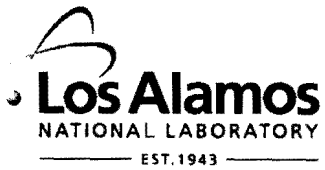
LA-UR- 09-00927

Approved for public release;
distribution is unlimited.

Title:	Trailblazing with Roadrunner
Author(s):	Paul Henning Andrew B. White, Jr.
Intended for:	Computing in Science and Engineering



Form 836 (7/06)

Trailblazing with *Roadrunner*

Paul Henning and Andrew B. White, Jr.
Los Alamos National Laboratory

In June 2008, a new supercomputer broke the petaflop/s performance barrier, more than doubling the computational performance of the next fastest machine on the Top500 Supercomputing Sites list (http://top500.org). This computer, named *Roadrunner,* is the result of an intensive collaboration between IBM and Los Alamos National Laboratory, where it is 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 neck-and-neck 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 towards the goal of exascale computing, 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 about 2.5 MW for 1.45 petaflop/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 2008 issue of *CiSE* [1] provided a glimpse into how computational scientists are adapting to and exploiting a variety of novel architectures. It can be done. 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 several decades ago. While there has been a shift to commodity clusters and changes in operating systems and communication infrastructure, the basic structure of our applications has not 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 the future of scientific computing as it is as a petaflop supercomputer. To support this claim, we'll start with a look at the changes occurring in processor design.

Processors are changing, for good reason

As in any business, the driving force in processor design is profit. 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. 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 re-think the general purpose processor. This section will illustrate some of these challenges to provide a rationale for the changes coming to hardware and software. This discussion follows that given by John Manferdelli [2].

One traditional approach to improving processor performance is simply to increase the clock frequencies of the processors. However, since power consumption is proportional to the clock frequency, the heat density per fixed area of processor chip increased to the point where simple cooling methods were inadequate. Designers are now lowering the clock frequencies of processors, 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 general-purpose processors since the late 1990s. These "superscalar" processors simultaneously execute multiple instructions on redundant functional units. In this scheme, the processor must automatically detect and avoid data dependencies between sequential instructions. Superscalar processors also employ pipelined, speculative and out-of-order execution, leading 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 promised performance improvements are ultimately limited by the nature of the instruction stream being executed. There is evidence that designers are moving back to more simple instruction scheduling models, and using the newly freed space for different purposes.

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, one can expect a single instruction to take ten or less clock cycles, while fetching data from main memory may take several hundred cycles. This problem is compounded when multiple instructions are being executed simultaneously. Traditionally, this difference has been alleviated by using larger hierarchies of fast cache memory, which acts as a bridge between the processor and the main system memory. However, this cache memory is expensive and

complex, and the deeper hierarchies appearing 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 is related to the increase in processor power consumption as the fabrication process size decreases. Companies are currently using a "45nm process," where a single transistor is approximately 6nm long and contains a dielectric layer that is 1.2nm thick. There are many complicated physical effects at this scale (cf. [4]), but the net effect is that transistors leak significant amounts of power. Advances in material sciences should help stem this problem, but the shrinking process size implies that there will soon be more transistors on a chip than we can afford to power simultaneously. It is expected that fine-grained power management features will appear in processors (or even to programmers), leading to heterogeneous performance across even homogeneous processors.

Even in the face of these significant challenges, processor designers must still meet the economic driver of providing a better user experience. Rather than pursuing even more complex single processors, they are now placing 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 single processors of even five years ago, but provide performance increases in aggregate. This trend shifts more work to the applications programmer. Not only do they have to make up the loss in single core performance through better optimization, but they also have to explore ways of parallelizing their applications to take advantage of more cores. While this is nothing new for the scientific computing community, it is a fundamental shift in the broader software industry. Additional software development challenges will be discussed in the next section.

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

Hardware changes will disrupt software development practices

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 program across functional units. Deep memory hierarchies, especially coupled with disjoint address spaces, require special attention to data motion costs. The short vector processors constrain data structure design. Developers must now 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 will have to evolve to new levels of complexity. One can no longer imagine that all processors have equal access to resources such as memory, network or I/O: tasks need to be scheduled on the processors with the best balance of functionality and resource access. Power management considerations may become explicit in programs, such as putting an idle functional unit into a reduced power state. And, 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 interrupt 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 may remove some of the challenges of thread programming, and innovations such as scout threads may provide more transparent performance increases.

Software solutions are also being developed, from new compiler technologies, to libraries and language extensions. For the most part, these tend to be proprietary solutions, useful on only one vendor's hardware. One exception to this is the *OpenCL* standard released by the Khronos Group [5]. This is an API for programming attached accelerator processors, such as GPGPUs. Companies are also pursing long-term strategies such as establishing research labs at universities to directly tackle some of today's challenging problems, as well as creating a stream of talented and experienced graduates. Vendors are also introducing some forms of "declarative" programming into their tools. This class of languages allows the programmer to focus on *what* should happen, rather than *how* the computer should execute the task.

While the intensity of these activities is encouraging and will certainly bring advances, they are not a sufficient solution for high-performance computing. Most of this work focuses on

programming a single chip or a desktop: large clusters of these complicated nodes do not command enough market share to warrant the investment. Another, somewhat subtle, challenge for the HPC market is a forced change in programming languages. The majority of the investment for new software tools is being focused on C/C++ compilers. While some of these developments will trickle down into Fortran tools, it is unlikely that Fortran will be well suited to take advantage of the new hardware.

Exploring the future with Roadrunner

Perhaps the greatest challenge that software developers face at this time can be simply termed "diversity". Hardware designers are providing a dizzying array of options, and each option may encourage several different programming approaches. Eventually, this period of rapid innovation will settle to 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 is flexible enough to confront most of our programming challenges in *Roadrunner*. This section contains only a very brief overview of the physical architecture of *Roadrunner*, preferring to concentrate on programming models. More details of the system can be found at http://www.lanl.gov/roadrunner.

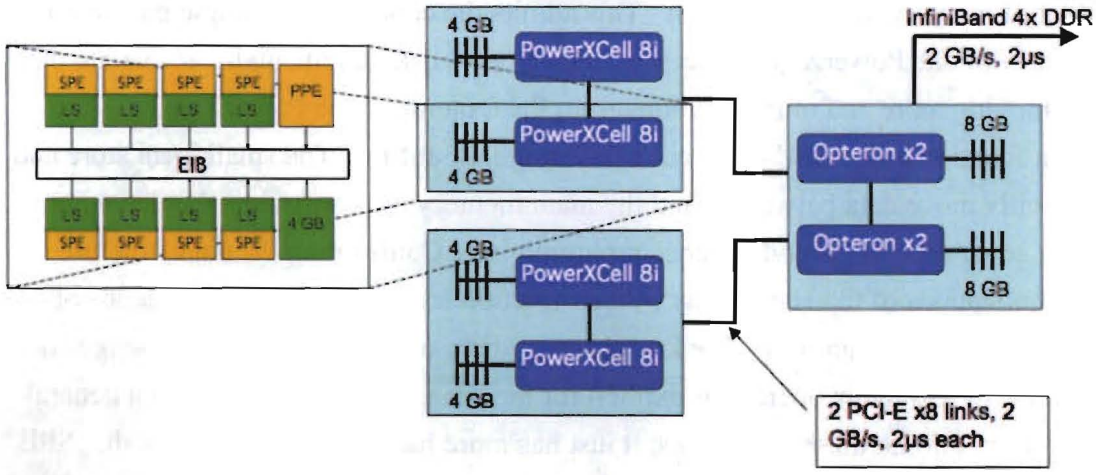


Figure 1: Schematic of a *Roadrunner* node. Communication channels are characterized by bandwidth and latency.

Overall, *Roadrunner* is configured as a relatively traditional cluster-of-clusters of node that uses InfiniBand for the interconnect and supports the HPC-standard MPI communications. At the node level, however, *Roadrunner* becomes unique. Figure 1 provides a conceptual schematic of the node. The "root" of the node, 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 created by Sony, Toshiba and IBM [6], and is itself a heterogeneous multicore chip as illustrated by the exploded view in the figure.

The PowerXCell 8i has a general-purpose core (the "PPE") and eight short-vector engines (the "SPEs"). The PPE has a traditional two-level cache, while the SPEs use a programmer-managed local store: 256 KB 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 128 128-bit registers and a statically scheduled, in-order, dual-issue instruction pipeline, and can achieve 12.8 Gflop/s in double precision. This gives each *Roadrunner* node over 400 Gflop/s peak.

The design of *Roadrunner* allows developers to gracefully transition their existing applications to the new architecture: MPI applications can run unchanged on the Opteron cluster-of-clusters. While this makes a nice starting point for developers, such applications can access only a small fraction (3.5%) of the peak performance of the machine. 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 and 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 have not 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 skilled scalar instruction programmers. The key observation to overcoming these challenges is simply that the SPE makes many operations explicit for the programmer. It is not that a general-purpose processor isn't doing these same tasks; it just has more hardware to do them with. SPE programs need to be cognizant of data locality, and see data motion in explicit instructions. But awareness of these issues is exactly what is needed to achieve high performance on cache-based general-purpose processors! We invariably obtain performance increases on our general-purpose processors when we apply the optimization lessons learned from porting the code to the SPE. This is a substantial benefit that will out-live any particular architecture.

While accelerating applications through function offload provides an expedient path to performance, the research potential of the machine is realized when developers start with a fresh look at application design. Rather than looking at the SPEs as accelerators for the Opterons, one can reverse the model and think of the Opterons as communication managers for the PowerXCell processors. While it is easiest to think of every SPE running the same instructions on different portions of data, they are independently programmable and can communicate with each other directly. This admits a variety of streaming and process ganging models. More opportunities arise when one discards the Opteron-PowerXCell pairing, and finds ways of distributing the work of one process across all forty 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 slow access to off-processor resources that will be seen in many-core designs.

All of these design techniques are being exploited in the development of high-performance applications for *Roadrunner*. As a result of a peer-reviewed competition, a number of research teams have been awarded time on *Roadrunner* for a variety of open-science applications. These applications cover a gamut of scientific fields (and scales!), from simulations of cellusomes and viral phylogenetics to supernovae light curves and the large-scale structure of the universe. In addition to the direct scientific contributions that will be made by these teams, we are studying the process of application development on each team, to better inform the next generation of application and tool developers.

Moving high-performance computing into the future

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 the delivery of the system. While this provides some continuity for the HPC community, we should not be complacent to the changes occurring in the broader market. Computer technology is changing, and while we cannot predict which technology path will become dominant, we can be assured that we will be programming differently in the future.

To make this transition as smooth as possible, we need to start by preparing ourselves. As you are designing an application, think about the implications of running on heterogeneous or hybrid architectures. Try to maximize the use of the short-vector and multithreaded processors 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 is between local memory and a processor, or between nodes in a parallel system.

The next step is to experiment and add to the body of knowledge in the community. 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®3 consoles. Experiment with functional programming languages, 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 also can pay dividends in terms of better utilization of today's technology.

Finally, we need to engage the broader community to ensure that standards and tools will support the needs of HPC. People are just beginning to consider how to develop tools that can alleviate some of the new burdens placed on the programmer; now is the time to share our experience 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 have made, while looking for the insights that will come from a fresh perspective on our long-standing problems.

References

1. *Computing in Science & Engineering*, 10, no. 6 (Nov-Dec 2008).
2. Manferdelli, John L. "The Many-Core Inflection Point for Mass Market Computer Systems," *CTWatch Quarterly* 3, no. 1 (February 2007), http://www.ctwatch.org/quarterly/articles/2007/02/the-many-core-inflection-point-for-mass-market-computer-systems.
3. Cotofana, Sorin and Vassiliadis, Stamatis. "On the Design Complexity of the Issue Logic of Superscaler Machines," Proceedings of the 24th Euromicro Conference 1 (August 1998): 227-284.
4. Tsividis, Yannis. *Operation and Modeling of the MOS Transistor*, 2nd ed. New York: Oxford University Press, 2003.
5. Munshi, Aaftab, ed., *The OpenCL Specification.* Version 1.0, Revision 29. http://www.khronos.org/registry/cl/specs/opencl-1.0.29.pdf
6. *Cell Broadband Engine Architecture.* Version 1.02. October 11, 2007.

Paul Henning *is a research scientist at Los Alamos National Laboratory. 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 is a member of SIAM and the ACM. Contact him at phenning@lanl.gov.*

Andrew B. White *is the deputy associate director for theory, simulation and computation at Los Alamos National Laboratory. White has a PhD in applied mathematics from the California Institute of Technology. Contact him at abw@lanl.gov.*