ACCELERATOR ARCHITECTURES

Sanjay Patel
Wen-mei W. Hwu
University of Illinois at Urbana-Champaign

We are entering the golden age of the computational accelerator. The commercial accelerator space is vibrant with activity from semiconductor vendors, large and small, that are designing accelerators for graphics, physics, network processing, and a variety of other applications. System vendors are introducing tools and programming systems to lower the barriers to entry for software development for their platforms. We are already seeing the initial stream of applications that benefit from these accelerators, and there are definite signs that more are yet to come. The research space is blossoming with very broad, multidisciplinary activity in advanced research and development for new classes of accelerator architecture and applications to tap into their power.

It is our honor to serve as coeditors of this special issue of *IEEE Micro* on accelerator architectures. There is much to be said about this area, and the authors of our articles have provided a good sampling of the commercial and academic work in this emerging area.

As with any emerging area, the technical delimitations of the field are not well established. The challenges and problems associated with acceleration are still formative. The commercial potential is not generally accepted. There is still the question, "If we build it, who will come?"

In this introductory article to this issue, let us spend some time trying to define the area of computational acceleration, discuss some of the architectural trade-offs, clarify some of the issues, drawbacks, and advantages of applications development on accelerator architectures, and try to articulate who might come if we build it.

What is an accelerator?

Let us first attempt to better define the notion of an accelerator. An accelerator is a separate architectural substructure (on the same chip, or on a different die) that is architected using a different set of objectives than the base processor, where these objectives are derived from the needs of a special class of applications. Through this manner of design, the accelerator is tuned to provide higher performance at lower cost, or at lower power, or with less development effort than with the general-purpose base hardware. Depending on the domain, accelerators often bring greater than a 10× advantage in performance, or cost, or power over a general-purpose processor. It's worth noting that this definition is quite broad, covering everything from special-purpose function units added to a base processor, to computational offload units, to separate, special processors added to the base platform. Examples of accelerators include floating-point coprocessors, graphics processing units (GPUs) to accelerate the rendering of a vertex-based 3D model into a 2D viewing plane, and accelerators for the motion estimation step of a video codec.

We view an accelerator as an augmentation of a base-class, general-purpose, or commodity system. As such, the accelerator is added to the system to achieve greater functionality or performance. Figure 1 shows the general system architecture of an accelerator, where the accelerator is attached to the base processor via an interconnect. Many variations of this model are possible, including the accelerator connected via a system bus such as PCI Express or HyperTransport. This is a relatively low-cost path, given the commod-

Published by the IEEE Computer Society

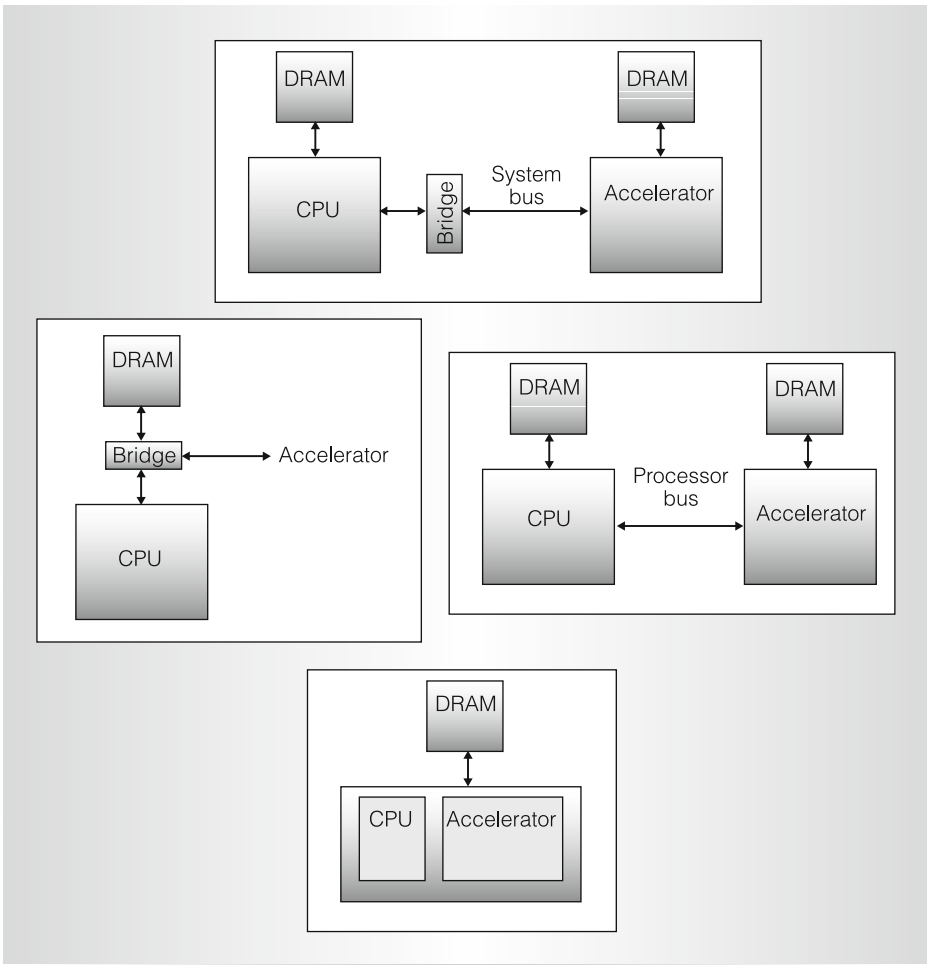

Figure 1. Several generalized accelerator system architectures.

ity support for these protocols, but, because these buses are intended to support a wide variety of devices, they are of typically modest performance and capability. PCI Express 2.0, for example, provides up to 16 Gbytes/s of bandwidth at microseconds of latency.

Some accelerator domains require tighter coupling between the accelerator and the processor, and for such domains other models may be more appropriate. The accelerator can be attached via the processor bus, such as a front-side bus, where the accelerator would be in a processor-like socket in the system, as is the case with the AMD Torrenza.[1] This model can provide higher bandwidth at lower latency, and with tighter integration than the system bus (for example, direct access to processor memory, possibly with coherence activity), but at a higher cost, which might not be feasible for cost-sensitive markets.

Even tighter coupling is possible with the accelerator directly on the same die as the processor, as is used with the CellBE processor. The choice of particular integration model depends on the nature of the domain, the volume of chips its market can support, and the general cost of solution it can bear. In this view, the accelerator can be thought of as a heterogeneous extension to the base platform.

Accelerators can have macroarchitectures that span from fixed-function, special-purpose chips (early generations of graphics chips were of this variety) to highly programmable engines tuned to the needs of a particular domain (latest-generation

graphics chips are of this variety). The choices of macroarchitecture are driven primarily by the diversity of computation in the domain requiring the accelerator. The more varied the computation, the more programmable the accelerator will need to be.

Generally speaking, accelerator architectures maximize throughput per unit area of silicon, (or depending on product and technology constraints, throughtput per watt), by invariably exploiting parallelism via fine-grained data-parallel hardware. Almost all exploit some form of multiword single-input, multiple-data (SIMD) operation, as SIMD hardware can have better throughput per area over a more general parallelism model; however, this passes some optimization cost on to the developer. Various memory models are employed: some accelerators have software-managed memories, some have hardware-managed caches, some have direct hardware support for interprocessor communication, some have very high-bandwidth channels to external memory. Accelerators also tend to use specialized, fixed-function hardware for frequent, regular computation. When contrasted to general-purpose CPU architectures, which are optimized for low latency and a richer application programming model, the fine-grained parallel accelerator architectures appear more akin to digital signal processors (DSPs) and early RISC processors that moved the performance burden out of the hardware into the software layers.

Viewed more from a market-driven perspective, accelerators arise because small sets of economically relevant applications demand more performance or more functionality than the base platform can provide. The economics of the situation justify the inclusion of additional hardware. History has taught us that this is a precarious path, though. Many ideas for computational acceleration have been proposed; many companies have attempted commercial solutions and found little success. People often fail to include the solution's longevity in their feasibility assessment of the technology. Eventually, the general-purpose device usurps the accelerator, as Moore's law reduces the cost of performance in the CPU. Eventually, the market for the accelerator erodes, and the accelerator dies. Examples include floating-point coprocessors, audio DSPs for PCs, and video decode acceleration chips. A sustainable accelerator model requires an application domain where "too much performance is never enough."

Application domains

Although it would be highly presumptuous of us to attempt to articulate future applications that will demand acceleration, we can examine a few domains that demand it today. Graphics, gaming, network processing (which includes TCP offload, encryption, deep packet inspection, intelligent routing, IPTV, and XML processing), and video encoding are the well-known spaces in which commercial accelerator chips for improving system performance are generally commercially viable. Markets exist for specialized chips for image processing and specialized functional blocks for SoCs for mobile devices (to improve overall performance per watt). Scientific computing, oil and gas exploration, and financial modeling have also been strong markets in which the accelerator model has provided value, particularly as more computation is done on an interactive, client-side basis in order to drastically reduce delay to discovery or decision making.

Why does the acceleration model work well for these domains? Primarily, these domains fit the mold in which "too much performance is never enough." Additional performance provided by the base platform is too costly (in dollars or in watts) or not presently possible, thus a customized solution makes economic sense.

From a more technical perspective, these domains are amenable to an accelerator-based solution, for which a combination of parallelism, pipelining, and regularity of computation are necessary. Accelerators use parallelism to gain their performance advantage over the general-purpose processors; thus, the key computation in the domain of interest must exhibit substantial parallelism to take advantage of the accelerator. Since the accelerator is working in concert with

the CPU, which is running the bulk of the application code, the application must be amenable to offloading via a relatively long-latency and low-bandwidth interconnect to the accelerator. For this, the application must deliver computational work units of sufficient size to the accelerator, and in the general case, send them in a decoupled, pipelined fashion to the accelerator, without stalling on the return of the results. Some accelerators exploit the regularity of the computation or communication patterns of a particular domain to gain their advantage over the general-purpose solution. Embedding specialized hardware paths and logic into the accelerator results in computationally denser substructures than using general-purpose paths. If a substantial amount of the computation can be mapped onto the special paths, the architecture can realize a net gain.

Application development

An important criterion for the viability of a particular accelerator technology is the cost of development for applications to tap into the technology. Traditionally, most accelerator studies have focused on the level of speedup one can achieve with accelerators, with little or no attention paid to the development effort required for achieving the speedup. There are few domains that require performance (or lower power) at any cost. In a typical application scenario, however, engineering cost can quickly erode the performance benefit. A sensible vice president of engineering will always question the cost involved in achieving the speedup. If the costs are too high, the accelerator platform will be correctly rejected over the lower-performing, lower-cost, general-purpose solution. It is therefore important that one understands the major sources of cost in using accelerators in application development.

There are, in general, four major contributors to this cost. First, the nature of an accelerator as a separate architectural entity working in concert with the general-purpose CPU makes the application development intrinsically more complex. As we described earlier, the application must be offloaded. An application developer must identify the portions of the computation to offload, isolate the data structures required by these portions, manage the transfer of these data structures to the accelerator memory (if the accelerator has a separate address space from the main processor), synchronize between the main processor and the accelerator, transfer the result data back to the main memory, and integrate the results into the original data structures. These offloading tasks are in addition to the original development effort required for the non-accelerated software. What makes this process even more difficult is that data transfer between the CPU and the accelerator is often a major performance overhead, and so needs to be tuned and optimized. To achieve the desired performance benefits, the application developer must overlap the computation with data transfer in a pipelined fashion, which often requires major changes to the program structure surrounding those components to be offloaded. One architectural approach to reduce this complexity is to have a shared memory space (potentially cache coherent) between the main processor and the accelerator, and this is the option provided by the AMD Torrenza and Intel CSI bus models for accelerator integration. Gelado et al. proposed architectural enhancements that allow entire data structures to be hosted by either the CPU memory or the accelerator memory on a dynamic basis to relieve the application developers from the burden of manually having to manage this process.[2]

The second source of complexity arises from the current lack of a standard architectural model (de facto or otherwise) for accelerators. General-purpose processing has benefited greatly from standardization around the PC architecture and its various flavors. Software developers can develop code knowing that their code will have a significant current and future base of installed hardware on which it can run, and that their software can be ported to similar platforms with well-understood costs. Some acceleration domains, such as raster-based graphics, have well-defined standards for applications development, which in return has provided a commercial explosion of applications that use the

accelerator hardware. Most accelerator spaces are, however, less mature, much more fragmented, and currently highly dynamic. An application developer considering acceleration of a computationally intensive kernel has a variety of off-the-shelf architectures to choose from—such as GPU, floating-point arrays, and FPGA architectures—each with its own application development environment and source language! For each of these hardware choices, the longevity of the programming interface is uncertain: it is unclear whether future chips from the same vendor will continue to support the same interface. In the worst case, an application developer needs to craft a different version of the application for each accelerator. In the GPU space, attempts have been made to take advantage of the standard graphics libraries such as OpenGL to provide a uniform programming model across different GPU implementations, such as the RapidMind API or the emerging OpenCL standard. Such approaches, however, are limited by the restrictions imposed by the underlying low-level graphics interfaces, which has prompted several vendors to introduce models of their own that map directly down to their hardware. AMD has provided support using the Brook+ streaming model for acceleration onto its ATI GPUs. Nvidia's CUDA (discussed in another article in this issue) is a shared memory-like data parallel programming model for Nvidia GPUs. We are even seeing generalizations of these vendor-specific models, such as the recent MCUDA tool[3] that enables CUDA source code to be efficiently targeted to both Nvidia GPUs and multicore CPUs using compiler transformations to automatically remap the application using threading granularities and memory types appropriate for the particular target architecture. For many accelerators, however, the development tools are less mature, less stable, and with fewer features than for the PC-based environment, adding risks to the accelerator-based development effort.

The third source of complexity arises from the fact that accelerators are designed to maximize computation throughput, which is often achieved at the expense of ease of programmability. Accelerators often have software-managed memories, special-purpose hardware, or raw hardware interfaces, all of which require additional software complexity for management or tuning in return for the higher performance. For example, software-managed SRAMs offer a sizable increase in on-chip storage density over hardware-managed caches, but require additional work on part of the developer to manage the SRAMs and to remap data structures to the address space of the local memories. Several proposed programming models attempt to abstract away some of these details. For example, the Sequoia model offers a virtualized, self-managed set of on-chip memory types that can be automatically targeted to different hardware architectures, insulating the developer from managing their own on-chip memory.[4]

The fourth source of complexity comes from optimization. Those seeking to accelerate their application are fundamentally looking at a multivariable optimization problem that includes parallelization. Getting good performance on an accelerator often requires sophisticated code and data structure transformations to expose the right variety and amount of parallelism (SIMD parallelism, in most cases) to effectively utilize chip resources and to adhere to the idiosyncrasies of the underlying hardware. One typically needs to perform simultaneous optimization of the algorithm, data structure selection, threading granularity, data-tiling dimensions, register usage, data prefetch distance, and loop unrolling. These parameters are often not orthogonal to each other: changing one may require changing another due to limited resources, such as the threading degree and local register file size in a CUDA program. The performance effect of varying these parameters is often nonintuitive and requires actual code development and execution time measurements to quantify. The Spiral project seeks to automate the tuning process for digital signal processing algorithms.[5] Ryoo et al. use a parameterized programming style and automated parameter space search and pruning methodology to reduce the programming efforts required to achieve opti-

mal combination of parameter values.[6] On a macroscopic level, Amdahl's law must be contended with: the nonaccelerated portion of the application can start to dominate performance, requiring optimization or acceleration of its own. Often the scope of the acceleration effort increases as new bottlenecks become exposed, and a larger fraction of the original code base requires optimization.

A quick survey of accelerator projects

The vibrancy of accelerators can be strongly felt in the amount of commercial activity in the area. Major semiconductor vendors are introducing fine-grained data-parallel architectures for general-purpose acceleration, with their products commercially anchored in the graphics acceleration space.

Nvidia and AMD both have announced general-purpose computing products (hardware and software) that use their 3D graphics accelerator architectures. Products from both companies were formative for the general-purpose GPU (GPGPU) movement that spawned much of the commercial GPU computing activity. Nvidia recently announced the G280, a 240-core chip that supports nearly 1 Tflops of peak single-precision performance and is programmable via the CUDA model. AMD broke the 1-Tflops mark with its ATI Radeon HD 4800 series GPU, which has 1.2-Tflops peak performance. Intel has a project code-named Larrabee that is intended to be an accelerator for visual computing. According to an official Intel statement,

> *The Larrabee architecture will be Intel's next step in evolving the visual computing platform. The Larrabee architecture includes a high-performance, wide SIMD vector processing unit (VPU) along with a new set of vector instructions including integer and floating point arithmetic, vector memory operations and conditional instructions. In addition, Larrabee includes a major new hardware coherent cache design enabling the many-core architecture. The architecture and instructions have been designed to deliver performance, energy efficiency and general purpose programmability to meet the demands of visual computing and other workloads that are inherently parallel in nature.*

The CellBE was similarly envisioned to be a general-purpose accelerator rooted in the broader consumer market. IBM, in partnership with Sony and Toshiba, developed the architecture as a general computing accelerator for consumer applications, such as the Playstation 3 game console, but also to span into specialized acceleration for supercomputers (for example, Roadrunner, http://www.lanl.gov/orgs/hpc/roadrunner).

Looking past the major vendors, we also notice a flurry of activity in smaller companies hoping to tap into an emerging accelerator market. XMOS, Ambric, Plurality, Movidia, Stream Processors, Tilera, Element CXi, Clearspeed, AGEIA, Rapport, and Cavium are a few of the companies that have publicly announced chips that provide acceleration for particular application spaces—nearly all of them providing ASIC-based solutions that involve general-purpose acceleration using tens to hundreds of cores. For areas where the acceleration demand is high, but for which no ASIC solution is yet available, developers are opting to use FPGA-based solutions. FPGAs developed by Altera and Xilinx are integrated by system integrators such as Nallatech, XtremeData, and Alpha Data into accelerator boards for commodity systems. In a few cases, owing to the extreme regularity of computation and communication, FPGA-based solutions can provide substantial value over even fine-grained general-purpose accelerators such as GPUs. For example, FPGA solutions for accelerating logic simulation are particularly profitable because individual bit operations in the logic to be simulated can be mapped directly into data paths in hardware.

In the research sphere, several academic projects in the recent past have led to commercial impact in the accelerator area. Such seminal projects include the Imagine at Stanford[7] and Raw at MIT.[8] These projects laid the groundwork for current

thinking in accelerator architecture, providing early examples of highly parallel chips and simple, effective programming models. Presently, Intel has ongoing projects in its TeraScale Computing initiative, involving algorithms, applications, tools, and architectures, including the 80-core, 1-Tflops Polaris chip announced in 2007.[9] Intel has codified a vision for future accelerator applications as those that involve recognition, mining, and synthesis. At University of California, Davis, Baas et al. are working on high-density computational structures configured in an asynchronous array.[10] At University of Illinois at Urbana-Champaign, we are leading an effort called the Rigel project, which is a 2,000+-core, fine-grained multiple-instruction-stream, multiple-data-stream (MIMD) accelerator for visual computing applications, such as real-time computer vision and imaging.[11] Rigel pushes the envelope on throughput per unit area, making trade-offs that promote effective programmability for parallel applications with minimal compromise to performance. Rigel will be capable of multiple Tflops of peak performance, and programmable via a bulk synchronous programming model.

Toward the general-purpose accelerator

It is an exciting time for computational accelerators, as evidence mounts that we are nearing transformative thresholds in computer performance that will enable breakthroughs in many fields, including interactive, intelligent visual computing applications. This, perhaps, is the driver that has caused the GPU's transformation into a general-purpose accelerator, and it may make the accelerator the dominant silicon component in commodity computing systems. One possible grand unified architecture contains a base general-purpose processor that is relatively small, necessary for legacy and control applications (which are not worth parallelizing to the accelerator) and an accelerator that provides the computing horsepower and commands the bulk of the silicon. The accelerator architecture will most likely consist of a fine-grained array of cores, architected to optimize performance per area or watt. As silicon architects, we largely understand how to provide value here.

The larger and mostly open question is how to reduce the cost of software development to utilize this grand unified architecture of base processor and accelerator. For non-performance-sensitive applications (of which there are many), the easiest path to market is to run on the base processor. Performance-sensitive applications will need to be optimized to take advantage of the fine-grained array; this, as we have articulated, is a path with many pitfalls, all of which erode the value proposition to software developers.

As advanced developers and system researchers, we must undertake the task of charting the roadmap of feasible solutions. Some issues will require attention from the commercial sphere: a lack of standards and shifting architectural roadmap will impede application development en masse, but this can be solved with foresight and commercial collaboration. Other issues require more research: effective parallelism models, tools, languages, APIs, and frameworks for application development are needed to lower the cost of developing applications for accelerators. Researchers should continue to understand the problems, and they should build prototype solutions.

In this issue

It is our pleasure to introduce this *IEEE Micro* special issue on accelerator architectures. In this issue, we have included an invited article from Garland et al. of Nvidia that describes some of the experiences, optimization strategies, pitfalls, and successes in porting applications to its GPU-based computing accelerators using the CUDA programming environment.

We also received a good number of submissions to the open call for papers for this issue, from which we selected four articles that provide a good sampling of advances in the broad space of accelerator applications and architectures. Woo et al. describe the POD architecture, an application accelerator that can attach to a base processor using 3D integration, such as die stacking. Bougard et al. apply accelerator architecture principles to the domain of

software-defined radios, and demonstrate a chip design that achieves substantial instruction throughput at very low power, which is particularly important for wireless devices. Wen et al. describe a hybrid on-chip dual-mode memory system, which can function as a purely software-managed SRAM or as a hardware-managed cache when the application's reference stream is irregular.

Last, we include an article outside the norm, on a potentially interesting specialized application domain involving bioimplantable devices. In this article, Jin and Cheng describe a set of benchmarks that represent the base computation for applications that could potentially improve human health and viability.

We hope you enjoy this issue! MICRO

References

1. M. Hummel, M. Krause, and D. O'Flaherty, ''AMD and HP: Protocol Enhancements for Tightly Coupled Accelerators,'' white paper, AMD, 2007; http://www.hp.com/techservers/hpccn/hpccollaboration/ADCatalyst/downloads/AMD_HPTCAWP.pdf.
2. I. Gelado et al., ''CUBA: An Architecture for Efficient CPU/Co-processor Data Communication,'' *Proc. 22nd Ann. Int'l Conf. Supercomputing* (ICS 08), ACM Press, pp. 299-308.
3. J. Stratton, S. Stone, and W.-m. Hwu, ''MCUDA: An Efficient Implementation of CUDA Kernels on Multi-cores,'' tech. report IMPACT-08-01, Univ. of Illinois at Urbana-Champaign, 2008.
4. K. Fatahalian et al., ''Sequoia: Programming the Memory Hierarchy,'' *Proc. 2006 ACM/IEEE Conf. Supercomputing* (SC 06), ACM Press, 2006; http://doi.acm.org/10.1145/1188455.1188543.
5. M. Puschel et al., ''SPIRAL: Code Generation for DSP Transforms,'' *Proc. IEEE*, vol. 93, no. 2, Feb. 2005, pp. 232-275.
6. S. Ryoo et al., ''Optimization Principles and Application Performance Evaluation of a Multithreaded GPU Using CUDA,'' *Proc. 13th ACM SIGPLAN Symp. Principles and Practice of Parallel Programming* (PPoPP 08), ACM Press, 2008, pp. 73-82.
7. U. Kapasi et al., ''The Imagine Stream Processor,'' *Proc. 2002 IEEE Int'l Conf. Computer Design* (ICCD 02), IEEE CS Press, 2002, pp. 282-288.
8. M.B. Taylor et al., ''The Raw Microprocessor: A Computational Fabric for Software Circuits and General-Purpose Programs,'' *IEEE Micro*, vol. 22, no. 2, Mar./Apr. 2002, pp. 25-35.
9. J. Held, J. Bautista, and S. Koehl, ''From a Few Cores to Many: A Tera-scale Computing Research Overview,'' white paper, Intel, 2006; ftp://download.intel.com/research/platform/terascale/terascale_overview_paper.pdf.
10. Z. Yu et al., ''An Asynchronous Array of Simple Processors for DSP Applications,'' *IEEE Int'l Solid-State Circuits Conference* (ISSCC 06), IEEE Press, 2006, pp. 428-429.
11. J. Kelm et al., ''Rigel: A Scalable Processor Architecture for 1000+ Core Computing,'' tech. report IMPACT-08-02, Univ. of Illinois at Urbana-Champaign, 2008.

Sanjay J. Patel is an associate professor of electrical and computer engineering and Willett Faculty Scholar at the University of Illinois at Urbana-Champaign. His research interests include high-performance and massively parallel chip architectures, parallel programming models, and visual computing applications. He has considerable commercial chip design experience, working and consulting for a number of companies, including Digital Equipment Corporation and Intel. From 2005 to 2008, he was the CTO and Chief Architect of AGEIA Technologies, a fabless semiconductor industry that developed chips for accelerating physical simulation for video games. Patel earned his BS, MS, and PhD in computer science and engineering from the University of Michigan, Ann Arbor. He is a member of the IEEE.

Wen-mei W. Hwu is Sanders-AMD Endowed Chair Professor in the Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign. He also directs the IMPACT research group (Illinois Microarchitecture Project utilizing Advanced Compiler Technology, www.crhc.uiuc.edu/Impact) and is hardware lead of the NSF Petascale Computer Project awarded to the University of Illinois

and IBM in 2007. His research interests are in architecture, implementation, and programming tools for parallel computer systems—in particular, novel computer architectures and the compiler techniques they require. For his contributions in research and teaching, he received the Eta Kappa Nu Holmes MacDonald Outstanding Teaching Award, the ACM SigArch Maurice Wilkes Award, the ACM Grace Murray Hopper Award, the Tau Beta Pi Daniel C. Drucker Eminent Faculty Award, and the ACM/IEEE ISCA Most Influential Paper Award. Hwu has a PhD in computer science from the University of California, Berkeley. He is a fellow of both the IEEE and the ACM.

Direct questions and comments about this special issue to Sanjay Patel, 262 Coordinated Sciences Laboratory, MC 228, 1308 West Main St., Urbana, IL 61801-2307; sjp@illinois.edu.