Blue Gene/L programming and operating environment

J. E. Moreira
G. Almási
C. Archer
R. Bellofatto
P. Bergner
J. R. Brunheroto
M. Brutman
J. G. Castaños
P. G. Crumley
M. Gupta
T. Inglett
D. Lieber
D. Limpert
P. McCarthy
M. Megerian
M. Mendell
M. Mundy
D. Reed
R. K. Sahoo
A. Sanomiya
R. Shok
B. Smith
G. G. Stewart

*With up to 65,536 compute nodes and a peak performance of more than 360 teraflops, the Blue Gene®/L (BG/L) supercomputer represents a new level of massively parallel systems. The system software stack for BG/L creates a programming and operating environment that harnesses the raw power of this architecture with great effectiveness. The design and implementation of this environment followed three major principles: **simplicity**, **performance**, and **familiarity**. By specializing the services provided by each component of the system architecture, we were able to keep each one simple and leverage the BG/L hardware features to deliver high performance to applications. We also implemented standard programming interfaces and programming languages that greatly simplified the job of porting applications to BG/L. The effectiveness of our approach has been demonstrated by the operational success of several prototype and production machines, which have already been scaled to 16,384 nodes.*

Introduction

The Blue Gene*/L supercomputer has been developed by IBM in collaboration with Lawrence Livermore National Laboratory (LLNL). Several installations of Blue Gene/L (BG/L) are currently planned, both in the United States and abroad. The flagship system, with 65,536 compute nodes and more than 360 teraflops of peak computing power, will be located at LLNL in Livermore, California. With its extreme scalability and raw performance, BG/L represents a new level of massively parallel systems. The system software team faced a major challenge in designing and implementing a programming and operating environment that could harness the power of this new architecture.

In designing the BG/L system software, we followed three major principles: *simplicity*, *performance*, and *familiarity*. Because we targeted BG/L primarily for scientific computations, we kept the system software simple for ease of development and to enable high reliability. For example, we impose a simplifying requirement that the machine be used only in a strictly space-sharing mode—only one (parallel) job can run at a time on a BG/L partition. Furthermore, we support only one thread of execution per processor. Another major simplification is the preclusion of demand paging support in the virtual memory system, thus limiting the virtual memory available per node to the physical memory size.

These simplifications lead directly to performance benefits that allowed us to take advantage of hardware features and deliver a high-performance system with no sacrifice in stability and security. For example, strict space-sharing enables us to use efficient, user-mode communication without protection problems. It also ensures that there is always a dedicated processor behind each application-level thread, which leads to more deterministic execution and higher scalability. The simplified virtual memory system ensures that there are no page faults or translation lookaside buffer misses during program execution on the compute nodes, leading to higher and more deterministic performance.

Finally, we developed a programming environment based on familiar languages and libraries without penalizing simplicity or performance. This adherence to language and Message Passing Interface (MPI) library standards has enabled the porting of several large scientific applications to BG/L with relatively modest efforts.

Critical to our system software strategy (and to the whole project strategy) is the concept of specialization

0018-8646/05/$5.00 © 2005 IBM

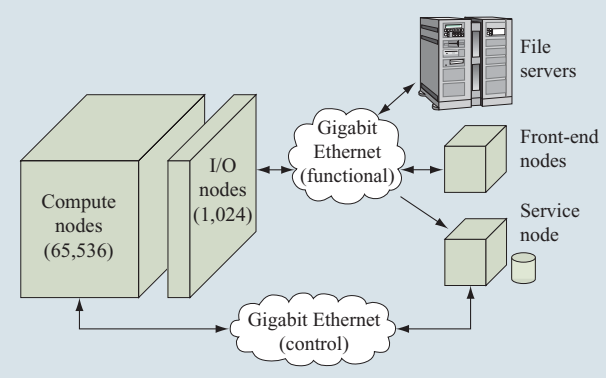

Figure 1

High-level architectural view of a complete Blue Gene/L system.

of services by different components of the system architecture. A complete BG/L system consists of a *computational core*, a *file server*, a set of *front-end nodes*, and a *service node*. The computational core is the BG/L machine proper, whose hardware was developed entirely from the ground up to support efficient execution of massively parallel message-passing applications. The computational core consists of a large number of compute nodes (65,536 in the case of LLNL) and a somewhat smaller number of input/output (I/O) nodes (1,024 in the case of LLNL). The file server, front-end nodes, and service node are commercially available machines that respectively provide file services, program development services, and control services to the computational core. This separation and specialization of services allowed us to optimally design the BG/L computational core for the execution of parallel applications.

Our system software strategy reflects the system organization described above in many of its aspects. To implement the system software stack, we leveraged existing components in support of our three major principles—simplicity, performance, and familiarity—while judiciously implementing critical components from a clean slate. For example, we use standard IBM XL compilers for PowerPC*, thus providing optimizing solutions for Fortran, C, and C++. The compilers run on the front-end nodes and are augmented with a back end that takes advantage of the new floating-point architecture in our chips. Similarly, we use Linux** on the I/O nodes to provide a rich functional interface to the machine, while writing our own kernel for the performance-critical compute nodes. Our MPI solution leverages the MPICH2 library [1] from Argonne National Laboratory (ANL) to produce an MPI implementation that exploits the communication hardware of our compute nodes. The control system for the core runs on the service node and uses a commercially available database, IBM DB2*, as a repository for static and dynamic system states. It communicates with the core through a special-purpose control network with proprietary protocols.

The validity and effectiveness of our approach to a programming and operating environment for BG/L has been demonstrated by success with several prototypes: a 4,096-compute-node system operating at 500 MHz (pass 1 chips), a 2,048-compute-node system operating at 700 MHz (pass 2 chips), and a 16,384-compute-node system operating at 700 MHz (pass 2 chips). The last prototype received the award for number one position in the TOP500 list [2] of the most powerful computers in the world when it achieved more than 70 Tflops in the Linpack benchmark. We also successfully ported a variety of production applications to BG/L, including SAIC** Adaptive Grid Eulerian (SAGE), FLASH [3], and Parallel Dislocation Simulator (ParaDiS) [4]. (These applications are discussed below in the experience section.) Results from additional benchmarks and applications in BG/L are presented in [5]. The porting efforts typically took only a few days each, and the applications displayed good performance and scalability on BG/L. In late 2004, IBM also completed the installation of the first phase (16,384 compute nodes) of the LLNL machine.

This paper is organized as follows. We describe the overall organization of the BG/L system, explaining the roles of the different components from a system software perspective. We then present the system software components for the I/O nodes and compute nodes. (That is, the software components for the computational core.) This is followed by sections describing the system software components for the front-end nodes and for the service node. After presenting a brief summary of the experience we have had operating the prototype machines, we present our conclusions.

Overall organization

A high-level architectural view of a complete BG/L system is shown in **Figure 1**. The BG/L machine proper is the computational core of the system, comprising 65,536 compute nodes and 1,024 I/O nodes. A Gigabit Ethernet (functional) network connects the computational core to a service node (for control of the machine), front-end nodes (where users compile, submit, and interact with their jobs), and parallel file servers. The service node also connects to the computational core through a separate (control) Gigabit Ethernet network that is used for direct manipulation of the hardware, as explained below. The computational core of 65,536 compute nodes is partitioned into 1,024 logical processing sets, called *psets*. Each pset consists of one I/O node running Linux and 64 compute nodes running a custom compute node kernel

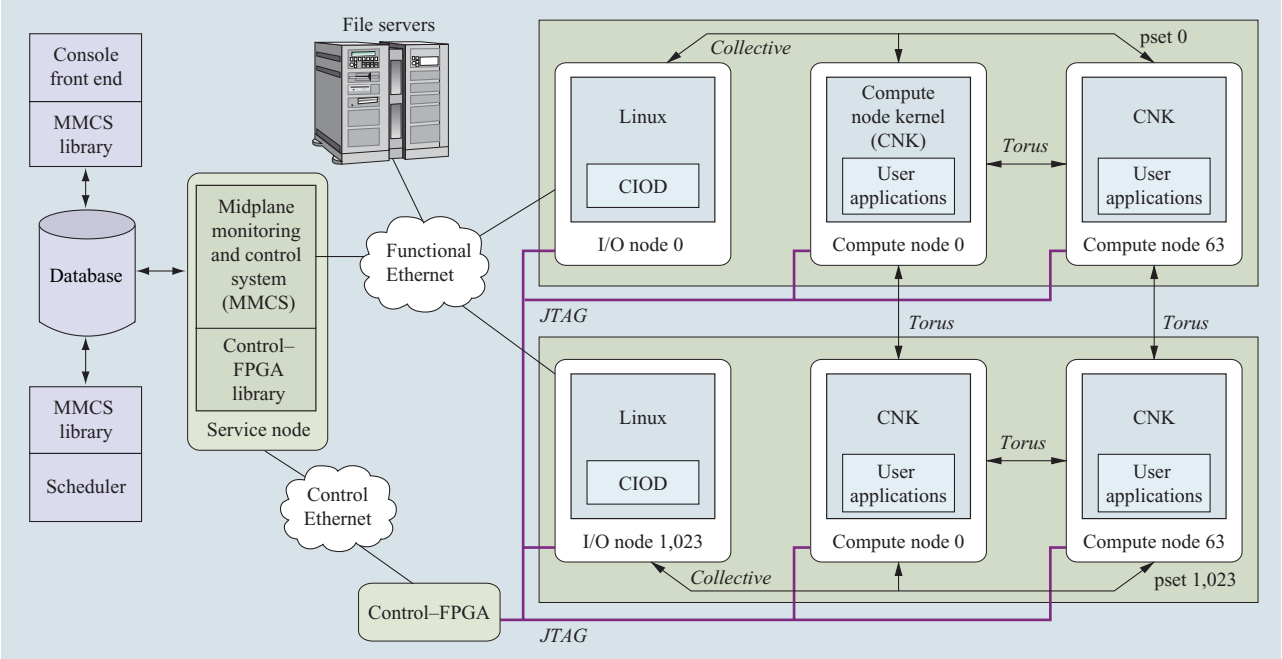

Figure 2

High-level view of the Blue Gene/L system software architecture.

(CNK). Psets are not physical entities in the architecture. They are assembled logically from the compute and I/O nodes of a partition by assigning compute nodes to a particular I/O node. There is a certain degree of flexibility in assigning nodes to a pset, and their configuration is part of machine setup. In particular, alternative configurations are possible, with a ratio of I/O-to-compute nodes from 1:8 to 1:128. That is, psets can be as small as eight compute nodes and as large as 128 compute nodes (conditional upon appropriate hardware being present).

From the system software perspective, the machine can be divided into three major components: a *computational volume*, a *functional surface*, and a *control surface*, each with its dedicated hardware for execution. We note that this approach of specialized hardware and software functions is similar to that adopted in ASCI Red [6]. A high-level view of the BG/L system software architecture is shown in **Figure 2**.

The computational volume is implemented by kernels and runtime libraries for the compute nodes that support the execution of user applications. The compute nodes of a pset are viewed as computational engines attached to their I/O node, which acts as the head of the pset.

The functional surface supports application compilation, job launch and control, application debugging, and I/O operations of running applications. The I/O nodes, front-end nodes, and service node all run system software that implements this functional surface and exposes the machine to the outside world. All interactions of the outside world to and from the computational processes running on the compute nodes go through the I/O nodes and the functional Gigabit Ethernet network.

The control surface is implemented exclusively in the service node. Operations such as machine booting, monitoring of environmental data (temperature, voltages), and critical error reporting are performed through the control surface. A single multiprocessor service node implements all of the control services for a 65,536-compute-node system, including services that would be performed by embedded service processors in conventional machines.

System software for the I/O nodes

The I/O nodes run Linux and play two roles in BG/L. First, they are responsible for implementing job launch and control in their respective psets. Second, they are also the primary offload engines for most system services required by running applications. User code never executes directly on the I/O nodes.

The Linux kernel that executes in the I/O nodes (currently Version 2.4.19) is based on a distribution for IBM PowerPC 440GP (PPC440) processors. Although

BG/L uses standard PPC440 cores, the overall chip and system design required changes in the booting sequence, interrupt management, memory layout, floating-point unit (FPU) support, and device drivers of the standard Linux kernel. There is no basic I/O system (BIOS) in the BG/L nodes; thus, the configuration of a node after power-on, as part of the initial program load (IPL), is initiated by the service node through the control network. We modified the interrupt and exception-handling code to support the custom BG/L interrupt controller (BIC). The implementation of memory management services in the Linux kernel for the I/O nodes remaps the torus and collective network devices of BG/L to user space. The kernel also supports the new Gigabit Ethernet media access controller (EMAC4) through the appropriate device drivers and has been extended to save and restore the double FPU registers (specific to BG/L) in each context switch.

The nodes in the BG/L machine are diskless; thus, the initial root file system is provided by a ramdisk loaded into the memory of an I/O node during IPL time. The ramdisk contains shells, simple utilities, shared libraries, and network clients such as `ftp` and `nfs`. After the Linux kernel boots with the ramdisk, it can use the `nfs` client to mount file systems from the file servers.

Because the level 1 (L1) caches of the two processors in an I/O node are not coherent, the current version of Linux runs on only one of the PPC440 cores. The second central processing unit (CPU) is captured at boot time in an infinite loop, and stays out of the way. We are still evaluating mechanisms to leverage the second processor in an I/O node, pursuing two strategies: symmetric multiprocessor (SMP) mode and virtual device mode. We have successfully compiled an SMP version of the kernel after reimplementing all required interprocessor communication mechanisms, because the BG/L BIC is not compliant with the OpenPIC Standard [7]. In this mode, the L1 caches are disabled in kernel mode and processes have affinity to one CPU. Forking a process in a different CPU requires additional parameters to the system call. The performance and effectiveness of this solution is still an open issue. A second, more promising mode of operation runs Linux in one of the CPUs, while the second CPU implements a virtual network interface. In this scenario, the torus and collective devices are handled by the second processor and are not directly visible to the Linux kernel. Transfers between the two processors appear as virtual direct memory access transfers.

Program launch, signaling, termination, and file I/O are accomplished using point-to-point messages between the compute node and its I/O node over the collective network. This functionality is provided by the control and I/O daemon (CIOD) running in the I/O nodes. CIOD provides job control and I/O management on behalf of all of the compute nodes in the pset. When launching a job, CIOD loads the executable code on all of the compute nodes of the pset and then sends a start message to the CNKs in those nodes. After the processes are running, CIOD listens for messages from the CNKs that contain descriptions of I/O operations to be performed by CIOD on behalf of processes running on the compute nodes. Messaging between CIOD and CNK is synchronous. All file and socket I/O operations are blocking on the application side.

System software for the compute nodes

Compute nodes execute a single-user, dual-threaded (one thread per processor), minimalist custom kernel, the CNK. The CNK accomplishes a role similar to that of PUMA [8] in ASCI Red by controlling the Blue Gene/L compute nodes. It provides a simple, flat, fixed-size 512-MB address space with no paging. (The LLNL machine is configured with 512 MB per compute node. Other configurations, with more or less memory, are possible.) The kernel and application program share the same address space, with the kernel residing in protected memory at address 0, and the application program image loaded above, followed by its heap and stack. The kernel protects itself by appropriately programming the PPC440 memory management unit. Physical resources—torus and collective networks, locks, barriers, and scratchpad—are partitioned between application and kernel. The entire torus network is mapped into user space to obtain better communication efficiency, while one each of the two collective channels is made available to the kernel and user application.

The CNK presents a familiar Portable Operating System Interface Standard (POSIX) interface. We have ported the GNU `glibc` runtime library and provided support for basic file I/O operations through system calls. The I/O operations are not performed by CNK directly. Rather, they are shipped to the CIOD in the I/O node of the pset for execution, and the results are received back (see the preceding section on I/O node system software). Multiprocessing services, such as `fork` and `exec`, are meaningless in this kernel and have not been implemented.

The message-passing software that allows application processes running on different compute nodes to communicate with one another is organized in three layers:

- The *packet layer*, a thin software library that provides functions to directly access BG/L network hardware.
- The *message layer*, which provides a low-latency, high-bandwidth point-to-point message-delivery system.

- The *Message Passing Interface (MPI)*, the application-level communication library. This software is described in further detail in [9, 10].

We support two modes of execution for a compute node in CNK: *coprocessor* mode and *virtual node* mode. In coprocessor mode, a single dual-threaded process runs on the compute node, with access to the entire address space of the node. The main application thread runs in a non-preemptible manner on the main processor. The coprocessor is used as an offload engine for the main processor that can be exploited by user-level code. In particular, it performs many of the message-passing services necessary for internode communication. The user application can also offload computations to the coprocessor using a coroutine programming model. The coroutine model is implemented by two function calls from the main thread. The `co_start` call starts the execution of a function in the coprocessor and then continues the main thread. The `co_join` call waits for a previously started coprocessor function to terminate. Because the L1 caches of the two processors are not coherent, all coherence has to be handled explicitly in software by user code.

In virtual node mode, two single-threaded processes run on the compute node, one bound to each processor. Each process has access to half of the node memory, and the processes share the torus and collective communication devices. The two processes can communicate only through message passing, just as if they were running on distinct compute nodes. Coprocessor and virtual node modes can be chosen on a per-partition basis, and the machine can be divided to run multiple jobs simultaneously, each in its own mode. In the case of the LLNL machine, if it were running entirely in virtual node mode, the machine would appear to effectively have 131,072 compute nodes.

In both coprocessor and virtual node modes, intranode communication between the two processors is performed through a non-L1-cached region of shared memory. This allows us to circumvent the lack of coherence in the L1 caches of both processors. In coprocessor mode, the intranode communication is used to implement the offload mechanisms for communication and computation. In virtual node mode, the intranode communication is used to implement virtual torus and collective networks, so that the two processes inside a node can communicate just as they would with processes in different nodes.

System software for the front-end nodes

Compilers and debuggers run on the front-end nodes, which are the only nodes where users log in. Job submission is also performed from the front-end nodes, although the job launch and control facilities run on the service node.

Blue Gene/L presents a familiar programming model and a standard set of tools. We have ported the GNU tool chain (`binutils`,[1] `gcc`, `glibc`, and `gdb`) to BG/L and set it up as a cross-compilation environment. There are two cross-targets: Linux for I/O nodes and the CNK for compute nodes. The compiler team from the IBM Toronto Laboratory has ported the XL compiler suite to provide advanced optimization support for languages such as Fortran 90, C, and C++. In particular, the compilers support automatic code generation to exploit the two-way single-instruction multiple-data (SIMD) FPU attached to each processor in the compute node. The compilers are further described in [11].

System software for the service node

The control infrastructure is a critical component of our design. It provides a separation between execution mechanisms in the BG/L core and policy decisions in external nodes. The BG/L node operating systems (Linux for I/O nodes and CNK for compute nodes) implement services and are responsible for local decisions that do not affect the overall operation of the machine. A global operating system makes all global and collective decisions, interfaces with external policy modules (e.g., job schedulers), and performs a variety of system management services, including machine booting, system monitoring, and job launching. The global operating system runs on the service node and is called the core management and control system (CMCS).

CMCS contains two major processes that exploit different paths into the system. The midplane monitoring and control system (MMCS) uses the restricted control network to directly manipulate hardware for configuration, initialization, and operation. The control and I/O manager (CIOMAN) uses the functional Ethernet to interface with the local operating systems to support job execution. System monitoring is performed by a combination of MMCS and CIOMAN services, and services in the I/O nodes. For security and reliability reasons, the control paths into the core are not accessible to user processes running in the compute or front-end nodes. MMCS operations over the control network include the following:

- Low-level hardware operations, such as turning on power supplies, monitoring temperature sensors and fans, and reacting accordingly (i.e., shutting down a machine if temperature exceeds some threshold).

[1] BINary UTILities, which support the GNU compilers by providing programs that manipulate binary (machine-readable but not human-readable) object code and executable files.

- Configuring and initializing the control–FPGA and the link and BG/L chips.
- Reading and writing configuration registers, static random access memory (SRAM), and resetting the cores of a BG/L chip. These operations can be performed with no code executing in the nodes, which permits machine initialization and boot, nonintrusive access to performance counters, and post-mortem debugging.

At the core of the control system is a DB2 relational database. Databases naturally provide scalability, reliability, security, and logging. The control system database contains static state (i.e., the physical connections between hardware components) and dynamic state (i.e., machine partitions, partition configuration, partition assignment, and node state). The database is both a repository for read-only configuration information and an interface for all of the visible states of a machine. External entities (such as a job scheduler) can manipulate this state by changing the data in the database, which in turn invokes functions in the CMCS processes.

Machine initialization and booting

The machine initialization and boot process is performed through the restricted control network. It consists of partition allocation, followed by booting the nodes in the partition.

Partition allocation identifies a set of unused nodes as a candidate for a new partition. The control system computes a personality for each component node: torus coordinates, collective address, routing information, memory sizes, and other information. The personality information replaces the static BIOS information in traditional architectures.

Next, MMCS uses the JTAG path to write the personality and a small boot loader into each component of the partition. It then uses a communication protocol over JTAG with the boot loader to load the appropriate boot image for each node. We use one boot image for all of the compute nodes and another boot image for all of the I/O nodes. The boot image for the compute nodes contains the code for the CNK and is approximately 128 KB in size. The boot image for the I/O nodes contains the code for the Linux operating system (approximately 0.5 MB in size) and the image of a ramdisk that contains the root file system for the I/O node (also approximately 0.5 MB in size). After an I/O node boots, it can mount additional file systems from external file servers. Since the same boot image is used for each node, additional node-specific configuration information—such as torus coordinates, collective addresses, media access control or Internet Protocol (IP) addresses—must be loaded into the personality area of each node. On I/O nodes, the personality is exposed to user processes through an entry in the proc file system. On compute nodes, a system call is provided to request the node personality.

System monitoring

BG/L system monitoring is accomplished through a combination of I/O node and service node functionality. Each I/O node is a full Linux machine and uses Linux services to generate system logs.

A complementary monitoring service for BG/L is implemented by the service node through the control network. Device information, such as fan speeds and power-supply voltages, can be obtained directly by the service node through the control network. The compute and I/O nodes use a communication protocol to report events that can be logged or acted upon by the service node. This approach establishes a completely separate monitoring service independent of any other infrastructure (collective and torus networks, I/O nodes, and Ethernet networks) in the system. Therefore, it can be used to retrieve important information, even in the case of many system-wide failures.

Job execution

Job execution is accomplished through a combination of I/O nodes and service-node functionality. When submitting a job for execution, the user specifies the desired shape and size of the partition to execute that job. The scheduler selects a set of compute nodes to form the partition. The compute (and corresponding I/O) nodes selected by the scheduler are configured into a partition by the service node using the control network. We developed techniques applicable to Blue Gene/L to efficiently allocate nodes in a toroidal machine [12]. Once a partition is created, a job can be launched through the I/O nodes in that partition using the CIOD–CIOMAN path, as explained above.

Experience

An analysis of application performance on BG/L is outside the scope of this paper; we refer the interested reader to [5]. In this paper, we report qualitatively on a sample of user experiences with our prototype systems to demonstrate the effectiveness of our approach. We report on our own experience with the Linpack benchmark and the experiences of our collaborators with three important applications: SAGE, FLASH, and ParaDiS.

Linpack is the first benchmark we run when we assemble a BG/L prototype system. In fact, we consider Linpack to be a part of our test and bring-up process. After assembling a new system, we start a bring-up process that involves a combination of specialized test software and the production system software stack. The

bring-up process begins with power-on and ends when we obtain a valid Linpack run under production system software. For each of the 2,048- and 4,096-compute-node systems, the entire bring-up process took approximately one week. We obtained a measured performance of 11.6 Tflops for the 500-MHz, 4,096-compute-node system and 8.7 Tflops for the 700-MHz, 2,048-compute-node system, putting both systems among the ten fastest computers (at the fourth and the eighth position, respectively) in the TOP500 list of June 2004. Bring-up for the 16,384-compute-node prototype took somewhat longer (approximately one month) and was done concurrently with hardware assembly. Linpack achieved 70.7 Tflops on that prototype, placing it in the first position on the TOP500 list of November 2004.

SAGE is a hydrodynamics code that is representative of a large fraction of the compute cycles spent by the large ASCI machines at Los Alamos National Laboratory (LANL). SAGE was ported to BG/L by the performance evaluation team from LANL in September 2003, when our largest system had only 128 compute nodes. SAGE contains approximately 130,000 lines of code and, despite problems with our early-generation compilers, was ported in only two days. More recent measurements on larger BG/L systems show that SAGE scales better on BG/L than on state-of-the-art IBM POWER4* systems (although the per-processor performance is better for POWER4).

FLASH is an astrophysics code from the Center for Astrophysical Thermonuclear Flashes at the University of Chicago that was ported to BG/L in March of 2004. It took only two days for a joint team from IBM and ANL to port FLASH, including the time to adjust build files and port several libraries on which FLASH depended. During those two days, we also found a few bugs in the BG/L system software exposed by the port. The FLASH code itself ran unmodified in BG/L and has demonstrated perfect scalability up to the maximum system size tested (16,384 compute nodes).

Finally, ParaDiS—a dislocation dynamics code from LLNL—is considered one of the most important applications for their BG/L machine. The code is relatively new and still evolving; it has been running on BG/L since the first 32-node prototype was available in August 2003. Because ParaDiS has been evolving with BG/L, it is more difficult to quantify the amount of effort that went specifically into the BG/L port. It has run successfully on ever-increasing system sizes, up to 2,048 compute nodes. Measurements by LLNL show that ParaDiS displays per-processor performance and scalability characteristics on BG/L that are very similar to those of their flagship Linux Multiprogrammatic Capability Cluster, which uses much-higher-frequency (2.4-GHz) Intel Xeon** processors.

Conclusions

We have described the Blue Gene/L system software architecture, founded on the principles of simplicity, performance, and familiarity. Partitioning the system software functionality among dedicated pieces of hardware—compute nodes, I/O nodes, front-end nodes, and service node—has allowed us to keep each component simple and endow it with appropriate hardware resources for fast execution. As a result, the organization of the system software with a computational volume, a functional surface, and a control surface closely reflects the overall hardware architecture of the BG/L system.

The BG/L system software stack offers a familiar programming environment to the application programmer. The use of standard MPI libraries and programming languages (Fortran, C, and C++) facilitates the development of new applications and the porting of existing ones to BG/L. Our experience with benchmarks and production applications shows that they can be ported with modest effort (of the order of days) and typically achieve very good performance and scalability.

Acknowledgments

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. B517552.

References

1. MPICH and MPICH2 homepage; see *http://www-unix.mcs.anl.gov/mpi/mpich*.
2. H.-W. Meuer, E. Strohmaier, and J. Dongarra, TOP500 Supercomputer Sites; see *http://www.netlib.org/benchmark/top500.html*.
3. ASC/Alliances Center for Astrophysical Thermonuclear Flashes, University of Chicago; see *http://flash.uchicago.edu/website/home/*.
4. V. Bulatov, W. Cai, J. Fier, M. Hiratani, G. Hommes, T. Pierce, M. Tang, M. Rhee, K. Yates, and T. Arsenlis, "Scalable Line Dynamics in ParaDiS," *Proceedings of SC04*, 2004; see paper at *http://www.sc-conference.org/sc2004/schedule/pdfs/pap206.pdf*.
5. G. Almási, S. Chatterjee, A. Gara, J. Gunnels, M. Gupta, A. Henning, J. E. Moreira, B. Walkup, A. Curioni, C. Archer, L. Bachega, B. Chan, B. Curtis, M. Brodowicz, S. Brunett, E. Upchurch, G. Chukkapalli, R. Harkness, and W. Pfeiffer, "Unlocking the Performance of the BlueGene/L

Supercomputer," *Proceedings of SC04*, 2004; see *http://www.sc-conference.org/sc2004/schedule/pdfs/pap220.pdf*.
6. D. S. Greenberg, R. Brightwell, L. A. Fisk, A. B. Maccabe, and R. E. Riesen, "A System Software Architecture for High-End Computing," *Proceedings of SC97*, 1997, pp. 1–15; see *http://www.sandia.gov/ASCI/Red/*.
7. Open Programmable Interrupt Controller (PIC) Register Interface Specification, Revision 1.2, October 1995, Advanced Micro Devices and Cyrix Corporation; see *http://www.printk.net/pub/docs/openpic/openpic_specification.pdf*.
8. S. R. Wheat, A. B. Maccabe, R. Riesen, D. W. van Dresser, and T. M. Stallcup, "PUMA: An Operating System for Massively Parallel Systems," *Proceedings of the 27th Hawaii International Conference on System Sciences*, 1994, pp. 56–65; see *http://www.cs.unm.edu/~jhorey/L4-papers/puma-overview.pdf*.
9. G. Almási, C. Archer, J. Gunnels, P. Heidelberger, X. Martorell, and J. E. Moreira, "Architecture and Performance of the BlueGene/L Message Layer," *Proceedings of the 11th European PVM/MPI Users' Group Meeting*, 2004, pp. 405–414.
10. G. Almási, C. Archer, J. G. Castaños, M. Gupta, X. Martorell, J. E. Moreira, W. Gropp, S. Rus, and B. Toonen, "MPI on BlueGene/L: Designing an Efficient General Purpose Messaging Solution for a Large Cellular System," *Proceedings of the 10th European PVM/MPI Users Group Meeting*, 2003; see paper at *http://www-fp.mcs.anl.gov/~toonen/Papers/mpich2-bgl-pvmmpi-2003.pdf*.
11. L. Bachega, S. Chatterjee, K. A. Dockser, J. A. Gunnels, M. Gupta, F. G. Gustavson, C. A. Lapkowski, G. K. Liu, M. P. Mendell, C. D. Wait, and T. J. C. Ward, "A High-Performance SIMD Floating Point Unit for BlueGene/L: Architecture, Compilation, and Algorithm Design," *Proceedings of the 13th International Conference on Parallel Architecture and Compilation Techniques (PACT'04)*, 2004, pp. 85–96; see paper at *http://www.research.ibm.com/people/g/gupta/pact04.pdf*.
12. E. Krevat, J. G. Castaños, and J. E. Moreira, "Job Scheduling for the Blue Gene/L System," *Proceedings of the 8th International Euro-Par Conference*, 2002, pp. 38–54.

Received June 7, 2004; accepted for publication October 5, 2004; Internet publication April 5, 2005

José E. Moreira *IBM Systems and Technology Group, 3605 Highway 52 N., Rochester, Minnesota 55901 (jmoreira@us.ibm.com)*. Dr. Moreira received B.S. degrees in physics and electrical engineering in 1987 and an M.S. degree in electrical engineering in 1990, all from the University of São Paulo, Brazil. He received his Ph.D. degree in electrical engineering from the University of Illinois at Urbana–Champaign in 1995. Since joining IBM in 1995, he has been involved in several high-performance computing projects, including the teraflop-scale ASCI Blue-Pacific, ASCI White, and Blue Gene/L. Dr. Moreira was a manager at the IBM Thomas J. Watson Research Center from 2001 to 2004; he is currently the Lead Software Systems Architect for the IBM eServer Blue Gene solution. Dr. Moreira is the author of more than 70 publications on high-performance computing. He has served on various thesis committees and has been the chair or vice-chair of several international conferences and workshops. Dr. Moreira interacts closely with software developers, hardware developers, system installers, and customers to guarantee that the delivered systems work effectively and accomplish their intended missions successfully.

George Almási *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (gheorghe@us.ibm.com)*. Dr. Almási is a Research Staff Member at the IBM Thomas J. Watson Research Center. He received an M.S. degree in electrical engineering from the Technical University of Cluj-Napoca, Romania, in 1991 and an M.S. degree in computer science from West Virginia University in 1993. In 2001 he received a Ph.D. degree in computer science from the University of Illinois at Urbana–Champaign; his thesis dealt with ways of optimizing and compiling MATLAB code. For the last three years, Dr. Almási has been working on various aspects of the Blue Gene system software environment, including the MPI communication libraries.

Charles Archer *IBM Systems and Technology Group, 3605 Highway 52 N., Rochester, Minnesota 55901 (archerc@us.ibm.com)*. Mr. Archer is a software engineer working on the Blue Gene/L project. He received a B.S. degree in chemistry and a B.A. degree in mathematics from Minnesota State University at Moorhead, and an M.S. degree in chemistry from Columbia University. He is currently a graduate student in computer science at the University of Minnesota. Mr. Archer has worked on the OS/400* PASE project and grid computing. His current role is development, optimization, and maintenance of the Blue Gene/L message-passing software stack.

Ralph Bellofatto *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (ralphbel@us.ibm.com)*. Mr. Bellofatto is a Senior Software Engineer. He has been responsible for various aspects of hardware system verification and control system programming on the Blue Gene/L project. He received B.S. and M.S. degrees from Ithaca College in 1979 and 1980, respectively. He has worked as a software engineer in a variety of industries. Mr. Bellofatto's interests include computer architecture, performance analysis and tuning, network architecture, ASIC design, and systems architecture and design. He is currently working on the control system for Blue Gene/L.

Peter Bergner *IBM Systems and Technology Group, 3605 Highway 52 N., Rochester, Minnesota 55901 (bergner@us.ibm.com)*. Dr. Bergner is a member of the Blue Gene/L software team working on compiler and runtime library

development. He received a Ph.D. degree in electrical engineering from the University of Minnesota in 1997. His thesis work involved developing techniques for minimizing spill code in graph coloring register allocators. Dr. Bergner has worked in a variety of areas since joining IBM, including AS/400* compiler optimizer development and Linux kernel and compiler development for PowerPC hardware platforms.

José R. Brunheroto *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (brunhe@us.ibm.com).* Mr. Brunheroto is a Senior Software Engineer at the IBM Thomas J. Watson Research Center. He received a B.S. degree in electrical engineering from Escola Politécnica, University of São Paulo, Brazil. Mr. Brunheroto is currently working on his M.A. degree in computer science at Columbia University. His research interests include computer architecture, simulators (single-node and distributed), and performance tools.

Michael Brutman *IBM Systems and Technology Group, 3605 Highway 52 N., Rochester, Minnesota 55901 (brutman@us.ibm.com).* Mr. Brutman has worked for IBM since 1992. Before joining the Blue Gene project in early 2003, he concentrated on operating systems implementation and performance analysis. Mr. Brutman has worked primarily on the control system and debugger support for the compute node kernel as part of the Blue Gene team.

José Gabriel Castaños *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (castanos@us.ibm.com).* Dr. Castaños joined the Blue Gene project in 2000 after receiving his Ph.D. degree in computer science at Brown University. His initial assignment as a Research Staff Member involved the development of applications for high-performance computing. He later became one of the technical leaders of the Blue Gene/L systems software and worked on many of its components: integrated software development environment, simulators, kernels, runtime libraries, and management infrastructure. Dr. Castaños received his undergraduate degrees in systems analysis (1988) and operations research (1989) at the Universidad Católica, Buenos Aires, Argentina.

Paul G. Crumley *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (pgc@us.ibm.com).* Mr. Crumley has worked in the IBM Research Division for more than 20 years. His work and interests span a wide range of projects including distributed data systems, high-function workstations, operational processes, and, most recently, cellular processor support infrastructure.

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. In 1992 he received a Ph.D. degree in computer science from the University of Illinois at Urbana–Champaign and has worked with the IBM Research Division since then. Dr. Gupta has coauthored several papers in the areas of high-performance compilers, parallel computing, and high-performance Java** Virtual Machines.

Todd Inglett *IBM Systems and Technology Group, 3605 Highway 52 N., Rochester, Minnesota 55901 (tinglett@us.ibm.com).* Mr. Inglett graduated with a B.S. degree in computer science from the University of Wisconsin in 1987, joining ETA Systems, Inc., as a software engineer. He began work at IBM in Rochester, Minnesota, in 1989, and was part of the Andrew Project and Andrew Consortium as a partner with Carnegie Mellon University from 1989 to 1993. He then worked on IBM internal development tools, including Apache, from 1994 to 1998, and on porting Linux to the PowerPC 64-bit architecture from 1999 to 2002. Mr. Inglett has been a member of the Blue Gene/L software development team since 2002. He has made various contributions to the project, including the I/O node Linux kernel, parts of the control system, file system performance, and system bring-up.

Derek Lieber *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (lieber@us.ibm.com).* Mr. Lieber received his B.S. degree in physics from Lebanon Valley College in 1975. In 1983 he joined the IBM Thomas J. Watson Research Center, where he is currently a Senior Software Engineer. Mr. Lieber was the technical leader and main implementer of the Jikes Research Virtual Machine (RVM) runtime environment. The Jikes RVM has been recognized by institutions worldwide as a excellent platform for research in Java. He was the main designer and implementer of the Blue Gene/L operating environment. Mr. Lieber has more than 20 publications in high-performance computing.

David Limpert *IBM Systems and Technology Group, 3605 Highway 52 N., Rochester, Minnesota 55901 (limpert@us.ibm.com).* Mr. Limpert is a Senior Technical Staff Member responsible for the software development and delivery of the Blue Gene/L supercomputer system. In 1977 he joined IBM in Rochester, Minnesota, after receiving a B.A. degree in mathematics and computer science from Saint Mary's University of Minnesota. He has a wide range of experience in the field of systems software engineering. His areas of responsibility have ranged from design and implementation of microcode for terminal and server system devices to technical project leader for complete software systems. During the 1980s and early 1990s, he led software development teams in the integration of the emerging PC business with IBM midrange server systems as cooperative processing workstations. Those efforts progressed from simple terminal device emulators to full-feature distributed processing networked environments. Mr. Limpert shifted focus in the late 1990s to the design of network-connected thin clients and their use of Linux as an embedded operating system.

Patrick McCarthy *IBM Systems and Technology Group, 3605 Highway 52 N., Rochester, Minnesota 55901 (pjmccart@us.ibm.com).* Mr. McCarthy has worked for IBM since 1984 and has spent most of his career in kernel development, including the OS/400 kernel and microkernel and the Linux port to the 64-bit PowerPC. He is currently working on the development of the BG/L supercomputer. His main focus is on the kernels which run on the compute and I/O nodes in the Blue Gene/L system. Mr. McCarthy has also worked on the development of parts of the Blue Gene/L control system.

Mark Megerian *IBM Systems and Technology Group, 3605 Highway 52 N., Rochester, Minnesota 55901 (megerian@us.ibm.com)*. Mr. Megerian is a Senior Software Engineer in the IBM Rochester Laboratory. He graduated from Michigan State University with a degree in computer science in 1987, joining IBM upon graduation. His past responsibilities have included database and SQL development on the iSeries* platform. Mr. Megerian is currently the control system team leader for BlueGene/L.

Mark Mendell *IBM Software Group, Toronto Laboratory, 8200 Warden Avenue, Markham, Ontario, Canada L6G 1C7 (mendell@ca.ibm.com)*. Mr. Mendell graduated from Cornell University in 1980 with a B.S. degree in computer engineering. He received his M.S. degree in computer science from the University of Toronto in 1983. At the University of Toronto, he helped develop the Concurrent Euclid, Turing, and Turing Plus compilers, and worked on the Tunis operating system project. In 1991 he joined IBM, working initially on the AIX* C++ compiler from V1.0 to V5.0. He has been the team leader for the TOBEY Optimizer Group since 2000. Mr. Mendell implemented the automatic compiler support of the dual floating-point unit (FPU) for the Blue Gene/L project.

Michael Mundy *IBM Systems and Technology Group, 3605 Highway 52 N., Rochester, Minnesota 55901 (mmundy@us.ibm.com)*. Mr. Mundy joined IBM in 1989 as a developer and tester for the MVS operating system. His past projects have included Qshell (a shell and utilities package), base POSIX enablement, Pthreads, TCP/IP sockets, and the Distributed Computing Environment (DCE). Since 1993 he has worked on i5/OS* and its predecessors to provide open industry-standard interfaces. Mr. Mundy is currently working on the compute node kernel of Blue Gene/L.

Don Reed *IBM Systems and Technology Group, 3605 Highway 52 N., Rochester, Minnesota 55901 (donreed@us.ibm.com)*. Mr. Reed received his computer science degree from the University of South Dakota. At IBM he has worked on control systems for DASD manufacturing, the port of Linux to 64-bit PowerPC, and process and resource management for the OS/400 kernel. His current role is development and maintenance of the hardware initialization and control system software stack of Blue Gene/L.

Ramendra K. Sahoo *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (rsahoo@us.ibm.com)*. Dr. Sahoo received a B.E. degree (with honors) in mechanical engineering from the National Institute of Technology, Durgapur, India, in 1990, and an M.S. degree (research) from the Indian Institute of Technology, Chennai, in 1992. In 1998 he received his Ph.D. degree from the State University of New York at Stony Brook. Prior to joining the Ph.D. program at Stony Brook, he worked as an assistant manager (CAD and analysis) at TATA Motors Engineering Research Center. Since 2000 he has worked at the IBM Thomas J. Watson Research Center and is currently a member of the Blue Gene Systems Software Group. Dr. Sahoo's research interests include distributed and fault-tolerant computing, numerical and parallel algorithms, databases, and artificial intelligence. He has published 35 papers in refereed international journals and conferences in the area of scientific computing, electronic packaging, data mining, parallel computing, and artificial intelligence. Dr. Sahoo holds seven patents in the area of distributed and fault-tolerant computing; he is a member of the IEEE and ASME.

Alda Sanomiya *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (sanomiya@us.ibm.com)*. Mrs. Sanomiya received a B.S. degree in computer science from the Universidade de São Paulo, Brazil. She began working with IBM Brazil in 1986 and joined the IBM Research Division in 2000. Mrs. Sanomiya was responsible for the first port of Linux to the Blue Gene/L I/O nodes; she has also worked on control systems for the Blue Gene/L project.

Richard Shok *IBM Systems and Technology Group, 3605 Highway 52 N., Rochester, Minnesota 55901 (shok@us.ibm.com)*. Mr. Shok has worked for IBM for more than six years. During that time, he has been involved in OS internals, retail middleware, and custom programs for various IBM customers. Mr. Shok has spent the last year working on the Blue Gene/L project; he is primarily responsible for software builds, integration, and packaging.

Brian Smith *IBM Systems and Technology Group, 3605 Highway 52 N., Rochester, Minnesota 55901 (smithbr@us.ibm.com)*. Mr. Smith has worked at IBM in Rochester, Minnesota, for the past year as a co-op student. His work primarily involves MPI development and optimization, and porting applications for Blue Gene/L. In early 2005 he began working full-time at IBM Rochester, after graduating from Iowa State University with an M.S. degree in computer engineering.

Gordon G. (Greg) Stewart *IBM Systems and Technology Group, 3605 Highway 52 N., Rochester, Minnesota 55901 (gregstewt@us.ibm.com)*. Mr. Stewart is a Senior Software Engineer in the eServer Custom Technology Center at IBM in Rochester, Minnesota. He received a B.S. degree in mathematics and an M.S. degree in computer science from Northern Illinois University in 1974 and 1976, respectively. After spending two years at Baxter/Travenol Laboratories as a systems programmer, Mr. Stewart joined IBM in 1978 to work on the development of the IBM System/38*. Since then, he has worked chiefly in architecture and development in various areas of the System/38, AS/400, and iSeries operating systems, particularly in data communications, availability/recovery, and the POSIX and C language environments. He has also contributed to the development of several application middleware products, including Domino* for AS/400 and Net.Data*. In 2000 he joined the eServer Custom Technology Center, where he has worked on the Blue Gene/L project, primarily in development of the control system. Mr. Stewart has received several patents and IBM Technical Achievement Awards.