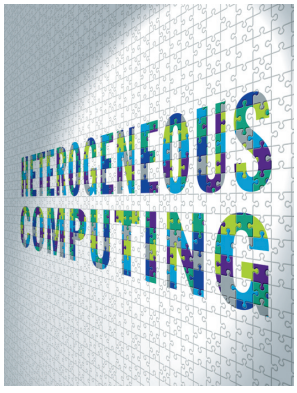
HETEROGENEOUS COMPUTING



Ravi Iyer
Intel

Dean Tullsen
University of California, San Diego

The microprocessor industry experienced a significant technology shift when multicore processors were introduced. Not only did parallelism immediately become pervasive, but chip designers became chiefly responsible for the nature of the parallelism that is exposed to the user. They decide what type of core, how many cores, which interconnect to use, and how to distribute available area among caches, cores, and other components. The chip architect makes much more complex decisions about how to best use the available transistors on the chip than ever before.

Soon, an additional design dimension become apparent—what level of diversity of computational resources should be supported on chip? Initial general-purpose microprocessor designs were all homogeneous, which had advantages in design, testing, and programmability. But those designs begged the question—once you have placed one or more identical cores onto the chip, is the very best use of transistor budget (or power budget) to continue to stamp out identical cores, or are those transistors best used to provide more diverse capabilities? That diversity can come in the form of heterogeneous general-purpose cores, specialized cores, accelerators, or even configurable fabric.

The value of heterogeneous computing becomes even more apparent when thread-level parallelism falls short of hardware parallelism, due to a lack of threads or to power constraints (dark silicon). When three threads are running on an eight-core homogeneous processor, those five idle cores provide no value whatsoever. But on a heterogeneous processor, even idle cores provide value—they could present a more efficient host for one of the running threads, either now or in a future phase of execution.

In this issue

There are many different ways to introduce heterogeneity to the computing landscape, and the articles in this special issue reflect the spectrum of approaches taken to add diversity to our computational capabilities. Perhaps the most natural way to introduce diversity on the chip is to aggregate computing resources that were already diverse. GPUs have evolved into more general-purpose (GPGPU) engines, which allowed us to separate regular, parallel, streaming segments of our workload from the less parallel, control-intensive segments that still run most effectively on the CPU. In "Scalable Heterogeneous CPU-GPU Computations for Unstructured Tetrahedral Meshes," Johannes Langguth et al. study this partitioning between CPUs (Intel Xeon Phi and Xeon E5-26xx processors) and GPUs (Nvidia K20 GPUs) for unstructured 3D tetrahedral meshes. They analyze the performance on two platforms and show that combining the CPU and GPU execution capacity clearly provides a performance advantage over the GPU-only approach for irregular applications.

"Enabling Portable Optimizations of Data Placement on GPU" by Guoyang Chen et al. addresses the dynamic data placement challenge and its impact on GPU performance. The authors introduce a new framework called Porple, which allows for customized, on-the-fly placement in memory based on memory type and the input dataset. They show significant performance improvements compared to typical programmer-driven data placement. Their data placement engine adapts to new memory systems and should provide insights to the reader for possible future memory hierarchy optimizations.

In "Achieving Exascale Capabilities through Heterogeneous Computing," by Michael Schulte et al., the authors describe the hardware and software challenges in building heterogeneous exascale systems with integrated CPUs and GPUs. They describe AMD's vision for exascale computing, outlining the APU approach, necessary hardware support, and

Published by the IEEE Computer Society

memory and programming challenges. They also describe physical constraints, including power, thermal, reliability, and resilience. The article gives the reader an inside view into considerations for developing real heterogeneous solutions for exascale computing.

In addition to providing CPU-GPU heterogeneity, we can provide diversity within the CPU array via heterogeneous CPU cores, each targeted at a distinct workload segment or a distinct performance/power level. In "Kernel-to-User-Mode Transition-Aware Hardware Scheduling," Nikola Markovic et al. take a new approach to thread scheduling by identifying specific multithreaded application bottlenecks (such as thread synchronization) and proposing a scheduling mechanism that can be implemented in hardware. The proposed KUTHS policy recognizes critical sections by monitoring kernel transitions and then maps the execution of critical code sections on large cores to achieve performance gains.

As systems become more heterogeneous, programming those systems becomes more challenging. "Understanding Portability of a High-Level Programming Model on Contemporary Heterogeneous Architectures" by Amit Sabne et al. addresses one of those challenges—the wide diversity of programming models, memory models, and architectures of existing accelerators. The authors present a single high-level programming model that can be ported to various accelerator architectures. They propose HeteroIR, a high-level architecture-independent intermediate representation to map high-level programming models to heterogeneous architectures, and they study the performance portability on three different platforms and show the efficacy and tradeoffs.

The value of heterogeneity is not exclusive to the processing elements, because programs also use the interconnect and memory hierarchy in diverse ways. "Designing Efficient Heterogeneous Memory Architectures" by Evgeny Bolotin et al. presents a model and analysis of energy, bandwidth, and latency for current and emerging DRAM technologies. This enables a detailed study of heterogeneous memory systems that combine memory technologies with different attributes. We hope readers find the analysis useful in determining the best configuration for heterogeneous memory systems and making tradeoff decisions in the future.

The first six articles exploit intentional heterogeneity. The last article examines unintentional heterogeneity, which is introduced by process variation. In "Decoupled Control and Data Processing for Approximate Near-Threshold Voltage Computing," Ismail Akturk et al. describe how process variation due to near-threshold voltage operation introduces both error-prone and reliable cores. The authors use control and data partitioning to develop error-tolerant architectures. Some emerging applications, such as Recognition, Mining, and Synthesis (RMS) applications, can tolerate errors in data processing but still require reliable cores for control processing. Therefore, the authors propose the decoupled control and data processing solution for approximate near-threshold voltage computing.

The seven articles in this special issue should give insight into the opportunities and challenges in heterogeneous architectures, ranging in topics from heterogeneous CPUs, GPUs, accelerators, control and data decoupling, data placement, memory hierarchies, and programming models. We hope you enjoy exploring the potential insights and advances in heterogeneous architecture research that these articles provide. MICRO

Ravi Iyer is a senior principal engineer, CTO, and director in New Business Initiatives at Intel. His interests include systems on chip and chip multiprocessors, including novel cores, accelerators, innovative cache and memory hierarchies, quality of service, heterogeneous architectures, algorithms and workloads, and performance and power analysis. Iyer has a PhD in computer science from Texas A&M University. He is an IEEE Fellow. Contact him at ravishankar.iyer@intel.com.

Dean Tullsen is a professor in the Computer Science and Engineering Department at the University of California, San Diego. His research focuses on parallel architectures, including multithreading, multicores, and datacenters. Tullsen has a PhD from the University of Washington. He is a fellow of IEEE and the ACM. Contact him at tullsen@cs.ucsd.edu.