RC24146 (W0701-007) January 2, 2007
Computer Science

IBM Research Report

The Blue Gene/L Supercomputer: A Hardware and Software Story

José E. Moreira, Valentina Salapura
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

The Blue Gene/L Supercomputer: A Hardware and Software Story[1]

José E. Moreira, Valentina Salapura

IBM Thomas J. Watson Research Center
Yorktown Heights, NY 10598-0218
{jmoreira,salapura}@us.ibm.com

Abstract

The Blue Gene/L system at the Department of Energy Lawrence Livermore National Laboratory in Livermore, California is the world's most powerful supercomputer. It has achieved groundbreaking performance in both standard benchmarks as well as real scientific applications. In that process, it has enabled new science that simply could not be done before. Blue Gene/L was developed by a relatively small team of dedicated scientists and engineers. This article is both a description of the Blue Gene/L supercomputer as well as an account of how that system was designed, developed, and delivered. It reports on the technical characteristics of the system that made it possible to build such a powerful supercomputer. It also reports on how teams across the world worked around the clock to accomplish this milestone of high-performance computing.

1. Introduction

The Blue Gene/L system [15] at the Department of Energy Lawrence Livermore National Laboratory in Livermore, CA (LLNL) [26] is the world's most powerful supercomputer [30]. Its 65,536 dual-processor compute nodes deliver over 360 Tflops of peak processing power. Sustained performance of over 280 Tflops has been demonstrated on the LINPACK benchmark used for the TOP500 ranking [29], and performance of over 200 Tflops has been achieved in QBox [12],[13], a real scientific application from LLNL.

Blue Gene/L derives its computing power from scalability [1],[6],[18]. Each computing node is optimized to deliver great computational rate per Watt and to operate with other nodes in parallel. The result is a machine that can scale to very large sizes and deliver unmatched aggregate performance.

The LINPACK score for the LLNL Blue Gene is 4 times higher than the first non-Blue Gene/L system in the TOP500 list. (The second highest score in that list is a smaller instance of Blue Gene/L, at the IBM Thomas J. Watson Research Center.) The figure for QBox is over half of the machine's peak performance and more than twice the peak performance of the fastest non-Blue Gene systems in the world. These figures are even more impressive when we take into the account that the Blue Gene/L machine at LLNL occupies about a quarter of the area and consumes only a quarter of the power of those competing systems. Also, the entire Blue Gene/L system at LLNL can be administered by a single person.

The Blue Gene/L at LLNL was delivered on time and on budget, under fluid requirements, by a team that is approximately 10-20% of the size that normal practices would dictate. This paper is an account of how this was accomplished. Blue Gene's success was made possible by a combination of design decisions that led to a simple yet powerful computer system and by strong team work across sub-teams around the world. Close cooperation between IBM and LLNL led to a system that was powerful, stable, and useful for real science.

This paper is organized as follows. Section 2 describes the design principles that were followed in the development of the Blue Gene/L supercomputer. Section 3 summarizes

[1] This paper is an expanded version of two other papers: (1) *Designing a Highly-Scalable Operating System: The Blue Gene/L Story*, to be published in Proceedings of SC'06, and (2) *Delivering Teraflops: An Account of how Blue Gene was Brought to Life*, in Proceedings of JVA 2006.

the Blue Gene/L system architecture and Section 4 describes its main software components. Section 5 talks about team organization and how the work was distributed. Section 6 reports on the experience of building and testing Blue Gene/L and how it grew over time to its final size at LLNL. Section 7 reports on the impact that Blue Gene/L is having on real science and Section 8 presents our conclusions and discusses what the future may hold for Blue Gene.

2. Design Principles

Three main design principles were essential to the successful development of the Blue Gene/L supercomputer: simplicity, efficiency, and familiarity. In this section, we discuss how each of these principles was applied to the hardware and software of Blue Gene/L.

Of the three principles, simplicity was the most important and pervasive. That is a more subtle concept than it seems. What it really means is that the Blue Gene/L team chose a design that let them focus their resources into those areas that had a high performance payoff. The simplicity of Blue Gene/L shows in various dimensions: chip design, physical packaging, and software structure.

One part of the simplicity story is to know what to reuse from existing systems and what to redesign/reimplement specifically for Blue Gene/L. On the hardware side, for example, the PowerPC processors in the compute nodes were reused as is from a standard *Application Specific Integrated Circuit* (ASIC) library that IBM provides. The floating-point unit was modified from an existing implementation to double its computation rate. The networks were designed from scratch to enable high performance communication and support scaling. On the software side, Linux was used to provide file system and socket I/O functionality, whereas a lightweight kernel and certain MPI collective operations were specifically implemented for Blue Gene/L, again in support of scalability.

The other, and probably more important, part of simplicity is maintaining the overall system concept simple. For example, Blue Gene/L only supports spatial partitioning. A large Blue Gene/L machine can be subdivided into multiple independent partitions and each partition can only run one application program at a time. Eliminating the possibility of time-sharing resources greatly simplified the software design and allowed for better performance.

When we discuss the technical characteristics of the system in Section 3, we will point out more specific examples of how the simplicity principle was applied.

Efficiency was the next most important design principle for Blue Gene/L. This was a system intended to deliver unprecedented levels of performance, which could only be accomplished by paying focused attention on efficiency of each component and of the overall system. There are two main dimensions to the efficiency of Blue Gene/L: single-node efficiency and scalability.

Blue Gene/L compute nodes operate at a relatively low frequency of 700 MHz. Yet, they achieve per node performance on real applications that is on-par with much more expensive (and power hungry) server-grade processors. This is made possible by a variety of factors. First, the low frequency processors in Blue Gene/L consume very little area and power, so it was possible to package two of those processors in a single chip. Second, the chip offers a very wide memory bus. Combined with the tight integration between processors and memory controller within the compute ASIC, this delivers a high-performance memory system that can keep the processors properly fed with data. Third, there are several architectural innovations within the chip, like a level-2 cache with prefetching engine [24], that are effective in speeding up the execution of scientific applications. Finally, a shallow and optimized software stack removes much of the overhead in memory management found in other system.

From a scalability perspective, Blue Gene/L performed well primarily for two reasons. First, the tight integration between networks and processors within the compute ASIC delivered a feature-rich interconnect system with low latency and high bandwidth. We call the interconnect feature-rich because it has several architectural features (e.g., broadcast, multicast, combining operations) that are useful for scientific computing. Second, again a shallow and optimized software stack reduces overhead and exploits the architectural features, leading to high application scalability and performance.

System support for scaling to large configurations allows real world applications to deliver a large fraction of peak performance. It also ensures we can achieve high power/performance efficiency through software that can efficiently take advantage of the system. After all, inefficient software on efficient hardware is no better than inefficient hardware [22],[23].

We note that the simplicity of Blue Gene/L design actually enabled its efficiency. For example, the strict spatial partitioning mentioned above eliminates the need of protection between jobs. This, in turn, allows user applications to get direct access to the communication hardware, greatly reducing software stack overhead.

Familiarity was the third and last main design principle applied to Blue Gene/L. By familiarity we mean that we wanted to build a system that did not look too different or too difficult to program when compared to other systems already in use. Familiarity in Blue Gene/L was accomplished both through hardware and software.

Blue Gene/L uses PowerPC 32-bit processors. This instruction set architecture has been around for approximately 20 years and there is a large body of software and expertise available for it. The processors were paired with a reasonable amount of memory (on the order of 256 or 512 MB per processor), thus creating an environment that could run conventional software. This allowed us to leverage compilers, libraries, and operating systems that already existed for the PowerPC processors.

Familiarity at the user application level was also ensured by using standard compilers (in particular the XL family of compilers for C, C++, and FORTRAN) and libraries (in particular MPI), so that existing parallel codes could be easily ported to Blue Gene/L. There was significant work in optimizing the compilers and libraries for Blue Gene/L, but the interface presented to the programmer is something they are familiar with. We note that familiarity in Blue Gene/L was accomplished without sacrificing simplicity and/or efficiency.

3. System Architecture

We briefly discuss the overall system architecture of Blue Gene/L in this section. For a more thorough description, the reader is referred to [9]. A Blue Gene/L system consists of a *compute section*, a *file server section*, and a *host section*. See Figure 1 for a high-level view of a Blue Gene/L system. The compute and I/O nodes form the computational core of Blue Gene/L, what we call the compute section. The host section consists of the *service node* and one or more *front-end nodes*. The service node controls the compute section through an Ethernet *control network*. The control traffic is converted to lower level protocols (e.g., JTAG) before actually making to the compute and I/O nodes. The front-end nodes provide the user interface to the system, supporting job compilation, job launch and job debugging. The file server section consists of a set of file servers. The I/O nodes in the compute section connect the core to file servers and front-end nodes through a separate Ethernet *functional network*.

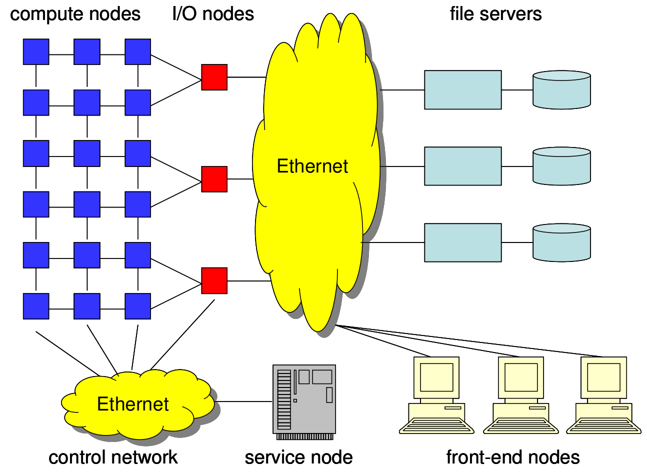

Figure 1: High-Level View of a Blue Gene/L System.

3.1. Compute Section

The compute section of a Blue Gene/L system is what is usually called a Blue Gene/L machine. It is where the application programs run. The Blue Gene/L compute section consists of a three-dimensional rectangular array of compute nodes, interconnected in a toroidal topology along the *x*, *y*, and *z* axes. Each compute node can execute one or two user application processes, as described below. I/O nodes, distinct from the compute nodes and not in the toroidal interconnect, are also part of the compute section. I/O and compute nodes are interconnected by a collective network, and each I/O node can communicate with the outside world through a 1-Gbit Ethernet link.

The entire system is built around a system-on-a-chip (SOC) concept [3]. The basic building block of Blue Gene/L is the Blue Gene/L Compute ASIC (BLC). The internals of that ASIC are shown in Figure 2. The BLC contains two non-coherent PowerPC 440 cores, each with its own private L1 cache (split for instructions and data, with 32 KB each). Associated with each core is a small (2 KB) L2 cache that acts as a prefetch buffer and matches the 32-byte cache line size of the L1 to the 128-byte cache line size of the L3. Completing the on-chip memory hierarchy is 4 MB of embedded DRAM (eDRAM) that is configured to operate as a shared L3 cache. Also on the BLC is a memory controller (for external DRAM) and interfaces to the five networks used to interconnect Blue Gene/L compute and I/O nodes: torus, collective, global barrier, Ethernet, and control network.

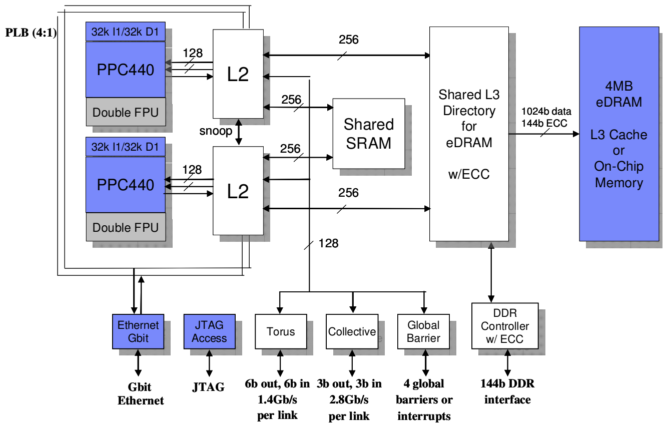

Figure 2: Blue Gene/L Compute ASIC (BLC).

The interconnection networks are primarily used for communication primitives used in parallel high-performance computing applications. These primitives include message passing for data sharing and exchange, barrier synchronization, and global operations such as reduction operations. The main interconnection network is the torus network, which provides high performance data communication with low latency and high throughput.

A collective network supports efficient collective operations, such as reduction. Arithmetic and logical operations are implemented as part of the communication primitives to facilitate low-latency collective operation, since it is recognized that current applications spend increasing percentage of time performing global operations.

The compute ASIC, designed specifically for Blue Gene/L, contains the processors, interconnection networks, and memory controllers that are used to build the system. One of these compute ASICs, combined with conventional memory chips constitutes a compute node of Blue Gene/L. The Blue Gene/L I/O nodes are assembled exactly the same way. Blue Gene/L compute and I/O nodes both use this ASIC with 512 MB of external memory (a 1 GB option is also available for the compute nodes), but their roles are quite different, as described in Section 4. Only compute nodes are interconnected through the torus network. Conversely, only I/O nodes have their Ethernet port connected to the functional network. The I/O nodes plug into an Ethernet fabric together with the file servers and front-end nodes. The collective and global barrier networks interconnect all (compute and I/O) nodes. The control network is used to control the hardware from the service node.

Blue Gene/L, large as it is, is built of essentially two kinds of chips: the compute ASICs described above, designed and built by IBM specifically for Blue Gene/L, and memory chips that are commercially available. There are also link ASICs that are used to redrive the network signals across racks and to partition the machine. There are 9 memory chips for every compute ASIC and approximate 20 compute ASICs for every link ASIC. The link ASIC is much simpler than the compute ASIC, and also it does not run any software.

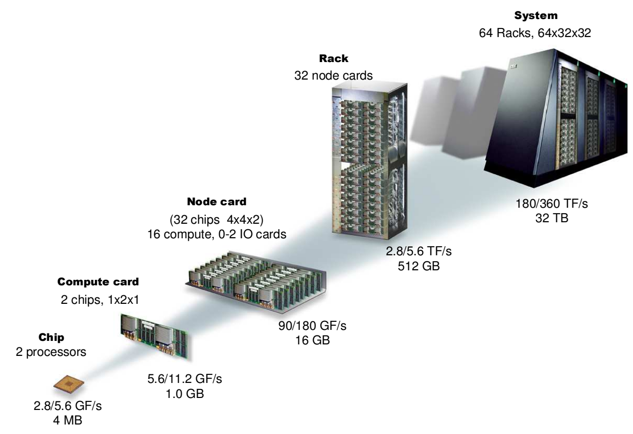

Figure 3: Build Blue Gene/L, from chip to system.

Having only one chip to design means just one design team. It also means just one test, validation and bring up team. It means just one chip to simulate and, obviously, one chip to get right. It also greatly simplifies the rest of the system design, since boards have to be designed for just a few different kinds of components. Blue Gene/L is built of seven kinds of cards: (1) compute card, (2) I/O card, (3) node card, (4) link card, (5) service card, (6) midplane card, and (7) clock card. An exploded view of Blue Gene/L, showing how we assemble the parts from the compute ASIC up is show in Figure 3.

The compute card holds two compute ASICs and 18 or 36 memory chips (9 or 18 for each compute ASIC). They implement the compute nodes. The I/O cards are very similar (two compute ASICs and 18 memory chips), but they expose the Ethernet network in the ASICs, so that they can be used as I/O nodes. The node card is where 16 compute and (up to) two I/O cards can be plugged. It implements the interconnection network among those nodes. Each link card contains six link chips. There are 16 node cards and four link cards per midplane. Together with a service card they plug into a midplane card, forming a midplane. The service card allows external control of the midplane from the host section. There is one clock card per rack. Its function is to distribute the master Blue Gene/L clock signal to the midplanes in the rack and to other racks.

Midplanes are units of 512 compute nodes, in an 8 x 8 x 8 arrangement. Each midplane can have a variable number of I/O nodes, from 8 to 64. (Compute nodes are associated to I/O nodes in an 8:1 to 64:1 ratio.) Each midplane also has 24 link chips used for inter-midplane connection to build larger systems. Midplanes are arranged two to a rack, and racks are arranged in a two-dimensional layout of rows and columns. Because the midplane is the basic replication unit, the dimensions the array of compute nodes must be a multiple of 8. The Blue Gene/L machine at LLNL in particular, is of size 64 x 32 x 32, along the *x*, *y*, and *z* axes respectively. Because of cabling and other packaging restrictions, not every size with dimensions multiple of 8 is legal, but we have delivered Blue Gene/L systems with 1, 2, 4, 8, 10, 20, and 64 racks, and many other sizes are possible (including bigger than 64).

Blue Gene/L can be partitioned along midplane boundaries. A partition is formed by a rectangular arrangement of midplanes. Each partition can run one and only one job at any given time. During each job, the compute nodes of a partition are in one of two modes of execution: coprocessor mode or virtual node mode. All compute nodes stay in the same mode for the duration of the job. These modes of execution are described in more detail below.

3.2. File Server Section

The file server section of a Blue Gene/L system provides the storage for the file system that runs on the Blue Gene/L I/O nodes. Several parallel file systems have been ported to Blue Gene/L, including GPFS, PVFS2, and Lustre [21]. To feed data to Blue Gene/L, multiple servers are required. The Blue Gene/L system at LLNL, for example, uses 224 servers operating in parallel. Data is striped across those servers, and a multi-level switching Ethernet network is used to connect the I/O nodes to the servers. The servers themselves are standard rack-mounted machines, typically Intel, AMD or POWER processor based.

3.3. Host Section

The host section for a Blue Gene/L system consists of one service node and one or more front-end nodes. These nodes are standard POWER processor machines. At LLNL, the service node is a 16-processor POWER4 machine, and each of the 14 front-end nodes is a PowerPC 970 blade.

The service node is responsible for controlling and monitoring the operation of the compute section. The services it implements include: machine partitioning, partition boot, application launch, standard I/O routing, application signaling and termination, event monitoring (for events generated by the compute and I/O nodes), and environmental monitoring (for things like power supply voltages, fan speeds, and temperatures).

The front-end nodes are where users work. They provide access to compilers, debuggers and job submission services. Standard I/O from user applications is routed to the submitting front-end node.

4. Blue Gene System Software

To support execution of application processes, compute nodes run a lightweight operating system called the Compute Node Kernel (CNK). This simple kernel implements only a limited set of services.

Scientific middleware for Blue Gene/L includes a user-level library implementation of the MPI standard, optimized to take advantage of Blue Gene/L networks, and various math libraries, also in user level. Implementing all the message passing functions in user mode had the effect of simplifying the supervisor (kernel) code of Blue Gene/L, thus facilitating development, testing, and debugging.

The Blue Gene/L I/O nodes run a port of the Linux operating system. The I/O nodes act as gateways between the outside world and the compute nodes, complementing the services provided by CNK with file and socket operations, debugging, and signaling.

This split of functions between I/O and compute nodes, with the I/O nodes dedicated to system services and the compute nodes dedicated to application execution, resulted in a simplified design for both components. It has also been fundamental in enabling Blue Gene/L scalability and robustness, and in achieving a deterministic execution environment [17],[20],[27].

We now describe in more detail the operating system solution for Blue Gene/L. We start with the overall architecture and proceed to describe the role of the separate components. As previously mentioned, the software architecture reflects to a great degree the hardware architecture of Blue Gene/L.

4.1. Overall operating system architecture

A key concept in the Blue Gene/L operating system solution is the organization of compute and I/O nodes into logical entities called *processing sets* or *psets*. A pset consists of one I/O node and a collection of compute nodes. Every system partition, in turn, is organized as a collection of psets. All psets in a partition must have the same number of compute nodes, and the psets of a partition must cover all the I/O and compute nodes of the partition. The psets of a partition never overlap. The supported pset sizes are 8, 16, 32, 64 and 128 compute nodes, plus the I/O node.

The psets are a purely logical concept implemented by the Blue Gene/L system software stack. They are built to reflect the topological proximity between I/O and compute nodes, thus improving communication performance within a pset. The regular assignment of compute to I/O nodes enforced by the pset concept allows us to simplify the system software stack while delivering good performance and scalability. With a static assignment of I/O to compute nodes, it becomes easier to separate operating system responsibilities. To understand those responsibilities, it is useful to have a picture of the job model for Blue Gene/L.

A Blue Gene/L job consists of a collection of *N* compute processes. Each process has its own private address space and two processes of the same job communicate only through message passing. The primary communication model for Blue Gene/L is MPI. The *N* compute processes of a Blue Gene/L job correspond to tasks with ranks 0 to *N*-1 in the MPI_COMM_WORLD communicator.

Compute processes run only on compute nodes; conversely, compute nodes run only compute processes. The compute nodes of a partition can all execute either one process (in coprocessor mode) or two processes (in virtual node mode) each. In coprocessor mode, the single process in the node has access to the entire node memory. One processor executes user code while the other performs communication functions. In virtual node mode, the node memory is split in half between the two processes running on the two processors. Each process performs both computation and communication functions. The compute node kernel implements these models in the compute nodes. See [9] and [32] for related work in other systems.

I/O nodes behave more like conventional computers. In fact, each I/O node runs one image of the Linux operating system. It can offer the entire spectrum of services expected in a Linux box, such as multiprocessing, file systems, and a TCP/IP communication stack. These services are used to extend the capabilities of the compute node kernel, providing a richer functionality to the compute processes. Due to the lack of cache coherency between the processors of a Blue Gene/L node, we only use one of the processors of each I/O node. The other processor remains idle.

4.2. The compute node kernel

The compute node kernel (CNK) accomplishes a role similar to that of PUMA [32],[33] in ASCI Red by controlling the Blue Gene/L compute nodes. It is a lean operating system that performs a simple sequence of operations at job start time. This sequence of operations happens in every compute node of a partition, at the start of each job:

1. It creates the address space(s) for execution of compute process(es) in a compute node.
2. It loads code and initialized data for the executable of that (those) process(es).
3. It transfers processor control to the loaded executable, changing from supervisor to user mode.

The CNK consumes only 1 MB of memory. It can create either one address space of 511 MB for one process (in coprocessor mode) or two address spaces of 255 MB each for two processes (in virtual node mode). (1023 and 511 MB respectively, with the 1 GB memory option.) The address spaces are flat and fixed, with no paging. The entire mapping is designed to fit statically in the TLBs of the PowerPC 440 processors.

The loading of code and data occurs in push mode. The I/O node of a pset reads the executable from the file system and forwards it to all compute nodes in the pset. The CNK in a compute node receives that executable and stores the appropriate memory values in the address space(s) of the compute process(es).

Once the CNK transfers control to the user application, its primary mission is to "stay out of the way". Since there is only one thread of execution per processor, there is no scheduling for the kernel to perform. Also, the memory space of a process is completely covered by the TLB in the processor running that process, so there is no memory management to perform. In normal execution, processor control stays with the compute process until it requests an operating system service through a system call. Exceptions to this normal execution are caused by hardware interrupts: either timer alarms requested by the user code or an abnormal hardware event that requires attention by the compute node kernel.

When a compute process makes a system call, three things may happen:

1. "Simple" system calls that require little operating system functionality, such as getting the time or setting an alarm, are handled locally by the compute node kernel. Control is transferred back to the compute process at completion of the call.
2. "I/O" system calls that require infrastructure for file systems and IP stack are shipped for execution in the I/O node associated with that compute node. (That is, the I/O node in the pset of the compute node.) The compute node kernel waits for a reply from the I/O node, and then returns control back to the compute process.
3. "Unsupported" system calls that require infrastructure not present in Blue Gene/L, such as *fork* and *mmap*, are returned right away with an error condition.

In Blue Gene/L we have implemented 68 system calls from Linux and an additional 18 CNK-specific calls [15]. The other Linux system calls are unsupported. In our experience, very seldom does a scientific application need one of the unsupported system calls. When it does, we adopt a combination of two solutions: either (1) change the application or (2) add the required system call.

There are two main benefits from the simple approach for a compute node operating system: robustness and scalability. Robustness comes from the fact that the compute node kernel performs few services, which greatly simplifies its design, implementation, and test. Scalability comes from lack of interference with running compute processes. Previous work by other teams [20] has identified system interference as a major source of performance degradation in large parallel systems. The effectiveness of our approach in delivering a system essentially free of interference has been verified directly through measurements of system noise [6],[17],[27] and indirectly through measurements of scalability all the way to 131,072 tasks in real applications [13],[28],[34].

4.3. The role of the I/O node

The I/O node plays a dual role in Blue Gene/L. On one hand, it acts as an effective master of its corresponding pset. On the other hand, it services requests from compute nodes in that pset. Jobs are launched in a partition by contacting corresponding I/O nodes. Each I/O node is then responsible for loading and starting the execution of the processes in each of the compute nodes of its pset. Once the compute processes start running, the I/O nodes wait for requests from those processes. Those requests are mainly I/O operations to be performed against the file systems mounted in the I/O node.

Blue Gene/L I/O nodes execute an embedded version of the Linux operating system. We call it embedded because it does not use any swap space, it has an in-memory root file system, it uses little memory, and it lacks the majority of daemons and services found in a server-grade configuration of Linux. It is, however, a complete port of the Linux kernel and those services can be, and in various cases have been, turned on for specific purposes. The Linux in Blue Gene/L I/O nodes includes a full TCP/IP stack, supporting communications to the outside world through Ethernet. It also includes file system support. Various network file systems have been ported to the Blue Gene/L I/O node, including GPFS, Lustre, NFS, and PVFS2 [21].

Blue Gene/L I/O nodes never run application processes. That duty is reserved to the compute nodes. The main user-level process running on the Blue Gene/L I/O node is the control and I/O daemon (CIOD). CIOD is the process that links the compute processes of an application running on compute nodes to the outside world. To launch a user job in a partition, the service node contacts the CIOD of each I/O node of the partition and passes the parameters of the job (user ID, group ID, supplementary groups, executable name, starting working directory, command line arguments, and environment variables). CIOD swaps itself to the user's identity, which includes the user ID, group ID, and supplementary groups. It then retrieves the executable from the file system and sends the code and initialized data through the collective network to each of the compute nodes in the pset. It also sends the command-line arguments and environment variables, together with a start signal.

Figure 4 illustrates how I/O system calls are handled in Blue Gene/L. When a compute process performs a system call requiring I/O (e.g., *open*, *close*, *read*, *write*), that call is trapped by the compute node kernel, which packages the parameters of the system call and sends that message to the CIOD in its corresponding I/O node. CIOD unpacks the message and then reissues the system call, this time under the Linux operating system of the I/O node. Once te system call completes, CIOD packages the result and sends it back to the originating compute node kernel, which, in turn, returns the result to the compute process. This simple model works well for transactional operations, such as read and write, which have a clear scope of operation and represent the bulk of I/O in scientific computing. It does not support operations such as memory mapped files, but those are uncommon in scientific computing.

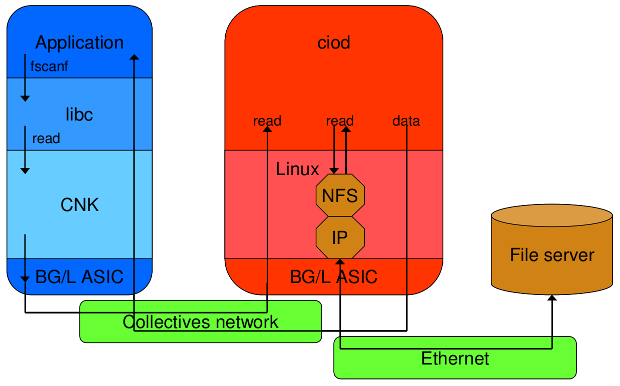

Figure 4: Function Shipping from CNK to CIOD.

There is a synergistic effect between simplification and separation of responsibilities. By offloading complex system operations to the I/O node we keep the compute node operating system simple. Correspondingly, by keeping application processes separate from I/O node activity we avoid many security and safety issues regarding execution in the I/O nodes. In particular, there is never a need for the common scrubbing daemons typically used in Linux clusters to clean up after misbehaving jobs. Just as keeping system services in the I/O nodes prevents interference with compute processes, keeping those processes in compute nodes prevents interference with system services in the I/O node. This isolation is particularly helpful during performance debugging work. The overall simplification of the operating system has enabled the scalability, reproducibility (performance results for Blue Gene/L applications are very close across runs [17]), and high-performance of important Blue Gene/L applications.

4.4. The role of the service node

The Blue Gene/L service node runs its control software, typically referred to as the Blue Gene/L control system. The control system is responsible for operation and monitoring of all compute and I/O nodes. It is also responsible for other hardware components such as link chips, power supplies, and fans. But this functionality is outside the scope of this paper. Tight integration between the Blue Gene/L control system and the I/O and compute nodes operating systems is central to the Blue Gene/L software stack. It represents one more step in the specialization of services that characterize that stack.

In Blue Gene/L, the control system is responsible for setting up system partitions and loading initial code and state in the nodes of a partition. The Blue Gene/L compute and I/O nodes are completely stateless: no hard drives and no persistent memory. When a partition is created, the control system programs the hardware to isolate that partition from others in the system. It computes the network routing for the torus, collective and global interrupt networks, thus simplifying the compute node kernel. It loads operating system code for all compute and I/O nodes of a partition through the dedicated control network. It also loads an initial state in each node (called the personality of the node). The personality of a node contains information specific to the node. Key components of the personality of I/O and compute nodes are shown in Table 1.

Table 1: Personalities of Compute and I/O Nodes.

Compute node personality	I/O node personality
• Memory size • Bit-steering for memory configuration • Physical location • x, y, z torus coordinates • Pset number and size • Routes for the collectives network	• Memory size • Bit-steering for memory configuration • Physical location • Pset number and size • Routes for the collectives network • MAC and IP address • Broadcast and gateway IP addresses • Service node IP address • NFS server address and export directory • Security key

5. Team Organization

The Sun never sets on the Blue Gene team. Cliché, but true. The project was started by a handful of people at the IBM T.J. Watson Research Center in Yorktown, with support from Lawrence Livermore National Laboratory.

Over the years, it grew to a team of over 100 people during its peak. The major development places were Yorktown, NY and Rochester, MN. But there were people working on Blue Gene/L in Raleigh (NC), Haifa (Israel), Delhi and Bangalore (India).

The approximately 12-hour difference between the western (Livermore) and eastern (Delhi) extremes of the Blue Gene/L development world offered both opportunities and challenges.

From an opportunity perspective, work never stopped. When it was middle of the night in the US, the teams in Israel and Delhi could continue working, sharing the available development prototypes. They would communicate their findings to the US teams early in the morning and those teams could pick up from there.

From a challenge perspective, coordination was certainly not trivial. The distance between the sites meant that face-to-face meetings were unusual (particularly with the non-US sites). It also created difficulties when people in two different sites, many hours apart (e.g., Livermore and Haifa), had to work side-by-side to solve a particular problem or address a crisis.

As we mentioned, coordination with the non-US sites, because of the distance and time difference, required particular care. During the initial development phase, we assigned specific, fairly independent pieces of the project to those sites. For example, Haifa assumed responsibility for developing a job control (including scheduling) solution for Blue Gene/L, while Delhi took responsibility of application checkpoint services. That allowed those sites to operate autonomously, typically with a weekly coordination meeting. As the project evolved and we moved into system integration and testing, and deployment to actual customers, the level of interaction had to increase. In the end, Blue Gene/L was a successful model of inter-site collaboration, and all the teams involved share the pride of participating in this project.

6. Building and Testing Blue Gene/L

Building a 65,536-node compute system does not happen overnight. Getting to a working 64-rack system at LLNL was an exercise in doubling.

The first midplane (half-rack) of Blue Gene/L became operational in September of 2003 in Yorktown, NY. That system was ranked in the 73rd position in the TOP500 list of November 2003. This was a 500 MHz prototype system.

The first multi-rack Blue Gene/L was a two-rack system assembled in Rochester, MN in March 2003. That system was later expanded to four racks in May 2004. The four-rack system was ranked in the 4th position in the June 2004 TOP500 list. This system also used the first iteration 500 MHz compute ASIC.

We then started an effort to build a 16-rack system in Rochester, MN using the second and final iteration of the compute ASIC, running at 700 MHz. This system was built to serve as a testbed for the integrated system. With four rows of four racks each, it was big enough for testing most of the software scaling issues, as well as network connections and file system operations. The first 8 racks became operational in August of 2004 and it was the first system to surpass the Earth Simulator in the Linpack benchmark. The complete 16 racks were assembled in September of 2004, and this system was ranked 1st in the November 2004 TOP500 list. From that point on and to this day, Blue Gene/L has had the top spot in that list.

A 16-rack system, organized as two rows of eight racks each, was installed in LLNL and passed acceptance test in December 2004, just in time to celebrate the Holidays. The particular configuration was chosen so that it could be expanded later, towards its final configuration.

In the beginning of 2005, we started parallel efforts to build two 32-rack systems, one at LLNL (expanding from the 16 already there) and one at the IBM facility in Rochester, MN. Both of these systems were configured as four rows of eight racks each. They were configured identically so that problems identified in one system (LLNL) could be debugged in the other (Rochester). Also, scalability testing of software continued in the Rochester system while application work proceeded in the LLNL system. The LLNL system became operational in March of 2005, while the Rochester system was completed one month later, in April of 2005. The expanded LLNL system was ranked 1st in the June 2005 TOP500 list.

In August of 2005, the 32 racks in Rochester were shipped to LLNL and integrated with the racks already there into a single 64-rack system, in its final configuration of eight rows of eight racks each. The system was operational after only two weeks. Some problems surfaced when the system reached its final size, which required software fixes and workarounds. Nevertheless, the system completed all its acceptance tests at LLNL by the end of September 2005. By this time it was already being used by many scientists and engineers at LLNL to accomplish breakthrough results. The final system again reached the 1st spot in the TOP500 list of November 2005. The 64 rack Blue Gene/L system installed at the Department of LLNL is shown in Figure 5.

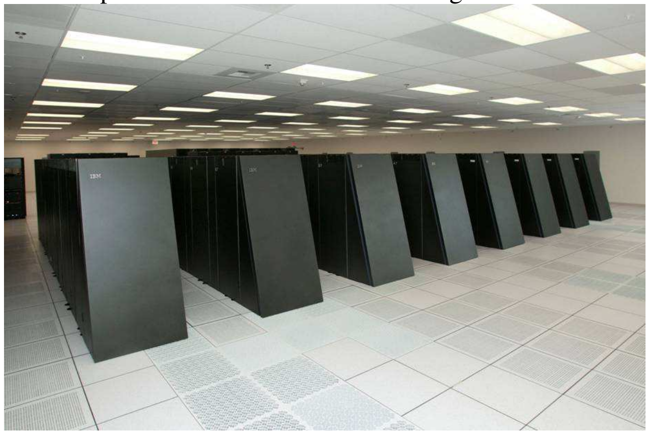

Figure 5. The Blue Gene/L system at LLNL.

7. The Impact of Blue Gene/L

Blue Gene/L was designed to be a breakthrough science machine. That is, to deliver a level of performance for scientific computing that enables entire new studies and new applications.

There are approximately 20 Blue Gene/L installations around the world. Even though most of them are small compared with the LLNL system (1 or 2 racks), they are mostly being used in support of new science. In this section we discuss just some of the important applications of Blue Gene/L in scientific research.

One of the main areas of applications of the LLNL system is in materials science. Scientists at LLNL use a variety of models, including quantum molecular dynamics [13], classical molecular dynamics [28], and dislocation dynamics [4], to study materials at different levels of resolution. Typically, each model is applicable to a range of system sizes being studied. First principle models can be used for small systems, while more phenomenological models have to be used for large systems. Blue Gene/L is the first system that is allowing scientist at LLNL to cross those boundaries. They can actually use first principle models in systems large enough to validate the phenomenological models, which in turn can be used in even larger systems.

Applications of notable significance at LLNL include ddcMD [28], a classical molecular dynamics code that has been used to simulate systems with approximately half a billion atoms (and in the process win the 2005 Gordon Bell award), and QBox [12],[13], a quantum molecular dynamics code that is, at the time of this writing, the highest performing application on Blue Gene/L.

Other success stories for Blue Gene/L include: (1) Astrophysical simulations in Argonne National Laboratory (ANL) using the FLASH code [2],[8]; (2) Global climate simulations in the National Center for Atmospheric Research (NCAR) using the HOMME code; (3) Biomolecular simulations at the T.J. Watson Research Center using the Blue Matter code [7],[10]; (4) Quantum chromo dynamics (QCD) at IBM T.J. Watson Research Center and LLNL, San Diego Supercomputing Center, Juelich Research Center, Massachusetts Institute of Technology, Boston University, University of Edinburgh, and KEK (Japan) using a variety of codes. The lattice QCD code by the Watson and LLNL team shows perfect scaling up to 65,536 nodes with 131,072 processors delivering 70.9 Teraflops. This is a previously unattained performance level and arguably poises lattice QCD and Blue Gene to produce the next generation of strong interaction physics theoretical results. This result [31] won the 2006 Gordon Bell special achievement award and is an indicator of simulation-based science applications enabled by the Blue Gene system. One of the most innovative uses of Blue Gene/L is as the central processor for the new large scale LOFAR radio telescope in the Netherlands [25].

Several results about Blue Gene/L performance are published in the literature [1],[5],[13],[14],[18],[28],[31],[34]. We here focus on just a few examples to illustrate the system performance in benchmarks and applications. First, we compare the performance of a single Blue Gene/L node executing the serial version of the NAS parallel benchmarks (class A) on top of two different operating systems: the CNK normally used on the Blue Gene/L compute nodes, and Linux normally used on the Blue Gene/L I/O nodes. Those results are shown in Table 2. It is clear that the performance benefit of CNK can be substantial, with the execution time increasing up to 300% for the IS benchmark. The reason for performance difference is that Linux has to handle TLB misses during execution, while CNK does not, as it maps the entire node memory in the TLB of the processor. The impact is worse for code that touches memory randomly, like IS.

Table 2: Single-node Performance for NAS Benchmarks on CNK and Linux.

	CNK		Linux			
	Time(s)	Speed (Mops)	Time(s)	Speed (Mops)	% Time	% Speed
is.A	5.7	14.7	23.0	3.6	303.0	-75.5
ep.A	172.6	3.1	177.2	3.0	2.7	-3.2
lu.A	380.3	313.7	504.0	237.0	32.5	-24.5
sp.A	375.0	227.0	517.4	164.3	38.0	-27.6
cg.A	27.1	55.0	32.3	46.3	19.2	-15.8
bt.A	522.6	322.0	613.0	274.4	17.3	-14.8

Illustrating the issue of scalability, Figure 6 shows the performance behavior of Flash (an astrophysics code from the University of Chicago) [8] on various systems. This is a weak scaling experiment, so ideal behavior would be for the execution time to stay constant as the number of processors increase. We observe that that is true only for the Blue Gene/L machines. All other machines eventually reach a point where the execution time grows significantly worse with the system size. Figure is a speedup curve for Miranda (a hydrodynamics code from LLNL) on Blue Gene/L [5]. This is a strong scaling experiment, and we observe an almost linear improvement in performance from 8192 to 65536 processors. Contributing to the scalability of Flash and Miranda on Blue Gene/L are the low-overhead user-mode MPI (enabled by the kernel) and the non-interference of system services on applications.

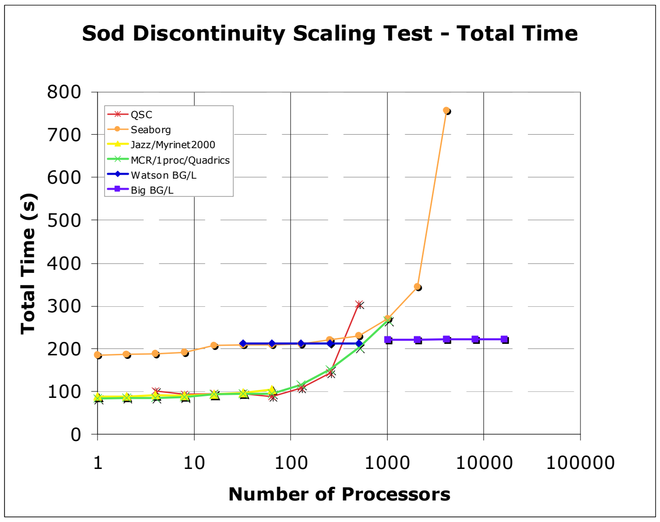

Figure 6: Performance of Flash in Different Parallel Systems (Weak Scaling).

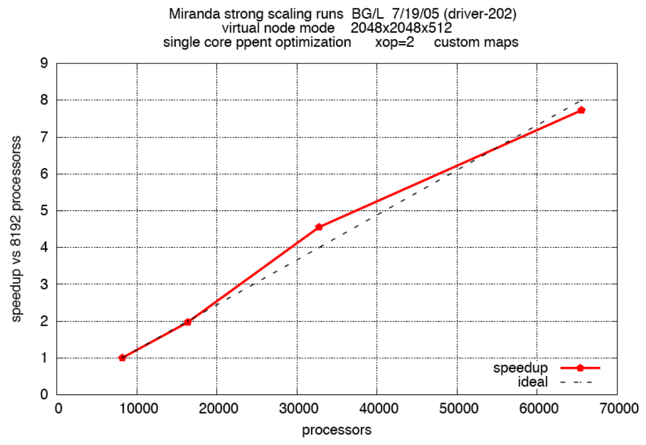

Figure 7: Strong Scaling of Miranda on Blue Gene/L (used with permission from the authors of [5]**).**

Finally, Figure 8 shows the performance of Linpack, the benchmark for the TOP500 list, on Blue Gene/L for different numbers of nodes. We observe an essentially linear scaling of performance (Tflops) with the number of nodes, all the way to the full machine size (65,536 compute nodes). Furthermore, the 280 Tflops mark represents better than 75% of the peak machine performance.

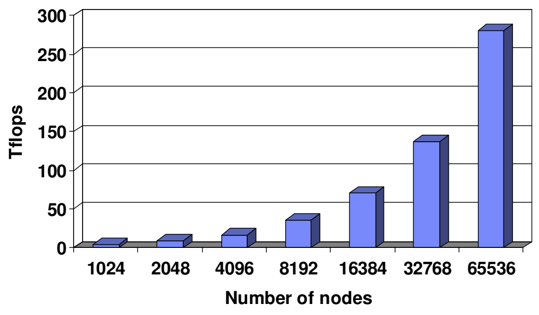

Figure 8: Performance of Linpack, the benchmark for TOP500, on Blue Gene/L.

8. Conclusions and a Look into the Future

Blue Gene/L was the first system in the Blue Gene family. We certainly do not want it to be the last. The major contribution of Blue Gene/L was to prove that it can be done! It is possible to build a machine with 100,000+ processors, and this machine can be reliable and operate within reasonable requirements of power, cooling and floor space. Furthermore, it is possible to run real applications that extract the power of a machine of this size. By doing that, Blue Gene/L changed much of the perception on the limits of scalability for parallel processing.

We can imagine two independent, and complementary paths for the evolution of Blue Gene/L into the future. First, as VLSI technology improves, we can implement essentially the same system with increasingly powerful nodes. With time, those nodes can have more and faster processors, and more memory. Also, the interconnection networks can get faster. Note that it is important to keep node reliability, which can be a challenge with more complex and larger nodes. Second, we can expand the system in size. From an architectural and packaging perspective, we could build a Blue Gene/L system with over 2 million nodes! That is 32 times larger than the LLNL system. Of course, such a system would have a failure rate that is 32 times larger, use 32 times more floor space, be 32 times more expensive to buy and operate, and require 32 times more power and cooling. Therefore, its usefulness is questionable. But systems a couple of times larger than LLNL's might make sense.

More interesting is the question of what is the next breakthrough in large scale parallel processing? I guess if we knew, it would not be the next breakthrough anymore. Many frontiers are open, such as specialized and reconfigurable hardware, which can offer orders of magnitude improvement in performance. Also, there is a productivity frontier. Can we drastically simplify the job of using the massive computing power offered by Blue Gene/L and its successors, maybe at the expense of some loss of efficiency?

One of the most rewarding aspects of the Blue Gene project is to see scientists and engineers getting work done that they could not do before this machine existed. We can only hope that projects like Blue Gene will continue to improve the tools those scientists and engineers have at their disposal and open new horizons for humankind.

Acknowledgments

The BlueGene/L project has been supported and partially funded by the Lawrence Livermore National Laboratories on behalf of the United States Department of Energy, under Lawrence Livermore National Laboratories Subcontract No. B517552.

This work has benefited from the cooperation of many individuals in IBM Research (Yorktown Heights, NY), IBM Engineering & Technology Services (Rochester, MN), IBM Systems and Technology Group (Rochester, MN) and IBM Microelectronics (Burlington, VT and Raleigh, NC).

References

[1] G. Almasi, S. Chatterjee, A. Gara, J. Gunnels, M. Gupta, A. Henning, J.E. Moreira, B. Walkup. *Unlocking the performance of the BlueGene/L supercomputer*. IEEE/ACM SC04, Pittsburgh, PA, November 2004.

[2] ASC/Alliances Center for Astrophysical Thermonuclear Flashes, University of Chicago; see http://flash.uchicago.edu/website/home/.

[3] A. Bright, M. Ellavsky, A. Gara, R. Haring, G. Kopcsay, R. Lembach, J. Marcella, M. Ohmacht, V. Salapura. Creating the BlueGene/L supercomputer from low power SoC ASICs. ISSCC 2005 – IEEE International Solid-State Circuits Conference, February 2005.

[4] V. Bulatov, W. Cai, J. Fier, M. Hiratani, G. Hommes, T. Pierce, M. Tang, M. Rhee, R.K. Yates, and T. Arsenlis. *Scalable line dynamics in ParaDiS*. IEEE/ACM SC04, Pittsburgh, PA, November 2004.

[5] A.W. Cook, W.H. Cabot, M.L. Welcome, P.L. Williams, B.J. Miller, B.R. de Supinski, R.K. Yates. *Tera-scalable algorithms for variable-density elliptic hydrodynamics with spectral accuracy*. IEEE/ACM SC05, Seattle, WA, November 2005.

[6] K. Davis, A. Hoisie, G. Johnson, D. J. Kerbyson, M. Lang, S. Pakin and F. Petrini. *A performance and scalability analysis of the BlueGene/L architecture*. IEEE/ACM SC04, Pittsburgh, PA, November 2004.

[7] Blake G. Fitch, Aleksandr Rayshubskiy, Maria Eleftheriou, T.J. Christopher Ward, Mark E. Giampapa, Michael C. Pitman, Robert S. Germain. Blue Matter: Approaching the Limits of Concurrency for Classical Molecular Dynamics. Supercomputing 2006, November 2006.

[8] B. Fryxell, K. Olson, P. Ricker, F. X. Timmes, M. Zingale, D. Q. Lamb, P. MacNeice, R. Rosner, and H. Tufo. *FLASH: An adaptive mesh hydrodynamics code for modeling astrophysical thermonuclear flashes*. Astrophysical Journal Supplement, 131:273, 2000.

[9] A. Gara, M. A. Blumrich, D. Chen, G. L.-T. Chiu, P. Coteus, M. E. Giampapa, R. A. Haring, P. Heidelberger, D. Hoenicke, G. V. Kopcsay, T. A. Liebsch, M. Ohmacht, B. D. Steinmacher-Burow, T. Takken, and P. Vranas. Overview of the Blue Gene/L system architecture. IBM Journal of Research and Development. Vol. 49, no. 2/3, March/May 2005, pp. 195–212.

[10] R. S. Germain, Y. Zhestkov, M. Eleftheriou, A. Rayshubskiy, F. Suits, T. J. C. Ward, and B. G. Fitch. *Early performance data on the Blue Matter molecular simulation framework*. IBM Journal of Research and Development. Vol. 49, no. 2/3, March/May 2005, pp. 447–456.

[11] D.S. Greenberg, R. Brightwell, L.A. Fisk, A.B. Maccabe, and R.E. Riesen. *A system software architecture for high-end computing*. IEEE/ACM SC97, San Jose, CA, November 1997, pp. 1–15;

[12] Francois Gygi, Erik W. Draeger, Martin Schulz, Bronis R. De Supinski, John A. Gunnels, Vernon Austel, James C. Sexton, Franz Franchetti, Stefan Kral, Christoph Ueberhuber, Juergen Lorenz. Large-Scale Electronic Structure Calculations of High-Z Metals on the BlueGene/L Platform. *2006 Gordon Bell Prize.* Supercomputing 2006, November 2006.

[13] F. Gygi, E.W. Draeger, B.R. de Supinski, R.K. Yates, F. Franchetti, S. Kral, J. Lorenz, C.W. Ueberhuber, J.A. Gunnels, J.C. Sexton. Large-scale first-principles molecular dynamics simulations on the BlueGene/L platform using the Qbox code. IEEE/ACM SC05, Seattle, WA, November 2005.

[14] A. Henning. BlueGene/L: Improving application memory performance on a massively parallel machine. M.E. Thesis. Cornell University. 2005.

[15] IBM Blue Gene Team. Blue Gene: A vision for protein science using a petaflop supercomputer. IBM Systems Journal, 40(2), 2001.

[16] IBM Corporation. Blue Gene/L: Application development. 2006. http://www.redbooks.ibm.com/abstracts/sg247179.html?Open

[17] E. Ipek, B.R. de Supinski, M. Schulz, and S.A. McKee. *An approach to performance prediction for parallel applications.* 2005 Euro-Par, Lisbon, Portugal, August 2005.

[18] S. Louis, B.R. de Supinski. *BlueGene/L: Early application scaling results*. BlueGene System Software Workshop February 23-24, 2005, Salt Lake City, Utah. http://www-unix.mcs.anl.gov/~beckman/bluegene/SSW-Utah-2005/BGL-SSW22-LLNL-Apps.pdf

[19] 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, and G. G.

Stewart. *Blue Gene/L programming and operating environment*. IBM Journal of Research and Development. Vol. 49, no. 2/3, March/May 2005.

[20] F. Petrini, D. Kerbyson and S. Pakin. *The case of the missing supercomputer performance: achieving optimal performance on the 8,192 processors of ASCI Q*. IEEE/ACM SC03, Phoenix, AZ, November 2003.

[21] R. Ross, J.E. Moreira, K. Cupps, W. Pfeiffer. *Parallel I/O on the IBM Blue Gene/L system*. Blue Gene/L Consortium Quarterly Newsletter. Argonne National Laboratory. 1st quarter 2006. http://www-fp.mcs.anl.gov/bgconsortium/ file%20system%20newsletter2.pdf.

[22] V. Salapura, R. Bickford, M. Blumrich, A. Bright, D. Chen, P. Coteus, A. Gara, M. Giampapa, M. Gschwind, M. Gupta, S. Hall, R.A. Haring, P. Heidelberger, D. Hoenicke, G.V. Kopcsay, M. Ohmacht, R.A. Rand, T.Takken, and P.Vranas. Power and Performance Optimization at the System Level. ACM Computing Frontiers 2005, Ischia, Italy, May 2005.

[23] V. Salapura, R. Walkup, A. Gara. Exploiting Workload Parallelism for Performance and Power Optimization in Blue Gene. IEEE Micro, September/October 2006.

[24] V. Salapura, J. R. Brunheroto, F. Redìgolo, D. Hoenicke, A. Gara. Exploiting eDRAM bandwidth with data prefetching: simulation and measurements. HPCA-13 – IEEE International Symposium on High-Performance Computer Architecture, February 2007.

[25] K. van der Schaaf. *Blue Gene in the heart of a wide area sensor network*. QCDOC and Blue Gene: Next Generation of HPC Architecture Workshop. Edinburgh, UK, October 2005.

[26] M. Seager. *The BlueGene/L computing environment*. Lawrence Livermore National Laboratory. October 2003. http://www.llnl.gov/asci/platforms/bluegene/ papers/16seager.pdf.

[27] K. Singh, E. Ipek, S.A. McKee, B.R. de Supinski, M. Schulz, and R. Caruana. *Predicting parallel application performance via machine learning approaches*. Concurrency and Computation: Practice and Experience. 2006. *To appear*.

[28] F.H. Streitz, J.N. Glosli, M.V. Patel, B. Chan, R.K. Yates, B.R. de Supinski, J. Sexton, J.A. Gunnels. *100+ TFlop solidification simulations on BlueGene/L*. Gordon Bell Prize at IEEE/ACM SC05, Seattle, WA, November 2005.

[29] University of Mannheim, University of Tennessee, and NERSC/LBNL. TOP500 Supercomputer sites. http://www.top500.org/.

[30] University of Tennessee. HPC Challenge Benchmark. http://icl.cs.utk.edu/hpcc/.

[31] P. Vranas, G. Bhanot, M. Blumrich, D. Chen, A. Gara, P. Heidelberger, V. Salapura, J. Sexton. The BlueGene/L Supercomputer and Quantum ChromoDynamics. *2006 Gordon Bell Prize*. Supercomputing 2006, November 2006.

[32] 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.

[33] S.R. Wheat, A.B. Maccabe, R. Riesen, D.W. van Dresser, and T.M. Stallcup. *PUMA: An Operating System for Massively Parallel Systems*. Scientific Programming, vol 3, 1994, pp. 275-288.

[34] *A. Yoo, E. Chow, K. Henderson, W. McLendon, B. Hendrickson, U. Catalyurek. A scalable distributed parallel breadth-first search algorithm on BlueGene/L. IEEE/ACM SC05, Seattle, WA, November 2005.*