Systems research challenges: A scale-out perspective

T. Agerwala
M. Gupta

A scale-out system is a collection of interconnected, modular, low-cost computers that work as a single entity to cooperatively provide applications, systems resources, and data to users. The dominant programming model for such systems consists of message passing at the systems level and multithreading at the element level. Scale-out computers have traditionally been developed and deployed to provide levels of performance (throughput and parallel processing) beyond what was achievable by large shared-memory computers that utilized the fastest processors and the most expensive memory systems. Today, exploiting scale-out at all levels in systems is becoming imperative in order to overcome a fundamental discontinuity in the development of microprocessor technology caused by power dissipation. The pervasive use of greater levels of scale-out, on the other hand, creates its own challenges in architecture, programming, systems management, and reliability. This position paper identifies some of the important research problems that must be addressed in order to deal with the technology disruption and fully realize the opportunity offered by scale-out. Our examples are based on parallelism, but the challenges we identify apply to scale-out more generally.

Introduction

Computer system designs have been driven by emerging applications and the ever-growing demands of these applications. The computational and storage needs of workloads in several areas such as scientific and technical computing, games, digital media, and data analytics are growing exponentially. Some examples of system demands in various domains are the following:

- *Life sciences:* A simulation of ten microseconds of protein-folding for a single 92,224-atom system (the protein ApoA1) consumes about 4.6×10^{19} floating-point operations, requiring 53 days on a supercomputer delivering 10-teraflop/s sustained performance on the application. Studying the mechanisms of folding of such a protein requires conducting several of these simulations [1, 2].
- *Climate modeling:* Studying future climate change as a result of human activity using the Community Climate System Model (CCSM2) will require, for a multi-century integration at a target resolution of 20 km in the atmosphere and 10 km in the ocean, about 25 years of computation on a system sustaining 10-teraflop/s performance. Again, multiple ensemble simulations are needed for a meaningful study.
- *Digital entertainment:* Performing ray tracing in a game scene at a resolution of $1,024 \times 1,024$ pixels, at 50 frames per second, with a realistic requirement of 10 rays per pixel, requires greater than 2-teraflop/s performance, which is more than two orders of magnitude higher than the performance achievable with modern personal computers and is also beyond the announced capabilities of the new generation of game consoles being introduced into the market.
- *Data analytics:* Billions of documents are accessible from the Internet, occupying several petabytes of storage, with millions of new documents being added daily. Performing analytics to find patterns, trends, and relationships over all of that data requires capabilities beyond the reach of even the most powerful supercomputers today. For example, the Web Fountain project [3] at IBM Research involves adding a series of annotations to about five billion documents (representing the relevant part of the web-

0018-8646/06/$5.00 © 2006 IBM

accessible data of interest), requiring an estimated 4×10^{18} operations in aggregate which should be completed within a day in order to be useful.

Systems-level performance must meet the demands of applications, even though power dissipation is creating a fundamental discontinuity in the development of microprocessor technology. While technology scaling has allowed us to continue to increase the number of transistors on a chip, power densities have grown with every CMOS generation as designers have pursued higher operating frequencies. Given the lack of any other low-power and high-volume technology than CMOS, in the future, the frequency of microprocessors must grow at a much lower rate than in the past decade. Even with advances in base silicon, on-chip interconnects, and cooling technologies, microprocessor performance growth rates have slowed substantially. However, chip-level performance growth can still be sustained at historic levels through higher levels of on-chip functional integration. The fundamental reason for this is the nonlinear relationship between power and performance. It may be possible, for example, to decrease the power of a system tenfold with only a threefold reduction in performance. Higher performance at the chip level would then be obtained through higher levels of on-chip functional integration.

We believe that the disruption caused by power dissipation will affect all forms of computing systems. Scale-out systems represent a natural initial target for exploring solutions to this problem, given the high computational needs and the use of parallelism in their workloads. Systems management, reliability, and Amdahl's law[1] are challenges for scale-out systems. We require a holistic approach that considers all aspects of system design such as architecture, operating systems, compilers, and runtime systems, as well as workload characteristics and programming methodologies, in order to develop breakthrough solutions.

This paper presents some of the challenges, indicated throughout the text in italics, that must be addressed in order to realize the full benefit of scale-out. Many of these issues have long been recognized as research problems. The discontinuity in the development of microprocessor technology requires us to address these problems with a sense of urgency and in a holistic manner by considering all aspects of systems design.

Research challenges

Architecture

For certain workloads, we can exploit the power/performance tradeoff at the systems level, leading, in the limit, to massively scalable computers. The Blue Gene*/L (BG/L) project [4] represents a research experiment that pushed the limits of scale-out computing for a significant class of scientific computations. The BG/L system design was driven, at every level, with an almost maniacal focus on power efficiency. A design decision was made to use low-power embedded processors in the system. Furthermore, BG/L uses system-on-a-chip technology to integrate, along with two processor cores, powerful interconnection networks with special support for collective communications, a prefetch buffer (L2 cache), and embedded DRAM serving as the L3 cache in the compute node chip. A single rack of BG/L, which is air-cooled, contains 2,048 processors, with an aggregated peak performance of 5.7 teraflop/s. The target set at the beginning of the project was to develop a 65,536-node system, with a peak performance of 367 teraflop/s. [This system was successfully delivered to Lawrence Livermore National Laboratory (LLNL) in September 2005.]

BG/L employs a novel, hierarchical software architecture to support high levels of scalability and is an excellent example of the use of chip-level functional integration and massive parallelism to obtain an order of magnitude improvement in performance (as well as in performance per watt and performance per square foot of floor space) at the high end of supercomputing.

The building blocks of scale-out systems must also address rising power dissipation and the memory wall problem.[2] These considerations are driving us toward accelerators, multiple cores on a chip, and multithreading.[3] All of these architectural approaches require greater exploitation of parallelism in applications. Well-designed accelerators and offload engines can provide significantly improved power-efficient computing on certain classes of applications such as graphics, game physics, and network protocol processing, to name just a few. The Cell Broadband Engine** Processor [5] is a great example of a power-efficient programmable accelerator. Multiple on-chip power-efficient cores can provide similar advantages for multithreaded applications. We foresee, at least in the near future, much more diversity in chips and systems as system designers experiment with ways to satisfy the workload demands.

Architectural research must be driven by application needs in different domains. We must systematically identify the pertinent characteristics, especially the amount of inherent parallelism and data access patterns, for a wide range of emerging applications. The goal is to design

[1] Amdahl's law states that the performance benefits to scale-out are limited by the percentage of code that is sequential.

[2] By Little's law, the amount of concurrency needed to hide the latency of memory accesses will continue to increase as the gap between memory and processor speed grows. Since the memory latency is improving at a rate of only roughly 6% each year, the gap is projected to continue growing even as the increase in processor speed decreases from the historic rate of about 60% each year to about 20% each year.

[3] We are discussing a chip with multiple processors, supporting a shared-memory model, as a building block of a scale-out platform, although such a chip would normally be associated with scale-up computing.

"*balanced*" *systems consisting of modular, cost-effective building blocks and interconnects and to identify chip architectures that are* "*optimal*" *with respect to the number and type of accelerators, the number of cores on a chip, the degree of multithreading, and the organization and amount of on-chip cache*.

Scale-out systems in which the elements are chips that consist of multiple, possibly heterogeneous cores and accelerators, with standard interconnects, can allow designs to be customized to different classes of workloads, keeping development and test costs manageable while improving time to market. *To realize this benefit, the design automation research community must develop languages and tools that allow us to quickly design and build high-performance systems following an approach similar to that used today for System-on-a-Chip (SoC) designs*.

Improving the single-thread performance of workloads has become a significant challenge, given the technology discontinuity and diminishing returns from using additional transistors in complex processors. *Architects must focus more on innovative uses of multiple cores, such as architectural support for speculative parallelization or multithreading, with appropriate compiler assistance, to improve the performance of codes that have not been explicitly parallelized.*

Programming

For an arbitrary program written in a serial manner using a language such as C, C++, or even Fortran, it is hard for a compiler to automatically extract parallelism, although years of research in parallelizing compiler technology have led to limited success in some application domains [6]. This has led to informal standards such as OpenMP**, through which a programmer can provide directives for parallelization [7]. Most of the popular languages also allow concurrency to be expressed directly, either using libraries (e.g., using POSIX** threads with C) or as part of the core language (e.g., multithreading in Java**) [8]. However, managing concurrency explicitly is a difficult and error-prone task, and it is hard for programmers to reason about too many different concurrent threads of execution. Those factors often lead to artificial constraints on the degree of parallelism expressed in many applications. In other cases, issues such as serial steps in the computation, true data-sharing patterns, and synchronization costs limit the amount of parallelism that can be exploited.

Data parallelism is a popular form of parallelism exploited by many applications. Variants of this programming style include single program multiple data (SPMD), vectorization, and use of single instruction multiple data (SIMD) instruction sets; the parallelism may be expressed directly at the language level or via libraries, such as message passing interface (MPI) [9]. Communication costs and load imbalances can limit the degree of scale-out that can be exploited. Another problem relates to the fact that most scale-out platforms support multiple forms of parallelism, message-passing parallelism across nodes, and shared-memory parallelism (and possibly SIMD, in addition) within a node. The programming task is complicated by multiple levels of problem decomposition [10].

It is widely accepted that the difficulty of programming high-performance computing systems is a significant barrier to widespread adoption of supercomputers. Several emerging languages and programming environments provide higher levels of abstraction to the programmer, especially support for global address space. Examples include Global Arrays [11], UPC [12], Co-Array Fortran [13], and recently proposed languages such as Chapel and X10. Achieving both high productivity and high performance with the same language and programming style remains a significant challenge. *There is a need for new languages and programming environments that support high programmer productivity and effective exploitation of scale-out systems*.

In recent years, there has been a dramatic growth in the popularity of high-productivity environments (in a desktop setting) such as MATLAB** [14], which are often used for prototyping solutions in specific domains such as signal processing or financial modeling. However, the performance of programs in such environments is often orders of magnitude lower than that obtainable with mainstream programming models such as C++ and MPI. This forces the end users to translate those programs into conventional languages for use in production, leading to a loss of productivity. *We must support such domain-specific high-productivity programming environments with higher performance on parallel systems.*

Tools to debug parallel programs and to help in understanding the performance of parallel programs are not very mature [10]. Often they do not scale to large systems, for example, because the volume of data being collected or displayed becomes too large. It is particularly difficult for debuggers to reproduce bugs that are not deterministic. A significant challenge involving performance tools is to support monitoring at the right level of granularity, and to relate the monitoring information with the original program source code. Another challenge is to provide more assistance to the end user in diagnosing performance problems, based on the automated analysis of collected data. *We need better tools, which combine ease of use, effectiveness in helping to find problems, and scalability, to allow programmers to use large scale-out systems more productively*.

Compilers and runtimes

Compilers and runtime systems must help bridge the gap between higher-level programming models and the

growing complexity of underlying hardware. The efficient use of heterogeneous cores and/or accelerators to leverage their inherent power efficiency requires effective support from compilers and optimized libraries.

An important research area is language and compiler co-design to identify ways in which a programmer can convey relevant information at a high level (such as parallelism and identification of high-cost operations) and enable the compiler to generate efficient code for the accelerators or additional cores with suitable runtime system support.

Given the trend toward late binding of programming components, e.g., using dynamically linked libraries with static languages or using dynamic languages such as Java, there is a clear opportunity for further optimization at deployment time or runtime. For example, it may be possible to replace a heavyweight protocol, such as SOAP (simple object access protocol), in a web services application with a lighter-weight protocol such as RMI (remote method invocation) or direct method call, by exploiting the relationship between the target platforms on which the clients and server for that application are deployed. Another powerful optimization technique is to perform a search over the optimization space at deployment time; this has been used for successful deployment of libraries such as ATLAS [15] and FFT-W [16], and holds great promise if applied more broadly to software components. The research challenge in dynamic compilation is to keep the overhead of compilation low. A number of techniques, such as using higher levels of optimization only for hot sections of the code [17] or utilizing history information from prior runs [18], have been developed to help reduce these overheads.

Operating systems

The operating system provides the services for ensuring secure execution and good utilization of resources in the presence of failures, dynamic changes in workload demands, and potential security attacks. The ability to virtualize resources and dynamically add and remove resources is a powerful technique for dealing with both changing workload demands and failures [19]. True virtualization of all resources in a scale-out system, including processor, memory, network, and storage, at acceptable performance overheads remains a challenge. Automatically identifying when to move resources, either proactively or in a responsive manner, to meet the quality-of-service goals (with possibly differentiated service) for dynamically changing workloads, and transparent recovery from failures are associated system management challenges. The operating system policies related to virtual memory management, scheduling of processes and threads, and file I/O can have a considerable impact on application performance; examples include selection of a suitable page size for memory regions corresponding to different data structures and the use of a suitable caching and prefetching mechanism to support efficient file I/O operations. The policies may have to be adapted for different kinds of workloads. For example, parallel scientific workloads usually require a greater emphasis on reducing the jitters caused by scheduling of daemons on different nodes [20]. A multithreaded workload with high memory access costs requires greater emphasis on preserving data locality while scheduling threads [21]. *We need to develop autonomic features for operating systems that enable dynamic policy adaptation and choice of algorithmic parameters on the basis of monitoring of the runtime behavior of the workload and a suitable cost model.*

System management

The total cost of ownership of server systems is increasingly dominated by system management [22, 23] and the problem is worse for scale-out systems. Some of the activities that account for a significant part of these costs include software installation and configuration, system upgrades and change management, security management, and dealing with failures. The lack of a good repository of system information and inefficiencies in operations such as booting, mounting of file systems, and detection and isolation of failures can contribute further to the above costs. If the management costs grow linearly with the number of nodes, large scale-out systems become unattractive.

For BG/L, the team set a goal to simplify management and ensure that an order-of-magnitude increase in the number of nodes (over other high-end supercomputers) did not require a corresponding increase in the number of system administrators. The traditional firmware in BG/L has been replaced by software residing on the service node, which manipulates, via standard Internet Protocol (IP) network packets, remote translators (built using field-programmable gate arrays) to perform the control operations. The system management software uses scalable collective operations to broadcast common information across nodes (such as kernels and program executables) and to send node-specific "personality" information. Booting an entire 65,536-node system is completed in less than seven minutes (and has been further accelerated significantly with new prototype software), and parallel job launch on the full system is completed in less than 20 seconds. The computational core of BG/L is kept stateless to simplify system management, and software needed on the compute nodes for a job is "deployed" as part of the job launch. All system state [including machine configuration and historical information on jobs, environmental parameters, and RAS (reliability, availability, and serviceability) events] is stored in a database on the

service node, allowing a system administrator to access a tremendous amount of information relevant to the health of the machine using SQL queries.

Heterogeneous scale-out systems amplify the above system management problems. *Management of scale-out systems must be simplified so that management costs do not scale linearly (or even close to linearly) as the number of nodes increases. This will require a systems management architecture that supports scalability of management operations and systems that are self-configuring and self-healing.*

Most server workloads of interest involve complex interactions between different subsystems, both "vertically" (e.g., across a software stack comprising an application, J2EE** container, Java Virtual Machine, and operating system) [24] and "horizontally" (e.g., across different tiers of an e-business workload—web server, application server, and database server) [25, 26]. Modular scale-out architectures enable application-optimized subsystems, including accelerators, to be deployed as "appliances" in these tiers. *Effective workload management is needed to provide quality-of-service guarantees for the overall service requests based on end-to-end performance monitoring, analysis across the different tiers, and dynamic movement of resources.*

As software components have become increasingly complex, there has been a tendency to design them with many configuration "knobs" to enable customization. The interaction of several components in an overall system can lead to a combinatorial explosion in the number of possible configuration parameters. Erroneous settings of configuration parameters can lead to failures and performance problems. *Designing components with built-in autonomic features and integrating them in such a manner that the configuration parameters can be set automatically will greatly enhance productivity and simplify system management.*

As the number of nodes in a system increases, the mean time between failures (MTBF) decreases. Both hardware and software failures account for this decrease in reliability. On modern systems, software failures are often dominant. If system software fails once a year on a node, and the failures on different nodes of a system are independent, a node would be expected to fail once roughly every two hours in a 4,096-node system and once every eight minutes in a 65,536-node system. There is relatively little business incentive to reduce the error rate (after correction) in commodity hardware below a level of once in a few years [10]. Hence, even though commodity hardware is becoming increasingly powerful, large systems built with it can have significant reliability problems unless special features are added to enhance reliability.

In mission-critical scale-out systems, it is important to ensure that single failures do not bring down the entire system. This requires research on end-to-end systems design (including storage, servers, and networks), novel techniques for capturing state, error detection, dynamic system reconfiguration, and rapid checkpoint restart.

Algorithms and software scaling

Successful utilization of large scale-out systems requires innovations in algorithms and software. To illustrate, we describe some of the challenges the BG/L team faced in scaling applications to tens of thousands of processors and finally to more than a hundred thousand processors.[4] Note that BG/L was designed, from the beginning, to support scalability of applications, with features such as a hierarchical system software environment with low computational noise and simple, reliable software on compute nodes, balanced networks, special hardware and software support for collective communication operations, and low-latency communication for short messages.

Memory on the BG/L compute node is limited for several reasons (cost and power/performance). The resulting reduced memory/compute ratio is one of the major constraints in the porting of applications to BG/L. Given that BG/L systems have significant aggregate memory (currently, 512 GB in a rack), in many cases memory constraints on a single node were indicators of an underlying scalability problem with the application.

The presence of a serial component in the application is clearly an inhibitor to scalability, as shown by Amdahl's law. In the original version of UMT2K, a photon-transport code on an unstructured mesh, the mesh was partitioned in a serial step using the Metis library [27]. This partitioning method used a table dimensioned by the number of partitions squared. The table grew too large to fit into the memory of a BG/L node when the number of partitions exceeded approximately 4,000. This limitation was overcome by modifying the code to use a parallel implementation of Metis, and the code was successfully run on larger configurations, including the full 65,536-node system.

Some of the codes encountered the well-known problems of load imbalance and communication overhead. Some examples are given in the following paragraphs.

ParaDiS (Parallel Dislocation Simulator) is a code developed at LLNL for direct computation of the plastic strength of materials by tracking the simultaneous motion of millions of dislocation lines. Strong scaling runs

[4]Previous results on scaling of MPI applications on other platforms had necessarily been limited (by hardware existence) to fewer than ten thousand processors. Furthermore, many previous studies had shown problems with the scaling of applications to thousands of processors due to factors such as computational noise [20].

showed a speedup of 1.8 times when doubling the processor count from 4,096 to 8,192, and a speedup of 2.8 times in quadrupling the processor count to 16,384 [28]. Further analysis showed that these results (which still represented compute capability more than two orders of magnitude higher than previously achieved for the code) were limited by dynamic load imbalances in the code, which are now being addressed.

The Miranda code (a high-order hydrodynamics code) from LLNL initially had disappointing speedups from scaling to 16,384 nodes. However, tuning the performance of the `MPI_Alltoallv` collective communication operation (which now achieves more than 97% of the torus network bandwidth) led to excellent scaling [28]. The Qbox quantum molecular dynamics code from LLNL and HOMME climate modeling code from the National Center for Atmospheric Research (NCAR) had some scaling problems at counts of 16,384 and 32,768 nodes, which were resolved by mapping the processes better to the 3D torus topology of BG/L [28, 29]. On certain kinds of problems, the Raptor code (an Eulerian adaptive mesh refinement code) [28] had severe scaling problems at the 16,384-node level because of the overloading of networks. These issues were resolved by introducing a pacing capability in the MPI implementation on BG/L.

Some of the relatively new codes that were attempted on BG/L were designed for scaling to thousands of nodes; they incorporated algorithmic improvements such as hierarchical organization that allowed a single global all-to-all communication to be replaced with several all-to-all communications over smaller domains in parallel, and the use of multidimensional data distribution. Examples of such codes, which have worked well on BG/L, include Qbox (derived from older codes called Jeep and GP), CPMD, and ddcMD [28, 29]. These codes have been successfully scaled to 131,072 processors, representing scaling about two orders of magnitude higher than that previously achieved on any other platform. In particular, ddcMD and CPMD became the first set of codes to break the barrier of 100-teraflop/s sustained performance for a real application on any platform, respectively achieving 101 and 110 teraflop/s in production on different problems. Qbox has achieved a performance of more than 60 teraflop/s. For these applications, the crucial scaling step typically involved going beyond two thousand processors, since most inherent scaling limitations in the code showed up at that level. Overall, we found that several codes scaled successfully on BG/L to unprecedented levels of parallelism, ranging from tens of thousands to more than a hundred thousand processors, although in some cases the applications and system software had to be modified to eliminate specific bottlenecks. *To leverage the benefits of scale-out, researchers must develop innovative algorithms, libraries, and system software to systematically remove the Amdahl bottlenecks.*

Conclusions

We face a technology discontinuity due to power dissipation which is manifested by slowing growth in microprocessor frequency while the performance requirements of applications continue to grow at a faster rate. This creates an exciting challenge and opportunity for system designers to play an increasing role in delivering performance growth. We must accelerate the pace of innovation in architecture, programming languages, operating systems, compilers, and runtime systems. Furthermore, we must develop holistic approaches that consider the interactions among all of these components to arrive at true breakthrough solutions.

A key foundation for many of these innovations will be the exploitation of greater degrees of scale-out at multiple levels. There is a clear need for system designers to work with application developers to ensure that the applications exhibit high degrees of parallelism and to systematically remove barriers to scaling—performance, reliability, and availability, and ease of system management.

Acknowledgments

We wish to thank Bruce D'Amora, Jose Castanos, John Field, Alan Gara, Robert Germain, Daniel Gruhl, Eric Kronstadt, Jaime Moreno, and Bob Walkup for valuable discussions. We also thank the entire Blue Gene team for their technical direction and results, some of which were presented in this paper. The Blue Gene/L project has been supported and partially funded by the Lawrence Livermore National Laboratory on behalf of the United States Department of Energy under Lawrence Livermore National Laboratory Subcontract No. B517522.

References

1. J. C. Phillips, G. Zheng, S. Kumar, and L. V. Kale, "NAMD: Biomolecular Simulation on Thousands of Processors," *Proceedings of SC'2002: High Performance Networking and Computing*, Baltimore, 2002; see *http://sc-2002.org/program_tech.html.*
2. B. G. Fitch, R. S. Germain, M. Mendell, J. Pitera, M. Pitman, A. Rayshubskiy, Y. Sham, F. Suits, W. Swope, T. J. C. Ward, Y. Zhestkov, and R. Zhou, "Blue Matter, an Application

Framework for Molecular Simulation on Blue Gene," *J. Parallel & Distr. Computing* **63**, No. 7–8, 759–773 (2003).
3. D. Gruhl, L. Chavet, D. Gibson, J. Meyer, P. Pattanayak, A. Tomkins, and J. Zien, "How to Build a WebFountain: An Architecture for Very Large-Scale Text Analytics," *IBM Syst. J.* **43**, No. 1, 64–77 (2004).
4. Special issue on Blue Gene, *IBM J. Res. & Dev.* **49**, No. 2/3 (2005).
5. H. P. Hofstee, "Power Efficient Processor Architecture and the Cell Processor," *Proceedings of the 11th International Symposium on High Performance Computer Architecture*, IEEE Computer Society, San Francisco, 2005, pp. 258–262; see *http://doi.ieeecomputersociety.org/10.1109/HPCA.2005.26*.
6. R. Eigenmann, J. Hoeflinger, and D. Padua, "On the Automatic Parallelization of the Perfect Benchmarks," *IEEE Trans. Parallel & Distr. Syst.* **9**, No. 1, 5–23 (1998).
7. OpenMP: Simple, Portable, Scalable SMP Programming; see *http://www.openmp.org*.
8. J. Gosling, B. Joy, G. Steele, and G. Bracha, *The Java™ Language Specification*, Third Edition, Addison-Wesley Professional, Indianapolis, June 2005.
9. M. Snir, S. Otto, S. H. Lederman, D. Walker, and J. Dongarra, *MPI: The Complete Reference*, Vol. 1, MIT Press, Cambridge, MA, 1996.
10. S. L. Graham, M. Snir, and C. A. Patterson, Eds., *Getting Up to Speed: The Future of Supercomputing*, National Academies Press, Washington, DC, 2004.
11. J. Nieplocha, R. J. Harrison, and R. J. Littlefield, "Global Arrays: A Portable Shared Memory Model for Distributed Memory Computers," *Proceedings of the 1994 IEEE Supercomputing Conference*, 1994, pp. 340–349.
12. W. Carlson, J. M. Draper, D. E. Culler, K. Yelick, E. Brooks, and K. Warren, "Introduction to UPC and Language Specification," *Technical Report CCS-TR-99-157*, IDA (Institute for Defense Analysis) Center for Computing Sciences, Alexandria, VA, 1999.
13. R. Numrich and J. Reid, "Co-Array Fortran for Parallel Programming," *ACM Fortran Forum* **17**, No. 2, 1–31 (1998).
14. C. Moler, "Numerical Computing with MATLAB"; see *http://www.mathworks.com/moler*.
15. R. C. Whaley and J. Dongarra, "Automatically Tuned Linear Algebra Software," *Proceedings of the ACM/IEEE SC 1998 Conference (SC'98): High Performance Networking and* Computing, 1998, p. 38.
16. M. Frigo, "A Fast Fourier Transform Compiler," *Proceedings of the 1999 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'99)*, Atlanta, May 1999, pp. 169–180; see *http://www.fftw.org/pldi99.pdf*.
17. M. Arnold, S. Fink, D. Grove, M. Hind, and P. Sweeney, "Adaptive Optimization in the Jalapeño JVM," *ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'00)*, Minneapolis, October 2000, pp. 47–65.
18. M. Serrano, R. Bordawekar, S. Midkiff, and M. Gupta, "Quasi-Static Compilation for Java," *ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'00)*, Minneapolis, October 2000, pp. 66–82.
19. J. Jann, L. M. Browning, and R. S. Burugula, "Dynamic Reconfiguration: Basic Building Blocks for Autonomic Computing on IBM pSeries Servers," *IBM Syst. J.* **42**, No. 1, 29–37 (2003).
20. F. Petrini, D. Kerbyson, and S. Pakin, "The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8,192 Processors of ASCI Q," *Proceedings of the 2003 ACM/IEEE Conference on Supercomputing*, Phoenix, November 2003, p. 55.
21. M. S. Squillante and E. D. Lazowska, "Using ProcessorCache Affinity Information in Shared-Memory Multiprocessor Scheduling," *IEEE Trans. Parallel & Distr. Syst.* **4**, No. 2, 131–143 (1993).
22. P. Horn, "Autonomic Computing: IBM's Perspective on the State of Information Technology," October 15, 2001; see *http://www.research.ibm.com/autonomic/manifesto/*.
23. J. Kephart and D. Chess, "The Vision of Autonomic Computing," *Computer* **36**, No. 1, 41–50 (2003).
24. C. R. Attanasio, J.-D. Choi, N. Dubey, K. Ekanadham, M. Gupta, T. Inagaki, K. Ishizaki, J. Jann, R. D. Johnson, T. Nakatani, I. Park, P. Pattnaik, M. Serrano, S. E. Smith, I. Steiner, and Y. Shuf, "Whole-Stack Analysis and Optimization of Commercial Workloads on Server Systems," *Proceedings of the Network and Parallel Computing: IFIP International Conference,* Wuhan, China, October 2004; *Lecture Notes in Computer Science* **3222**, 5–8 (2004).
25. J. Aman, C. K. Eilert, D. Emmes, P. Yocom, and D. Dillenberger, "Adaptive Algorithms for Managing a Distributed Data Processing Workload," *IBM Syst. J.* **36**, No. 2, 242–283 (1997).
26. A. Dan, H. Ludwig, and G. Pacifici, "Web Services Differentiation with Service Level Agreements" IBM developerWorks, May 2003; see *http://www.ibm.com/developerworks/library/ws-slafram/*.
27. Metis home page; see *http://glaros.dtc.umn.edu/gkhome/views/metis/*.
28. G. Almasi, G. Bhanot, A. Gara, M. Gupta, J. Sexton, B. Walkup, V. V. Bulatov, A. W. Cook, B. R. de Supinski, J. N. Glosli, J. A. Greenough, F. Gygi, A. Kubota, S. Louis, T. E. Spelce, F. H. Streitz, P. L. Williams, R. K. Yates, C. Archer, J. Moreira, and C. Rendleman, "Scaling Physics and Material Science Applications on a Massively Parallel Blue Gene/L System," *Proceedings of the 19th International Conference on Supercomputing*, Cambridge, MA, June 2005, pp. 246–252.
29. G. Almasi, G. Bhanot, D. Chen, M. Eleftheriou, B. Fitch, A. Gara, R. Germain, J. Gunnels, M. Gupta, P. Heidelberg, M. Pitman, A. Rayshubskiy, J. Sexton, F. Suits, P. Vranas, B. Walkup, C. Ward, Y. Zhestkov, A. Curioni, W. Andreoni, C. Archer, J. Moreira, R. Loft, H. Tufo, T. Voran, and K. Riley, "Early Experience with Scientific Applications on the Blue Gene/L Supercomputer," *Proceedings of the Euro-Par 2005 Parallel Processing: 11th International Euro-Par Conference* Lisboa, Portugal, 2005; *Lecture Notes in Computer Science* **3648**, 560–570 (2005).

Received November 2, 2005; accepted for publication November 21, 2005; Internet publication February 22, 2006

Tilak Agerwala *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (tilak@us.ibm.com)*. Dr. Agerwala is vice president, systems, at IBM Research. His primary area of research is high-performance computing systems. Dr. Agerwala is responsible for all IBM advanced systems research programs in servers and supercomputers. He received his Ph.D. degree in electrical engineering from The Johns Hopkins University. Dr. Agerwala is a Fellow of the Institute of Electrical and Electronics Engineers and a member of the Association for Computing Machinery.

Manish Gupta *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (mgupta@us.ibm.com)*. Dr. Gupta is a Research Staff Member and Senior Manager of the Emerging System Software Department at the IBM Thomas J. Watson Research Center. His group has developed system software for the Blue Gene/L machine and conducts research on software issues for high-performance server systems. Since 1992, when he received a Ph.D. degree in computer science from the University of Illinois at Urbana–Champaign, he has worked in the IBM Research Division. Dr. Gupta has coauthored several papers in the areas of high-performance compilers, parallel computing, and high-performance Java Virtual Machines.