

TRAILBLAZING WITH ROADRUNNER

By Paul Henning and Andrew B. White Jr.

IN JUNE 2008, A NEW SUPERCOMPUTER BROKE THE PETAFL0P/S PERFORMANCE BARRIER, MORE THAN DOUBLING THE COMPUTATIONAL PERFORMANCE OF THE NEXT FASTEST MACHINE ON THE TOP500 SUPERCOMPUTER LIST ([HTTP://TOP500.ORG](http://top500.org)). THIS COMPUTER, NAMED ROADRUNNER, IS THE RESULT OF AN INTENSIVE COLLABORATION BETWEEN IBM

and Los Alamos National Laboratory, where it's now located. Aside from its performance, Roadrunner has two distinguishing characteristics: a very good power/performance ratio and a "hybrid" computer architecture that mixes several types of processors. By November 2008, the traditionally architected Jaguar computer at Oak Ridge National Laboratory was tied with Roadrunner in the performance race, but it requires almost 2.8 times the electric power of Roadrunner. This difference translates into millions of dollars per year in operating costs.

As supercomputer designers push on toward the exascale computing goal, power consumption becomes a major challenge. Current power estimates for exascale computers range from many tens to low hundreds of megawatts for the computer and memory alone, discounting storage and environmental conditioning. For comparison, Roadrunner requires roughly 2.5 MW for 1.45 petaflops/s peak. The combination of computing performance and power efficiency in Roadrunner shows one of the principal advantages to considering hybrid architectures. However, this advantage comes with the challenge of learning a new programming paradigm. The November/December 2008 issue of *CiSE*¹ provided a glimpse into how computational scientists are adapting to and exploiting a variety of novel architectures. We can do this—there are many benefits, but there are challenges.

It would be nice to think that specialized processors and hybrid systems are simply fads that will disappear soon. Scientific computing has enjoyed a fairly idyllic era since the transition from vector processors to massively parallel processing in the 1990s. Although there has been a shift to commodity clusters and changes in operating systems and communication infrastructure, the applications' basic structure hasn't needed significant change. Unfortunately,

challenges in processor design and fabrication are bringing this era to a close. In many ways, Roadrunner is just as important as a glimpse into scientific computing's future as it is as a petaflop supercomputer.

Processors Are Changing and for Good Reason

As in any business, profit is the driving force in processor design. Without an improved user experience to generate sales, companies have no economic incentive to bring new designs to market. Traditionally, this improvement in user experience came through performance advances in general-purpose processors (GPPs). However, this path has bifurcated: the rapid proliferation of embedded processors has created demand for specialized low-power designs, and a variety of challenges to traditional means of improving performance is causing designers to rethink GPPs. Here, we illustrate some of these challenges to provide a rationale for the changes coming to hardware and software, which John Manferdelli discusses in his article, "The Many-Core Inflection Point for Mass Market Computer Systems."²

One traditional approach to improving processor performance is simply to increase the processors' clock frequencies. However, because power consumption is proportional to clock frequency, the heat density per fixed area of the processor chip increased to the point where simple cooling methods were inadequate. Designers are now lowering processors' clock frequencies, negating the "free" performance improvements that applications were getting from successive generations of faster single processors.

Another technique for transparently improving performance is instruction-level parallelism, common in GPPs since the late 1990s. These *superscalar* processors simultaneously execute multiple instructions on redundant functional units and must automatically detect and avoid data

dependencies between sequential instructions. Superscalar processors also employ pipelined, speculative, and out-of-order execution that lead to a combinatorial number of gates related to dependency checking, branch prediction, and instruction scheduling.³ These techniques create processor designs that are difficult to design and verify; yet, the nature of the instruction stream that executes ultimately limits the promised performance improvements. Designers are moving back to simpler instruction-scheduling models and are using the newly freed space for different purposes.⁴

The final architectural challenge is the growing discrepancy between the time required to execute an instruction and the time required to retrieve data from memory. For most current processors, we can expect a single instruction to take 10 or fewer clock cycles, whereas fetching data from main memory might take several hundred cycles. This problem is compounded when the processor executes multiple instructions simultaneously. Traditionally, programmers have worked around this difference by using larger hierarchies of fast cache memory, which act as a bridge between the processor and the main system memory. However, this cache memory is expensive and complex, and the deeper hierarchies that appear in contemporary processors are increasingly difficult to verify for correctness. Designers are beginning to introduce new memory subsystems to processors, including crossbar switches and programmer-controlled local storage.

Looking beyond processor design, another broad category of challenges deals with the increase in processor power consumption as the fabrication process size decreases. Companies currently use a 45-nm manufacturing process in which SRAM cell area is measured in fractions of square micrometers.⁵ Many complicated physical effects exist at this scale (see, for example, Yannis Tsividis's book, *Operation and Modeling of the MOS Transistor*⁶), but the net effect is that transistors leak significant amounts of power. Advances in material sciences should help stem this problem, but the shrinking feature size implies that we'll soon have more transistors on a chip than we can afford to power simultaneously. It's expected that fine-grained power management features will appear in processors (or even to programmers), leading to heterogeneous performance across even homogeneous processors.

In the face of these significant challenges, processor designers must still meet the economic driver of providing a better user experience. Rather than pursue even more complex single processors, they place multiple copies of

a processor onto one chip to form a multicore processor. Each of these cores tends to be slower and less complex than the single processors of just five years ago, but they provide performance increases in aggregate. However, this trend shifts more work to the application programmer. Not only do programmers have to make up the loss in single-core performance through better optimization, but they must also explore ways of parallelizing their applications to take advantage of more cores. Although this is nothing new for the scientific computing community, it's a fundamental shift in the broader software industry.

In addition to moving to multicore designs, companies have introduced chips that contain a mix of general- and special-purpose cores. These heterogeneous multicore chips represent the most significant challenge (and opportunity!) for software developers. In contemporary heterogeneous multicore chips, special-purpose cores tend to be short-vector processors, which are especially useful in computer graphics applications. Whereas GPPs have had some form of vector operations available for some time (SSE, AltiVec, and so on), offloading this workload to a stand-alone processor allows for greatly increased parallelism. Scientific programmers certainly use vector processors, but expect to see much more specialized processors appearing in the future, such as cryptographic engines, compression, video decoding, and so on. These will pose special problems for the high-performance computing (HPC) community, both in terms of how (or if) to use them, as well as managing the power they draw when not in use.

Hardware Changes That Disrupt Software Development

The changes occurring in computer architectures are creating a ripple effect in the software development arena, even for traditionally serial applications. The availability of specialized processors forces developers to decompose their programs across functional units. Deep memory hierarchies, especially coupled with disjoint address spaces, require special attention to data motion costs. Short-vector processors constrain data structure design, so developers must look to parallelism, often in terms of multithreading, for performance gains.

Even in the HPC community, where programming has always involved some level of adaptation to distinctive hardware, programs must evolve to new levels of complexity. We can no longer imagine that all processors have equal access to resources such as memory, network, or I/O: developers

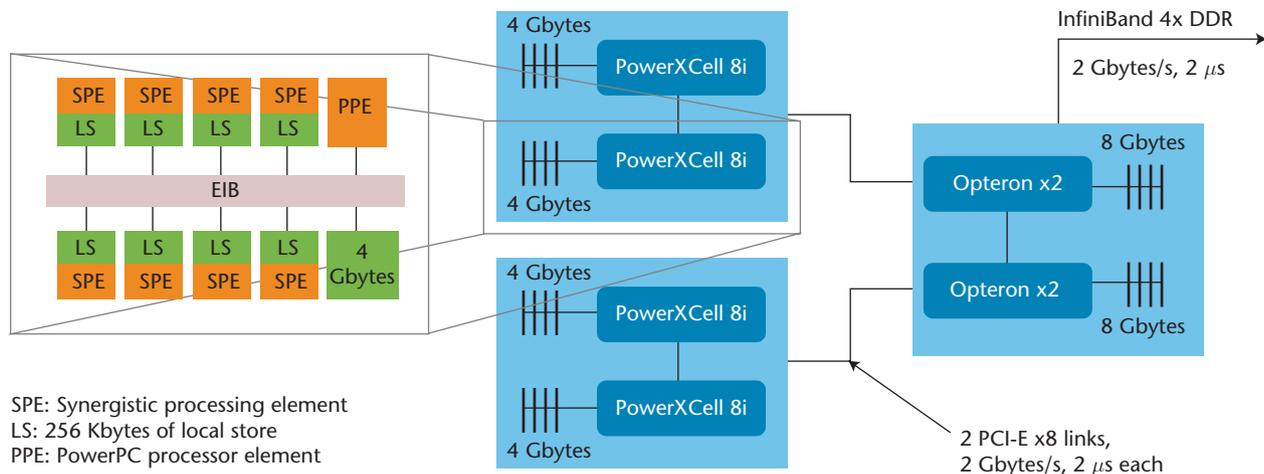


Figure 1. A Roadrunner node schematic. Bandwidth and latency characterize communication channels.

must schedule tasks on the processors with the best balance of functionality and resource access. Power management considerations can become explicit in programs, such as putting an idle functional unit into a reduced power state. Moreover, as system sizes continue to increase, reliability and resilience become significant issues. Can we detect and recover from hard, soft, and even silent data corruption? How do we restart calculations on systems with a mean time between interrupts measured in hours?

Compounding these technical challenges is the dire lack of parallelism experience among the general software developer community. The state of the tools available to developers makes this problem worse: threading libraries and primitives added to fundamentally serial languages are challenging to use. Hardware vendors have recognized these problems and are working on a variety of solutions. In the hardware itself, transactional memory could remove some of the challenges of thread programming, and innovations such as scout threads might provide more transparent performance increases.

Hardware vendors are also creating software solutions to address these problems, from new compiler technologies to libraries and language extensions. But for the most part, these tend to be proprietary, useful on only one vendor's hardware. One exception to this is the Khronos Group's OpenCL standard,⁷ which is an API for programming attached accelerator processors, such as general-purpose computations on graphics processing units (GPGPUs). Some companies have pursued long-term strategies such as establishing research labs at universities to directly tackle some of today's challenging problems and creating a stream of talented and experienced graduates. Vendors have also introduced forms of declarative programming into their tools to help programmers focus on what should happen, rather than how the computer should execute the task.

Although the intensity of these activities is encouraging and will certainly bring advances, they're not a sufficient solution for HPC. Most of this work focuses on programming a single chip or a desktop: large clusters of these complicated nodes don't command enough market share to warrant the investment. Another, somewhat subtle challenge for the HPC market is a forced change in programming languages. Hardware vendors have focused most of their investment for new software tools on C/C++ compilers. Although some of these developments will trickle down into Fortran tools, it's unlikely that Fortran will be well-suited to take advantage of the new hardware.

A Comprehensive Approach

Perhaps the greatest challenge that software developers face at this time can be simply termed as diversity. Hardware designers have provided a dizzying array of options, and each one could encourage several different programming approaches. Eventually, this period of rapid innovation will settle to a smaller set of stable technologies, but we don't have the luxury of waiting until that happens. Fortunately, the HPC community already has a tool that's flexible enough to confront most of our programming challenges in Roadrunner (www.lanl.gov/roadrunner).

At the top level, LANL configured Roadrunner as a relatively traditional cluster of clusters that uses InfiniBand for the interconnect and supports HPC-standard message-passing interface (MPI) communications. At the node level, however, Roadrunner becomes unique. Figure 1 provides a conceptual node schematic. The node's root—at least with respect to the network—is a blade server with two dual-core AMD Opteron processors. Attached to that are two accelerator blade servers based on the IBM PowerXCell 8i processor. This processor conforms to the Cell Broadband Engine architecture specification that Sony, Toshiba, and IBM created⁸ and is itself a heterogeneous multicore chip.

The PowerXCell 8i has a general-purpose core (the PowerPC processor element [PPE]) and eight short-vector engines (the synergistic processing element [SPEs]). The PPE has a traditional two-level cache, whereas the SPEs use a programmer-managed local store: 256 Kbytes for program text and data. The programmer explicitly moves data from main memory to the local store using asynchronous communication calls, allowing truly overlapped communication and computation. Each SPE contains 128-bit registers and a statically scheduled, in-order, dual-issue instruction pipeline and can achieve 12.8 Gflops/s in double

precision. This gives each Roadrunner node a more than 400 Gflops/s peak.

Roadrunner's design lets developers gracefully transition their existing applications to the new architecture: MPI applications can run unchanged on the Opteron cluster of clusters. Although this makes a nice starting point for developers, such applications can access only a small fraction (3.5 percent) of the machine's peak performance. Accelerating these applications requires identifying portions of the code to move to the PowerXCell processors. As provisioned, Roadrunner has equal numbers of PowerXCell processors and Opteron cores and the same amount of memory available to each. This admits the conceptually simple pairing of one Opteron core with one PowerXCell processor. Developers can incrementally accelerate their applications by moving more functions from the Opteron to the PowerXCell.

Moving a function to the SPE can seem a daunting task at first. The small local store and the need to explicitly move data between it and the main memory are constraints that haven't been relevant for some time in general-purpose programming. Confronting the data structure and alignment implications of the vector-only SPE instruction set can shake the confidence of even the most skilled scalar instruction programmers. The key observation to overcoming these challenges is simply that the SPE makes many operations explicit for programmers. It's not that a GPP isn't doing these same tasks, it just has more hardware with which to do them. SPE programmers must be cognizant of data locality and will see data motion in explicit instructions. But awareness of these issues is exactly what programmers require to achieve high performance on cache-based GPPs! We invariably obtain performance increases on our GPPs when we apply the optimization lessons learned from porting code to the SPE. This is a substantial benefit that will outlive any particular architecture.

Although accelerating applications through function off-load provides an expedient path to performance, developers realize the machine's research potential when they start with a fresh look at application design. Rather than look at the SPEs as Opteron accelerators, we can reverse the model and think of the Opterons as communication managers for the PowerXCell processors. Although it's easiest to think of every SPE running the same instructions on different portions of data, they're independently programmable and can communicate with each other directly. This admits a variety of streaming and process ganging models. More opportunities arise when we discard the Opteron-PowerXCell pairing and find ways to distribute the work of

ADVERTISER INFORMATION • JULY/AUGUST 2009		
Advertiser AAPT 2009	Page 49	
Advertising Personnel		
Marion Delaney IEEE Media, Advertising Dir. Phone: +1 415 863 4717 Email: md.ieeemedia@ieee.org		
Marian Anderson Sr. Advertising Coordinator Phone: +1 714 821 8380 Fax: +1 714 821 4010 Email: manderson@computer.org		
Sandy Brown Sr. Business Development Mgr. Phone: +1 714 821 8380 Fax: +1 714 821 4010 Email: sb.ieeemedia@ieee.org		
Advertising Sales Representatives	Email: dg.ieeemedia@ieee.org	Email: jmd.ieeemedia@ieee.org
Recruitment:	Northwest/Southern CA Tim Matteson Phone: +1 310 836 4064 Fax: +1 310 836 4067 Email: tm.ieeemedia@ieee.org	US Central Darcy Giovingo Phone: +1 847 498 4520 Fax: +1 847 498 5911 Email: dg.ieeemedia@ieee.org
Mid Atlantic Lisa Rinaldo Phone: +1 732 772 0160 Fax: +1 732 772 0164 Email: lr.ieeemedia@ieee.org	Japan Tim Matteson Phone: +1 310 836 4064 Fax: +1 310 836 4067 Email: tm.ieeemedia@ieee.org	US West Lynne Stickrod Phone: +1 415 931 9782 Fax: +1 415 931 9782 Email: ls.ieeemedia@ieee.org
New England John Restchack Phone: +1 212 419 7578 Fax: +1 212 419 7589 Email: j.restchack@ieee.org	Europe Hilary Turnbull Phone: +44 1875 825700 Fax: +44 1875 825701 Email: impress@impressmedia.com	Europe Sven Anacker Phone: +49 202 27169 11 Fax: +49 202 27169 20 Email: sanacker@intermediapartners.de
Southeast Thomas M. Flynn Phone: +1 770 645 2944 Fax: +1 770 993 4423 Email: flyntom@mindspring.com	Product:	
Midwest/Southwest Darcy Giovingo Phone: +1 847 498 4520 Fax: +1 847 498 5911	US East Joseph M. Donnelly Phone: +1 732 526 7119	

one process across all 40 processors on the node. By treating the node as a many-core processor, developers can better understand the mismatch between high on-processor performance and the slow access to off-processor resources that we'll see in many-core designs.

Developers have exploited all of these design techniques when creating high-performance applications for Roadrunner. As a result of a peer-reviewed competition, LANL awarded time on Roadrunner to 10 research teams for a variety of open-science applications. These applications ran the gamut of scientific fields (and scales!), from simulations of cellosomes and viral phylogenetics to supernovae light curves and a large-scale structure of the universe. In addition to the direct scientific contributions that these teams have made, we're studying the application development process in each team to better inform the next generation of application and tool developers.

Bringing a new large-scale computational resource online takes considerable time and effort. This fact alone buffers the HPC community from the most rapid technological changes, as technology choices are often made well in advance of system delivery. Although this provides some continuity for the HPC community, we shouldn't be complacent to the changes occurring in the broader market. Computer technology is changing, and although we can't predict or prescribe which technology path will eventually dominate, we can be assured that we'll be programming differently in the future. If nothing else, the lines of code dedicated to explicit control of memory, functional unit power, resilience, and data communication will soon outnumber the lines of code doing the real computation.

To make this transition as smooth as possible, we must start by preparing ourselves. As you design an application, think about the implications of running on heterogeneous or hybrid architectures. Try to maximize the short-vector and multithreaded-processor uses that we have today, without assuming that a compiler will take care of this for you. Think about the costs of moving data around the system, whether that's between local memory and a processor or between nodes in a parallel system.

The next step is to experiment and add to the community's body of knowledge. Write applications for alternative processors, such as FPGAs or Cell processors. Learn to program a small, accelerated system, such as a workstation with a GPGPU or a cluster of Sony PlayStation consoles. Experiment with functional programming lan-

guages, such as Clojure or Haskell, to see how this class of languages can provide a powerful abstraction from the hardware. These sorts of investments prepare us for the future, but they can also pay dividends in terms of better utilization of today's technology.

Finally, we must engage the broader community to ensure that standards and tools support HPC needs. People are just beginning to consider how to develop tools that can alleviate some of the new burdens placed on programmers—now's the time to share the experience we've gained from years of programming parallel systems. As the industry struggles with the rapid rise of parallel computing, we can act as mentors, helping avoid the mistakes that we've made, while looking for the insights that will come from a fresh perspective on our long-standing problems.



References

1. *Computing in Science & Eng.*, special issue on novel architectures, Nov./Dec. 2008; www2.computer.org/portal/web/csdl/magazines/cise#3.
2. J.L. Manferdelli, "The Many-Core Inflection Point for Mass Market Computer Systems," *CTWatch Quarterly*, vol. 3, no. 1, 2007; www.ctwatch.org/quarterly/articles/2007/02/the-many-core-inflection-point-for-mass-market-computer-systems.
3. S. Cotofana and S. Vassiliadis, "On the Design Complexity of the Issue Logic of Superscaler Machines," *Proc. 24th Euromicro Conf.*, IEEE CS Press, 1998, pp. 227–284.
4. J. Kahle et al., "Introduction to the Cell Multiprocessor," *IBM J. Research and Development*, vol. 49, nos. 4–5, 2005, pp. 589–694.
5. K. Mistry et al., "A 45nm Logic Technology with High-k+Metal Gate Transistors, Strained Silicon, 9 Cu Connect Layers, 193 nm Dry Patterning, and 100% Pb-Free Packaging," *Proc. Int'l Electron Devices Meeting, (IEDM 2007)*, IEEE CS Press, 2007, pp. 247–250.
6. Y. Tsividis, *Operation and Modeling of the MOS Transistor*, 2nd ed., Oxford Univ. Press, 2003.
7. A. Munshi, ed., "The OpenCL Specification," v. 1.0, revision 29, 2008; www.khronos.org/registry/cl/specs/opencl-1.0.29.pdf.
8. IBM, "Cell Broadband Engine Architecture," v. 1.02, 2007; [www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/1AEEE1270EA2776387257060006E61BA/\\$file/CBEA_v1.02_11Oct2007_pub.pdf](http://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/1AEEE1270EA2776387257060006E61BA/$file/CBEA_v1.02_11Oct2007_pub.pdf).

Paul Henning is a research scientist at Los Alamos National Laboratory (LANL). His research interests include high-performance computing and domain-specific languages for scientific computing. Henning has a PhD in computer science from the University of Iowa. He's a member of SIAM and the ACM. Contact him at phenning@lanl.gov.

Andrew B. White Jr. is the deputy associate director for theory, simulation, and computation at LANL. His research interests include high-performance computing, reliability, and resiliency. White has a PhD in applied mathematics from the California Institute of Technology. Contact him at abw@lanl.gov.