Exascale Computing

The history of high-performance computing (HPC) spans almost seven decades, and has seen a factor of 10 trillion increase in speed since the first-generation vacuum-tube-based von Neumann computers of the late 1940s. This extraordinary advance greatly exceeds that of any other human technology. And it's not that we initially got it wrong and then later finally got it right. Rather, each decade saw a performance gain of at least two orders of magnitude, steadily harnessing the accumulating advances of the basic enabling device technologies in logic, memory, and data communication. Despite this apparent consistency of progress, the technologies driving performance growth as well as the innovations in programming models and operational methods that have delivered it have changed markedly and repeatedly to sustain this growth.

In the most recent epoch, after 20 years of improvements to the multiprocessor, distributed-memory message-passing strategy, significant changes are taking place, again driven by technological change. Teraflops were achieved in 1997 and Petaflops in 2008. This last milestone was accomplished without significant disruption to programmers employing conventional

1521-9615/13/$31.00 © 2013 IEEE
Copublished by the IEEE CS and the AIP

Steven Gottlieb and Thomas Sterling
Indiana University

methods, despite a dramatic change occurring in 2004/2005, in which the speed of the individual processor core flat-lined due to limitations in power consumption. However, it was clear, even then, that scaling current technologies to exaflops through incremental extensions of past practices would consume much too much power to be practical. This special issue of *CiSE* addresses the deep questions of the challenges currently facing sustained performance growth to exascale and beyond, the opportunities to do so, the new architecture designs that might make it possible, and the programming models and support software methods that will employ it for future applications in science, technology, commerce, and defense.

What the Future Holds—and Still Needs

The advent of multicore sockets and GPU accelerators offer possible performance growth through raw semiconductor technology improvements, but also impose unprecedented challenges in efficiency, scaling, power, and reliability, as well as programmer productivity. Achieving exaflops speed will require new programming techniques, but what of the billions of dollars of investment in past software development and mainstream markets? How will the field of HPC continue to leverage the strength of COTS technologies and the economy of scale of mass-produced computing and memory components if exascale may need something different? Will the highest-end systems become increasingly limited in the classes of problems they can serve, or will new execution models, architectures, and programming techniques evolve to meet these challenges? This special issue of *CiSE* brings together expert views to illuminate the possible approaches.

Before we get more deeply into the challenges of exascale computing, we should talk briefly about the need. From November 2008 to October 2009, there was a series of eight Scientific Grand Challenges Workshops (sponsored by the US Department of Energy Office of Advanced Scientific Computing Research and coordinated by Paul Messina) that asked scientists to assess their need for exascale computing. The workshops covered climate science, high-energy physics, nuclear physics, fusion energy, nuclear energy, biology, material science and chemistry, and national security. The workshop reports (http://science.energy.gov/ascr/news-and-resources/workshops-and-conferences/grand-challenges) detail what could be done with exascale computers.

In December 2009, Rick Stevens and Andy White led a workshop on architectures and technology for extreme-scale computing that brought together scientists and computer scientists from industry, national laboratories, and universities to examine the challenges, some of the potential solutions, and the research that would need to be done to achieve exascale computing by 2018. A key concept is the codesign of the hardware, system software, and applications software to assure that they all work together. Three codesign teams have been funded to study materials in extreme environments, advanced reactors, and combustion in turbulence (http://science.energy.gov/ascr/research/scidac/co-design). There's also an ongoing international effort in software design (www.exascale.org).

If you don't expect to be computing at the exascale level, is there a reason for you to be interested in the current issue? We think so—because the technology that will be needed for exascale will require great improvements in energy efficiency and cost-effectiveness at the node level, this technology might also wind up on your desktop, and departmental systems at the petaflop/s level might become affordable. Although your level of concurrency might be smaller than required for exascale, it will be much higher than what's required on today's desktops.

Contributions to This Special Issue

We kick off this issue with "Exascale Computing Trends: Adjusting to the 'New Normal' for Computer Architecture," by Peter Kogge and John Shalf. Kogge chaired a 2008 DARPA-funded study on technology challenges of building exascale systems (www.cse.nd.edu/Reports/2008/TR-2008-13.pdf). Kogge and Shalf detail why the single-processor speed increases we've seen in the past won't continue, and how the key to exascale computing is a vastly increased level of parallelism and much greater attention to data movement. They also discuss how many picojoules a floating-point operation or a dynamic RAM (DRAM) access cost now and in the future.

The second article—"Programming for Exascale Computers," by Bill Gropp and Marc Snir—deals with the quite significant challenges ahead for application developers who want to know whether their codes will need to be completely rewritten. At this point, the answer isn't completely clear, but Gropp and Snir summarize what programmers will be dealing

Exascale System Shift to Runtime Technologies

As Exascale systems emerge around the end of this decade, two distinct forms are likely to be developed: an incremental extension of conventional heterogeneous systems and a new class of architectures with lightweight cores and tightly integrated networking supporting some form of global address space. The first class will provide important extensions in support of continuity and legacy codes and skill sets. The latter class will incorporate and integrate innovative structures, semantics, and mechanisms to achieve greater scalability, efficiency, generality, portability, and user productivity. It will also embody new principles for superior reliability and power consumption. But both classes of computing systems are likely to share one major innovation in common—the shift from pure static resource management and task scheduling to dynamic adaptive control. Currently, many complex applications measuring performance may exhibit poor efficiencies. Runtime software can support introspection to use continuing information about system execution to guide its control and improve operation.

Runtime system software provides a layer of dynamic execution control between the application programming interface (and its compiler) and the physical computing platform as represented by the operating system. While runtime systems have a long history of yielding high-level system abstractions for many programming models, with a few exceptions it hasn't found favor in high-performance computing. This is due to inherent overheads and the assumption that adding more work (the runtime actions) to reduce time to solution is contradictory. However, recent evidence associated with some classes of application algorithms has suggested that opportunities exist in which runtime systems may in fact be capable of improving overall performance. It's noted that while well-tuned benchmarks may deliver high performance efficiencies such as High Performance Linpack (HPL), many real-world applications will deliver less than 10 percent floating-point efficiency.

Although a runtime system might add some additional work, this could be much less than the potential performance advantage gained through its use. Global barriers, especially within the Bulk Synchronous Parallel (BSP) modality, permit progress in computation to proceed at the rate of the slowest thread at each stage. Yet in many cases, any single task in the following stage is only dependent on a small part of the work preceding it; its precedence constraints. There are complicated interrelationships among the latencies of remote access, the overheads of thread scheduling and context switching, the management of locality both laterally (across nodes) and vertically (through the memory hierarchy), the control of addressing, and the order of thread scheduling. Ideally, the critical path of execution (the longest single sequence of operations from start to culmination) will receive highest priority of resource access. The management and juggling of all these operational variables is the venue of the runtime system and the performance opportunity space in which it controls system execution in support of user applications.

It might not be easy to schedule applications which exhibit varying resource demands per local task at compile (or load) time. Such dynamic problems require adaptive methods to schedule tasks and resources on demand to achieve higher efficiency. Early methods of load balancing, over subscription, work stealing, and inspector-executor approaches have explored this space with some success for specific problems. Charm++ is another early effort, embodying even more adaptive mechanisms, with particular focus on molecular dynamics. More recently, there has been an increasing interest in the development and use of runtime software packages. Examples include Cilk, Thread Building Blocks (TBB), High-Performance ParalleX (HPX), Open Common Runtime (OCR), and X10 runtime, among others.

with, what approaches will be feasible, and the pros and cons of trying to evolve current code to exascale hardware.

Finally, in "PaRSEC: Exploiting Heterogeneity to Enhance Scalability," George Bosilca and his colleagues describe a runtime system and technique for programming that help the application programmer spend less time concentrating on the details of the hardware and how the data needs to be distributed among the processors. This approach to programming should be a real help when dealing with heterogenous compute nodes, and can be tested and used well before the dawn of exascale hardware.

We hope that you enjoy this issue, and we can assure you that you'll be hearing about exascale computing for quite some time. Although the authors for this special issue topic are currently working in the US, the effort to produce hardware and software for computing at the exascale level is an international one. We wouldn't be surprised to see the first exascale computer produced outside the US.

Steven Gottlieb *is a distinguished professor of physics at Indiana University, where he directs the PhD minor in scientific computing. He's also the Associate Editor in Chief of CiSE. His research is in lattice quantum chromodynamics (QCD). Gottlieb has a PhD in physics from Princeton University. Contact him at sg@indiana.edu.*

Thomas Sterling *is a professor of informatics and computing at Indiana University. He also serves as the executive associate director of the Center for Research in Extreme-Scale Technologies (CREST) and as its chief scientist. He has conducted research in parallel computing systems in industry, academia, and government centers, and currently leads a team of researchers at Indiana University to derive the advanced ParalleX execution model and develop a proof-of-concept reference implementation to enable a new generation of extreme-scale computing systems and applications. Sterling has a PhD in computer science from MIT. He's the recent inaugural winner of the Exascale Vanguard Award. Contact him at tron@indiana.edu.*

IEEE Software
Bridging Software Communities through Social
Twin Peaks of Requirements and Architecture
Safety-Critical Software