An Overview of the Blue Gene/L System Software Organization

George Almási[1], Ralph Bellofatto[1], José Brunheroto[1], Călin Caşcaval[1], José G. Castaños[1], Luis Ceze[2], Paul Crumley[1], C. Christopher Erway[1], Joseph Gagliano[1], Derek Lieber[1], Xavier Martorell[1], José E. Moreira[1], Alda Sanomiya[1], and Karin Strauss[2]

[1] IBM Thomas J. Watson Research Center
Yorktown Heights, NY 10598-0218
{gheorghe,ralphbel,brunhe,cascaval,castanos,pgc,erway,
jgaglia,lieber,xavim,jmoreira,sanomiya}@us.ibm.com
[2] Department of Computer Science
University of Illinois at Urbana-Champaign
Urabana, IL 61801
{luisceze,kstrauss}@uiuc.edu

Abstract. The Blue Gene/L supercomputer will use system-on-a-chip integration and a highly scalable cellular architecture. With 65,536 compute nodes, Blue Gene/L represents a new level of complexity for parallel system software, with specific challenges in the areas of scalability, maintenance and usability. In this paper we present our vision of a software architecture that faces up to these challenges, and the simulation framework that we have used for our experiments.

1 Introduction

In November 2001 IBM announced a partnership with Lawrence Livermore National Laboratory to build the Blue Gene/L (BG/L) supercomputer, a 65,536-node machine designed around embedded PowerPC processors. Through the use of system-on-a-chip integration [10], coupled with a highly scalable cellular architecture, Blue Gene/L will deliver 180 or 360 Teraflops of peak computing power, depending on the utilization mode. Blue Gene/L represents a new level of scalability for parallel systems. Whereas existing large scale systems range in size from hundreds (ASCI White [2], Earth Simulator [4]) to a few thousands (Cplant [3], ASCI Red [1]) of compute nodes, Blue Gene/L makes a jump of almost two orders of magnitude.

The system software for Blue Gene/L is a combination of standard and custom solutions. The software architecture for the machine is divided into three functional entities (similar to [13]) arranged hierarchically: a *computational core*, a *control infrastructure* and a *service infrastructure*. The I/O nodes (part of the control infrastructure) execute a version of the Linux kernel and are the primary off-load engine for most system services. No user code directly executes on the I/O nodes. Compute nodes in the computational core execute a single user, single process minimalist custom kernel, and are dedicated to efficiently run user applications. No system daemons or sophisticated system services reside on compute nodes. These are treated as externally controllable

entities (i.e. devices) attached to I/O nodes. Complementing the Blue Gene/L machine proper, the Blue Gene/L complex includes the service infrastructure composed of commercially available systems, that connect to the rest of the system through an Ethernet network. The end user view of a system is of a flat, toroidal, 64K-node system, but the system view of Blue Gene/L is hierarchical: the machine looks like a 1024-node Linux cluster, with each node being a 64-way multiprocessor. We call one of these logical groupings a *processing set* or *pset*.

The scope of this paper is to present the software architecture of the Blue Gene/L machine and its implementation. Since the target time frame for completion and delivery of Blue Gene/L is 2005, all our software development and experiments have been conducted on architecturally accurate simulators of the machine. We describe this simulation environment and comment on our experience.

The rest of this paper is organized as follows. Section 2 presents a brief description of the Blue Gene/L supercomputer. Section 3 discusses the system software. Section 4 introduces our simulation environment and we conclude in Section 5.

2 An Overview of the Blue Gene/L Supercomputer

Blue Gene/L is a new architecture for high performance parallel computers based on low cost embedded PowerPC technology. A detailed description of Blue Gene/L is provided in [8]. In this section we present a short overview of the hardware as background for our discussion on its system software and its simulation environment.

2.1 Overall Organization

The basic building block of Blue Gene/L is a custom system-on-a-chip that integrates processors, memory and communications logic in the same piece of silicon. The BG/L chip contains two standard 32-bit embedded PowerPC 440 cores, each with private L1 32KB instruction and 32KB data caches. Each core also has a 2KB L2 cache and they share a 4MB L3 EDRAM cache. While the L1 caches are not coherent, the L2 caches are coherent and act as a prefetch buffer for the L3 cache.

Each core drives a custom 128-bit double FPU that can perform four double precision floating-point operations per cycle. This custom FPU consists of two conventional FPUs joined together, each having a 64-bit register file with 32 registers. One of the conventional FPUs (the primary side) is compatible with the standard PowerPC floating-point instruction set. We have extended the PPC instruction set to perform SIMD-style floating point operations on the two FPUs. In most scenarios, only one of the 440 cores is dedicated to run user applications while the second processor drives the networks. At a target speed of 700 MHz the peak performance of a node is 2.8 GFlop/s. When both cores and FPUs in a chip are used, the peak performance per node is 5.6 GFlop/s.

The standard PowerPC 440 cores are not designed to support multiprocessor architectures: the L1 caches are not coherent and the architecture lacks atomic memory operations. To overcome these limitations BG/L provides a variety of synchronization devices in the chip: lockbox, shared SRAM, L3 scratchpad and the blind device. The

lockbox unit contains a limited number of memory locations for fast atomic test-and-sets and barriers. 16 KB of SRAM in the chip can be used to exchange data between the cores and regions of the EDRAM L3 cache can be reserved as an addressable scratch-pad. The blind device permits explicit cache management.

The low power characteristics of Blue Gene/L permit a very dense packaging as shown in Figure 1. Two nodes share a node card that also contains SDRAM-DDR memory. Each node supports a maximum of 2 GB external memory but in the current configuration each node directly addresses 256 MB at 5.5 GB/s bandwidth with a 75 cycle latency. Sixteen compute cards can be plugged in a node board. A cabinet with two midplanes contains 32 node boards for a total of 2048 CPUs and a peak performance of 2.9/5.7 TFlops. The complete system has 64 cabinets and 16 TB of memory.

In addition to the 64K compute nodes, BG/L contains a number of I/O nodes (1024 in the current design). Compute nodes and I/O nodes are physically identical although I/O nodes are likely to contain more memory. The only difference is in their card packaging which determines which networks are enabled.

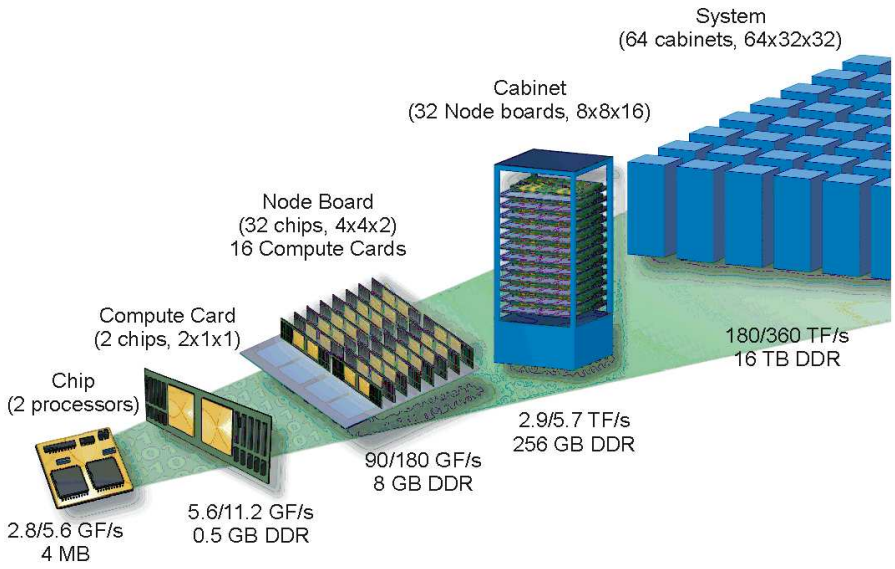

Fig. 1. High-level organization of the Blue Gene/L supercomputer. All 65,536 compute nodes are organized in a $64 \times 32 \times 32$ three-dimensional torus.

2.2 Blue Gene/L Communications Hardware

All inter-node communication is done exclusively through messages. The BG/L ASIC supports five different networks: torus, tree, Ethernet, JTAG, and global interrupts. The

main communication network for point-to-point messages is a three-dimensional torus. Each node contains six bi-directional links for direct connection with nearest neighbors. The 64K nodes are organized into a partitionable 64x32x32 three-dimensional torus. The network hardware in the ASICs guarantees reliable, unordered, deadlock-free delivery of variable length (up to 256 bytes) packets, using a minimal adaptive routing algorithm. It also provides simple broadcast functionality by depositing packets along a route. At 1.4 Gb/s per direction, the unidirectional bisection bandwidth of a 64K node system is 360 GB/s. The I/O nodes are not part of the torus network.

The tree network supports fast configurable point-to-point messages of fixed length (256 bytes) data. It also implements broadcasts and reductions with a hardware latency of 1.5 microseconds for a 64K node system. An ALU in the network can combine incoming packets using bitwise and integer operations, and forward a resulting packet along the tree. Floating point reductions can be performed in two phases (one for the exponent and another one for the mantissa) or in one phase by converting the floating-point number to an extended 2048-bit representation. I/O and compute nodes share the tree network. Tree packets are the main mechanism for communication between I/O and compute nodes.

The torus and the tree networks are memory mapped devices. Processors send and receive packets by explicitly writing to (and reading from) special memory addresses that act as FIFOs. These reads and writes use the 128-bit SIMD registers.

A separate set of links provides global OR/AND operations (also with a 1.5 microseconds latency) for fast barrier synchronization.

The Blue Gene/L computational core can be subdivided into *partitions*, which are electrically isolated and self-contained subsets of the machine. A partition is dedicated to the execution of a single job. The tree and torus wires between midplanes (half a cabinet) are routed through a custom chip called the Link Chip. This chip can be dynamically configured to skip faulty midplanes while maintaining a working torus and to partition the torus network into multiple, independent torii. The smallest torus in BG/L is a midplane (512 nodes). The tree and torus FIFOs are controlled by device control registers (DCRs). It is possible to create smaller meshes by disabling the FIFOs of the chips at the boundary of a partition. In the current packaging schema, the smallest independent mesh contains 128 compute nodes and 2 I/O nodes.

Finally, each BG/L chip contains a 1Gbit/s Ethernet macro for external connectivity and supports a serial JTAG network for booting, control and monitoring of the system. Only I/O nodes are attached to the Gbit/s Ethernet network, giving 1024x1Gbit/s links to external file servers.

Completing the Blue Gene/L machine, there is a number of service nodes: a traditional Linux cluster or SP system that resides outside the Blue Gene/L core. The service nodes communicate with the computational core through the IDo chips. The current packaging contains one IDo chip per node board and another one per midplane. The IDo chips are 25MHz FPGAs that receive commands from the service nodes using raw UDP packets over a trusted private 100 Mbit/s Ethernet control network. The IDo chips support a variety of serial protocols for communication with the core. The I^2C network controls temperature sensors, fans and power supplies. The JTAG protocol is used for reading and writing to any address of the 16 KB SRAMs in the BG/L chips,

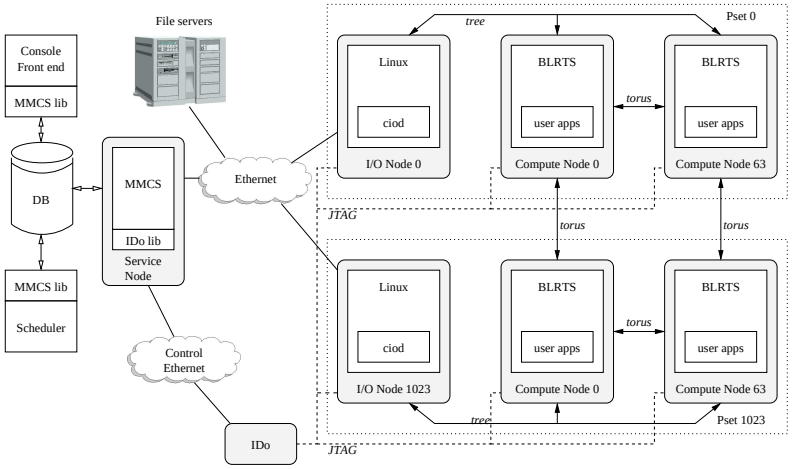

Fig. 2. Outline of the Blue Gene/L system software. The computational core is partitioned into 1024 logical processing sets (psets), each with one I/O node running Linux and 64 compute nodes running the custom BLRTS kernel. External entities connect the computational core through two Ethernet networks for I/O and low level management.

reading and writing to registers and sending interrupts to suspend and reset the cores. These services are available through a direct link between the IDo chips and nodes, and bypass the system software running on the target nodes.

3 Blue Gene/L System Software

To address the challenges of scalability and complexity posed by BG/L we have developed the system software architecture presented in Figure 2. This architecture is described in detail in this section.

3.1 System Software for the I/O Nodes

The Linux kernel that executes in the I/O nodes (currently version 2.4.19) is based on on a standard distribution for PowerPC 440GP processors. Although Blue Gene/L uses standard PPC 440 cores, the overall chip and card design required changes in the booting sequence, interrupt management, memory layout, FPU support, and device drivers of the standard Linux kernel. There is no BIOS in the Blue Gene/L nodes, thus the configuration of a node after power-on and the initial program load (IPL) is initiated by the service nodes through the control network. We modified the interrupt and exception handling code to support Blue Gene/L's custom Interrupt Controller (BIC). The implementation of the kernel MMU remaps the tree and torus FIFOs to user space. We support the new EMAC4 Gigabit Ethernet controller We also updated the kernel to save and restore the double FPU registers in each context switch.

The nodes in the Blue Gene/L machine are diskless, thus the initial root file system is provided by a ramdisk linked against the Linux kernel. The ramdisk contains shells, simple utilities, shared libraries, and network clients such as `ftp` and `nfs`.

Because of the non-coherent L1 caches, the current version of Linux runs on one of the 440 cores, while the second CPU is captured at boot time in an infinite loop. We are investigating two main strategies to effectively use the second CPU in the I/O nodes: SMP mode and virtual mode. We have successfully compiled a SMP version of the kernel, after implementing all the required interprocessor communications mechanisms, because the BG/L's BIC is not OpenPIC [6] compliant. In this mode, the TLB entries for the L1 cache 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 is the core of a virtual network card. In this scenario, the tree and torus FIFOs are not visible to the Linux kernel. Transfers between the two CPUs appear as virtual DMA transfers.

We are also investigating support for large pages. The standard PPC 440 embedded processors handle all TLB misses in software. Although the average number of instructions required to handle these misses has significantly decreased, it has been shown that larger pages improve performance [23].

3.2 System Software for the Compute Nodes

The "Blue Gene/L Run Time Supervisor" (BLRTS) is a custom kernel that runs on the compute nodes of a Blue Gene/L machine. BLRTS provides a simple, flat, fixed-size 256MB address space, with no paging, accomplishing a role similar to PUMA [19]. 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 PowerPC MMU. Physical resources (torus, tree, mutexes, barriers, scratchpad) are partitioned between application and kernel. In the current implementation, the entire torus network is mapped into user space to obtain better communication efficiency, while one of the two tree channels is made available to the kernel and user applications.

BLRTS presents a familiar POSIX interface: we have ported the GNU Glibc runtime library and provided support for basic file I/O operations through system calls. Multi-processing services (such as fork and exec) are meaningless in single process kernel and have not been implemented. Program launch, termination, and file I/O is accomplished via messages passed between the compute node and its I/O node over the tree network, using a point-to-point packet addressing mode. This functionality is provided by a daemon called CIOD (Console I/O Daemon) running in the I/O nodes. CIOD provides job control and I/O management on behalf of all the compute nodes in the processing set. Under normal operation, all messaging between CIOD and BLRTS is synchronous: all file I/O operations are blocking on the application side. We used the CIOD in two scenarios:

1. driven by a console shell (called CIOMAN), used mostly for simulation and testing purposes. The user is provided with a restricted set of commands: run, kill, ps,

set and unset environment variables. The shell distributes the commands to all the CIODs running in the simulation, which in turn take the appropriate actions for their compute nodes.
2. driven by a job scheduler (such as LoadLeveler) through a special interface that implements the same protocol as the one defined for CIOMAN and CIOD.

We are investigating a range of compute modes for our custom kernel. In *heater* mode, one CPU executes both user and network code, while the other CPU remains idle. This mode will be the mode of operation of the initial prototypes, but it is unlikely to be used afterwards. In *co-processor* mode, the application runs in a single, non-preemptable thread of execution on the main processor (cpu 0). The coprocessor (cpu 1) is used as a torus device off-load engine that runs as part of a user-level application library, communicating with the main processor through a non-cached region of shared memory. In *symmetric* mode, both CPUs run applications and users are responsible for explicitly handling cache coherence. In *virtual node* mode we provide support for two independent processes in a node. The system then looks like a machine with 128K nodes.

3.3 System Management Software

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. Local node operating systems (Linux for I/O nodes and BLRTS for compute nodes) implement services and are responsible for local decisions that do not affect overall operation of the machine. A "global operating system" makes all global and collective decisions and interfaces with external policy modules (e.g., LoadLeveler) and performs a variety of system management services, including: (i) machine booting, (ii) system monitoring, and (iii) job launching.

In our implementation, the global OS runs on external service nodes. Each BG/L midplane is controlled by one Midplane Management and Control System (MMCS) process which provides two paths into the Blue Gene/L complex: a custom control library to access the restricted JTAG network and directly manipulate Blue Gene/L nodes; and sockets over the Gbit/s Ethernet network to manage the nodes on a booted partition.

The custom control library can perform:

- low level hardware operations such as: turn on power supplies, monitor temperature sensors and fans, and react accordingly (i.e. shut down a machine if temperature exceeds some threshold),
- configure and initialize IDo, Link and BG/L chips,
- read and write configuration registers, SRAM and reset the cores of a BG/L chip. As mentioned in Section 2, 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.

This path into the core is used for control only; for security and reliability reasons, it is not made visible to applications running in the BG/L nodes. On the other hand,

the architected path through the functional Gbit/s Ethernet is used for application I/O, checkpoints, and job launch.

We chose to maintain the entire state of the global operating system using standard database technology. Databases naturally provide scalability, reliability, security, portability, logging, and robustness. The database contains static state (i.e., the physical connections between each HW component) and dynamic state (i.e., how the machine is partitioned, how each partition is configured, and which parallel application is running in each partition). Therefore, the database is not just a repository for read-only configuration information, but also an interface for all the visible state of a machine. External entities (such as a scheduler) can manipulate this state by invoking stored procedures and database triggers, which in turn invoke functions in the MMCS processes.

Machine Initialization and Booting. The boot process for a node consists of the following steps: first, a small boot loader is directly written into the (compute or I/O) node memory by the service nodes using the JTAG control network. This boot loader loads a much larger boot image into the memory of the node through a custom JTAG mailbox protocol.

We use one boot image for all the compute nodes and another boot image for all the I/O nodes. The boot image for the compute nodes contains the code for the compute node kernel, and is approximately 64 kB in size. The boot image for the I/O nodes contains the code for the Linux operating system (approximately 2 MB in size) and the image of a ramdisk that contains the root file system for the I/O node. 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, tree addresses, MAC or IP addresses) must be loaded. We call this information the *personality* of a node. In the I/O nodes, the personality is exposed to user processes through an entry in the `proc` file system. BLRTS implements a system call to request the node's personality.

System Monitoring in Blue Gene/L 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 Blue Gene/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 that is independent of any other infrastructure (tree and torus networks, I/O nodes, Ethernet network) in the system. Therefore, it can be used even in the case of many system-wide failures to retrieve important information.

Job Execution is also accomplished through a combination of I/O nodes and service node functionality. When submitting a job for execution in Blue Gene/L , 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 have developed techniques for efficient allocation of nodes in a toroidal machine that are applicable to Blue Gene/L [16]. Once a partition is created, a job can be launched through the I/O nodes in that partition using CIOD as explained before.

3.4 Compiler and Run-time Support

Blue Gene/L presents a familiar programming model and a standard set of tools. We have ported the GNU toolchain (binutils, gcc, glibc and gdb) to Blue Gene/L and set it up as a cross-compilation environment. There are two cross-targets: Linux for I/O nodes and BLRTS for compute nodes. IBM's XL compiler suite is also being ported to provide advanced optimization support for languages like Fortran90 and C++.

3.5 Communication Infrastructure

The Blue Gene/L communication software architecture is organized into three layers: the *packet layer* is a thin software library that allows access to network hardware; the *message layer* provides a low-latency, high bandwidth point-to-point message delivery system; *MPI* is the user level communication library.

The packet layer simplifies access to the Blue Gene/L network hardware. The packet layer abstracts FIFOs and devices control registers into torus and tree *devices* and presents an API consisting of essentially three functions: initialization, packet send and packet receive. The packet layer provides a mechanism to use the network hardware but doesn't impose any policies on its use. Hardware restrictions, such as the 256 byte limit on packet size and the 16 byte alignment requirements on packet buffers, are not abstracted by the packet layer and thus are reflected by the API. All packet layer send and receive operations are non-blocking, leaving it up to the higher layers to implement synchronous, blocking and/or interrupt driven communication models. In its current implementation the packet layer is stateless.

The message layer is a simple active message system [12, 17, 21, 22], built on top of the torus packet layer, which allows the transmission of arbitrary messages among torus nodes. It is designed to enable the implementation of MPI point-to-point send/receive operations. It has the following characteristics:

No packet retransmission protocol. The Blue Gene/L network hardware is completely reliable, and thus a packet retransmission system (such as a sliding window protocol) is unnecessary. This allows for stateless virtual connections between pairs of nodes, greatly enhancing scalability.

Packetizing and alignment. The packet layer requires data to be sent in 256 byte chunks aligned at 16 byte boundaries. Thus the message layer deals with the packetizing and re-alignment of message data. Re-alignment of packet data typically entails memory-to-memory copies.

Packet ordering. Packets on the torus network can arrive out of order, which makes message re-assembly at the receiver non-trivial. For packets belonging to the same message, the message layer is able to handle their arrival in any order. To restore order for packets belonging to *different* messages, the sender assigns ascending numbers to individual messages sent out to the same peer.

Cache coherence and processor use policy. The expected performance of the message layer is influenced by the way in which the two processors are used (as discussed in Section 3.2). Co-processor mode is the only one that effectively overlaps computation and communication. This mode is expected to yield better bandwidth, but slightly higher latency, than the others.

MPI: Blue Gene/L is designed primarily to run MPI [20] workloads. We are in the process of porting MPICH2 [5], currently under development at Argonne National Laboratories, to the Blue Gene/L hardware. MPICH2 has a modular architecture. The Blue Gene/L port leaves the code structure of MPICH2 intact, but adds a number of plug-in modules:

Point-to-point messages. The most important addition of the Blue Gene/L port is an implementation of ADI3, the MPICH2 Abstract Device Interface [14]. A thin layer of code transforms e.g. MPI `Request` objects and `MPI_Send` function calls into calls into sequences of message layer function calls and callbacks.

Process management. The MPICH2 process management primitives are documented in [7]. Process management is split into two parts: a process management interface (PMI), called from within the MPI library, and a set of process managers (PM) which are responsible for starting up/shutting down MPI jobs and implementing the PMI functions.

MPICH2 includes a number of process managers (PM) suited for clusters of general purpose workstations. The Blue Gene/L process manager makes full use of its hierarchical system management software, including the CIOD processes running on the I/O nodes, to start up and shut down MPI jobs. The Blue Gene/L system management software is explicitly designed to deal with the scalability problem inherent in starting, synchronizing and killing 65,536 MPI processes.

Optimized collectives. The default implementation of MPI collective operations in MPICH2 generates sequences of point-to-point messages. This implementation is oblivious of the underlying physical topology of the torus and tree networks. In Blue Gene/L optimized collective operations can be implemented for communicators whose physical layouts conform to certain properties.

- The torus hardware can be used to efficiently implement broadcasts on contiguous 1, 2 and 3 dimensional meshes, using a feature of the torus that allows depositing a packet on every node it traverses. The collectives best suited for this (e.g. `Bcast`, `Allgather`, `Alltoall`, `Barrier`) involve broadcast in some form.
- The tree hardware can be used for almost every collective that is executed on the `MPI_COMM_WORLD` communicator, including reduction operations. Integer operand reductions are directly supported by hardware. IEEE compliant floating point reductions can also be implemented by the tree using separate reduction phases for the mantissa and the exponent.

– Non `MPI_COMM_WORLD` collectives can also be implemented using the tree, but care must be taken to ensure deadlock free operation. The tree is a locally class routed network, with packets belonging to one of a small number of classes and tree nodes making local decisions about routing. The tree network guarantees deadlock-free simultaneous delivery of no more than two class routes. One of these routes is used for control and file I/O purposes; the other is available for use by collectives.

4 Blue Gene/L Simulation Environment

The first hardware prototypes of the Blue Gene/L ASIC are targeted to become operational in mid-2003. To support the development of system software before hardware is available, we have implemented an architecturally accurate, full system simulator for the Blue Gene/L machine [11]. The node simulator, called `bglsim` , is built using techniques described in [18, 15]. Each component of the BG/L ASIC is modeled separately with the desired degree of accuracy, trading accuracy for performance. In our simulation environment, we model the functionality of processor instructions. That is, each instruction correctly changes the visible architectural state, while it takes one cycle to execute. We also model memory system behavior (cache, scratch-pad, and main memory) and all the Blue Gene/L specific devices: tree, torus, JTAG, device control registers, etc. A `bglsim` process boots the Linux kernel for the I/O nodes and BLRTS for the compute nodes. Applications run on top of these kernels, under user control.

When running on 1.2 GHz Pentium III machine, `bglsim` simulates an entire BG/L chip at approximately 2 million simulated instructions per second – a slow-down of about 1000 compared to the real hardware. By comparison, a VHDL simulator with hardware acceleration has a slow-down of 10^6, while a software VHDL simulator has a slow-down of 10^9. As an example, booting Linux takes 30 seconds on `bglsim` , 7 hours on the hardware accelerated VHDL simulator and more that 20 days on the software VHDL simulator.

Large Blue Gene/L system are simulated using one `bglsim` process per node, as shown in Figure 3. The `bglsim` processes run on different workstations and communicate through a custom message passing library (*CommFabric*), which simulates the connectivity within the system and outside. Additionally, different components of the system are simulated by separate processes that also link in CommFabric. Examples are: the IDo chip simulator, a functional simulator of an IDo chip that translates packets between the virtual JTAG network and Ethernet; the Tapdaemon and EthernetGateway processes to provide the Linux kernels in the simulated I/O nodes with connectivity to the outside network, allowing users to mount external file-systems and connect using telnet, ftp, etc. We use this environment to develop our communication infrastructure, the control infrastructure and we have successfully executed the MPI NAS Parallel Benchmarks [9].

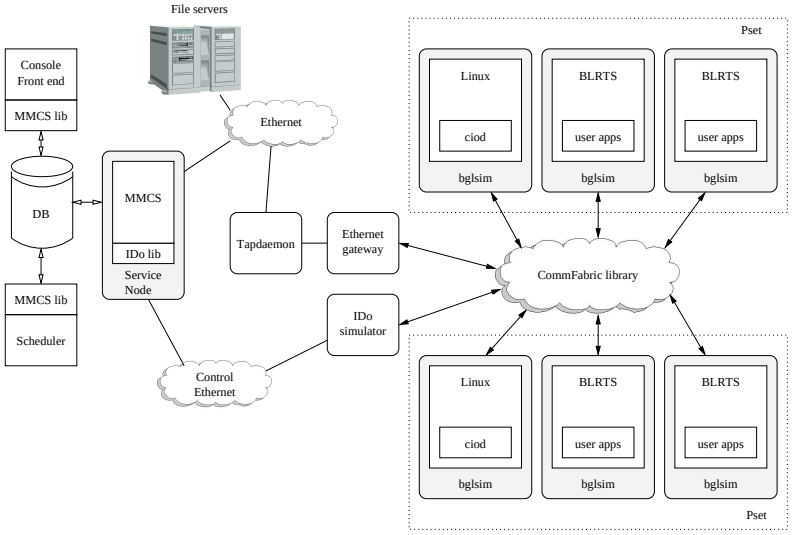

Fig. 3. Overview of the Blue Gene/L simulation environment. Complete Blue Gene/L chips are simulated by a custom architectural simulator (`bglsim`). A communication library (CommFabric) simulates the Blue Gene/L networks.

5 Conclusions

Blue Gene/L is the first of a new series of high performance machines being developed at IBM Research. The hardware plans for the machine are complete and the first small prototypes will be available in late 2003.

In this paper, we have presented a software system that can scale up to the demands of the Blue Gene/L hardware. We have also described the simulation environment that we are using to develop and validate this software system. Using the simulation environment, we are able to demonstrate a complete and functional system software environment before hardware becomes available. Nevertheless, evaluating scalability and performance of the complete system still requires hardware availability. Many of the implementation details will likely change as we gain experience with the real hardware.

References

1. ASCI Red Homepage. `http://www.sandia.gov/ASCI/Red/`.
2. ASCI White Homepage. `http://www.llnl.gov/asci/platforms/white`.
3. Cplant homepage. `http://www.cs.sandia.gov/cplant/`.
4. Earth Simulator Homepage. `http://www.es.jamstec.go.jp/`.
5. The MPICH and MPICH2 homepage. `http://www-unix.mcs.anl.gov/mpi/mpich`.
6. *Open Firmware Homepage*. `http://www.openfirmware.org`.
7. Process Management in MPICH2. Personal communication from William Gropp.

8. N. R. Adiga et al. An overview of the BlueGene/L supercomputer. In *SC2002 – High Performance Networking and Computing*, Baltimore, MD, November 2002.
9. G. Almasi, C. Archer, J. G. Castanos, M. G. 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. Submitted for publication to the 2003 Euro PVM/MPI workshop.
10. G. Almasi et al. Cellular supercomputing with system-on-a-chip. In *IEEE International Solid-state Circuits Conference ISSCC*, 2001.
11. L. Ceze, K. Strauss, G. Alm´asi, P. J. Bohrer, J. R. Brunheroto, C. Caşcaval, J. G. Castanos, D. Lieber, X. Martorell, J. E. Moreira, A. Sanomiya, and E. Schenfeld. Full circle: Simulating Linux clusters on Linux clusters. In *Proceedings of the Fourth LCI International Conference on Linux Clusters: The HPC Revolution 2003*, San Jose, CA, June 2003.
12. G. Chiola and G. Ciaccio. Gamma: a low cost network of workstations based on active messages. In *Proc. Euromicro PDP'97, London, UK, January 1997, IEEE Computer Society.*, 1997.
13. D. Greenberg, R. Brightwell, L. Fisk, A. Maccabe, and R. Riesen. A system software architecture for high-end computing. In *Proceedings of Supercomputing 97*, San Jose, CA, 1997.
14. W. Gropp, E. Lusk, D. Ashton, R. Ross, R. Thakur, and B. Toonen. MPICH Abstract Device Interface Version 3.4 Reference Manual: Draft of May 20, 2003. `http://www-unix.mcs.anl.gov/mpi/mpich/adi3/adi3man.pdf`.
15. S. A. Herrod. *Using Complete Machine Simulation to Understand Computer System Behavior*. PhD thesis, Stanford University, February 1988.
16. E. Krevat, J. Castanos, and J. Moreira. Job scheduling for the Blue Gene/L system. In *Job Scheduling Strategies for Parallel Processing*, volume 2537 of *Lecture Notes in Computer Science*, pages 38–54. Springer, 2002.
17. S. Pakin, M. Lauria, and A. Chien. High performance messaging on workstations: Illinois Fast Messages (FM) for Myrinet. In *Supercomputing '95, San Diego, CA, December 1999*, 1995.
18. M. Rosenblum, S. A. Herrod, E. Witchel, and A. Gupta. Complete computer simulation: The SimOS approach. *IEEE Parallel and Distributed Technology,*, 1995.
19. L. Shuler, R. Riesen, C. Jong, D. van Dresser, A. B. Maccabe, L. A. Fisk, and T. M. Stallcup. The PUMA operating system for massively parallel computers. In *In Proceedings of the Intel Supercomputer Users' Group. 1995 Annual North America Users' Conference*, June 1995.
20. M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI - The Complete Reference, second edition*. The MIT Press, 2000.
21. T. von Eicken, A. Basu, V. Buch, and W. Vogels. U-net: A user-level network interface for parallel and distributed computing. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles, Copper Mountain, Colorado*, December 1995.
22. T. von Eicken, D. E. Culler, S. C. Goldstein, and K. E. Schauser. Active Messages: a mechanism for integrated communication and computation. In *Proceedings of the 19th International Symposium on Computer Architecture*, May 1992.
23. S. J. Winwood, Y. Shuf, and H. Franke. Multiple page size support in the Linux kernel. In *Proceedings of Ottawa Linux Symposium*, Ottawa, Canada, June 2002.