources, including the ability to provide for the isolation of software and hardware components for programmability, correctness and performance.

We should create architectural designs that attain composable performance and power in a highly multiprogrammed environment. We need to be able to sandbox programs from other running programs --- the programmer should continue to be able to code as if her program is the only one running on the hardware. This also involves isolating communication between threads in the same program and across programs. Architectural support is critical to these goals; it increases the security of the system and enables the enforcement of quality of service even on commodity systems.

We need to rethink virtual memory and protection in the era of ubiquitous concurrent systems. Moreover, systems software and runtime system need significantly-enhanced communication with hardware and software monitors, managing the global environment in a heterogeneous system, for potentially diverse performance, reliability, and energy goals.

4.2.3 *Why Is This Research Transformative*

Society and large segments of our national economy have come to depend on substantial, continuous increase in performance with each microprocessor generation. We have managed to maintain this scaling for nearly four decades, doubling performance every 2 years. By significantly lowering the barrier to effective and productive parallel programming, this research will enable the rapid generation of new parallel software. This will allow computing to continue on the performance scaling path by harnessing available hardware parallelism.

4.2.4 *What Are the Deliverables*

The deliverables are architectural concepts and programming models to attain highly-programmable, high-performance, and correct 1000-core chips.

4.2.5 *Research Disciplines Involved*

Addressing the parallel programming problem is a multi-disciplinary challenge. It requires significant advances in computer architecture as well as other fields, including programming languages, compilers and operating systems. This is a unique opportunity for tight collaboration across several computer science disciplines. Computer architects will bring research innovation in execution cores, memory hierarchies and communication fabrics. Programming language research will be instrumental in developing the programming models. Compiler research will be necessary

to map the language features to the novel hardware mechanisms developed. Finally, operating system research will need to keep up with the scale of hardware, as well as virtualize and manage the new hardware resources developed to support better programmability.

4.2.6 *Risk to Industry and Society if Not Pursued*

Users in education, science, government, industry and commerce have come to depend on the repeated doubling of computer performance and cost-performance. This doubling will dramatically slow for sequential programmers. No longer can researchers or practitioners dream up a new idea whose sequential performance is too slow or costly and know that in four years technologists and architects will quadruple sequential performance. This will remove an important decades-old "yellow-brick road" for innovative ideas to improve society.

4.2.7 *Benefits of Success to Industry and Society*

The "yellow-brick road" to better performance and cost-performance can be extended by effective programming of parallel architectures. If successful, researchers and practioners can dream up ideas that are too slow or expensive today as long as they can express their solution to run on parallel architectures. While it is hard to predict the exact inventions to come, the past suggests that they may come at an increasingly rapid rate.

4.2.8 *Why Not Let Industry Do It*

The challenge of developing new models for parallel architectures requires a scientific examination of possibilities that change the landscape beyond the purview of any one company, and with the stops and starts that make the work pre-competitive. A chip company does not control software; a software company does not control the chips; a service company does not sell to most consumers; a search company seeks commodity hardware; and all successful companies are limited by backward compatibility. Consequently, it is very unlikely that industry alone will develop the new models required.

4.2.9 *Likelihood of Success*

The likelihood of success varies with computing platform and application domain. Parallel architectures for servers, cloud, and data center will likely succeed and scale. This is because they operate on vast parallel work, even as there are many challenges to maximize effectiveness and minimize costs. Parallel architectures for client and embedded devices will likely succeed at the small scale (e.g., ten cores) because of the data parallelism in some important applications. Success for client and embedded computers with large core counts will be challenging due to the very constrained energy requirements and the open questions about the nature of parallelism in their future workloads.

In practice, however, transformative technology typically comes from unexpected sources. By putting parallel programming in the hands of the masses, rather than an elite corps of highly-trained programmers (our current state), we will likely significantly accelerate the path to the next big transforming applications.

4.3 Hardware-Software Co-Design and Asymmetry

4.3.1 *Problem Statement and Goals*

Specialization of the hardware and system-software layers eliminates inefficiencies and overheads that come with the flexibility of general-purpose systems. It is possible to obtain orders of magnitude improvements in performance, performance per Watt, and performance per dollar. However, specialized hardware has previously been too difficult, too slow, and too expensive to develop, even when the hardware substrate itself is reconfigurable. Truly custom hardware such as ASICs also entails very high manufacturing costs. These obstacles have prevented the use of specialization even for applications of high strategic value, in cases where the software or algorithms are still evolving, as is the case with many important application categories, such as data mining, feature detection,

bioinformatics, and modeling and simulation. As a result, specialization has usually been limited to a small number of high-volume consumer applications.

Our goal is the development of the technologies necessary to deliver "turn-key" specialized computing systems quickly and economically. Specifically, we want to design specialized architectures that deliver up to 10,000x speed-up for particular applications for less than $10,000. The manufacturing costs should be no greater than they are for conventional commodity computing systems. Further, the methodology should scale across form factors and enable specialized systems ranging from handhelds to datacenters. From the user perspective, the specialized system should integrate seamlessly into existing software systems so that programmers can leverage the specialized hardware with minimal engineering overhead. The ultimate goal is a fully automated generation of application-specific hardware for each program.

4.3.2 *Research Thrust Description*

To attain our goals, we propose to (1) leverage existing reconfigurable hardware and programmable accelerators, (2) collaborate with Electronic Design Automation (EDA) researchers, (3) identify of the correct software abstractions, and (4) focus on important real-world applications.

4.3.2.1 Leverage Existing Reconfigurable Hardware and Programmable Accelerators

To date, architects have devised accelerators for many domains ranging from signal processing to biological simulation to computer graphics to climate modeling. These accelerators are generally purpose-built and require extensive, custom software support, but they also embody solutions to many problems that turn-key accelerators must face. These include, for instance, approaches to optimizing particular memory access patterns and methods for exploiting parallelism at fine and coarse granularities. In addition, many architectures include reconfigurable components that can provide large performance gains across many applications --- although smaller than full-custom accelerators.

Turn-key accelerators must integrate and extend these approaches where appropriate. Providing a widely applicable turn-key accelerator synthesis capability will also require, however, a careful rethinking of the interfaces that existing accelerators provide. This is so that they can work “fist in glove” with the CAD tools, software, and other components required to seamlessly build and integrate turn-key accelerators into existing computer systems.

4.3.2.2 Collaboration with Electronic Design Automation (EDA) Researchers

Many toolchains already exist for automatically generating hardware description language (HDL) implementations of a given function or code segment. However, significant challenges remain if we are to achieve our goal of turn-key custom processing.

First, most existing toolchains provide an architectural “schema” for the accelerators they generate. If the computation at hand is not a good fit for that schema, the resulting architecture may be suboptimal. This problem is especially difficult if the targeted computation is not a good candidate for conventional accelerator architectures.

Second, automatically integrating the hardware implementation into an existing system can be challenging, especially if the accelerator requires low-latency communication with other system components (e.g., the processor or shared memory). In this case, it is difficult to ensure that the design “meets timing” without introducing excessive delays.

Finally, to achieve turn-key customization, designs will either have to be “correct by construction,” or the toolchain will have to thorough test benchmarks automatically (which, themselves must be correct).

Meeting these goals will require new tools and new approaches to well-known and long-standing problems in EDA. Solving these problems in the context of turn-key optimization will require an orchestrated effort between architects and tool builders. It may be necessary to craft architectures that are amenable to practical EDA techniques.

4.3.2.3 Identification of the Correct Software Abstractions

For turn-key optimizers to be useful and provide maximum performance, they must satisfy two conflicting needs. First, it must be easy for programmers to use and to integrate into existing systems. Since turn-key optimizers, by definition, target existing code or an existing problem, system designers are tightly constrained in the software-level interface the accelerator must ultimately present. Second, maximizing flexibility in accelerator design (and therefore performance gains), will require decoupling the architecture's interface from the software-level interface.

The key to resolving these conflicting requirements lies in the compiler/runtime and the specification of both the baseline, unoptimized system and the optimizer itself. The compiler or runtime must be able to bridge the gap between the interface that the software requires and the interface that an efficient accelerator can present. For instance, a compiler might perform function inlining or outlining, identify parallelism, eliminate data sharing through coherent memory, or refactor key loops.

Furthermore, the compiler must be robust for any turn-key accelerator that the tool chain produces. This will require both remarkable flexibility and reliability. Recent work on provably correct optimizations may be of use in achieving this goal. As with the EDA challenges that turn-key accelerators present, close collaboration between compiler designers and architects will aid in providing tractable solutions.

4.3.2.4 Focus on Important Real-World Applications

While there are many basic research questions that are motivated by specialization, it is vital that the research be pursued in the context of strategically important real-world applications. Refining the design of turn-key accelerators will take some time, and will require close attention to the many concerns that arise in production computer systems. Focusing on the needs of these "real world" users will provide a positive feedback loop.

As researchers deploy a sequence of prototype turn-key accelerators across a range of domains, they will necessarily refine the architecture, tool chain, and software interface that the accelerators provide. The ultimate goal of the turn-key research program is to amortize the development cost and effort required to build these initial accelerator across a large number of future accelerators across a wide range of domains. If the initial accelerator targets are not representative of real-world applications, then this will not be possible.

4.3.3 *Why is this Research Transformative?*

Hardware specialization would allow orders of magnitude improvements in performance, performance per Watt, and performance per dollar. This has been a goal for decades. What has changed in recent years in the advent of multi-core and heterogeneous systems, including FPGAs with sufficient resources to implement a multitude of complex functions. Another important change in the landscape is the increasing severity of the power and memory walls, coupled with the likely end of Moore's Law. Straightforward replication of conventional general-purpose cores simply will not be sustainable. Specialization is necessary in any case to cope with these limitations, and fits naturally in the emerging ecosystem of solutions to these limits.

Reducing the design costs associated with specialization, and eventually achieving turn-key specialization, will bring the order-of-magnitude benefits of specialization to increasing numbers of applications. Automating specialization also allows it to be employed before software becomes stable, allowing the hardware to evolve with each stage of the software's development.

4.3.4 *What Are the Deliverables*

In the first three years, we will perform a full analysis of several strategically-important applications or application domains running on general-purpose machines, identifying their bottlenecks and major inefficiency sources. We will also manually design specialized systems to run each of these applications more efficiently. Within five years, we will propose an approach to build a custom supercomputer-in-a-rack. Within seven years, we will perform full analysis of additional applications or application domains, and complete the design and implementation of one or

more applications targetting the custom supercomputer-in-a-rack. After that, we would like to build a prototype of a multi-application customizable cloud supercomputer.

4.3.5 *Research Disciplines Involved*

Building robust, easily-designed, specialized systems requires coordinated efforts across a wide range of disciplines. The first discipline is applications. We need input from experts on the initial set of target applications. The second discipline is software engineering. We need to define the crisp boundaries to cleanly separate the accelerated portions of the code from unaccelerated portions. In addition, we must define the right software-level interfaces for communicating with the specialized hardware. The third discipline is system architecture. We need advances in hardware interfaces to integrate specialized processors into the larger system in an automated way. We need to specify a set of common interfaces that building block components can implement to facilitate easy integration. The fourth discipline is processor architecture. The architecture and microarchitecture of the specialized processors will influence their ultimate performance and efficiency. Ideally, a set commonly applicable design practices and primitives will emerge. The final discipline is automated design and verification. We must develop and refine tool flows that make it easy to construct specialized circuits for particular applications. Full automation is desirable, but there is a trade-off between the level of automation and the breadth of applicability.

4.3.6 *Risk to Industry and Society if Not Pursued*

The country (or countries) that masters designing high-efficiency, heterogeneous platforms in a fully automated fashion with rapid problem-to-system turn-around time has a high potential of being the new world leader in this area (the tools and hardware will drive high value exports). This country also has a high potential of being a leader in cloud computing services as well, as it would be able to provide higher efficiency at a significantly lower cost.

In addition, the past scaling of processor performance has driven advances in an enormous range of disciplines both within computing and in society at large. Although performance will continue to improve, continued scaling of performance at historic rates is in grave doubt as physical limitations become more severe. As a result, although scientists, engineers, and corporations will continue to find and develop new ideas using computer systems, they will be less able to quickly reduce those ideas to practice and less able to solve bigger and larger problems. The long term consequence will be a slowing of innovation and growth in computational sciences and compute-intensive business sectors.

4.3.7 *Benefits of Success to Industry and Society*

The benefits to society will be widespread and direct. Turn-key development and deployment of specialized, high-performance processors will enable corporations, researchers, and governments to quickly and affordably focus enormous computing power on critical problems in short order. Example applications include: climate modeling, drug discovery, advanced (real-time) image recognition, simulation of biological processes, simulation of a new products/prototypes, analyzing large social (and other) networks, and implementing/accelerating new business intelligence algorithms. The benefits to the computing industry in particular will be even larger.

4.3.8 *Why not Let industry Do It?*

The proposed vision is too high-risk for businesses to attempt. It requires coordinated efforts across many different disciplines and aims to develop a capability that might be deployed in 10-15 years. This is far beyond the planning horizon for most companies.

4.3.9 *Likelihood of Success*

The proposed vision is very ambitious and will require basic advances across many areas in order to be successful. As a result, the vision will drive valuable research even if the entire vision is not fully realized. Given the extremely aggressive goals set by the vision, complete success is by no means assured. However, even if we missed our target by an order of magnitude in cost, time-to-design, and performance, the project would still provide great benefits.

4.4 Domain Specific Languages

4.4.1 *Problem Statement and Goals*

The new era of explicit and heterogeneous parallelism presents a tremendous challenge to software development. Current parallel programming models are hardware-centric and require a different programming model for each flavor of parallel architecture (e.g., threads, locks, vectors, streaming data parallel, message passing, or specialized functions). This current state of affairs makes programming difficult and non-portable from one parallel architecture to the next. For explicit parallelism to be successful, we need programming models that allow the application developer to use a high level of abstraction, while still achieving very high parallel performance. To attain this goal, we examine Domain Specific Languages (DSLs) and the new parallel architectures that they will enable.

A DSL is a concise programming language with a syntax that is designed to naturally express the semantics of a narrow problem domain. DSLs can be used to improve the productivity of application developers and the efficiency and performance of the applications. This is because the DSL implementation can take advantage of high-level domain-specific optimizations that are inaccessible to general-purpose compilers for general-purpose software written at a lower level of abstraction. In particular, a high-level DSL compiler can exploit the semantics of the domain to compile efficient code for a given type of parallel machine or even generate specialized hardware. Example DSLs are languages (e.g., SQL, Matlab, Latex, Verilog, SPICE, make and sh) or libraries (e.g., OpenGL, DirectX, and Grand Central Dispatch) designed to express a particular class of applications or application components efficiently. In the context of parallel computing, a DSL provides a set of semantics that are expressive and natural for developers of a class of applications, yet describe the computation at a sufficiently high enough level that it is easy to exploit parallelism from the resulting code.

DSLs such as SQL, Matlab, make and OpenGL/DirectX are provably successful at enabling the exploitation of parallelism. SQL is used for database processing. SQL queries describe how to interface to a database in a way abstract enough to enable the optimization and parallelization of the query, and to substantially increase the productivity of application writers. Matlab is used by scientists and engineers to perform mathematical transformations involving matrices and linear algebra. It has matrix processing primitives that expose vast amounts of data parallelism. This has enabled companies to provide automatic parallelization tools for Matlab codes. Make is a simple language to describe process flows and is typically used in building systems. Pmake is a way of extracting very coarse grain parallelism from the dependencies in a make file. Finally, OpenGL and DirectX have traditionally been used for graphics rendering, although they are now growing into a general-purpose streaming computation language. Initially, both of these libraries provided only limited, but standardized, access to hardware for graphics rendering. Currently, many scientific problems are recast into the specialized type of data parallelism available in these DSLs and run on Graphics Processing Units (GPUs).

In this research thrust, our goal is to attain 5-10 commercially available DSLs in 10-15 years. These DSLs should have widespread commercial availability, and be used in industrial, academic and scientific settings. They should be "as successful as SQL" for the particular application domain covered. Another goal is the creation of DSL infrastructure that makes new DSLs easier to develop, and enables an optimization framework with compilation and mapping techniques to be used by multiple DSLs.

4.4.2 *Research Thrust Description*

DSL research requires an interdisciplinary team of experts that includes application domain experts, language designers, compiler researchers, and architects. We focus on the components of this research where computer architects are likely to be involved. We have identified the following research vectors: (1) identification of DSL application domains, (2) a DSL infrastructure for creating DSLs, (3) generation of prototype DSLs, and (4) push-button DSL acceleration. We now detail them.

4.4.2.1 Identification of DSL Application Domains

While existing successful DSLs cover databases, matrix-based programming, and graphics programming (extending into stream-based processing), there remains a wealth of significant application domains without DSLs to enable the

exploitation of parallelism. Among these domains are machine learning, speech recognition, graph-based computations and image and media processing.

We should perform research to characterize these domains and identify the characteristics of the DLSs that will be successful there. We can learn the elements of a successful DSL by examining past successes. For example, SQL is high-level enough to permit database optimization irrespective of the application supported. Implementers can write solutions that scale from the simplistic (e.g., access to a text file) to the most sophisticated (e.g., systems for airline reservations or stock markets). Moreover, SQL has enabled new storage and I/O architectures, which is largely where the bottlenecks lay for this domain.

Matlab is easy to use for matrix-based programming. It is forgiving of bad software engineering practices, enables incremental development, and provides a rich set of primitives well suited for its users.

Finally, OpenGL/DirectX has provided a standard method of accessing a 3D graphics accelerator. Critical to the success of this DSL has been the co-evolution of graphics architectures, which provide orders of magnitude performance gains compared to writing rendering algorithms in other languages that utilize the main CPU only. Hardware and DSL are continuing to evolve to expand support for these application classes.

From this list of successful DSLs, we can distill a few common features. One is that a successful DSL supports the programmer community it targets; it is much easier to use the DSL than not for that application domain. Another is that a DSL enables scalable parallelism if its semantics are crafted so that the parallelism is hidden from the developer. Finally, a DSL is particularly effective if it co-evolves with accelerator hardware that solves an important need.

4.4.2.2 A DSL Infrastructure for Creating DSLs

The history of successful DSLs has shown that success did not come with the first version of the DSL. Consequently, progress will be stymied if every version and every new DSL requires a significant investment of infrastructure resources. In addition, the actual semantics of the resulting DSLs are typically prone to subtle errors. Therefore, we should create a single meta-DSL embedding language with well defined semantics, which enables experimentation in both application creation and hardware/software architecture design and optimization. Such a DSL requires a supporting infrastructure, including compiler, runtime, and front-end development tools.

We hope that after some years of research and development, there can be widely-available open source DSL infrastructure and associated tools. This transfer of technology from research to product will foster further widespread DSL use. We would expect one or two research efforts to develop DSL infrastructure consisting of DSL embedding languages rooted in general-purpose languages (e.g., C++ or Scala), and an associated optimization framework for exploiting domain knowledge to generate optimized mappings to existing or new architectures. Other researchers involved in DSL development and architecture research would make use of these DSL infrastructures to create new DSLs and new architectures based on these DSLs.

4.4.2.3 Generation of Prototype DSLs

We should create 3-5 separate efforts to prototype new DSLs. These efforts should be carried out in parallel by different research groups. They should lead to the creation of a handful of new DSLs within five years. These DSLs will be a mixture of new application domains and rethinking of earlier work and domains. Each group should form an interdisciplinary team of experts. The result should be the DSL itself and a collection of candidate architectures that support them. Some of these architectures will be traditional multi-cores with small changes, while others will be GPUs, or hybrids of what we build today, or completely new architectures. Such DSLs and hardware architectures need to prove themselves useful by providing 10–100x efficiency gains. As with past DSL languages, fewer architectures than DSLs will be successful. A small number of architecture designs with sufficient generality, when implemented together with DSLs, may prove to be successful.

4.4.2.4 Push-Button DSL Acceleration

After sufficient expertise is available on a DSL, its use, and its architecture acceleration, an intriguing research direction is to automatically create and specialize candidate architectures. This research involves collecting several

sample applications, understanding what trade-offs are available in the architecture, and creating a hardware-software framework for automatic generation of a customized architecture. Automatic customization will enable lower power and less expensive deployment of hardware resources.

4.4.3 *Why is this Research Transformative*

Despite the availability of multi-core CPUs for several years, and parallel machines for several decades, commodity software makes tepid use of the available parallel resources. The creation of several new DSLs that unlock the ability to write software for a wide collection of domains has the potential to transform the industry and scientific landscapes. A properly designed DSL enables a relatively unsophisticated software engineer to write complex applications where parallelism is much more easily exploited than general-purpose languages.

4.4.4 *What Are the Deliverables*

The deliverables include both research goals, achievable through widespread community effort, and commercial goals, achieved through success in the research domain and technology transfer. Specifically, during years 1-5, our deliverables are 3-5 prototype DSL projects and 1-3 DSL infrastructure projects for creating DSLs. During years 6-10, our deliverables are 10-15 prototype DSLs, push-button DSL acceleration, and commercially-available or widely-distributed open-source DSL-creation tools. Finally, during years 11-15, the deliverables are 5-10 commercially-available DSL languages.

4.4.5 *Research Disciplines Involved*

Research into DSLs requires a vertically-integrated approach. Only through a close collaboration between application domain experts, language and systems researchers, and architects will useful DSLs evolve. Architects will play a key role in DSL research because they have traditionally focused on the design of the hardware-software interface.

4.4.6 *Risk to Industry and Society If Not Pursued*

Commodity applications have had only limited success in exploiting the threads and locks programming model. DSLs offer a promising path forward for applications to exploit parallel resources. The risks to industry and society if research into DSLs is not carried out are the continued inability of mainstream software to easily exploit mainstream hardware.

4.4.7 *Benefits of Success to Industry and Society*

The success of this research program will have significant impact in both industrial and scientific areas. On the scientific side, the successful history of DSLs such as Matlab and OpenGL (with accompanying high-performance GPUs) suggests that newly crafted DSLs will unlock new scientific discoveries as programmers are able to explore ever more complex problems. On the industrial side, having an established set of DSLs and hardware accelerators for popular application areas will enable new products with scalable performance and improved capabilities. This will also foster more parallel hardware innovation because it will be possible to use the DSL infrastructure to create optimized mappings from the existing DSL software to new parallel hardware.

4.4.8 *Why Not Let Industry Do It*

Academic research into DSLs has an important role that is both distinct and in some respects advantaged compared to industry. Academics are in a unique position to collaborate with application scientists to learn about their application domains and work in tight collaboration with them on DSL creation. Industry is generally more successful at creating DSLs for industrial purposes (e.g., game development or database interfacing). Moreover, as with all academic research efforts, the horizon can be longer. There is no need for hardware to be ready for sale in the first few years of language development, where new ideas are being explored with software developers. In other words, DSL research requires a longer horizon for development than industry can typically take.

4.4.9 *Likelihood of Success*

The history of successful DSLs suggests that a concerted effort to create new DSLs for as-yet unsupported application domains may be successful. However, the fact that these DSLs do not yet exist or, more worrisome, that some exist but do not expose scalable performance (e.g., php/ruby for web processing, perl for text file processing, or python for scripting automation tasks) provides a note of caution.

5 Educational Perspective

The professionals who will develop the future parallel hardware and software systems need to be properly trained. This requires that our universities offer a strong educational program in parallel systems. In this section, we outline our vision in this area, the challenges that we face to augment the undergraduate curriculum with parallelism, the approaches we might take, and our recommendations.

5.1 Vision

It has been true for a long time that computer hardware is an example of parallel implementation, although educators have not always emphasized this. On the other hand, most of the software written has been sequential software. Moving forward, we expect that much of the programming in the future will involve some form of parallel programming. Writing such software will require parallel thinking. Consequently, to prepare graduates of computer science and engineering, we need to ensure that all of the core concepts of parallel thinking are included in the required curriculum paths, and that programming assignments include substantial parallel programming.

To accomplish this, we need to integrate parallelism concepts across the entire curriculum, starting in the freshman year. Further, it would be helpful to encourage high schools to teach some very basic concepts of parallelism in their computer science curricula, so as students are aware of parallelism before reaching the freshman year.

5.2 The Challenges to Making this Happen

There are several obstacles to integrating parallelism into the preparation of computing professionals. The first one is that there is no universal agreement among educators as to what parallel concepts and techniques should be taught and when they should be taught. For example, should we start in the freshman course? If so, how do we need to change the freshman course to enable the concepts taught to be meaningful?

A second obstacle is that the pace of curriculum change in the university is very slow. Part of this is due to the need to build consensus. Part is unfortunately due to the inherent human resistance to change. Some faculty do not understand the value of teaching parallel thinking. Others have no experience doing so.

Finally, a third obstacle is that there is a lack of tools available to teach parallel programming. For example, tools for software debugging are still very primitive.

5.3 Approaches

There are opportunities to teach parallelism throughout the curriculum, from the introduction of fundamentals in early courses such as threads, locks, communication, and streaming, to the exploitation of parallelism in later courses such as data structures, algorithms, and theory.

With respect to fundamentals, core principles appear to be best explained within the context of a simple, low-level model, where the student uses a simple ISA augmented with fork/join and synchronization via spin locks. At the data structures level, one can easily introduce data structures that can handle parallel access. At the algorithm level, one can emphasize the inherent parallelism of some algorithms and the sequentiality of others. In theory courses, one can introduce models such as PRAM, which can be used to show the intrinsic value of parallel algorithms.

Educators should learn from approaches to add parallelism to the undergraduate curriculum that have already been tried successfully, or that can be tried with little danger of being disruptive to the entire curriculum. For example, computer architecture and operating systems are courses that already have parallelism built-in, wherein it may

simply be a matter of providing additional emphasis. In addition, a senior elective in parallel programming provides an opportunity for students to embrace parallelism in large software systems --- and, since the course comes late in the curriculum, it does not cause ripples in the pre-requisite structure. Finally, a parallelism course for non-majors with an emphasis on applications is a popular option, and has the potential to impact a large group of students.

5.4 Recommendations

We suggest several steps to making parallelism an integral part of the undergraduate curriculum.

First, we suggest making low-resistance changes to the curriculum to embed parallel thinking. These changes include (1) augmenting existing courses with simple additions where they make sense, (2) working with theory faculty so that they teach parallel algorithms, (3) adding senior electives as a function of individual faculty interests, and (4) teaching a non-majors course for those with applications amenable to parallel implementation.

Second, we should survey the university landscape for approaches that have been used to teach parallelism, looking for what has worked, and subsequently aggregating that information into an understanding of what is done now.

Finally, we should work with CRA, NSF, and other funding agencies to impart urgency in the need to teach parallel practices throughout the curriculum. This includes getting endorsement on the value of including parallelism already in the first courses.

6 Industry Collaboration

Traditionally, the work of academic computer architecture researchers has influenced the computer industry and vice-versa. As we reach what has been called the “parallelism crisis”, many business models in industry are at risk. Consequently, it is now especially critical to maintain a strong and productive relationship between industry and academic computer architecture researchers. It is now a key time to maintain a broad discussion between industry and academia on programming models, programming languages, programmer productivity, applications, and the computer architectures to support them all. In this section, we provide some thoughts on how to make this relationship more robust.

6.1 How Academics Can Contribute

Academics need to take a proactive approach and seek to influence industry in areas such as architecture, emerging programming models, emerging languages, emerging applications, software productivity issues, and key data center issues. There are several ways to accomplish this. One is to invite industry participants to panels, workshops, and other discussion forums on parallel computing that academics organize. Another way is for academics to prepare weak-long courses on parallel programming and computing for computer practitioners with little experience in parallelism. These courses can be taught at an academic institution, at an industrial site, or remotely over the internet. Finally, it is also possible to organize a few academics into a team that tours industrial sites, giving short talks to educate computer professionals about upcoming architectural challenges or trends.

6.2 How Industry Can Contribute

Industry can provide key input to academics. First and foremost, it can provide direction and input into the critical problems it faces in the areas mentioned; the academics can then orient their work appropriately. Industry can also provide access to experimental data, traces, or large data centers. It can help open avenues for technology transfer, by taking ideas and designs from academia. Finally, it can provide funding for academic research, given that many companies’ futures ride heavily on solving the problem of parallelism. Such funding and interaction could be modeled on the two Intel-Microsoft UPCRC centers, but it should include a much broader audience, with many more universities.

One short-term step toward this broad discussion forum between industry and academia that we seek could be industry tours. These would be a few key people from industry traveling together to several schools to discuss important issues on parallelism and programming models. These visits could be instrumental to educating faculty

about the urgent need to make progress in the area of parallel computing, and could have a catalyzing effect on the revamping of the course curricula to include concepts on parallelism.

7 The Funding Landscape for Computer Architecture Research

As part of the ACAR workshop, we held a panel in which Program Directors from NSF and DARPA discussed funding prospects and approaches with the workshop participants. The panel was asked for feedback on the workshop and on the current and near-future funding landscape.

Both NSF and DARPA agreed that the research topics documented in this report were appropriate for funded programs. However, there is a significantly different approach to programs between NSF and DARPA. NSF does not often start new programs, but funds ongoing, dynamically changing research within its core programs. This stems from NSF's charter to fund basic research rather than mission-oriented research. DARPA strategy, on the other hand, is centered on new game-changing ideas through the institution of new programs.

Overall, there was a consensus that funding for computer architecture research should be organized along larger, more ambitious projects that cover multiple layers of the computing stack. It should also involve a concerted, complementary effort by several funding agencies, focusing on the research thrusts identified here.

7.1 National Science Foundation (NSF)

Current computer architecture research is primarily supported in two divisions within the CISE Directorate, namely Computing and Communication Foundations (CCF) and Computer and Network Systems (CNS). CCF support for computer architecture resides in two clusters, namely Software and Hardware Foundations (SHF), which supports VLSI, design automation, hardware architectures, compilers, programming languages, software, HPC, and emerging architectures, and Algorithm Foundations (AF), which supports aspects of algorithms research. CNS maintains two core programs, namely the Computer Systems Research (CSR) and the Networking Technology and Systems (NeTS) programs. The former supports computer architecture research. Funding for computer architecture research increased over the last two years, primarily in the multi-core area and will continue at least at the current level with possibly moderate growth.

NSF has crosscutting programs supporting collaboration among several directorates. Examples include the Expeditions in Computing, Cyber-Enabled Discovery and Innovation, and the Cyber-Physical Systems programs. NSF also supports special programs that have been initiated outside of core, such as the High-End Computing University Research Activity supporting middleware research, Multi-core Chip Design and Architecture, as well as the Engineering Centers and the NSF Supercomputing Centers.

Two themes that run through these programs are captured by the Science and Engineering beyond Moore's Law (SEBML) and the Climate Research Initiative thrusts. SEMBL is an effort among four Directorates (CISE, Math and Physical Sciences, Engineering, and the Office of Cyber-infrastructure) to coordinate investment in the following areas: architectures, 3D and optical interconnects, reliability with unreliable components, power and energy considerations, abstract models, programming languages and software environments, algorithms, multi-core, and pervasive, distributed and mobile computing. SEBML supports research on multi-core as well as on non-silicon substrates such as quantum, bio, and nano. SEBML is not run as a separate program; it is funded out of existing areas. SEBML is already receiving significant funding increases.

Multi-core research is a well-recognized topic within the core of the CCF and CNS divisions, and is the subject of a large fraction of the proposals entertained at present. Rather than creating a new program for multi-core architectures, this research need is represented by the SEBML thrust.

The computer architecture research community can assist NSF by continuing to identify the key challenges in the area. It should take the time to present a clearer picture of the landscape --- what approaches may work and what are the hard challenges. One important point is that we should engage in research across the technology layers. Computer architecture research cannot be isolated from the higher-level layers such as system software and applications.

Moreover, while the computer architecture research community is very good at small research ideas and projects, it needs to think of ambitious research problems and strive for large projects --- Research Expedition-scale problems. Moreover, it needs to reach out to other disciplines for collaborative research.

At all times, the research community should identify fundamental research themes critical to parallel computer architecture and tie them to questions of national importance. In light of the interest in supporting cross-agency collaborative programs, it could make sense to explore the proposal of large inter-agency programs in this area.

7.2 Defense Advanced Research Projects Administration (DARPA)

DARPA is currently organized into seven offices. The most relevant ones for computer architecture research are the Information Processing Techniques Office (IPTO), which supports research, development, and prototyping that spans the information lifecycle of sense, process, understand, and apply, and the Transformational Convergence Technology Office (TCTO), which advances new crosscutting capabilities derived from a broad range of emerging technological and social trends, particularly in areas related to computing and computing-reliant subareas of the life sciences, social sciences, manufacturing, and commerce.

DARPA is structured around proposed projects supported by program managers. To work with DARPA it is important to become familiar with the challenges and opportunities of National Security. The way to obtain funding from DARPA involves putting novel ideas in a white paper and approach a program manager. The ideas need to be bold and risky. An unofficial set of questions that one should ask oneself as guidance for the items to be addressed in a proposal are known as Heilmeier's Catechism. They include: (1) What are you trying to do? (2) How is it done today? (3) What is new in your approach and why do you think it will be successful? (4) If you're successful, what difference will it make? (5) What are the risks and the payoffs? (6) How much will it cost? (7) How long will it take? and (8) What are the checks for determining success?

DARPA funding opportunities appear in Requests for Proposals (RFPs) and Broad Agency Announcement (BAAs) solicitations at its website.

8 Next Steps

Workshop participants identified four major research areas in parallel computer architecture that need investment, namely, (1) data centers and large-scale systems, (2) architectures to enhance programmability, (3) hardware-software co-design and asymmetry, and (4) domain specific languages. We recommend that each of these areas be the nucleus of research programs, or at least be consciously incorporated as elements of mission-oriented programs. The next steps involve working with our professional colleagues to publicize this report among funding agencies, industry, academic circles, and the broad computer science and engineering community.

We hope to reignite a sense of excitement for the entire computer architecture research community. We also hope to help blossom the abundant talent that exists among the many young faculty members who are now actively doing research in computer architecture --- many of them are overflowing with new ideas and plans. Finally, we hope to help the funding agencies steer the funding for computer architecture research to the most promising areas.

9 Acknowledgments

The organizers thank the CRA and the CCC for providing advice, direction and funding to make this workshop and report possible. In particular, we thank Bill Feiereisen, Andrew Bernat, and Erwin Gianchandani. Thanks also goes to the Steering Committee of this workshop, all of the workshop participants, and all of the researchers that sent in a position paper.

10 Appendix A: Call for Position Papers

Call for Position Papers
Advancing Computer Architecture Research
Computing Community Consortium (CCC)
http://www.cra.org/ccc/acar.php

Overview
Discontinuity-inducing trends such as the arrival of multi/many-cores, the reduced reliability of semiconductors, and the ever-presence of power constraints, are transforming the field of computer architecture. In particular, the ubiquity of multi-cores and the fact that much of the IT industry is relying on main-streaming parallel processing for survival is a truly seismic event. Momentous changes are about to happen in all domains, including portable clients, home and business computing, and datacenter/petascale computing. Multi/many-cores will have to evolve to enable and support high-productivity parallel software development and execution. At the same time, there remains a huge gap between the theoretical limits of instruction-level parallelism (ILP) and what processors actually attain. One has to wonder about the sequential execution model, is this really as good as it gets? While it may appear that way, novel robust techniques that effectively push ILP further may yet be invented. In this environment, we ask ourselves:

-What will be the computing platforms in 2020-2025?
-What are the major research challenges that must be overcome to create these platforms?
-What will be the impacts to and from the broader society at large?

To answer these questions, it is appropriate to organize a sequence of workshops that, building on the 2005 CRA workshop on Revitalizing Computer Architecture Research, focus on what role computer architecture research plays going forward. The goals of these workshops are:

- Clearly articulate an agenda and roadmap for computer architecture research. Such an agenda must be broadly endorsed by the research and industrial communities as well as be an effective vehicle for communicating to technical and non-technical national leaders.
- Create excitement and community building for computer architecture research and form lasting research partnerships between multiple computer architecture researchers.
- Unlock the potential of the many junior researchers in our community and ensure the continuous leadership of our nation in this area.
- Suggest how to structure funding and research programs in a way that is commensurate with computer architecture's central role in computer science, the IT industry, and the US economy.

Failure is not an Option: Popular Parallel Programming
This Call for Position Papers is for the first of two workshops and focuses on **Popular Parallel Programming**; the second workshop will focus on Extending the Current Sequential Programming Model. For this first workshop, members of the computer architecture community are invited to submit a 1-page position paper outlining their thoughts on the following Questions:

- How can computer architecture help enable ubiquitous parallel software development?
- How does the architecture most-effectively interact with the different layers of the software stack in parallel systems?
- What are the key parallel programming models requiring support; how to support multimodal parallelism?
- What is the role of re-configurability and heterogeneity in parallel systems: GPUs and other special-purpose parallel systems versus general-purpose parallel systems?
- How to effectively support continuous run-time optimization in parallel systems?
- What is the proper role of academic and state-sponsored research in parallel systems?
- How and whether to support parallel system building and prototyping efforts?

Potential contributors are encouraged to be brief, and keep the following in mind:

- The position paper should not be about what you are working on currently; it should be about a vision for parallel computing platforms available 10-15 years from now.
- Focus the paper on one of the following three areas: (1) Portable clients, (2) Home and business computing, and (3) Datacenter and peta-scale. Do not try and cover all three in one brief position statement.
- Keep the challenges of parallel computing central.
- The position paper should include: (1) Name, position and organization; (2) Area of focus among the three, and (3) Answer to the workshop Questions.
- The steering committee will select the workshop invitees based on the responses. The committee is looking for a wide range of insightful views.

Submit a 1-page PDF file to acar@cs.uiuc.edu by 6pm CST, Monday November 30, 2009.

Workshop Format, Dates and Location

The workshop will focus on the topic of Popular Parallel Programming, allowing the participants ample time for discussion. Attendees will be both academic researchers and representatives of industry and funding agencies. The workshop will be February 22-23, 2010 in San Diego, CA.

Organizing Committee

Organizers:
Josep Torrellas (Univ. of Illinois) and Mark Oskin (Univ. of Washingon).
Steering Committee:
Chita Das (NSF), William Harrod (DARPA), Mark Hill (Univ. of Wisconsin), James Larus (Microsoft), Margaret Martonosi (Princeton), Jose Moreira (IBM), Kunle Olukotun (Stanford), Mark Oskin (Univ. of Washington) and Josep Torrellas (Univ. of Illinois).

11 Appendix B: Workshop Attendees

Almadena Chtchelkanova, NSF
Anand Sivasubramaniam, Pennsylvania State University
Bill Feiereisen, CRA
Chita Das, Pennsylvania State University and NSF
Christos Kozyrakis, Stanford University
Dean Tullsen, University of California, San Diego
George Almasi, IBM Research
James Larus, Microsoft Research
Jon Hiller, ST Associates
Josep Torrellas, University of Illinois
Karin Strauss, Microsoft Research
Kevin Skadron, University of Virginia
Krishna Kant, NSF
Kunle Olukotun, Stanford University
Luis Ceze, University of Washington
Mark Hill, University of Wisconsin
Mark Oskin, University of Washington
Onur Mutlu, Carnegie Mellon University
Richard Murphy, Sandia National Laboratory
Sampath Kannan, NSF
Sarita Adve, University of Illinois
Satish Narayanasamy, University of Michigan
Steven Swanson, University of California, San Diego
William Harrod, DARPA
Yale Patt, University of Texas

12 Appendix C: Workshop Schedule

Schedule of the ACAR Workshop

Sunday, February 21st

6:00pm : *Dinner*

- Welcome, goals and charter.
- Keynote #1: George Almasi (IBM Research): "What were they thinking? - a system programmer's approach to bridging the gap between software and hardware "

Monday, February 22nd

8:30am : *Introduction*

9:00-9:45am : Keynote #2: Jim Larus (Microsoft Research): "Should We Fear Concurrency?"

10:00am-noon : *Attendees introduce their position papers with presentations (7 min each)*

12:00-12:30pm : *Lunch*

12:30-2:30pm : *Breakout in groups to address the key questions*

- General questions: (2-3 slides)
    - What are the computer architecture (CA) challenges to be solved by 2020-25 to enable ubiquitous parallel software systems?
        - What do the platforms look like?
        - What do the programming models look like?
        - What are the big questions that must be addressed?
    - What role do CA academics have in enabling ubiquitous parallel software systems?
        - On what horizon do we research?
        - What methodologies?
        - How do we make our output (ideas, students) relevant?
        - With what resources? If public funding, what is the case-for?
- Questions about the particular area assigned: (4 slides)
    - What is the detailed research program and roadmap?
    - What will the deliverables be at the program level?
    - Is the assigned area one of the 4-6 Recommended Research Thrusts by itself / should we combine / not discuss it?
- Optional: Are there other areas we should be focusing on (1-2 slides)?

3:00-4:30pm : *Plenary with one presentation per group*

4:30-5:45pm : *Groups reconvene. Possible group reassignment*

- Rework based on feedback
- Expand and structure the material for the final report

6:15pm : *Working dinner*

- Panel: Funding agency perspectives for parallel CA research in the next 10-15 years
- Panelists: Sampath Kannan, NSF (Chair); Almadena Chtchelkanova, NSF; Bill Feiereisen, CRA; Chita Das, Penn State University and NSF; Krishna Kant, NSF; William Harrod, DARPA

Tuesday, February 23rd

8:30am : *Reconvene, converge on the proposed Research Thrusts*

8:45-10:00am : *Breakout in groups.*

In addition to the several Research Thrusts groups, add three more groups:

- Educational Perspective
- Industry Collaboration
- Funding Collaborators (Bill Feiereisen, lead)

10:15am-12:15pm : *Plenary:*

- Each group presents
- Group discussion and feedback

12:15-1:00pm : *Lunch*

1:00-3:30pm : *Breakout*

- Each group finishes up the slides and the chapter of the report

3:45-4:45pm : *Plenary: Critique and assembly*

4:45-5:00pm : *Discussion and social (Beer and wine)*

- Feedback and next steps

13 Appendix D: Slides from the Working Groups

These are attached in a separate document.