

An Overview of the Blue Gene/L System Software Organization

George Almási¹, Ralph Bellofatto¹, José Brunheroto¹, Călin Cașcaval¹, José G. Castaños¹, Luis Ceze², Paul Crumley¹, C. Christopher Erway¹, Joseph Gagliano¹, Derek Lieber¹, José E. Moreira¹, Alda Sanomiya¹, and Karin Strauss²

¹ IBM Thomas J. Watson Research Center
Yorktown Heights, NY 10598-0218

{gheorghe, ralphbel, brunhe, cascaval, castanos, pgc, erway,
jgaglia, lieber, jmoreira, sanomiya}@us.ibm.com

² Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, 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 to deliver 360 Teraflops of peak computing power. With 65,536 compute nodes, Blue Gene/L represents a new level of scalability for parallel system, 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 a simulation framework that we have used to experiment with our ideas.

1 Introduction

In November 2001 IBM announced a partnership with Lawrence Livermore National Laboratory to build the Blue Gene/L supercomputer, a 65,536-node machine designed around embedded PowerPC processors. Through the use of system-on-a-chip integration [4], coupled with a highly scalable cellular architecture, Blue Gene/L will deliver 180 or 360 Teraflops of peak computing power, depending on 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 White³, Earth Simulator⁴) to a few thousands (Cplant⁵, ASCI Red⁶) 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 [7]) 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 through an Ethernet network. The end user's view of a system is of a flat, toroidal, 64K processor system, but the system's 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.

³ <http://www.llnl.gov/asci/platforms/white/>

⁴ <http://www.es.jamstec.go.jp/>

⁵ <http://www.cs.sandia.gov/cplant/>

⁶ <http://www.sandia.gov/ASCI/Red/>

2 An Overview of the Blue Gene/L Supercomputer

Blue Gene/L (BG/L) is a new architecture for high performance parallel computers based on low cost embedded PowerPC technology. A detailed description Blue Gene/L is provided in [3]. 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. The cores also have a 2KB L2 cache each and 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 440 FPU. 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 700Mhz the peak performance of a node is 2.8 GFlop/s. If both cores and FPUs in a chip are used, the peak performance per node is 5.6 GFlop/s.

The standard PPC440 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. 8KB 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 scratchpad. The PPC440 cores can interrupt each other and 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 can support a maximum of 2 GB external memory but in the current configuration with 256 Mb DDR chips each node can directly address 256 MB at 5.5 GB/s bandwidth and 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 larger memory. Their only difference is in their card packaging which determines the enabled networks.

2.2 Blue Gene/L Communications Hardware

The PowerPC processor(s) in the compute node can only address the node’s local memory. 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 3-D 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 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, broadcast and reductions of packets, 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, forwarding 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. Therefore, tree packets are the main mechanism for communication between these nodes.

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

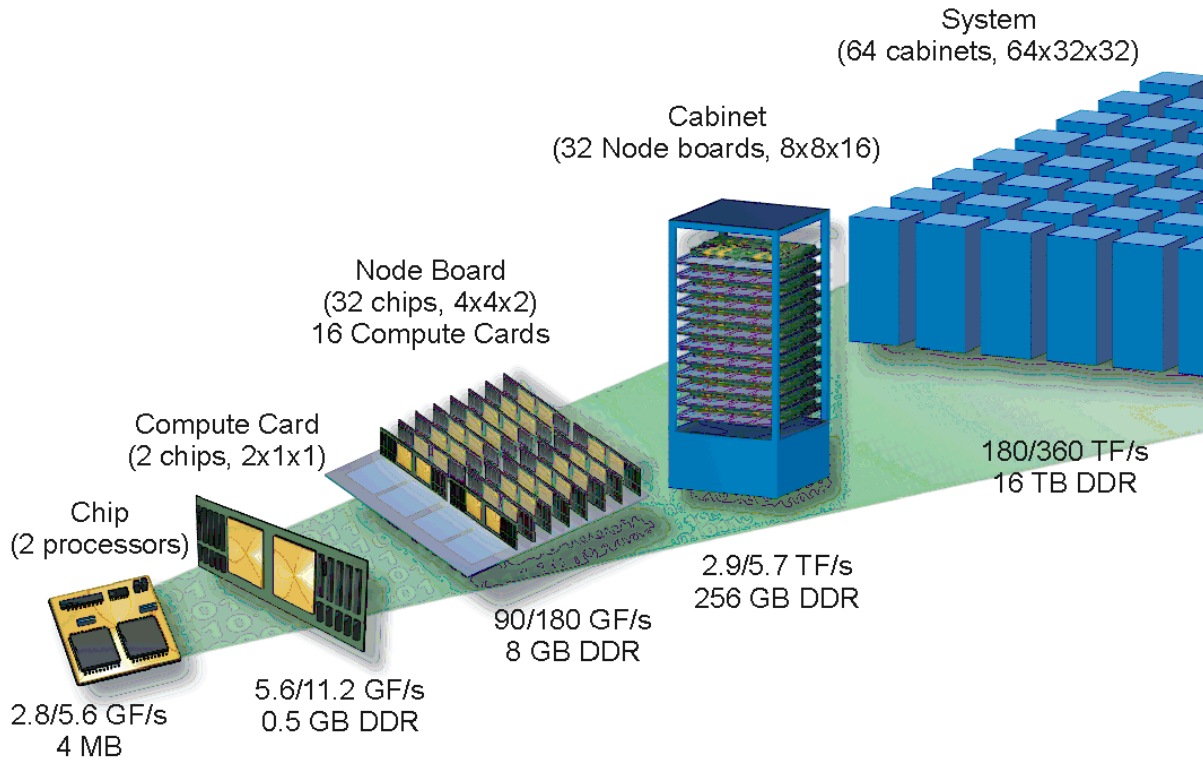


Fig. 1. High-level organization of the Blue Gene/L supercomputer. All 65,536 compute nodes are organized in a $64 \times 32 \times 32$ three-dimensional torus.

The Blue Gene/L computational core can be subdivided into *partitions*, which are completely (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 rack) 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 boundary chips of a partition. In the current packaging schema of node cards and boards, the smallest independent mesh that we can build contains 128 compute nodes and 2 I/O nodes.

Finally, each ASIC contains a 1Gbit/s Ethernet macro for external connectivity and supports a serial JTAG network for booting, control and monitoring of the system through an unarchitected network. Only I/O nodes are attached to a 1Gbit/s Ethernet network, giving 1000x1Gbit/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 100Mhz 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 SPI protocol reads a small EPROM that contains the serial number of each board. The JTAG protocol is used for reading and writing to any address of the 8KB SRAMs in the BG/L chips, 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 each node and do not require any software intervention in the target nodes.

3 Blue Gene/L System Software

Scalability and complexity are just a few of the challenges posed by a machine such as Blue Gene/L. To address these challenges we developed the system software architecture presented in Figure 2, and described in detailed in the next sections.

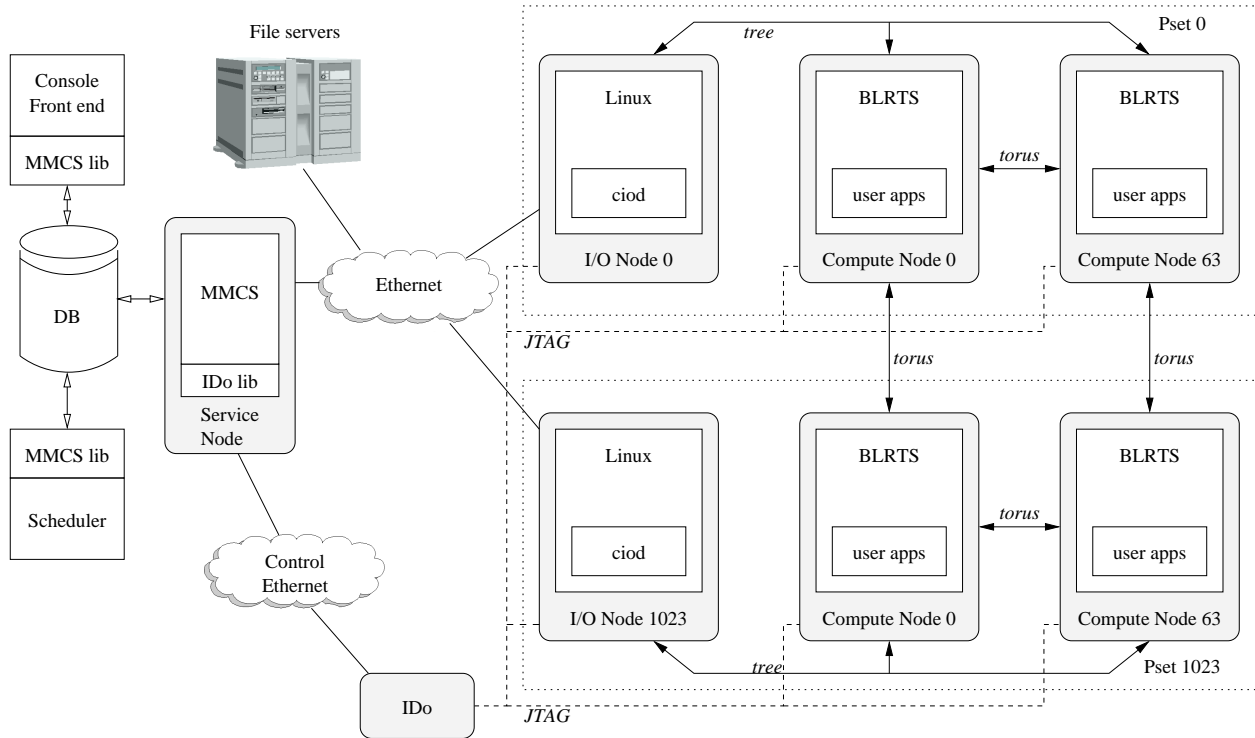


Fig. 2. Outline of the Blue Gene/L system software. The computational core is partitioned into 1024 logical processing sets (psets), each with one I/O node running Linux and 64 compute nodes running the custom BLRTS kernel. External entities connect the computational core through two Ethernet networks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

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 Monta Vista's 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 JTAG network. We also 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. The Gigabit Ethernet controller embedded in each Blue Gene/L chip is the new EMAC4, and we support it in the kernel. 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 a ramdisk linked against the Linux kernel provides the initial root file system. The contents of the ramdisk are also based on Monta Vista's distribution and includes shells, simple utilities, shared libraries, and network clients such as `ftp` and `nfs`.

Due to the lack of coherence of the L1 caches, the current version of Linux for Blue Gene/L uses only 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 (note that the Blue Gene/L's BIC is not OpenPIC [2] 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 functionality. The performance and effectiveness of this solution is still an open issue. A second, promising mode of operation runs Linux in only 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 large page support. The standard PPC440 embedded processors handle all TLB misses in software. The average number of instructions required to handle these misses has significantly decreased. Nevertheless, it has been shown that larger pages improve performance [18].

3.2 System Software for the Compute Nodes

The "Blue Gene/L Run Time Supervisor" (BLRTS, pronounced "blurts") is a new 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 [14]. 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 from the application by appropriately programming the PowerPC MMU. Resources (torus, mutexes, barriers, scratchpad) are partitioned between application and kernel. In the current implementation, the entire torus network is mapped into user space while the tree network is managed by the kernel.

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 not implemented. Program launch, termination, and file I/O is accomplished via messages passed between the compute node and its control node over the tree network, using a point-to-point packet addressing mode. This functionality is provided by a daemon called CIOD (Console IO Daemon) running in the I/O nodes. CIOD provides job control and I/O management on behalf of all the compute nodes in the pset. Under normal operation, all messaging between CIOD and BLRTS is synchronous: all file I/O operations are blocking on the application side. Currently we support only one application running on a subset of the nodes of the pset (the subset can be the entire pset, or if not, the nodes not in the subset will be idle). 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 initial mode of operation when the machine is deployed, 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 Blue Gene/L core is stateless when powered on (no nonvolatile memory, no disk) and all the basic control infrastructure is off-loaded to external service nodes. The service node performs a variety of system management services, including: (i) machine booting, (ii) system monitoring, and (iii) job execution. The control infrastructure is a critical component of our design: without it the machine will not boot.

We chose to maintain all the state of a Blue Gene/L system using standard database technology. Databases naturally provide scalability, reliability, security, portability, logging, and robustness. The Midplane Management and Control

System (MMCS) processes running in the service nodes provide two paths into the Blue Gene/L complex: a custom library to access the restricted JTAG network and directly manipulate Blue Gene/L nodes; and sockets over the 1Gbit/s Ethernet network to manage the I/O nodes and compute nodes on a booted partition.

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.

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 only 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 exported to user process through an entry in the `proc` file system. BLRTS implements a system call to request the node's personality.

System Monitoring 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 (and only the service node) through the control network. The (compute or 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 Job execution in Blue Gene/L 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. (Each compute node executes exactly one compute process of the parallel job.) The scheduler selects an appropriate 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 [10]. 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 infrastructure abstracts the hardware complexity by providing a layered API. At the bottom of the hierarchy the *packet layer* provides a *simple* abstraction that matches the hardware, and allows the transmission of single packets along the torus and tree networks. The next layer in the hierarchy, the *message layer*, allows the transmission of arbitrary sized buffers. Finally, the MPI library leverages the message layer to implement point-to-point communication and provides a familiar interface to users.

The packet layer is designed to simplify access to the network hardware. It abstracts the notion of FIFOs, memory mapped ports and device control registers (DCRs). It presents an API consisting of essentially three functions: initialization, packet send and receive. Communication endpoints are abstracted into torus and tree *devices*. Hardware restrictions, such as the 256 byte limit on packet size and the 16 byte alignment requirements on packet buffers, are reflected by the packet layer API.

The essential feature of the packet layer is that it provides a *mechanism* to use the network hardware but doesn't impose *policies* on how to use it. For example, it provides multiple devices without controlling their management; all send and receive operations are non-blocking leaving it up to the higher layers to implement synchronous, blocking and interrupt driven communication models. The packet layer is stateless.

The message layer is an active message system [17, 6, 12, 16] built on top of the packet layer that allows the transmission of arbitrary buffers among torus nodes. Like many other active message libraries, it is designed to work asynchronously: callbacks are invoked by the message layer when a message starts arriving, has arrived or has been sent. Because the message layer is designed to support MPI, it has a built-in notion of message matching: each message contains a unique combination of identifiers that can be used by MPI.

The message layer impose policy decisions:

- **Use of the second CPU.** Several strategies are under consideration for making use of the second processor in co-processor mode (see Section 3.2), by allowing it to service the tree/torus hardware. For example, sending a message involves packetizing it and sending the packets. The second processor can be involved in: sending packets from a common non-cacheable memory area, packetizing entire messages with various degrees of involvement by the first processor.
- **Out-of-order delivery of torus packets.** The Blue Gene/L torus network has no ordering guarantees; the message layer has to provide enough guarantees to allow software layered on top to work correctly. Within a message, out-of-order packets are handled by the unpackitizer when it reconstitutes the message out of its component packets.
- **Alignment issues.** As discussed before, the packet layer is alignment sensitive. The message layer handles message alignment problems, but requires at least one memory-to-memory copy when the *alignment phase* of the send and receive buffers differ. The memory copy can be performed either by the sender or the receiver depending on the amount of information available.
- **Polling vs. interrupts** The current implementation of the Blue Gene/L message layer is designed to be used in polling mode.

MPI: Blue Gene/L is designed primarily to run MPI [15] workloads. The degree to which the MPI implementation can be made fast and efficient will to a large extent drive user satisfaction with the machine.

Blue Gene/L MPI is based on MPICH2 [1] (currently in beta stage), a public domain implementation of the MPI 2 protocol. MPICH2 provides better scalability, lower overhead, and a modular software approach: it allows porting to a new architecture based on the so-called Abstract Device Interface [8] layer. A default implementation, called the channel interface (well suited for TCP/IP like protocols) is also provided in an extendible way.

Blue Gene/L MPI implements the abstract device interface using the active message paradigm of the message layer, and bypasses the channel interface. Here is a short list of the challenges we are facing in this implementation:

- **Scalability** of Blue Gene/L MPI is paramount. In many MPI implementations today, communication is established between every pair of MPI processes at startup time (e.g. in MPI LAM [11] TCP/IP sockets are opened from every MPI process to every other MPI process). The Blue Gene/L implementation of MPICH2 cannot afford to open communication between every pair of processes, because the necessary state information and buffers would exhaust a large part of the available memory on each processor. Thus, communication between MPI processes has to be opened and closed on a per-need basis, and often used communication patterns may have to be cached.
- **Latency, bandwidth and the second processor:** Use of the communication co-processor relieves the main processor of handling the network hardware, and allows the overlap of computation and communication; hence it allows optimal use of the available bandwidth. However, some latency is inevitably added to every message sent through the co-processor. Thus, in some cases (for example, when sending extremely short MPI messages in non-bandwidth-critical situations) sending packets directly from the computation processor may realize better latency without otherwise affecting performance.

- **Message ordering and MPI semantics:** As discussed above, the message layer does not guarantee ordered delivery of incoming messages. Fortunately for MPI, such guarantees are not really needed: the only order needed by MPI semantics is that of matching incoming messages in the order they were sent [15]. The message layer provides mechanisms to identify this order. However, out-of-order incoming messages can cause performance problems, because messages may have to be buffered to allow earlier messages (that have not yet arrived) to be matched first.
- **Collective operations:** The default implementation of MPI collective operations in MPICH2 relies on sequences of point-to-point messages, building a tree of connections to perform broadcasts and reductions. We exploit the special properties of the torus and tree networks for implementing optimized versions of the collective MPI operations as follows:
 - Use the *global combining tree* to perform `allreduce` operations on the `MPI_COMM_WORLD` communicator.
 - Setting the *deposit bit* of a torus packet causes it to be received by every node it touches on the way from the sender to the receiver. Deposit bits can be deployed to fill lines, planes and square volumes in the torus with data. Hence some categories of broadcasts are best done in the torus with deposit bits set.
 - Finally, the *barrier synchronization wires* can be used in MPI to implement very fast barriers.

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. The simulator, called `bglsim`, is built using the principles in [13, 9]. Each component of the BG/L ASIC is modeled separately with the desired degree of accuracy. Accuracy can be traded for performance. For our simulation environment, we model the processor instructions functionally, i.e., each instruction takes one cycle to execute. We also model cache behavior, memory system behavior and all the Blue Gene/L specific devices: tree, torus, jtag, device control registers, etc. `bglsim` implements enough functionality to run the Linux kernel for the I/O nodes and BLRTS for the compute nodes. Applications run on top of these kernels, under user control.

`bglsim` simulates an entire BG/L chip at approximately 2 million simulated instructions per second when running on 1.2 GHz Pentium III machine – 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 pure 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 than 20 days on the software VHDL simulator.

Each node in a large Blue Gene/L system is simulated by a `bglsim` process. Multiple `bglsim` processes run on different workstations and communicate through a custom message passing library called *CommFabric*. This library simulates the connectivity between BG/L chips, I/O chips, and the external world as outlined in Figure 3. We use this environment to develop our communication infrastructure, and we have successfully executed the MPI NAS Parallel Benchmarks [5].

The physical description of a the simulated BG/L system is contained in a database called *BGLMachine*. This database includes node specific data, such as serial numbers, and connectivity information, such as the placement of nodes, cards, boards and midplanes. *BGLMachine* also contains a description of a running simulation (such as the location of every `bglsim` process).

User applications, runtime libraries, and the kernel create packets and write them to memory mapped devices. `bglsim` interprets these write requests and forwards the packets to *CommFabric*. *CommFabric* analyzes the header of the packet and queries the *BGLMachine* database to determine packet routing.

Our simulation environment for large BG/L systems includes several components in addition to the instruction level simulator described previously. The I/O chip simulator is a functional simulator of an I/O chip that translates packets between the virtual JTAG network and Ethernet. The *Tapdaemon* and *EthernetGateway* processes provide Linux kernels in the simulated I/O nodes with the ability to mount external file systems. Users can also connect to the I/O nodes (telnet, ftp, etc.). The development of the control infrastructure is thus decoupled from the simulation environment.

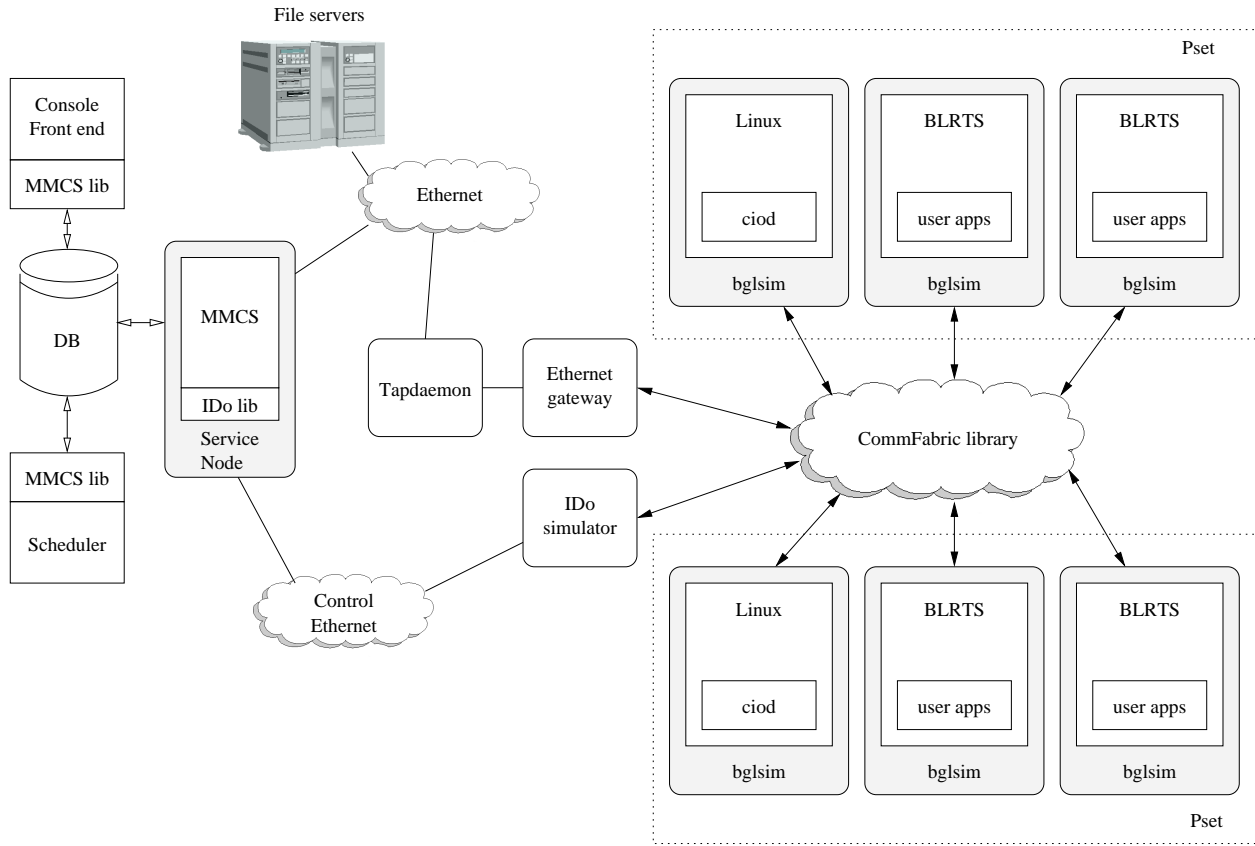


Fig. 3. Overview of the Blue Gene/L simulation environment. Complete Blue Gene/L chips are simulated by a custom architectural simulator (`bglsim`). A communication library (`CommFabric`) simulates the Blue Gene/L networks.

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. The MPICH and MPICH2 homepage. <http://www-unix.mcs.anl.gov/mpi/mpich>.
2. *Open Firmware Homepage*. <http://www.openfirmware.org>.
3. N. R. Adiga et al. An overview of the BlueGene/L supercomputer. In *SC2002 – High Performance Networking and Computing*, Baltimore, MD, November 2002.
4. G. Almasi et al. Cellular supercomputing with system-on-a-chip. In *IEEE International Solid-state Circuits Conference ISSCC*, 2001.
5. D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, D. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga. The NAS Parallel Benchmarks. *The International Journal of Supercomputer Applications*, 5(3):63–73, Fall 1991.
6. 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.
7. 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.
8. W. Gropp and E. Lusk. MPICH abstract device interface version 3.3 reference manual: Draft of october 17, 2002. <http://www-unix.mcs.anl.gov/mpi/mpich/adi3/adi3man.pdf>.
9. S. A. Herrod. *Using Complete Machine Simulation to Understand Computer System Behavior*. PhD thesis, Stanford University, February 1988.
10. 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.
11. N. J. Nevin. The performance of LAM 6.0 and MPICH 1.0.12 on a workstation cluster. Technical Report OSC-TR-1996-4, Ohio Supercomputing Center, Columbus, Ohio, 1996.
12. 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.
13. M. Rosenblum, S. A. Herrod, E. Witchel, and A. Gupta. Complete computer simulation: The SimOS approach. *IEEE Parallel and Distributed Technology*, 1995.
14. 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.
15. M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI - The Complete Reference, second edition*. The MIT Press, 2000.
16. 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.
17. T. von Eicken, D. E. Culler, S. C. Goldstein, and K. E. Schausser. Active Messages: a mechanism for integrated communication and computation. In *Proceedings of the 19th International Symposium on Computer Architecture*, May 1992.
18. 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.