Overview of the IBM Blue Gene/P project

IBM Blue Gene team

On June 26, 2007, IBM announced the Blue Gene/P™ system as the leading offering in its massively parallel Blue Gene® supercomputer line, succeeding the Blue Gene/L™ system. The Blue Gene/P system is designed to scale to at least 262,144 quad-processor nodes, with a peak performance of 3.56 petaflops. More significantly, the Blue Gene/P system enables this unprecedented scaling via architectural and design choices that maximize performance per watt, performance per square foot, and mean time between failures. This paper describes our vision of this petascale system, that is, a system capable of delivering more than a quadrillion (10^{15}) floating-point operations per second. We also provide an overview of the system architecture, packaging, system software, and initial benchmark results.

Introduction

The Blue Gene/L* (BG/L) supercomputer was introduced in November 2004, and its architecture, packaging, and system software were described previously [1]. While originally intended to be used for applications related to life sciences, such as large-scale molecular dynamics computations of protein folding, the repertoire of applications running on the BG/L systems has increased dramatically [2], with users reaping the benefits of its scalability, superior cost/performance, low power/gigaflop, high reliability, and high sustained flops (floating-point operations per second) when compared to competitive supercomputer systems. Applications on the BG/L system have won four Gordon Bell Prizes [3–6]. The largest BG/L installation had been in the number one position of the TOP500** list [7] for an unprecedented seven consecutive times since November 2004, and benchmark applications running on this system have won most of the High Performance Computing Challenge awards [8] for 2005–2007.

The BG/L supercomputer was designed for high-power efficiency, with a ratio of dissipated power to flops (W/flops) that is much lower than that of competitive supercomputer systems. This power efficiency made the BG/L system the "greenest" (i.e., most energy efficient) supercomputer [9], until the introduction of the IBM Blue Gene/P* (BG/P) system, and it has enabled computer facilities with limited power budgets to accommodate systems far more powerful than previously imagined, enabling substantial increases in the precision and complexity of the applications being run.

However, the simulation demands for broad classes of scientific phenomena continue to increase, and with it, the demand for ever more powerful machines also increases. In order to meet these demands, on June 26, 2007, IBM announced the BG/P system as the successor to the BG/L system. The largest installed BG/P system so far (a 16-rack installation in Juelich, Germany) achieved the number two spot of the November 2007 TOP500 list [7] and took over the top position in the Green500 list [9] as the most energy-efficient supercomputer. In this overview paper, we emphasize the major enhancements to the BG/P system over its predecessor. While our aim with the BG/P system is to push high-performance computing (HPC) to an unprecedented petascale, it maintains backward compatibility to BG/L application software. **Table 1** highlights the progression from the BG/L system to the BG/P system.

At a system architecture level, the BG/P system maintains many of the design choices of the BG/L system. A single ASIC (application-specific integrated circuit), the BG/P compute (BPC) chip, is used as the processor chip for both compute nodes and I/O nodes. Compute nodes are connected in a three-dimensional (3D) torus topology. Each compute node is also connected to a collective network, rooted at the I/O nodes. Thus, the network

0018-8646/08/$5.00 © 2008 IBM

Table 1 Comparison of the BG/L and BG/P systems.

	Property	*Blue Gene/L*	*Blue Gene/P*
Node properties	Node processors	Two PowerPC* 440	Four PowerPC 450
	Processor frequency	0.7 GHz	0.85 GHz
	Coherency	Software managed	SMP
	L1 cache (private)	32-KB I-cache + 32-KB D-cache per processor	32-KB I-cache + 32-KB D-cache per processor
	L2 cache (private)	Seven-stream prefetching	Seven-stream prefetching
		Two-line buffers/stream	Two-line buffers/stream
	L3 cache size (shared)	4MB	8MB
	Main store	512 MB and 1 GB	2 GB and 4 GB
	Main store bandwidth	5.6 GB/s (16 bytes wide)	13.6 GB/s (2 × 16 bytes wide)
	Peak performance	5.6 Gflops/node	13.6 Gflops/node
Torus network	Bandwidth	Core injects/receives packets 6 × 2 × 175 MB/s = 2.1 GB/s	DMA injects/receives packets 6 × 2 × 425 MB/s = 5.1 GB/s
	Hardware latency (nearest neighbor)	<1 μs	<1 μs
	Hardware latency (worst case, 72 racks)	7 μs (68 hops)	5 μs (68 hops)
Collective network	Bandwidth	3 × 2 × 350 MB/s = 2.1 GB/s	3 × 2 × 850 MB/s = 5.1 GB/s
	Hardware latency (round-trip worst case, 72 racks)	6.0 μs	5.0 μs
System properties (e.g., 72 racks)	Area	150 m^2	200 m^2
	Peak performance	410 Tflops	1 Pflops
	Total power (LINPACK)	1.9 MW	2.9 MW

topology for the BG/P system is the same as that for the BG/L system. However, the BG/P system incorporates significant enhancements to the hardware and software architectures. Most notably, BG/P nodes are now four-way symmetric multiprocessors (SMPs). The torus network logic has been enhanced with a direct memory access (DMA) engine, thereby offloading communication. Software provides flexibility for the ways in which the SMP nodes are used by applications in order to attain the most efficient parallelism. In the BG/P system, a significant portion of the software stack is open source, permitting a wide community of users to understand, leverage, and contribute to BG/P system software.

The BPC chip is based on IBM Cu-08 ASIC technology [10] (90-nm generation), which is the technology generation following the Cu-11 (130-nm) technology of the BG/L system. The denser Cu-08 technology allows the integration of roughly double the function on a similarly sized chip. As depicted in **Figure 1**, the BPC chip is a single ASIC with four IBM PowerPC 450 (PPC450)-embedded 32-bit processor cores, arranged as an SMP. A dual-pipeline floating-point unit (FPU) is attached to each PPC450 core. The design of this dual FPU is logically identical to the one used in the BG/L system. It supports two simultaneous double-precision floating-point calculations in SIMD (single-instruction, multiple-data) fashion, along with instruction set extensions for complex number arithmetic. The dual-pipeline FPUs can simultaneously execute two fused multiply–add instructions per machine cycle, each of which is counted as 2 FLOPs (floating-point operations). Thus, each processor unit (PPC450 and FPU) has a peak performance of 4 FLOPs per machine cycle, and the BPC chip with quadruple processor units rates at a peak performance of 16 FLOPs per cycle × 850 MHz, or 13.6 gigaflops (Gflops).

Figure 2 is a photograph of the BPC chip with its major regions indicated. Along with the above described quadruple processor units, the chip integrates a memory subsystem and a number of I/O subsystems to network the chips to one another and to external computers and

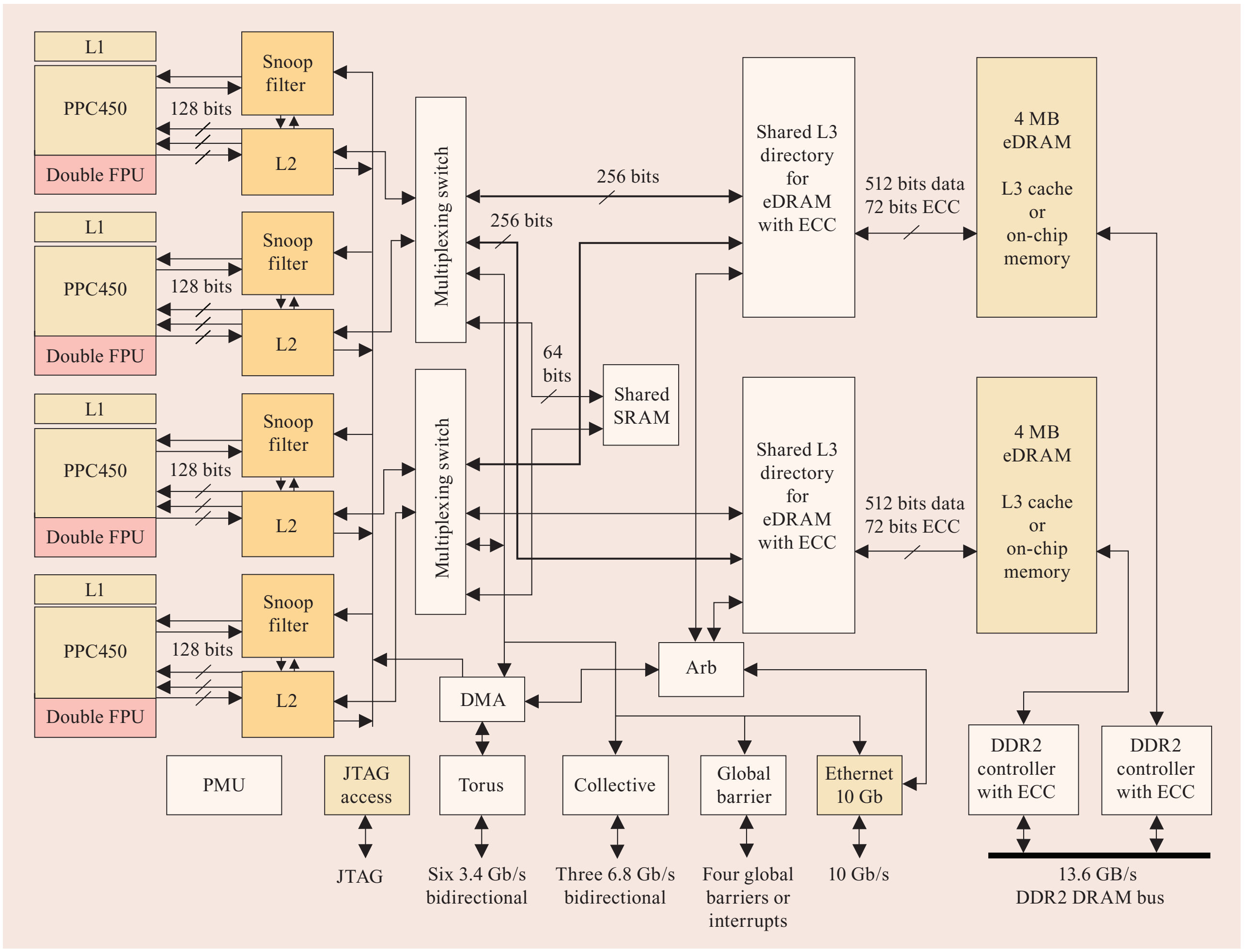


Figure 1

A BG/P compute (BPC) chip integrates four PowerPC 450 (PPC450) cores (each with a double FPU and an L1 cache consisting of a 32-KB instruction cache and a 32-KB data cache) with L2 and L3 cache, memory controllers, and various external network interfaces. (FPU: floating-point unit; torus: 3D torus network interface; DMA: direct memory access unit; collective: collective network interface; global barrier: global barrier interface; Arb: arbiter between DMA and 10-Gb Ethernet access to L3 cache; DDR: double-data rate; ECC: error checking and correction; PMU: performance monitor unit. *JTAG* stands for Joint Test Action Group and refers to an IEEE 1149.1 interface for control, monitoring, and debug.)

file systems. The memory subsystem and I/O networks are more fully described below.

Each BPC chip is packaged on a compute card, along with its associated memory chips. This combination of a BPC chip with its associated private memory represents a node. The BG/P system currently offers 2 GB of main memory per node, in the form of double-data-rate-2 (DDR2) SDRAM (synchronous dynamic RAM). As the DRAM market evolves, a future offering may increase this to 4 GB per node.

With 2 GB of main memory per node, the BG/P system has a ratio of 512 MB per processor core, if the memory is evenly divided among the four cores. The same ratio exists for the latest BG/L model, which has 1 GB per node, or 512 MB per core. Thus, with the transition from the BG/L to the BG/P system, the relative memory size per core has been maintained, easing software migration.

The performance improvement of the BG/P system over the BG/L system can mainly be attributed to two sources: 1) a twofold boost from dual cores to quad cores, and 2) a frequency enhancement from 700 MHz to 850 MHz. Other characteristics, such as memory subsystem performance and network performance, have been maintained or improved on a per-FLOP basis, so we expect the BG/P system to be 2.43 times faster on a per-node basis, when compared with the BG/L system. (This

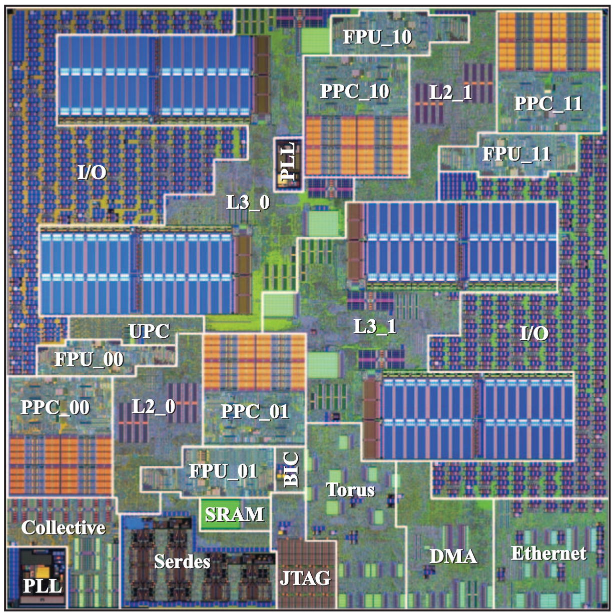


Figure 2

The Blue Gene/P compute (BPC) chip, showing the four PowerPC 450 processors (PPC_00, PPC_01, PPC_10, PPC_11) and their associated floating-point units (FPU_00, FPU_01, FPU_10, FPU_11), network interface controllers (torus with DMA, collective network, JTAG, and 10-Gb Ethernet), and memory hierarchy (L2 and L3). The four large arrays are 2-MB embedded DRAMs (eDRAMs), each used as L3 data caches. The BPC chip dimensions are 13.16 mm × 13.16 mm, and it contains 208M transistors, of which 88M are in the eDRAM arrays. (UPC: universal performance counter; PLL: phase-locked loop; serdes: serializer/deserializer; DMA: direct memory access.)

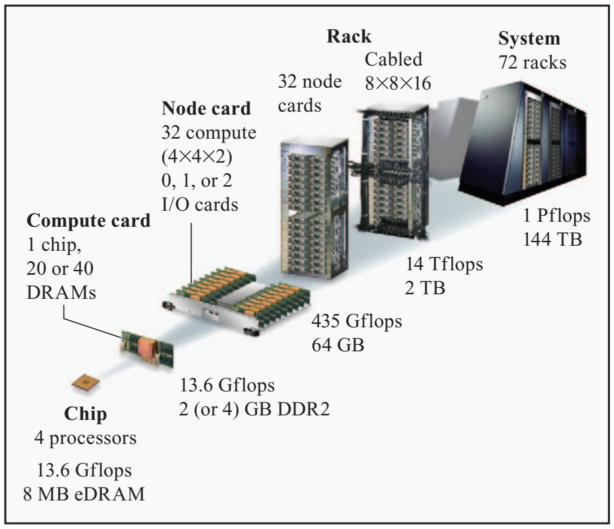


Figure 3

The packaging hierarchy of the BG/P system. (Gflops: gigaflops; Tflops: teraflops; Pflops: petaflops; eDRAM: embedded DRAM.)

comparison assumes that the BG/L and BG/P systems both use the same L1-cache write-through mode.) Another performance improvement, as mentioned, is due to the addition of a DMA engine to the torus network, which enables most of the network overhead to be offloaded from the cores. The DMA engine is described further in the section on DMA.

In addition to the compute nodes, the BG/P system contains a configurable number of I/O nodes. The I/O nodes are physically the same compute cards as described above, but their position in the system differentiates their logical function. I/O nodes have the 10-Gigabit Ethernet (GbE) interface enabled for communication with a file system and host computers. This is an upgrade from the 1 GbE used in the BG/L system. The peak unidirectional bandwidth of an I/O port is limited to 6.8 Gb/s by the internal collective network that feeds it. The number of I/O nodes is configurable, with a maximum I/O-to-compute-node ratio of 1:16, whereas the corresponding ratio for the BG/L system is 1:8. Combining these factors, the BG/P system has a net 3.4-fold increase in I/O bandwidth per physical unit (node card or rack, for a rack that is maximally configured with I/O cards) over the BG/L system, or 40% more I/O bandwidth per FLOP.

The packaging of the BG/P system is similar to that of the BG/L system (**Figure 3**). Thirty-two compute cards and, optionally, up to two I/O cards are packaged onto the next-level board, called the *node card*. Sixteen node cards are plugged from both sides into a vertical midplane card, completing an assembly of 512 compute nodes in an 8 × 8 × 8 configuration. The inbound and outbound network connections for this 512-way cube are routed to four *link cards* that carry a total of 24 Blue Gene/P link (BPL) chips. The assembly of 16 node cards, 4 link cards, and an additional service card is called a *midplane* or a *512-way*. The BPL chips are relatively simple switches that, depending on the size and configuration of a user partition of the system, route network signals back into the midplane (completing the wraparounds for an 8 × 8 × 8 torus) or route the network signals through cables to another midplane for larger partitions. Two midplanes, one on top of the other, complete a rack. Thus, a rack has 1,024 nodes, or 4,096 cores, giving a peak performance of 13.9 teraflops (Tflops). Scaling upward, a 72-rack system can package 72K nodes (288K cores) (where *K* stands for 1,024) into a 1-petaflops (Pflops) (peak) system, and larger configurations up to 256 racks (3.56 Pflops peak) are possible.

Blue Gene/P system integration

A critical component of any large supercomputer installation is a high-capacity and scalable parallel file

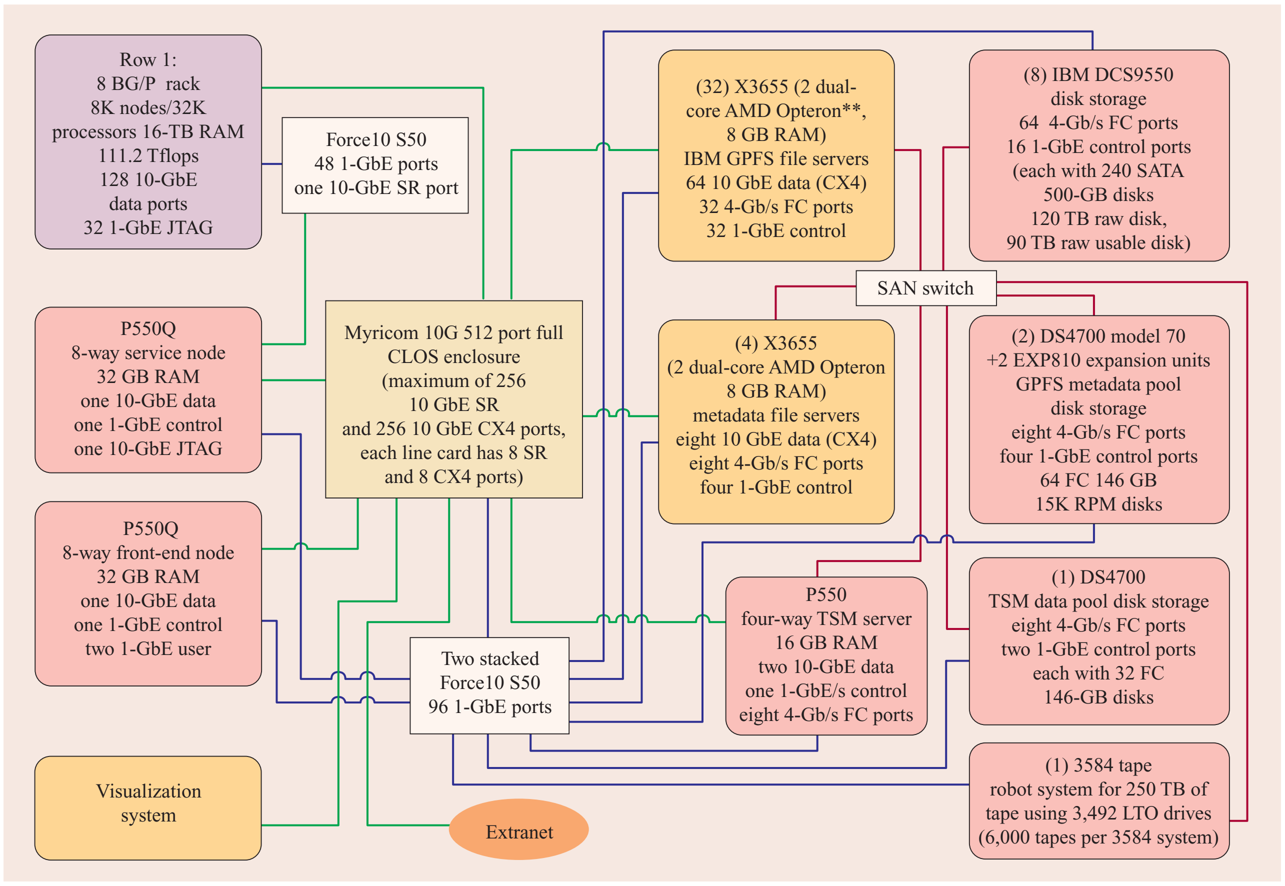


Figure 4

A typical architecture of a 111-Tflops, eight-rack Blue Gene/P system with a storage system supporting the IBM GPFS. Blue connections represent a 1 GbE (Gigabit Ethernet), green connections represent a 10 GbE, and red connections represent FiberChannel (FC). See text for further explanation. (SR: short-range optical fiber; CX4: copper connection; SATA: serial advanced technology attachment; LTO: linear tape-open.)

system. While customers are free to specify a file system of their choice to attach to a Blue Gene* system, IBM offers an integrated solution based on the General Parallel File System* (GPFS*). The GPFS implementation on the BG/P system remains largely the same as on the BG/L system. The GPFS runs on BG/P I/O nodes in much the same way as it does on conventional GPFS clusters. Because the I/O bandwidth of the BG/P system has significantly increased with respect to the BG/L system, the presence of a large number of I/O nodes in a BG/P configuration may pose significant challenges to a parallel file system from the perspective of sustained bandwidth delivery and network connectivity. The architecture of a host system that supports the BG/P system must take these requirements into consideration. In **Figure 4**, we show a proposed system configuration for a 111-Tflops eight-rack BG/P system, with 16 I/O nodes per rack, each operating at its unidirectional bandwidth limit of 850 MB/s. Apart from the eight BG/P racks, this example system proposal includes IBM System p5* 550Q servers used as front-end and service nodes, connected via Force10 S50 and Myricom 10G switches for the 1 GbE control networks and the 10 GbE user-data networks, respectively. IBM System x3655 GPFS file system servers and an IBM System p5 550 server, used as an IBM Tivoli* Storage Manager (TSM) backup server, drive the storage functions. The proposed GPFS storage system is based on IBM DCS9550 storage units, which under GPFS can each deliver approximately 2.1 GB/s of I/O throughput in a cost-effective manner, as well as IBM DS7400 storage units for GPFS metadata and for backup, and an IBM System Storage TS3500 Tape Library.

Cost/performance

One of the key design objectives for the Blue Gene family of supercomputers is to achieve a cost/performance ratio on par with the COTS (commodity-off-the-shelf)

Table 2 Power and space consideration for petascale systems based on various architectures (see Reference [7]).

	IBM BG/P	*IBM BG/L*	*IBM POWER5* 575*	*Cray XT3***	*Silicon Graphics Altix** 4700*
Nodes per rack	1,024	1,024	12	96	1
Processors per node	4	2	16	2	64
Clock speed (GHz)	0.85	0.7	1.5	2.4	1.6
FLOPs/clock	4	4	4	2	4
Peak performance per rack (Gflops)	13,926.4	5,734.4	1,152.0	921.6	409.6
Power required per rack (kW)	40.0	27.0	35	14.8	20.6
Gflops/kW	348.16	212.4	33	62.3	19.9
Racks required for 1 petaflops	72	174	868	1,085	2,441
Total power for 1 petaflops (MW)	2.9	4.7	28.6	16.2	50.2

approach. Three components dominate the cost of a modern supercomputer: memory, processor, and interprocessor networks. In order to minimize the cost, the BG/L and BG/P supercomputers use a balanced approach in which the memory cost and processor cost are roughly equal. Furthermore, we integrate the interprocessor networks directly onto the processor chip in order to avoid the expensive industry-standard links. This approach has attracted top-tier customers in HPC such as Lawrence Livermore National Laboratory (LLNL) and Argonne National Laboratory (ANL) to partner with IBM to develop future Blue Gene systems.

Low power

A hallmark of the Blue Gene family is the low-power design. We have previously demonstrated experimentally [1] and theoretically [11] that by using a clock frequency that is judiciously chosen to be relatively low, we can achieve an overall gain in rack-level performance and in power efficiency in terms of Gflops per watt.

In the current CMOS (complementary metal-oxide semiconductor) 90-nm generation, leakage power is as high as an average of 4 W per BPC chip, whereas leakage power was negligible in the previous generation. Nevertheless, the BG/P system has improved the ratio of Gflops per watt by about 50% compared with the BG/L system. A low-power design results in a higher integration level (more function per physical unit such as a chip, board, or rack), higher reliability, smaller footprint (i.e., floor space), energy savings, and an overall reduction in total cost of ownership over the lifetime of the system. **Table 2** provides a comparison of current supercomputers from the June 2007 TOP500 list. The last two rows highlight the space and power savings normalized to a petaflop. (Note that we scale these values to a petaflop for comparison purposes, regardless of whether a particular architecture can be scaled to a petaflop.)

Figure 5 compares the system power efficiency of the BG/P to the top systems on the November 2006 TOP500 list [7]. IBM systems are in yellow. Red Storm is a Cray XT3 system with AMD Opteron chips. Thunderbird is a Dell PowerEdge** cluster, using Intel Xeon** EM64T chips. Both Red Storm and Thunderbird are located at Sandia National Laboratories. ASC (Advanced Simulation and Computing) Purple is at the LLNL and is based on IBM chips. Consistent with this data, the BG/P and BG/L machines now occupy the top positions of the Green500 list [9].

Memory system

Each PPC450 core contains a separate L1 instruction cache (I-cache) and L1 data cache (D-cache). Both caches are 32 KB in size, have a line size of 32 bytes, and are 64-way set-associative with 16 sets. The PPC450 I-cache is *virtually tagged* (which means that the tag for any cache line is based on the upper bits of the virtual address), whereas the PPC450 D-cache is *physically tagged*, which means that the tag for a cache line is based on the upper bits of the real or physical address. For both caches, lines are replaced in round-robin order. I-cache line fills are performed through a dedicated 128-bit bus for each respective PPC450 core. The D-cache has two separate 128-bit data buses to and from L2, one for reading and one for writing. A line transfer between the L1 and L2 caches requires two consecutive 16-byte transfers and optimally returns the critical word first. The peak fill rate is 6.8 GB/s. The PPC450 memory system allows pages of memory to be mapped as write-through, cacheable, or cache inhibited. The D-cache supports a snooping cache coherence protocol, more specifically a write-invalidate cache coherence protocol.

The architecture of L2 remains almost the same as that of the BG/L system. Each L2 cache includes a read-only line store (L2R) and a write buffer (L2W). The L2R is small and essentially serves as a prefetch buffer for line fills. On L1-cache misses, it not only delivers the requested data to the L1, but also requests the data for the subsequent line from the L3 cache and stores this prefetched data for potential future L1-cache misses. The prefetch improves latency and sustainable bandwidth for regular access patterns such as streams or array accesses. The L2R is fully associative and includes 15 lines, each of which is 128 bytes. An L2R hit is expected to have a latency of 12 cycles and to match the peak fill rate of the L1 cache. The L2 prefetch is complemented by a prefetch unit inside the L3, which prefetches data from main memory into L3, anticipating future requests from the L2 cache.

The write-through policy of the L1 D-cache required that the coherence management of the prefetch buffer be improved with respect to the BG/L system. The granularity of prefetch buffer invalidation was changed from 128 bytes to 32 bytes, allowing the prefetcher to function efficiently in the presence of writes that modify the streamed-in data. The presence of the L2 write buffer allows the core to complete write-through stores very quickly and allows for the larger L2 and L3 line size to be aggregated.

The four L2 caches are connected via multiplexing switches to the L3 cache (see Figure 1). The L3 cache is constructed with embedded DRAM (eDRAM) and is 8 MB, organized as two banks of 4 MB each. It is shared by instructions and data. The L3 provides much higher bandwidth, lower power, and lower latency access to the working set of applications than off-chip memory. For many workloads, the bandwidth and latency restrictions of the off-chip main memory on performance are, therefore, significantly mitigated.

Since the L1 D-caches are designed to be operated in write-through mode on the BG/P system, the L3 cache has been improved for write throughput with respect to the BG/L system. The total number of on-chip L3 write-combining buffer entries has been increased from four 128-byte lines in the BG/L system to sixty in the BG/P system. In addition, the L3 directory has been optimized for write throughput, allowing a total of four write requests to complete every 425-MHz cycle.

The BPC chip also integrates dual DDR2 memory controllers, one associated with each L3 bank, and each with a 128-bit-wide data interface, running at half the processor frequency. This results in an off-chip memory bandwidth to floating-point performance ratio of 1 byte/FLOP. For a processor clock frequency of 850 MHz, the total memory bandwidth is 13.6 GB/s. The main memory is external to the BPC chip and is built from commodity DDR2 SDRAM devices. A flexible memory controller design and ECC (error checking and correction) algorithm allows the BG/P system to progress between 512-Mb, 1-Gb, and 2-Gb DRAM chip technologies as the DRAM industry evolves. For a 2-GB/node design, each BPC ASIC can address forty 512 Mb x8 DRAM chips (where *x8*—i.e., "by eight"—denotes the data bus width of an individual DRAM chip) or twenty 1 Gb x16 chips. For a 4-GB/node design, forty 1 Gb x8 chips or twenty 2 Gb x16 chips can be used. These configurations of DRAM chips provide strong ECC protection and can tolerate failed DRAM chips, as described below.

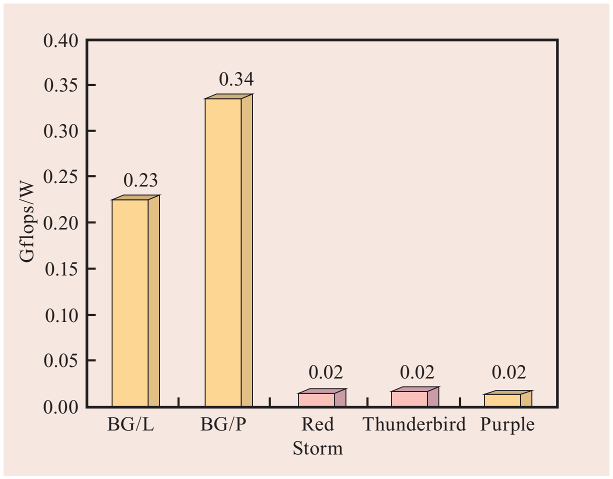


Figure 5

System power efficiencies of recent supercomputers on the TOP500 list.

Finally, the node memory subsystem additionally includes a shared SRAM. The shared SRAM is used for loading boot code and for out-of-band communication between the node kernels and host service processor. In addition, the shared SRAM provides the backing-store for a flexible number of low-latency 32-bit atomic counters. This builds on the functionality of the BG/L lockbox device that provided a fixed number of atomic single locks and two-core barrier operations. Using an active address decoder, the BG/P lockbox implements atomic counting operations (including fetch, fetch with increment, fetch with decrement, and fetch with clear), providing traditional counting semaphores in hardware. System and application software can build on these primitives to perform resource management, interprocessor communication, and barriers and state exchange, in addition to simple locking.

In summary, on a per-core basis, the L1, L2, and L3 cache sizes remain the same in the BG/P system as in the BG/L system. The L1, L2, and L3 caches as well as main memory bandwidths, measured in terms of bytes per

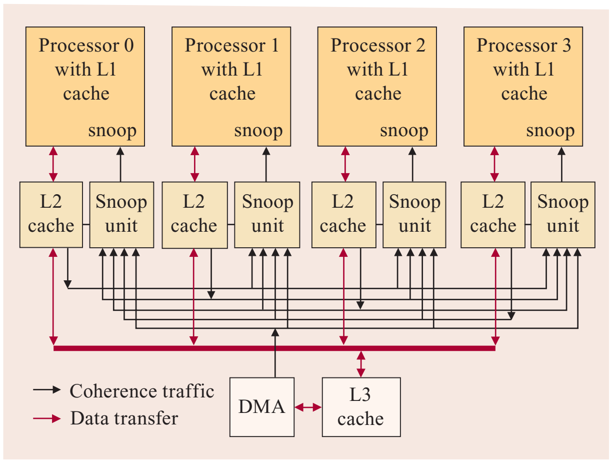


Figure 6

BG/P cache coherency is maintained among the four cores with a write-invalidate protocol. Irrelevant invalidate requests are filtered out by snoop filters.

FLOP, also remain the same as in the BG/L system. Thus, the BG/P platform maintains the overall memory system characteristics for applications, easing the migration of code from the BG/L system to the BG/P system while introducing improvements in performance and flexibility.

Node coherence

The BG/L system was based on the PowerPC 440 processor core, which did not support cache coherence. In the BG/P system, however, the PPC450 core provides hardware support for cache coherence and, therefore, allows BG/P nodes to be used as SMPs. As mentioned, data integrity between the processors is maintained with a cache coherence protocol based on write-invalidates, with all L1 caches operating in write-through mode. Every store not only updates the L1 cache of the issuing core but also immediately sends the write data via the L2 write buffer to the shared L3 cache. If two or more processors attempt to write to the same location simultaneously, then only one of them succeeds at the L3. The L2s broadcast an invalidate request for the write address to ensure that no stale copy of the same data will remain in the other L1s and L2s.

In SMP architectures, invalidate requests represent a significant fraction of all cache accesses, but only a small fraction of the invalidate requests will typically hit in any of the remote caches. This is particularly true of supercomputing applications in which optimization techniques such as data partitioning and data blocking will have increased locality of reference.

On the BPC chip, we designed a snoop filter (also known as a *coherence request filter*) that is associated with each of the four processor cores, as illustrated in **Figure 6**. By filtering out irrelevant invalidate requests, it significantly reduces the interference of invalidate requests with processor operations while adding minimal chip area, latency, or complexity. The snoop filter compares incoming invalidate requests to its own copy of the L1 address tags. If the L1 cache does not contain the address tag, the invalidate request is not relevant to the L1 cache of its processor, and the snoop filter suppresses transmission of the irrelevant invalidate request to the L1 cache. Thus, the number of actual invalidate requests presented to the L1 cache is reduced, thereby increasing performance and reducing power consumption.

Each L2 cache broadcasts invalidate requests to the remote snoop filters using a point-to-point coherence network. Memory system coherence also extends to the DMA engine. The DMA connects to the memory system at the L3. Because of the L1 write-through policy, all core writes are visible in the L3. When the DMA writes data from the network into the L3, corresponding invalidate requests are generated and sent to all four cores.

The snoop filters, therefore, receive multiple concurrent requests. Each snoop filter implements a separate block (or port filter) for each interconnect port. Thus, each snoop filter in Figure 6 has four separate port filters, three to handle requests from a remote L2 and one to handle requests from the DMA.

Each snoop port filter implements three different and complementary filtering algorithms, capturing various characteristics of the memory references. The first is a snoop cache that filters invalidate requests on the basis of temporal locality. This means that if a single invalidate request for a particular location was made, it is probable that another request to the same location will be made soon. This filter unit effectively records a subset of memory blocks that are not cached. The second filter, a set of stream registers, uses an orthogonal snoop filtering technique, exploiting the regularity of memory accesses. This filter unit records a superset of blocks that are cached. The third filter is a range filter, in which a range of addresses is specified. Depending on the selected mode, address ranges are set to filter all coherence requests with addresses either within or outside of the specified address range. Results of the three filter units are considered in a combined filtering decision. If any one of the filtering units decides that a snoop request should be discarded, the snoop request is discarded.

Early measurements show that the snoop filters reject about 90% of unnecessary invalidates, resulting in performance improvements up to 35%, depending on the application.

To ensure that memory synchronization occurs among the processors, our design supports the PowerPC instructions *msync* and *lwarx*/*stwcx*. The *lwarx* instruction loads a word and sets a reservation bit internal to the PPC processor. The *stwcx* instruction updates memory, but only if the reservation bit is set. Together, these two instructions allow the implementation of atomic memory updates. The *msync* instruction ensures that all scheduled instructions complete prior to completion of the *msync*. This is necessary because the PowerPC architecture supports out-of-order execution. When necessary, a global synchronization enforces completion of all snoops and memory writes before proceeding.

Networks

In the BG/P system, as in the BG/L system, three networks are used for node-to-node communication: a 3D torus network, a collective network, and a global barrier network. The internal logic of the torus and collective networks remains essentially unchanged from those of the BG/L system. However, on the BG/P system, these networks deliver more than twice as many bytes per cycle than on the BG/L system. This was accomplished by doubling the internal clock rates of the network logic and by raising the signaling rate by a factor of 2.43 with respect to the BG/L system through the use of IBM high-speed serial (HSS) I/O technology (see Serdes area in Figure 2). As a result, the ratio of I/O bandwidth to floating-point performance, as measured in bytes per FLOP, is preserved, easing the migration of BG/L code to the BG/P system.

DMA

On the BG/L system, the processor cores were responsible for injecting (or receiving) packets to (or from) the network. As mentioned, on the BG/P system, a DMA engine has been added to offload most of this responsibility from the cores, thus enabling better overlap of communication and computation. Specifically, the DMA interfaces with the torus network. Cores are still responsible for handling packets on the collective network. Because most Message Passing Interface (MPI) calls that use the collective network are both global in nature and blocking, there is little performance to be gained by offloading collective network traffic. The DMA is rich in function yet occupies only a small fraction of the chip. The major DMA constructs include injection and reception memory FIFO (first-in, first-out) buffers, injection and reception byte counters, and short (32-byte) message descriptors. The FIFOs and counters are organized into four groups so that each core can potentially reserve, manage, and use its own DMA resources. The DMA can send messages to other nodes or to itself, resulting in a local, intranode memory copy (or, optionally, simply a prefetch into L3).

Three message types exist: memory FIFO, direct put, and remote get. Each message is defined by a 32-byte message descriptor that contains such information as the message type, DMA resources used, a torus packet header specifying the destination, and a message length and pointer to the start of the message. For a memory FIFO message, the packets (including torus headers) are placed in a reception FIFO on the destination node. For a direct put, the payloads of the packets are deposited directly into an arbitrary memory buffer, for example, the reception buffer of an MPI_Irecv call. For a remote get, the payload of the message is itself one or more message descriptors. These descriptors are placed into an injection FIFO on the destination node for eventual processing by the DMA. This descriptor may correspond to a direct put of a long message back to the source node.

A memory FIFO can be of arbitrary length and located anywhere in (physically contiguous) memory. The DMA maintains pointers to the start, end, head, and tail of the FIFO. It is managed as a producer–consumer queue in a circular buffer. For (non-remote get) injection FIFOs, cores are the producers, placing message descriptors into the FIFO and advancing the tail pointer. The DMA is the consumer, reading the descriptor at the head of the FIFO, interpreting the descriptor, fetching the data from the memory, packetizing the data, injecting the packets into the torus network, and advancing the head pointer. For an injection FIFO that is used (exclusively) for remote get messages, the DMA is both the producer and the consumer. For reception FIFOs, the DMA is the producer, placing network packets into the FIFO and advancing the tail pointer. Cores are the consumers, reading packets in the FIFO and advancing the head pointer. There are 32 injection FIFOs per group (128 total) and 8 reception FIFOs per group.

In addition, there are 64 injection and 64 reception counters per group (for a total of 256 each). Each counter has a base physical address and a byte counter. Reception counters also have a maximum physical address. Each message must have an injection counter, and all direct put messages have a reception counter, each specified as a 1-byte identifier. Upon injection, the payload of the message is pulled from the base address of the injection counter plus an offset specified in the descriptor. This is done on a packet-by-packet basis. For each packet, the injection counter is decremented by the number of payload bytes in the packet. For direct put packets, the reception counter ID is placed in each packet, and the DMA computes the appropriate put offset for the packet. Upon reception, the payload of a packet is written starting at the base address of the reception counter plus the offset of the packet, and the reception byte counter is decremented by the number of payload bytes in the packet. To test for message completion, software can poll a counter to see whether it has reached an appropriate value (usually zero). Because only a limited number of counters exist, a variety of counter sharing and management techniques have been developed and implemented in the messaging software.

Torus packets can be up to 256 bytes in length (in multiples of 32). The torus hardware routing header occupies the first 8 bytes, and the DMA header occupies the next 8 bytes, leaving up to 240 bytes of payload data per packet.

The DMA uses physical addresses; thus, a single put message must read (or write) data from (or to) a physically contiguous buffer. The BG/P compute node kernel (CNK), which is optimized for scientific applications, has a simplified memory management policy in which, for most of user memory, the physical address equals the virtual address plus an offset. Under the CNK, DMA messaging occurs entirely in user space, and efficient user-space calls exist to do the virtual-to-physical translation. For most (nonstrided) large messages, a single descriptor and a single memory translation call can be used for the entire message.

The DMA can also generate interrupts for events such as a counter hitting zero, a FIFO becoming full, or an attempt to write to an invalid memory address.

The combination of the DMA, torus network, and memory system is capable of supporting high-bandwidth communications. Our measurements show that for a full 3D nearest-neighbor exchange (six neighbors per node), up to 93% of peak performance can be sustained. Because each link corresponds to 425 MB/s and the payload utilization of the link is 88% [payload bytes/(packet-size + CRC + acknowledgment packet)], the node simultaneously sends and receives 2.09 GB/s (equal to 425 MB/s $\times$ 6 $\times$ 0.88 $\times$ 0.93) for an aggregate rate of 4.18 GB/s.

Performance counters

Performance counters are particularly important for HPC systems, such as the Blue Gene system, in which performance tuning is critical to achieve high efficiency. Most traditional processor architectures support a limited number of counters for processor events and system events. In the BG/P system, we provide a performance counter unit that implements 256 64-bit counters. At any time, 256 events out of more than 1,000 events can be monitored concurrently. To efficiently implement this large number of counters, we designed a novel hybrid performance monitor. The 64-bit counters are split into a 12-bit low-order portion that is implemented as classical fast counters, as well as a 52-bit high-order portion that is implemented densely using an SRAM array. The SRAM array logic sequentially polls the low-order counters and increments an SRAM word if the overflow bit of the corresponding low-order counter is set.

Each counter can be configured individually to count in one of four different signal-level modes: level-sensitive events (low- or high-active) and edge-sensitive events (low-to-high or high-to-low transition). In addition, each counter can be configured to generate an interrupt if a specified threshold value is reached. The counter configuration is specified in a memory-mapped register block, which can be accessed from all processors in a shared memory space, if given access by the operating system.

Reliability, availability, and serviceability

For supercomputers, the sheer number of components requires a strong focus on reliability, availability, and serviceability (RAS), in both the hardware and the software design. The BG/P system reliability target is less than one fail per rack per year. In order to meet or exceed this reliability standard, the hardware features for RAS in the BG/P system largely follow the successful BG/L approach, with $N + 1$ redundant power supplies at all levels, $N + 1$ redundant fans, reliability-grade-1 processor chips, direct soldered memory, and spare wires in cables. The L1 and L2 caches and all major on-chip buses are parity-protected and the L3 cache is protected by ECC. The L1 cache and L2 cache will operate in write-through mode so that any new data is written directly to ECC-protected L3.

A major enhancement to the BG/P system is the ECC protection of the external DRAMs against both hard and soft errors. This was required because each BPC chip has two on-chip DRAM controllers, controlling twice the number of DRAMs as a BG/L compute (BLC) chip. In the BG/P system, each controller communicates with the external DDR2 SDRAMs via a 160-bit-wide bus, of which 128 bits are user data. In the BG/L system, the DRAM data bus was 144 bits wide. The additional bits in the BG/P system allow for storing address parity bits, spare bits, and enhanced ECC protection data. The ECC protection is such that it can correct either single or double symbol errors (where a symbol is three adjacent physical bits) and can tolerate the failure of a full x16 DRAM chip (or, equivalently, two adjacent x8 DRAM chips)—a feature called IBM Chipkill. Because DRAM chips constitute the vast majority of chips in the BG/P system, this strong ECC scheme is a major factor in maintaining the reliability of the system against both hard and soft errors. With the DRAMs thus protected, the principal remaining failure mode of a BG/P system is expected to be a hard fail of a BPC chip. Although all BPC and BPL chips are burnt-in (i.e., prestressed at elevated temperature and voltage), the failure rate of the machine is expected to be somewhat higher during the first year or so of operation, and thereafter to improve as early fails are removed from the system.

Software architecture overview

This section presents the software architecture for the BG/P system. A software overview article [12] appeared after the release of the BG/L system. Thus, while we present the overall BG/P software model here, we focus on the areas that involve the most significant changes between the BG/L and BG/P systems.

The BG/P system is being released at a transition point in computing. Previously, Moore's Law was satisfied by increasing the speed of each of the cores. For numerous reasons, upcoming generations of computers will achieve Moore's Law by increasing the number of cores. This will have a profound impact on the way in which we must perform computation. In the generation after the BG/P architecture, the number of processing elements in a supercomputer will be sufficiently large that a programming model other than a flat MPI programming model will be required. On the BG/P system, this upcoming need was taken into account as we designed the software and represents one of the significant differences from the BG/L system. For example, in this section, we discuss the SMP runtime mode, different potential models for producing a two-tiered programming model, and what we have done to make those efficient. Although potentially only a few applications will require such models to fully take advantage of the BG/P system, the BG/P system represents an excellent test vehicle, both for Blue Gene and for other platforms, on which application programmers can transition from the programming model of the previous generation to the necessities of next-generation programming models.

We address the following challenges associated with the system software on the BG/P system:

Scalability—The design of system software that scales to hundreds of thousands of cores requires careful software design and strict adherence to the principles embodied by the design. It is possible to approach scalability and, to a lesser extent, performance from two directions. Existing system software can be scaled by identifying the location of bottlenecks and addressing those areas, one by one, that are associated with a machine of a given size. As larger machines become available, new bottlenecks arise and are then addressed. An alternative approach is to start with a minimal-functionality system stack, but one that is designed to scale. The Blue Gene strategy to achieve scalability and high performance has been to start simply, for example, with space sharing, one job per partition, no paging, and one thread per core.
Performance—System software should not be the performance-limiting factor. We maintained the Blue Gene design philosophy that the machine speed should be driven by hardware constraints and that the system software should not impede the performance obtainable by hardware and application design.
Functionality—One of our ongoing goals with the BG/P system is to make the machine useful to more HPC applications, as well as to applications outside the HPC arena while continuing to maintain the performance required by HPC applications. To this end, we have increased the set of POSIX** (Portable Operating System Interface) functions we support and added binary compatibility with Linux**. By generalizing the messaging stack, we now have the ability to support a range of programming models including MPI, OpenMP**, ARMCI (Aggregate Remote Memory Copy Interface), Global Arrays, Charm++, and UPC (Unified Parallel C, a global address space extension of C). As with the BG/L system, we also provide a variety of important optimized numerical libraries, including ESSL (Engineering Scientific Subroutine Library), MASS (Mathematical Acceleration Subsystem) and MASSV, ScaLAPACK (Scalable Linear Algebra Package), FFTW (an implementation of fast Fourier transform developed at MIT), and glibc (GNU C library).
Wider accessibility—The Blue Gene platform currently excels at watts per FLOP [9], making it an attractive capacity machine in addition to a capability machine (see Figure 5). In order to enhance accessibility, we are exploring Linux on the compute node and using compute nodes as accelerators for programs such as MATLAB** or Java**, while still maintaining the necessary scalability and performance.

In the following sections, we describe each of the system software components. We focus on the areas that have undergone the most change comparing the BG/L and BG/P systems. We start by describing our open-source strategy on the BG/P system, which marks a significant change from the BG/L system. This is followed by a section on compiler support. We then describe the messaging stack and programming models. Following that, we describe the CNK (compute node kernel). We then describe the I/O node Linux, followed by a presentation of the BG/P control system. We finish this section by presenting our efforts to broaden the user base by describing our compute node Linux strategy and other efforts to make Blue Gene computing cycles available to a wider audience, including work on system-level acceleration on the Blue Gene platform.

Other crucial pieces of the software stack, for example, the IBM LoadLeveler* scheduler and the IBM GPFS, are available for BG/P systems. These have been described in the original BG/L software paper [12].

Open source

A foundation of the BG/P initiative is the fostering of collaboration around the core system software. A significant portion of the BG/P software stack is provided as open-source software that allows customers, laboratories, and universities to customize and optimize the code to meet their needs and to return innovations to the community. These innovations may be integrated back into the product.

All software that has a direct impact on an application in the system software stack is open-source software. This includes all layers of the communication libraries, for example, the highest level of MPI and Global Arrays down through the system programming interfaces that control the hardware through direct register access. Application owners are given the opportunity to fully understand performance by including these layers directly in their analysis.

To further computer science research in operating systems, we have made the lightweight CNK and the associated I/O path open source, and we include source for all system daemons on the I/O node. Linux and all drivers are open source as well. This gives users outside IBM a better understanding of performance impacts of modifications to the I/O paths in the existing system and allows development of alternative kernels and methods.

Enabling this open-source strategy are two software components, the Bootloader and Common Node Services (CNS), which provide a firmware layer resident on compute and I/O nodes. These components collaborate with the control system service node to perform early chip initialization, device configuration, and kernel load and to provide a common functional interface, used by all kernels, for RAS event capture and reporting.

Finally, administration and efficient use of the system via job scheduling often has unique requirements from installation to installation. The Web-based Blue Gene Navigator and scheduler APIs (application programming interfaces) are open source in order to allow for experimentation as well as advanced customization and automation.

Compiler support

BG/P system users develop code and run applications from front-end nodes (FENs), which are 64-bit POWER* systems typically in a blade form-factor. The FENs are installed with Blue Gene system-specific compilers. The application may be executed using a standard *mpirun* launch interface or more typically using a job scheduler such as LoadLeveler.

For Blue Gene systems, IBM provides XL C, C++, and Fortran compilers and supports the widely available gcc compiler. Only the XL compilers exploit the dual-pipeline FPU. For an individual core, the XL compiler optimization work done for the BG/L system [13–16] is readily applicable to the BG/P system, because the 16-byte aligned loads and stores and the dual-pipeline FPU retain their BG/L characteristics. Throughout the availability of the BG/L system, the optimizing capability of the XL compilers has steadily improved, particularly with respect to exploiting the dual-pipeline FPU. In a September 2007 release, the XL C/C++ 9.0 and XL Fortran 11.1 compilers have continued this trend.

Additionally, the latest compiler release supports OpenMP directives, allowing users and developers to leverage OpenMP parallelism on the node while using other mechanisms such as MPI to communicate between nodes. Some examples of the use of OpenMP are provided in the section "BG/P application performance and scaling."

Messaging framework and programming models

The BG/P system messaging framework has been redesigned, enhanced, and generalized. The new framework is implemented primarily in C++ and provides multiple levels of abstraction. One of its key design goals was to provide support for multiple programming models. Although MPI applications are still considered the primary targets of the BG/P system, alternative programming models that can exist on top of the new messaging stack include ARMCI from the PNNL (Pacific Northwest National Laboratory), GA (Global Arrays, matrix operations on large distributed matrices), Converse/Charm++ (an asynchronous parallel programming language from the University of Illinois), and UPC. **Figure 7** depicts the hierarchical structure of the messaging framework.

The foundation of the messaging stack is the Deep Computing Messaging Framework (DCMF) and the Component Collective Messaging Interface (CCMI). DCMF is a C++ library implementing efficient point-to-point messaging on top of the System Programming Interface (SPI). The SPI consists of a collection of C functions for low-level access and control of the network hardware. DCMF provides abstract interfaces that are based on C++ classes for the relevant pieces of the hardware, including the collective network, the torus network, and the global interrupt network (depicted in Figure 7 as a row of three device boxes), and for communication protocols using those devices. In addition to the standard point-to-point protocols, DCMF implements a novel multisend protocol. A multisend protocol is a type of aggregation of many point-to-point messages into one abstract operation that makes specific hardware-dependent optimizations possible for groups of messages. The multisend protocol in DCMF is the bridge that connects the abstract implementations of collective operations (in CCMI) to specific BG/P communication networks.

DCMF also provides a simple but powerful C language API that serves as the primary public interface for higher-level messaging systems such as MPI or ARMCI. The public DCMF API exposes three types of message-passing operations: a two-sided send, a multisend, and a one-sided get. All three have nonblocking semantics to facilitate overlapping of computation and communication. The user is notified about completions of communication events by means of completion callback functions. The API is based on the active message model. A reception callback function, registered through the API, is activated automatically at the receiving node on arrival of the message. One-sided messaging is supported by the get API call. It takes full advantage of the DMA hardware. In particular, its remote get mechanism avoids CPU intervention on the remote node by using the DMA hardware to fulfill the requested data transfer. The API also provides a registration framework for the supported messaging operations that enables coexistence of different higher-level messaging systems within the same application. For example, an application may issue both MPI and ARMCI calls.

SMP mode within a compute node is fully supported because the API calls are thread-safe. In particular, MPICH (a portable implementation of MPI) on the BG/P system can run in *thread multiple* mode, which is the highest possible MPI thread level. DCMF is fully compliant with MPI progress semantics by utilizing interrupts and dedicated lightweight communication threads.

Optimized collective operations for the BG/P system are implemented in the CCMI C++ library (see Figure 7). CCMI defines a set of components representing a flexible, programming-paradigm-independent framework for implementing collectives. In particular, it uses schedule and executor components to express the algorithm of a collective operation. A schedule defines the sources or targets of the incoming or outgoing messages at each step of the algorithm on the basis of a virtual network topology of the contributing processes. The executor queries the schedule in each step and performs the sending or receiving operations. Typically, each type of collective operation requires one executor that implements the necessary operation-specific optimizations. CCMI provides executors for barrier, broadcast, reduce, allreduce, and all-to-all collectives. An executor can take different schedules, optimized for different network hardware and physical topology; for example, rectangular blocks in the torus network have an optimized schedule that takes advantage of the deposit bit hardware feature. The actual data transfer in the executor is done by means of the multisend component. The executor can pick

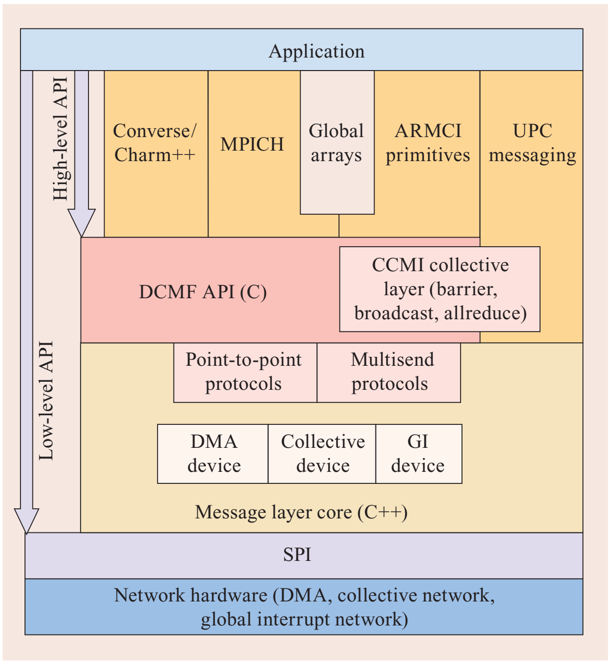


Figure 7

The Blue Gene/P messaging framework. (CCMI: component collective messaging interface; UPC: universal parallel C; DCMF: deep computing messaging framework; API: application-programming interface; DMA: direct memory access; GI: global interrupt; SPI: system programming interface.)

different multisend instances in order to perform the same collective operation on top of any of the various hardware networks. The actual multisend implementations come from the DCMF library. The CCMI framework is sufficiently flexible to support both synchronous and asynchronous collectives. An asynchronous low-latency broadcast is part of CCMI.

Compute node kernel

The BG/P CNK provides a 32-bit PowerPC SMP operating system that is binary compatible with I/O node Linux at the system call interface. To enhance performance and scalability, CNK restricts the number of processes per node to a maximum of four and the number of threads per process to a maximum of one per processor. CNK provides three major modes of operation to high-performance parallel applications: 1) SMP mode, supporting a single multithreaded application process per compute node, 2) dual mode, supporting two multithreaded application processes per compute node, and 3) quad mode, supporting four single-threaded application processes per compute node. In the BG/L literature [12], these application processes were typically called *virtual nodes*. Unlike normal fork/exec semantics, these application processes are created at application load time by CNK. Processor cores are statically assigned, or bound, to application processes. This strategy allows CNK to fairly apportion CPU and memory to processes, ensuring that all nodes and threads in the parallel job have the opportunity to sustain equal performance. Further, this strategy ensures minimal memory fragmentation, allowing the virtual memory manager (VMM) of the CNK to statically map the address space for each process using large translation lookaside buffers (TLBs), thereby avoiding any overhead and asynchrony due to TLB-miss handling. To ensure the most efficient use of memory, CNK application processes are single program, multiple data (SPMD), sharing a single copy of the text and read-only constant data, but with per-process private data and stack areas. CNK also supports POSIX shared memory allocation interfaces, allowing a flexible amount of memory to be set aside for sharing between the processes. This capability allows communication libraries, such as MPI, to more easily coordinate activities within a physical node while allowing applications to share a single copy of dynamic data, which is often large and read-mostly. This allocation strategy also enhances performance of the DMA SPI, described below, by simplifying virtual-to-physical address translation and allowing large messages to be sent with a single DMA operation.

Multithreading in the CNK is implemented with support for the complete Native POSIX Threading Library (NPTL), primarily for use in dual and SMP mode because the CNK restricts the number of application threads to one per core. An optional second *pthread* per core is supported. This additional thread, available to applications or system runtime environments, is intended to be used as a communication thread (CommThread). CommThreads are a nonportable extension to NPTL. They provide stateless user-space pthreads that are launched upon communication interrupts, global interrupts, and interprocessor (core-to-core) interrupts. In this way, CommThreads provide high-performance interrupt-driven communication to user-space threads. In the CNK, threads are managed with a per-core scheduler and in the typical case amounts to a single count-leading-zeros instruction, indicating which, if any, threads are ready to run on that core.

Functionally, the CNK provides a broader choice of application environments than that which is available on the BG/L system. In addition to MPI and OpenMP, the CNK supports scripting environments including Python. This was motivated by the fact that several applications of interest on the BG/P system use Python scripts to manage dynamically linked modules and application kernels that change as the application proceeds through phases and states.

A major component of CNK is a rich set of SPIs that provide low-level access to all devices in the BG/P system, as well as kernel and job state interfaces. The network SPIs of the CNK include torus packet-layer and torus DMA, collective network packet-layer and configuration, and global interrupt barrier and notification handling. Additionally, SPIs are provided for devices local to the node, including the lockbox, and the universal performance counters.

The SPIs closely follow the hardware interfaces. Thus, making effective use of the interfaces requires detailed knowledge of the hardware. Carefully coded applications, such as Blue Matter [17], have exploited these interfaces with unprecedented performance and scalability results. Typically, however, the SPI is used by higher-level libraries, as described in the previous section "Messaging framework and programming models." In the following paragraphs, we describe the SPIs.

The *DMA messaging SPI* provides access to all DMA resources such as counters and FIFO pointers. In addition, SPI system-level calls exist for allocating DMA resources such as counters and injection FIFOs. This permits a mixed-messaging mode in which, for example, MPI and an optimized SPI-based library each have their own FIFOs and counters. Although there is less flexibility in assigning reception FIFOs (as compared to the counters and injection FIFOs), the SPI permits software to register its own packet-handling functions. The SPI calls (which poll reception FIFOs) then dispatch the correct packet-handling function for each packet, thereby allowing packets from MPI and an SPI-based library to be received in the same reception FIFOs.

The *collective network SPI* includes optimized packet injection and reception functions and status polling operations. In addition, interfaces are provided to program the collective network route registers for construction of virtual trees within the collective network.

The *global interrupt SPI* allows applications to quickly establish a global barrier, a global notification, or an alert. These global operations are delivered to CommThreads as an interrupt.

The *lockbox SPI* provides interfaces to enable the allocation and use of atomic counters. Within each physical compute node, the lockbox is a shared resource among the application processes. Functions in the SPI include simple locks, barrier operations among a flexible number of participants, barrier with status notification to the participants, and the full set of PV (Proberen–Verhogen) counting semaphore operations.

The *universal performance counters SPI* provides the ability to configure, initialize, and query counters and to enable interrupts on the basis of count values. Typically, the universal performance counters are made available to applications via higher-level libraries such as PAPI (Performance Application Programming Interface).

The *CNK SPI* provides interfaces to quickly query the application process and thread mapping that is currently in effect, and to translate virtual to physical addresses as required for DMA. This SPI also supports RAS functions and parallel job query and management functions. The kernel SPI is also used extensively by CommThreads to selectively enable and acknowledge communication interrupts and to deliver interprocessor interrupts (IPIs).

Unlike the BG/L system, which used a separate kernel for verification, manufacturing tests, and diagnostics, the CNK directly supports all of this functionality via its *kernel-extension* interface. Hundreds of special CNK-extension test cases have been developed that run with full kernel privileges and, thus, can take advantage of the hardware beyond what is typically possible from applications. Further, unusual configuration scenarios can be created by these kernel-extension diagnostics that stress the hardware in nonstandard ways. In support of its use in verification, testing, and diagnostics, the CNK has a strong emphasis on reporting error conditions (also known as *RAS events*). The reporting is comprehensive and efficient and is done in a way that does not burden each compute node with large amounts of error-condition description code.

I/O node Linux

The BG/P system presents a challenge for Linux on the I/O node with a 10-Gb/s Ethernet interface. In order to obtain performance consistent with the philosophy of driving the hardware at its maximum rate, Linux on the I/O node requires an efficient SMP implementation. While Linux has support for SMP, there was no such support for the PPC450 processor. Fortunately, the PPC450 core supports the standard PowerPC locking primitives; thus, the PPC450 Linux implementation required modification only to the initialization paths and the TLB handling, which are unique to this processor and to Blue Gene systems. Because I/O nodes do not participate in torus communication, a DMA device was not needed for I/O node Linux.

The control and I/O daemon (CIOD) has been enhanced on the BG/P system to take advantage of the SMP environment and to provide better Linux semantics for redirected system calls from the CNK. The CIOD is organized in such a way that a Linux process represents each CNK that allows Linux to track file descriptors, file locks, current working directory, and similar state. This provides a CNK-based application with true Linux process semantics.

Control system

Blue Gene computers are configured, booted, and monitored using a control system [Midplane Management and Control System (MMCS)] that runs on a service node (SN). The SN is a commercial-grade 64-bit POWER5 processor-based server, located externally to the Blue Gene compute racks. The SN uses SUSE** Linux Enterprise Server version 10 (SLES10) as an operating system and is usually configured so that it is accessible only to administrators of the system.

Through a dedicated external Ethernet switch, the SN is attached to the service card of each midplane via a 1-Gb/s Ethernet connection and from there to a control FPGA (field programmable gate array chip) on each node card or link card. The FPGA controls card functions such as power supplies and temperature sensors, and it drives an IEEE 1149.1 interface, known as a *JTAG* interface, to each BPC or BPL chip for configuring, boot-code loading, monitoring, and debugging. This private control network has been upgraded from 100 Mb/s Ethernet for the BG/L system to 1 Gb/s Ethernet for the BG/P system, to better support these functions.

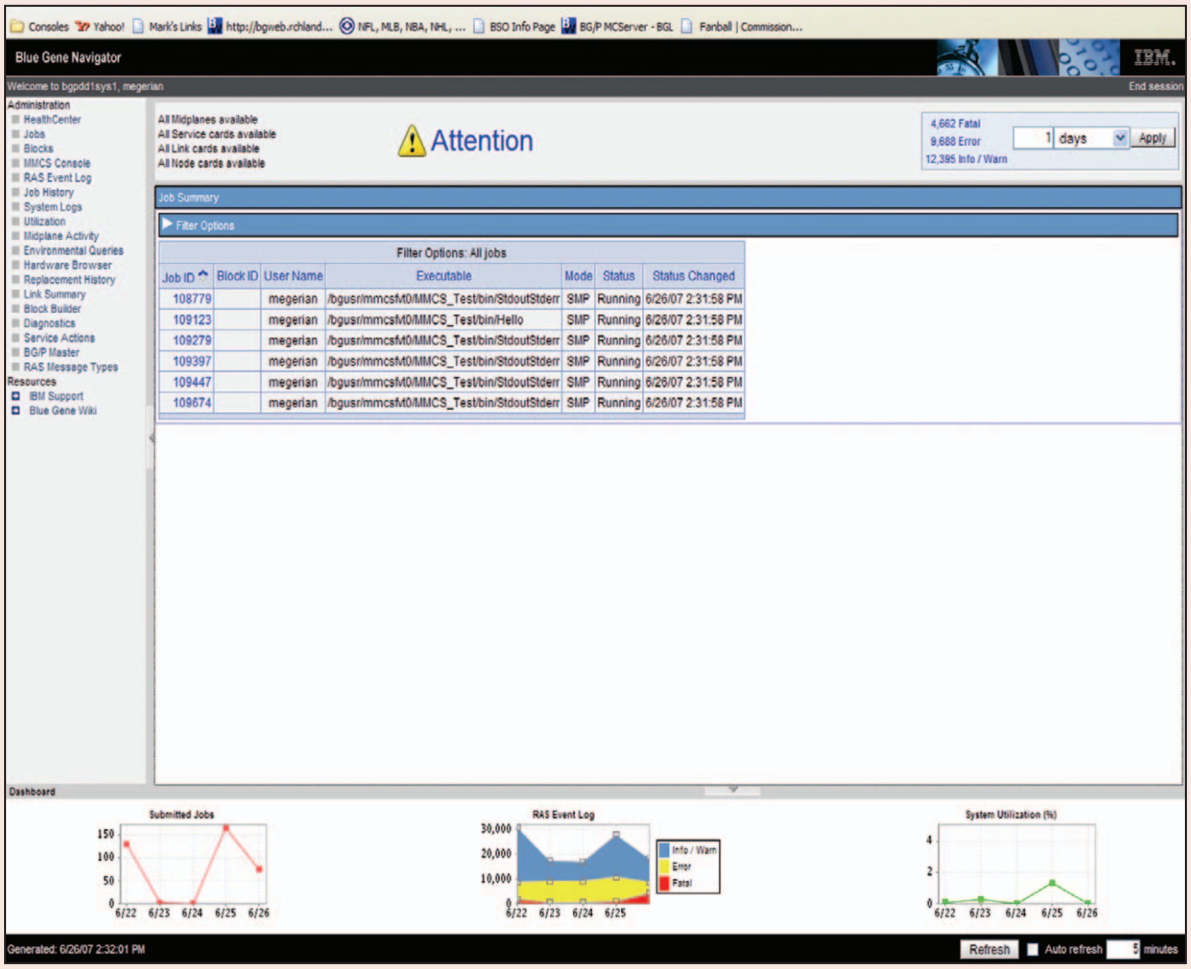
The software architecture of the MMCS for the BG/P system has been built on the strengths of its predecessor for the BG/L system, and it has improved areas that were previously weak. Decisions that proved successful on the BG/L system were adopted as elements of the MMCS, such as having one central process that spawned threads for each booted partition and using an IBM DB2* database as the active central communication mechanism for components of the control system.

Much of the BG/P system architecture remains the same as that of the BG/L system, such as the basic hardware layout, partitioning, and job handling. Thus, in order to provide a consistent feel for users, the associated aspects of the control system remain similar. The areas of improvement for the BG/P control system include the handling of RAS events, the Blue Gene Navigator (administrator console), the partitioning capability, and overall scalability. These four aspects are described in more detail below.

Control system RAS events

Any error event detected in any part of the system hardware will give rise to a *RAS event message*, which is sent to the MMCS server process and stored in a RAS database. In the BG/L system, the kernel software running on compute and I/O nodes can issue RAS messages as free-form text messages. Storing these RAS messages on each compute or I/O chip required significant code space.

For the BG/P system, we have designed a more structured RAS architecture. RAS messages are issued by the CardController processes of the control system for events related to power supplies, temperature monitors, and other support hardware. Also, the CNK software running on compute nodes and the Linux kernel running



Welcome to bgpdd1sys1, megerian

All Midplanes available
All Service cards available
All Link cards available
All Node cards available

4,662 Fatal
9,688 Error
12,395 Info / Warn
1 days

Filter Options: All jobs

Job ID	Block ID	User Name	Executable	Mode	Status	Status Changed
108779		megerian	/bgusr/mmcsM0/MMCS_Test/bin/StdoutStderr	SMP	Running	6/26/07 2:31:58 PM
109123		megerian	/bgusr/mmcsM0/MMCS_Test/bin/Hello	SMP	Running	6/26/07 2:31:58 PM
109279		megerian	/bgusr/mmcsM0/MMCS_Test/bin/StdoutStderr	SMP	Running	6/26/07 2:31:58 PM
109397		megerian	/bgusr/mmcsM0/MMCS_Test/bin/StdoutStderr	SMP	Running	6/26/07 2:31:58 PM
109447		megerian	/bgusr/mmcsM0/MMCS_Test/bin/StdoutStderr	SMP	Running	6/26/07 2:31:58 PM
109674		megerian	/bgusr/mmcsM0/MMCS_Test/bin/StdoutStderr	SMP	Running	6/26/07 2:31:58 PM

Generated: 6/26/07 2:32:01 PM

Figure 8

The integrated Blue Gene Navigator.

on I/O nodes may issue binary RAS messages. In all cases, the RAS messages are decoded and further handled by the control system software stack running on the SN. This design substantially reduces the kernel code space required for RAS messages. Concentrating the RAS handling in the control system has resulted in a more scalable and flexible RAS architecture. The identifying information is standardized, and supporting detail is given where necessary. Message text, severity codes, and recommended service actions can be adapted on the basis of the operational context of the machine (i.e., contexts relating to a card test, a manufacturing test, or in an installation at the customer site), with the aim of giving system operators in each context accurate and meaningful information upon an error event.

Control system Navigator

The Blue Gene Navigator is a browser-based administrator console that provides a comprehensive view of the status and activity of the machine (**Figure 8**). While the BG/L Navigator was received positively by users, a demand existed for additional customization features. On the BG/P system, the Navigator allows users to create plug-ins to suit their particular needs. For example, in the BG/L system, the Attention bar at the top would be activated whenever fatal RAS events occurred or when hardware was in an error state. For the BG/P system, users can write their own pieces of code to make the Attention flag active for their own purposes, such as for use when file system storage, certain hardware readings, or even utilization of the machine exceeds a certain threshold. The Navigator also supports custom plug-ins for the graphs in the dashboard section at the bottom of the screen and for the navigation pane of links along the left side. An end-user version of the Navigator has also been created for the BG/P system, which gives a read-only view of the machine but hides some features of the administrator console, such as those that relate to service actions and diagnostics. These changes improve the power and usefulness of the Blue Gene Navigator.

Control system partitioning
Blue Gene systems allow considerable flexibility in logically partitioning the machine into smaller blocks so that multiple smaller jobs can be run simultaneously for different users. Typically, such partitions are the size of one midplane (512 nodes) or, typically in powers of 2, up to the full size of the installed machine. Each such user partition will have a complete torus, tree, and global interrupt network. On the BG/L system, the smallest allowable partition sizes were 32 and 128 nodes. On these sub-midplane partitions, one or more torus dimensions cannot be closed, and the node topology will be a 3D mesh. On the BG/P system, we have extended the system software to support sub-midplane partitions of 16, 32, 64, 128, and 256 nodes. We also addressed the challenging problem of dynamic partitioning and provide such partitioning on the BG/P system. On the basis of user feedback, we have provided real-time notification APIs. Using this model, we do not provide all information but only provide change information, and we move from a poll model to an event model in which schedulers and application programs can be notified of state changes to partitions and active jobs.

Customer feedback also led us to address the issue of rebooting only a portion of a Blue Gene partition. On the BG/L system, a reboot of a partition required that all I/O nodes and compute nodes be rebooted. For the BG/P system, we allow users to reboot only compute nodes or to reboot only a subset of the I/O nodes. This is a significant improvement, because it allows a partition that is booted normally (with all I/O nodes having Linux started and with their Ethernet connections and mounted file systems functioning properly) to remain unaffected by a reboot of the compute nodes. This can lead to better stability of the I/O node complex while still giving users the flexibility of either leaving the compute nodes booted across multiple jobs or performing a reboot before each job starts. It also means that a failure on a single I/O node can be resolved without having an impact on the other I/O nodes.

Control system scalability
The Blue Gene control system must configure, track, and monitor systems ranging from 32-node test stands to large installations of hundreds of thousands of compute nodes and associated components. For the BG/P system, we have aggressively addressed bottlenecks that were experienced in the BG/L control system on larger installations, resulting in much better scalability.

Broader use of Blue Gene cycles
On the BG/P system, we provide support for using Blue Gene compute cycles beyond HPC applications while still maintaining the high performance mandated by HPC applications. This initiative for broader application support began in the later stages of BG/L system work. The work is currently in progress and is proceeding along many fronts. Here, we describe two such initiatives. The first investigates how to run Linux on the compute nodes in a scalable manner, and the second involves an infrastructure to allow Blue Gene to be used as a broader-based compute engine, taking full advantage of its tremendous ratio of flops to watts [9].

Blue Gene/P and Linux
The CNK described in previous sections is carefully designed for minimal overhead; thus, performance on the BG/P system is very close to the limits imposed by the hardware. On the other hand, the BG/P system is not confined to running specialized kernels on the compute nodes. Running Linux, for example, broadens the range of applications that can run on a Blue Gene system.

Many issues must be considered when using Linux. These issues concern how to boot Linux, how to launch applications, how to handle communication and file I/O, how to effectively utilize the DMA device, and how to ensure that it scales to machines that are the size of the Blue Gene machine. Booting Linux on a compute node leverages the work done to run a Linux image on the I/O nodes. The major change arises from the fact that the ramdisk must accommodate a larger number of dynamic libraries that are usually not required on the I/O nodes.

Launching applications requires cooperation between the control system and the I/O nodes and between the I/O nodes and the compute nodes. This is done using specialized daemons on the I/O nodes that accept application-launch instructions from the control system over the Ethernet network and pass the executable image to the compute nodes over the collective network. On the compute nodes, a different set of daemons copies the image into the ramdisk and launches the executable image using fork and exec.

Communications may occur between a compute node and external computers; between different compute nodes and, for example, a Web server running on one node and a database on another; or between compute nodes and a file system. The communication patterns can be handled in several ways. One option is to implement a complete TCP driver for the collective and torus networks and use

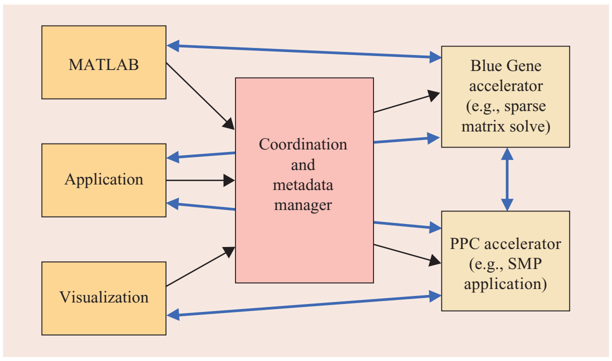

MATLAB
Application
Visualization
Coordination and metadata manager
Blue Gene accelerator (e.g., sparse matrix solve)
PPC accelerator (e.g., SMP application)

Figure 9

An architecture for system-level acceleration. Blue and black arrows indicate data and control paths, respectively. (PPC: PowerPC; SMP: symmetric multiprocessing.)

the I/O nodes as routers to external systems. This allows standard file system clients, such as NFS** (network file system) clients, directly on the compute nodes, solving the file I/O issue as well. Another option is to use specialized file system clients on the compute nodes that can trap file I/O requests and forward them over the collective network to the I/O nodes, where they will be served by the locally or the remotely mounted file system. In this model, requests for connections will also be forwarded to the I/O nodes, which will issue them on behalf of the compute nodes.

When running Linux on the compute nodes, a question exists as to how this will affect the performance of HPC applications and what can be done to mitigate any performance penalty. At least three important issues exist that must be addressed for high-performance Linux on compute nodes. First, noise is caused by daemons, and this can be ameliorated by synchronizing the daemons. Second, minimization of TLB misses can be accomplished by using very large pages. However, in a long-running Linux kernel, fragmentation can become an impediment to using very large pages, and careful coding must occur to allow very large pages after a Linux kernel has been running for a long time. Third, Linux must be programmed to be able to take advantage of the DMA hardware available on the BG/P system.

Utilizing the DMA device in the general case is challenging, because over time, after many processes have run, memory becomes fragmented, thereby making it difficult to obtain contiguous chunks for the DMA device. Currently, our approach is to simplify the problem and handle the common cases of few applications being run. Future work involves the examination of how to manage space for the DMA in general, without major modifications to Linux.

System-level acceleration

The cost of operating a Blue Gene system, especially in terms of power per FLOP, is significantly better than that of any other general-purpose scalable system available. To date, the Blue Gene system has been used to satisfy the capability needs of HPC customers. With the ever-increasing power concerns of data centers, the potential for the Blue Gene system to be an attractive offering for more generic computing cycles is great. A solution is needed, both in terms of system software stack support and a data manager that can make Blue Gene cycles available in a capacity manner as well. We have developed prototype architecture (depicted in **Figure 9**) that allows applications such as MATLAB running on a laptop, desktop, or other workstation to be offloaded to a Blue Gene system. A metadata server tracks the placement of data for particular applications. This information can then be used to run successive applications, such as a visualizer, on that data. While still in the early stages, this work provides an opportunity for allowing access to Blue Gene cycles to a much wider user base.

System software conclusion

In this "Software architecture overview" section, we have presented the system software on the BG/P system. Achieving the scalability and performance goals of Blue Gene software is a challenging and rewarding effort and requires applying the correct combination of discipline and innovation. The software solution available on the BG/P system will address wider needs of the community than those addressed by the previous generation while maintaining the performance that customers have come to expect from Blue Gene systems.

We also have presented two examples of software architecture and design that we are using to broaden the use of Blue Gene compute cycles. Work on both the Linux and the system-level acceleration is ongoing. When sufficient progress has been made, subsequent publications will report in detail on each of these areas.

BG/P application performance and scaling

Application enablement for petascale computation remains a challenging task and much work remains to develop petascale applications for the future. One of the key accomplishments of the BG/L project has been to provide clear proof that a broad spectrum of applications, spanning many different application domains, can be engineered to run efficiently on systems with hundreds of thousands of cores.

Of particular note are the applications ddcMD (domain decomposition and molecular dynamics) [3] and Qbox [4], which resulted in the winning of Gordon Bell Prizes in 2005 and 2006, respectively. DdcMD is a classical molecular dynamics program that has achieved more than 100 Tflops of sustained performance on the LLNL 64-rack BG/L system. Qbox carries out quantum electronic structure calculations and has achieved a sustained performance of more than 200 Tflops on the LLNL system.

Our key design goals for the BG/P system were to maintain and increase the strong application performance and scalability that the BG/L system has provided while simultaneously increasing the breadth of applications that the BG/P system supports. In this section, we provide some comparisons of application performance on the BG/L and BG/P systems, some examples of new applications enabled by the increased memory and SMP mode of the BG/P system, and finally a preliminary look at application scalability on the BG/P system.

A BG/L node has two cores, each capable of executing two fused floating multiply–add instructions per cycle, and runs at 700 MHz. This gives a BG/L node a peak performance of 5.6 Gflops [(0.7 GHz) × (4 FLOPs) × (2 cores)]. A BG/P node has four cores, each also capable of executing two fused floating multiply–add instructions per cycle, and runs at 850 MHz. This gives a BG/P node a peak performance of 13.6 Gflops [(0.85 GHz) × (4 FLOPs) × (4 cores)]. Thus, a BG/P node has a peak performance that is 2.43 times that of a BG/L node. Direct comparisons of BG/L and BG/P node performance are complicated because two different cache configurations are generally used on the BG/L system. One of these cache configurations is write-through mode with full recovery enabled for L1 cache parity errors. The second cache configuration is write-back mode with no recovery enabled for L1 cache parity errors. The first of these configurations is the normal execution mode on the BG/L system for jobs executing on large system partitions. The second configuration is often used for jobs running on smaller partitions because it can increase performance but at the cost of leaving an application vulnerable to soft errors. On the BG/P system, write-through is required to enable SMP mode, and the only cache configuration used is write-through mode with full recovery enabled. **Table 3** shows BG/P node performance relative to that of the BG/L system for a number of applications. These applications include MILC [MIMD (multiple-instruction stream, multiple-data stream) lattice computation] [18] (a quantum chromodynamics code), POP (Parallel Ocean Program) [19] (an ocean circulation code), UMT2k (Unstructured Mesh Transport) (see Reference [20]) (a photon transport code), sPPM (Simplified Piecewise Parabolic Method) (see Reference [20]) (which models 3D gas dynamics), SPHOT (see Reference [20]) (a 2D photon transport code), and CPMD [21] (which implements the Car–Parrinello molecular dynamics algorithm for materials modeling). We see that a BG/P node systematically outperforms a BG/L node, running in write-through mode with full recovery, by at least a factor of 2.43, which is the ratio of BG/P to BG/L peak performance. Improvements over the 2.43 factor are directly attributable to the changes in the memory system from the BG/L to the BG/P system. When comparing a BG/P node to a BG/L node running in write-back mode with no L1 recovery, we see performance improvements of 2.0 or more in all cases.

Table 3 Performance of a BG/P node relative to a BG/L node for a selection of applications. The BG/P system operates only in write-through (WT) mode with full L1-recovery enabled. *BG/L-WT* denotes a similarly configured BG/L node. *BG/L-WB* denotes a BG/L node configured for cache write-back without L1 recovery. This mode gives additional performance on a BG/L node but leaves the node exposed to soft errors.

	BG/P compared to BG/L-WT	*BG/P compared to BG/L-WB*
MILC	2.46	2.08
POP	2.61	2.01
UMT2k	2.81	2.28
sPPM	3.10	2.28
SPHOT	2.44	2.30
CPMD	2.40	2.27

Applications on the BG/P system can use a variety of run modes. These run modes include the following:

1. *SMP mode*: one MPI process per node, with one, two, three, or four threads per process.
2. *Dual mode*: two MPI processes per node, with one or two threads per process.
3. *Quad mode*: four MPI processes per node, with one thread per process.

In SMP or dual mode, OpenMP is the normal application method used to enable multiple threads. OpenMP is fully supported by the IBM XL compiler suite. **Table 4** shows some examples of performance achieved in dual and SMP modes using OpenMP to enable additional threads. Perfect speedups for OpenMP-enabled applications would correspond to achieving a factor-of-2 performance increase in dual mode and a factor-of-4 performance increase in SMP mode. Usually, however, this perfect speedup is not possible because most codes contain non-parallelizable elements. However, the table clearly shows significant performance improvements for all the applications listed.

The SMP and dual modes provide the added benefit of the ability to run applications that require large memory footprints per node by reducing the active processes per node. A BG/P node has 2 GB of memory. MPI applications that require less than 0.5 GB of memory per process can be executed in quad mode and take

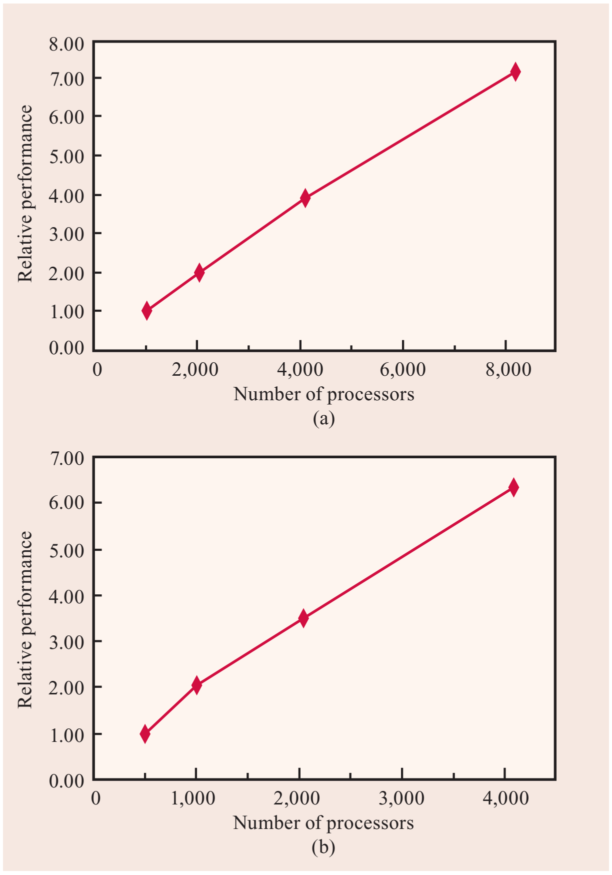


Figure 10

Scalability of (a) the LAMMPS application and (b) the HYCOM application. This figure is a "strong scaling plot" that shows the relative performance of the application for the same problem, as distributed across different numbers of BG/P processors.

advantage of all four cores by running one process on each core of the BG/P node. MPI-only applications that require more than 0.5 GB but less than 1 GB of memory per processor can be executed in dual mode. In this mode, only two processes are active, and peak performance is reduced by a factor of 2. Finally, MPI applications that require more than 1 GB of memory per process but less than 2 GB of memory per node can be run in SMP mode with only a single process active. This reduces peak performance by a factor of 4. It is possible to regain the lost performance by modifying MPI-only codes to use OpenMP. In this mode, there is one process per node, but multiple threads within that process allow utilization of some or all of the cores on the node. However, even before enabling OpenMP, these new BG/P run modes broaden the applications range of the BG/P system significantly beyond that available to the BG/L system. Two specific applications that illustrate this point are

Table 4 OpenMP scalability of a BG/P node for a selection of applications. Numbers are normalized to single-thread performance. The sPPM code superscales because the four threads accidentally cooperate to bring data to the caches from main memory. The SPHOT application shows perfect scaling for its main code segment. However, SPHOT has a serial component outside its main code segment that does not take advantage of OpenMP. Blank entries are not yet measured.

	Two OpenMP threads; dual mode	*Four OpenMP threads; SMP mode*
UMT2k	1.87	3.57
sPPM	—	4.30
SPHOT (main code)	—	4.00
CPMD	1.91	3.60

CTH [22] and AVUS [23]. CTH is an application from Sandia National Laboratory for modeling complex multidimensional, multimaterial problems. AVUS (Air Vehicles Unstructured Solver) is a computational fluid dynamics code. Running CTH in SMP mode with only one process active per node and AVUS in dual mode with only two processes active per node allowed the BG/P team to complete a submission to the TI 08 benchmark process of the U.S. government. Such a submission would not have been possible on the BG/L system because many of the required applications and workloads simply did not fit in the BG/L memory footprint.

As a final comment on BG/P application performance, note that in the BG/P network design, we paid very careful attention to scale network bandwidth with the increase in clock speed and number of cores per node. The BG/P system provides the same bytes-to-flops ratio as the BG/L system. This contributes to maintaining the application scaling properties that were demonstrated on the BG/L system, both in weak scaling (where performance per node is maintained as the problem size grows with increasing partition size) and in strong scaling (where performance increases with partition size for a constant size problem). As examples, we show strong scaling curves for two applications on the BG/P system. **Figure 10(a)** shows strong scaling for LAMMPS (Large-scale Atomic/Molecular Massively Parallel Simulator) [24], a molecular dynamics code, and **Figure 10(b)** shows strong scaling for the ocean modeling code HYCOM (HYbrid Coordinate Ocean Model) [25].

Conclusions

The BG/P system is the second generation of the Blue Gene family of supercomputers. It enables petascale computing within the power and floor-space budgets of existing data centers, and with high standards of RAS. With respect to its predecessor, the BG/L system, significant advances have been made in system architecture and system software, leading to a simpler programming model and increased usability and flexibility, while maintaining excellent performance and scalability. Consequently, a broader spectrum of applications will be able to run on the BG/P platform. Given its high power efficiency and reliability, resulting in a low total cost of ownership, the BG/P system is a compelling platform for HPC.

Acknowledgments

This work has benefited from the cooperation of many individuals at IBM Research (Yorktown Heights, New York), IBM Global Engineering Solutions (Rochester, Minnesota), and IBM Systems and Technology Group (East Fishkill, New York, Burlington, Vermont, and Raleigh, North Carolina). The Blue Gene/P project has been supported and partially funded by Argonne National Laboratory and the Lawrence Livermore National Laboratory on behalf of the U.S. Department of Energy, under Lawrence Livermore National Laboratory subcontract no. B554331.

*Trademark, service mark, or registered trademark of International Business Machines Corporation in the United States, other countries, or both.

**Trademark, service mark, or registered trademark of TOP500.org, Advanced Micro Devices, Inc., Cray, Inc., Silicon Graphics, Inc., Dell Computer Corporation, Intel Corporation, Institute of Electrical and Electronic Engineers (IEEE), Linus Torvalds, OpenMP Architecture Review Board, MathWorks, Inc., Sun Microsystems, Inc., or Novell, Inc., in the United States, other countries, or both.

References

1. "Blue Gene," *IBM J. Res & Dev*. **49**, No. 2/3 (2005), entire issue.
2. "Applications of Massively Parallel Systems," *IBM J. Res & Dev*. **52**, No. 1/2 (2008, this issue), entire issue.
3. F. H. Streitz, J. N. Glosli, M. V. Patel, B. Chan, R. K. Yates, and B. R. de Supinski, "100+ TFlop Solidification Simulations on Blue Gene/L," *Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, November 2005; 2005 Gordon Bell Prize winner; see *http://sc05.supercomputing.org/schedule/pdf/pap307.pdf*.
4. F. Gygi, E. W. Draeger, M. Schulz, B. R. de Supinski, J. A. Gunnels, and V. Austel, J. C. Sexton, et al., "Large-Scale Structure Calculations of High-Z Metals on the Blue Gene/L Platform," *Proceedings of the ACM/IEEE 2006 Conference on Supercomputing*, November 2006; 2006 Gordon Bell Prize for Peak Performance winner; see *http://sc06.supercomputing.org/schedule/pdf/gb104.pdf*.
5. P. Vranas, G. Bhanot, M. Blumrich, D. Chen, A. Gara, P. Heidelberger, V. Salapura, and J. C. Sexton, "The Blue Gene/L Supercomputer and Quantum Chromodynamics," *Proceedings of the ACM/IEEE 2006 Conference on Supercomputing*, November 2006; 2006 Gordon Bell Prize finalist; *http://sc06.supercomputing.org/schedule/pdf/gb110.pdf*.
6. J. N. Glosli, K. J. Caspersen, J. A. Gunnels, D. F. Richards, R. E. Rudd, and F. H. Streitz, "Extending Stability Beyond CPU Millennium: A Micron-Scale Atomistic Simulation of Kelvin–Helmholtz Instability," *Proceedings of the 2007 ACM/IEEE Conference on Supercomputing*, 2007; Gordon Bell Prize Winner; see *http://sc07.supercomputing.org/schedule/pdf/gb109.pdf*.
7. TOP500 Supercomputer Sites; see *http://www.top500.org*.
8. HPC Challenge Awards Competition; see *http://www.hpcchallenge.org/*.
9. The Green500 List; see *http://www.green500.org/*.
10. IBM Corporation,"ASICs"; see *http://www.ibm.com/chips/asics/products/stdcell.html*.
11. J. Castanos, G. Chiu, P. Coteus, A. Gara, M. Gupta, and J. Moreira, "Lilliputians of Supercomputing Have Arrived!" *CTWatch Quart*. **1**, No. 3, 21–26 (2005).
12. J. E. Moreira, G. Almási, C. Archer, R. Bellofatto, P. Bergner, J. R. Brunheroto, M. Brutman, et al., "Blue Gene/L Programming and Operating Environment," *IBM J. Res & Dev*. **49**, No. 2/3, 367–376 (2005).
13. IBM Corporation,"Exploiting the Dual Floating Point Units in Blue Gene/L," white paper (2006); see *http://www.ibm.com/support/docview.wss?uid=swg27007511*.
14. A. Martin and M. Mendell, "Blue Gene Compilers and Optimization," paper presented at Third BG/L Systems Software and Applications Workshop, Tokyo, April 19–20, 2006; see *http://www.cbrc.jp/symposium/bg2006/PDF/Martin.pdf*.
15. A. E. Eichenberger, P. Wu, and K. O'Bien, "Vectorization for SIMD Architectures with Alignment Constraints," *Proceedings of the ACM SIGPLAN 2004 Conference on Programming Language Design and Implementation*, 2004, pp. 82–93; see *http://domino.research.ibm.com/comm/research_projects.nsf/pages/cellcompiler.refs.html/$FILE/paper-eichen-pldi04.pdf*.
16. P. Wu, A. E. Eichenberger, A. Wang, and P. Zhao, "An Integrated Simdization Framework Using Virtual Vectors," *Proceedings of the 19th Annual International Conference on Supercomputing*, 2005, pp. 179–178; see *http://domino.research.ibm.com/comm/research_projects.nsf/pages/cellcompiler.refs.html/$FILE/paper-wu-ics05.pdf*.
17. B. G. Fitch, A. Rayshubskiy, M. Eleftheriou, T. J. C. Ward, M. Giampapa, M. C. Pitman, and R. S. Germain, "Blue Matter: Approaching the Limits Of Concurrency for Classical Molecular Dynamics," *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, 2006; see *http://doi.acm.org/10.1145/1188455.1188547*.
18. MIMD Lattice Computation (MILC) Collaboration; see *http://www.physics.indiana.edu/~sg/milc.html*.
19. Los Alamos National Laboratory, Climate, Ocean, and Sea Ice Modeling: POP; see *http://climate.lanl.gov/Models/POP/*.
20. Advanced Simulation and Computing; see *http://asc.llnl//gov*.
21. CPMD Consortium Page; see *http://www.cpmd.org/*.
22. Aeronautical Systems Center/Major Shared Resource Center, CTH; see *http://www.asc.hpc.mil/software/info/cth/*.
23. C. Hoke, V. Burnley, and G. Schwabacher, "Aerodynamic Analysis of Complex Missile Configurations Using AVUS (Air Vehicles Unstructured Solver)," *22nd Applied Aerodynamics Conference and Exhibit*, August 2004; see *http://www.aiaa.org/content.cfm?pageid=406&gTable=Paper&gID=23213*.
24. LAMMPS Molecular Dynamics Simulator; see *http://lammps.sandia.gov/*.
25. HYCOM Consortium; see *http://hycom.rsmas.miami.edu/*.

Received July 3, 2007; accepted for publication July 13, 2007; Internet publication January 18, 2008

IBM Blue Gene team *(gchiu@us.ibm.com)*. The Blue Gene team spans a wide range of technical disciplines and organizations within IBM. Blue Gene team members are listed below.

IBM Thomas J. Watson Research Center
Yorktown Heights, New York 10598: Gheorghe Almasi, Sameh Asaad, Ralph E. Bellofatto, H. Randall Bickford, Matthias A. Blumrich, Bernard Brezzo, Arthur A. Bright, Jose R. Brunheroto, Jose G. Castanos, Dong Chen, George L.-T. Chiu, Paul W. Coteus, Marc B. Dombrowa, Gabor Dozsa, Alexandre E. Eichenberger, Alan Gara, Mark E. Giampapa, Frank P. Giordano, John A. Gunnels, Shawn A. Hall, Ruud A. Haring, Philip Heidelberger, Dirk Hoenicke, Michael Kochte, Gerard V. Kopcsay, Sameer Kumar, Alphonso P. Lanzetta, Derek Lieber, Ben J. Nathanson, Kathryn O'Brien, Alda S. Ohmacht, Martin Ohmacht, Rick A. Rand, Valentina Salapura, James C. Sexton, Burkhard D. Steinmacher-Burow, Craig Stunkel, Krishnan Sugavanam, Richard A. Swetz, Todd Takken, Shurong Tian, Barry M. Trager, Robert B. Tremaine, Pavlos Vranas, Robert E. Walkup, Michael E. Wazlowski, Shmuel Winograd, Robert W. Wisniewski, Peng Wu.

IBM Global Engineering Solutions
Rochester, Minnesota 55901: Dennis R. Busche, Steven M. Douskey, Matthew R. Ellavsky, William T. Flynn, Philip R. Germann, Michael J. Hamilton, Lance Hehenberger, Brian J. Hruby, Mark J. Jeanson, Fariba Kasemkhani, Robert F. Lembach, Thomas A. Liebsch, Kelly C. Lyndgaard, Robert W. Lytle, James A. Marcella, Christopher M. Marroquin, Curt H. Mathiowetz, Michael D. Maurice, Eldon Nelson, Dennis M. Rickert, Glenn W. Sellers, John E. Sheets, Scott A. Strissel, Charles D. Wait, Bruce B. Winter, Cory J. Wood, Laura M. Zumbrunnen.
Bangalore, India: Meera Rangarajan.

IBM Systems and Technology Group
Rochester, Minnesota 55901: Paul V. Allen, Charles J. Archer, Michael Blocksome, Thomas A. Budnik, Sam D. Ellis, Michael P. Good, Thomas M. Gooding, Todd A. Inglett, Kerry T. Kaliszewski, Brant L. Knudson, Cory Lappi, Glenn S. Leckband, Susan Lee, Mark G. Megerian, Samuel J. Miller, Michael B. Mundy, Roy G. Musselman, Thomas E. Musta, Michael T. Nelson, Carl F. Obert, James L. Van Oosten, John P. Orbeck, Jeffrey J. Parker, Ruth J. Poole, Harold L. Rodakowski, Don D. Reed, Jeffrey J. Scheel, Faith W Sell, Richard M. Shok, Karl M. Solie, Gordon G. Stewart, William M. Stockdell, Andrew T. Tauferner, John Thomas.
Research Triangle Park, North Carolina 27709: Robert H. Sharrar.
Burlington, Vermont: Steven Schwartz.
Waltham, Massachusetts 02451: David L. Satterfield.

IBM Integrated Technology Delivery
Rochester, Minnesota 55901: Jeffery D. Chauvin.

IBM Haifa Research Laboratory
Mount Carmel, Haifa 31905, Israel: Edi Shmueli.

IBM Software Group
Rational, Markham, Ontario, Canada: Roch G. Archambault, Allan R. Martin, Mark P. Mendell, Guansong Zhang, Peng P. Zhao.
Bangalore, India: Indra Mani, Rohini Nair.

IBM Sales and Distribution
San Diego, California 92122: Rajiv D. Bendale.

IBM Zurich Research Laboratory
Zurich, Switzerland: Alessandro Curioni.

IBM India Research Laboratory
New Delhi, India: Yogish Sabharwal.

IBM Yamato Laboratory
Yamato, Japan: Jun Doi, Yasushi Negishi.