TOOLS AND ENVIRONMENTS FOR MULTICORE AND MANY-CORE ARCHITECTURES

Wu-chun Feng, *Virginia Tech*
Pavan Balaji, *Argonne National Laboratory*

Programming for serial computing is already a difficult undertaking that we have yet to master; programming for parallel computing will only exacerbate this difficulty. Fortunately, tools and environments are beginning to emerge that will let us take advantage of multicore and many-core architectures in a more productive manner.

In the past, computing speeds doubled every 18 to 24 months, increasing the clock speed and giving software a "free ride" to better performance. This free ride, however, is now over, and such automatic performance improvement is no longer possible. With clock speeds stalling out and computational horsepower instead increasing due to the rapid doubling of the number of cores per processor, serial computing is now dead, and the vision for parallel computing, which started more than 40 years ago, is a revolution that is now upon us.

With the advent of multicore chips—from the traditional AMD and Intel multicore to the more exotic hybrid multicore IBM Cell and the many-core AMD/ATI and NVIDIA graphics processing units—parallel computing across multiple cores on a single chip has become a necessity. However, writing parallel applications is a significant challenge that will create more-not less-problematic software.

A WORLD OF UBIQUITOUS PARALLELISM

Even without parallelism, software defects already account for up to 40 percent of system failures. Further, concurrency bugs and memory-related bugs cause more than 60 percent of system vulnerabilities. Consequently, even with serial computing, computing systems must often be rebooted, thus creating downtime and impacting availability. This is particularly insidious in data centers, where the average cost of an hour of downtime can run in the millions of dollars.

In short, programming for serial computing is already a difficult undertaking that we have yet to master; programming for parallel computing will only exacerbate this difficulty. For parallelism to succeed, it must ultimately produce better performance relative to speed, efficiency, and reliability. However, most programmers not only are ill-equipped to produce proper parallel programs, they also lack the tools and environments for producing such programs.

Dealing with these issues requires a suite of tools and environments that provide users and developers with convenient mechanisms for managing different resources in multicore environments. Resources include memory, cache and compute units, compilers that allow sequential

Published by the IEEE Computer Society

programs to automatically take advantage of multicore systems, strategies that allow users to analyze performance issues in multicore systems, and environments to control and eliminate bugs in parallel threaded programs. Thus, the purpose of this special issue is to present such latest advances in next-generation tools and environments for multicore and many-core architectures.

IN THIS ISSUE

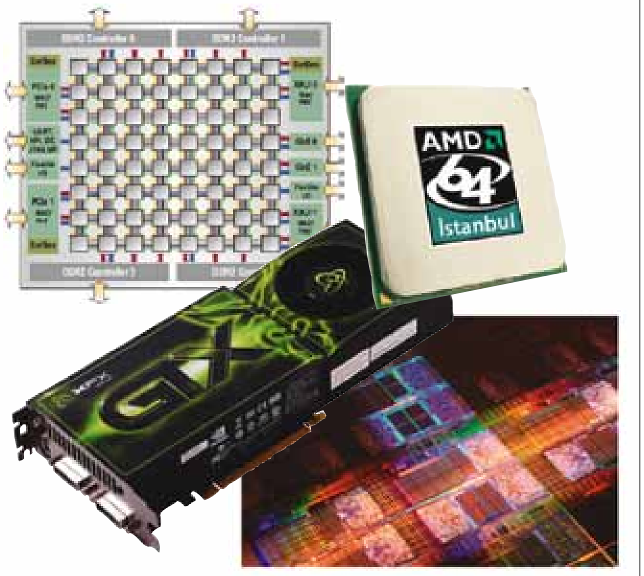
In this special issue, we take a closer look at some of the recent advances in tools and environments that have allowed users to take advantage of multicore and many-core architectures in an easy and productive manner.

In "Programming Multiprocessors with Explicitly Managed Memory Hierarchies," Scott Schneider, Jae-Seung Yeom, and Dimitrios S. Nikolopoulos take a look at modern many-core architectures such as the Cell Broadband Engine that require programmers to explicitly manage data movement in the memory hierarchy.

Managing the memory hierarchy in multicore processors introduces tradeoffs in terms of performance, code complexity, and optimization effort. While processors with coherent hardware caches simplify programming by freeing programmers from having to explicitly manage data, they come at the cost of more expensive hardware requiring more power and potentially providing worse performance. Explicitly managed caches, on the other hand, avoid the additional hardware and logic that is required for such automated cache management, but can make development more complex and error-prone. The authors study various programming models that provide different mechanisms to make the task of explicit memory/cache management easier for application developers. They compare these models using two applications and describe the pros and cons of each model.

With multicore and many-core architectures infiltrating every aspect of computing, including desktops and laptops, developers who were content with just sequential programs are now plunged into the sudden requirement to parallelize their applications. To address this issue, in "Cetus: A Source-to-Source Compiler Infrastructure for Multicores," Chirag Dave and colleagues describe an open source compiler infrastructure that allows for automatic parallelization of C programs on multicore architectures. The authors present an overview of Cetus and offer details about its working model.

Performance analysis of threaded programs has traditionally been one of the biggest challenges of the multicore era. What parts of the program are well parallelized; what parts have too fine-grained parallelism, causing locking and synchronization overheads; and what parts do not have enough parallelism, causing idleness, are all important issues that are not easy to identify. In "Identifying Performance Bottlenecks in Work-Stealing Computations," Nathan R. Tallent and John M. Mellor-Crummey propose a new profiling strategy for analyzing such issues, more specifically focusing on work-stealing programming models.


The many faces of multicore and many-core architectures. Images courtesy of http://images.google.com.

In "Eliminating Concurrency Bugs with Control Engineering," Terence Kelly and colleagues present approaches from control engineering that can be applied to multicore parallel program development, containing the behavior of complex systems and preventing runtime failures. They provide an in-depth look at the Gadara project, which uses discrete control theory to eliminate deadlocks in shared-memory multithreaded software, and discuss broader prospects for concurrency management based on discrete control theory.

We hope the articles in this special issue will provide relevant insights into the emerging ubiquitous world of parallel computing. C

__Wu-chun Feng__ is an associate professor in the Departments of Computer Science and Electrical and Computer Engineering at Virginia Tech. His research focuses on high-performance computing with specific interests in accelerator-based parallel computing, green supercomputing, dynamic multicore and many-core scheduling, and bioinformatics. Feng received a PhD in computer science from the University of Illinois at Urbana-Champaign. Contact him at feng@cs.vt.edu.

__Pavan Balaji__ is an assistant computer scientist in the Mathematics and Computer Science Division at Argonne National Laboratory. His research interests focus on parallel and distributed computing. Balaji received a PhD in computer science and engineering from Ohio State University. Contact him at balaji@mcs.anl.gov.