Blue Gene/L torus interconnection network

N. R. Adiga
M. A. Blumrich
D. Chen
P. Coteus
A. Gara
M. E. Giampapa
P. Heidelberger
S. Singh
B. D. Steinmacher-Burow
T. Takken
M. Tsao
P. Vranas

The main interconnect of the massively parallel Blue Gene®/L is a three-dimensional torus network with dynamic virtual cut-through routing. This paper describes both the architecture and the microarchitecture of the torus and a network performance simulator. Both simulation results and hardware measurements are presented.

Introduction

One of the most important features of a massively parallel supercomputer is the network that connects the processors together and allows the machine to operate as a large coherent entity. In Blue Gene*/L (BG/L), the primary network for point-to-point messaging is a three-dimensional (3D) torus network [1].

Nodes are arranged in a 3D cubic grid in which each node is connected to its six nearest neighbors with high-speed dedicated links. A torus was chosen because it provides high-bandwidth nearest-neighbor connectivity while also allowing construction of a network without edges. This yields a cost-effective interconnect that is scalable and directly applicable to many scientific and data-intensive problems. A torus can be constructed without long cables by connecting each rack with its next-to-nearest neighbor along each x, y, or z direction and then wrapping back in a similar fashion. For example, if there are six racks in one dimension, the racks are cabled in the order of 1, 3, 5, 6, 4, 2, 1. Also, because the network switch is integrated into the same chip that does the computing, no separate switch and adapter cards are required, as is typical in other supercomputers. Previous supercomputers, such as the Cray T3E [2], have also used a torus network.

The torus network uses both dynamic (adaptive) and deterministic routing with virtual buffering and cut-through capability [3]. The messaging is based on hardware packets of variable length. A robust error-recovery mechanism that involves an acknowledgment and retransmission protocol is implemented across the physical links that connect two BG/L compute nodes.

These features provide excellent communication characteristics for a large variety of communication patterns found in most scientific applications. This paper is organized as follows: The architecture and microarchitecture are presented, followed by an overview of the custom-designed simulator that was used to make many of the design decisions and provided expected-performance figures. Sample performance studies done on the simulator are presented, as are measurements on the fully functioning hardware, which are compared with the expected-performance numbers from the simulator.

Torus network

In this section, the torus network architecture and microarchitecture are presented. The torus network router directs variable-size packets, each $n \times 32$ bytes, where $n = 1$ to 8 "chunks." Messages, such as those conforming to the Message Passing Interface Standard (MPI), may consist of many packets that are constructed, sent, and received by software running on one or both associated BG/L processors. The first eight bytes of each packet contain link-level protocol information (e.g., sequence number); routing information, including destination; virtual channel and size; and a byte-wide cyclic redundancy check (CRC) that detects header data corruption during transmission.

In addition, a 24-bit CRC is appended to each packet, along with a one-byte valid indicator. The valid indicator is necessary, since packets can be forwarded before being entirely received. This CRC permits checking of each packet as it is sent over each link. A time-out mechanism is used for retransmission of corrupted packets. Use of the eight-bit packet header CRC is an optimization that permits early detection of packet header errors because the header CRC is included in the full packet CRC. The error detection and recovery protocol is similar to that used in IBM High Performance Switch

0018-8646/05/$5.00 © 2005 IBM

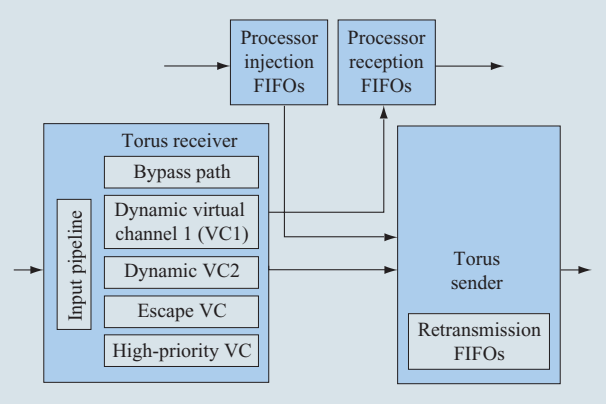

Figure 1

General structure of the torus router. There are six interconnected receiver/sender pairs (only one pair is shown here).

(HPS) interconnection networks and in the HIPPI-6400 standard [4].

As in the BG/L collective network, each sender (receiver) also maintains a 32-bit link-level CRC that includes all data packets sent (received) successfully on the link. At the end of a run or at a checkpoint, each sender link-level CRC matches its corresponding receiver link-level CRC, thereby providing a mechanism to detect 24-bit packet-level CRC escapes. If not, the results of the run are not to be trusted, and the job should be restarted from the last checkpoint. Since BG/L is massively parallel, an efficient, simple, and local error-recovery mechanism (as opposed to an end-to-end message retransmission approach) was paramount.

For routing, the header includes six "hint" bits that indicate the directions in which the packet may be routed. For example, hint bits of 100100 mean that the packet can be routed in the $x+$ and $y-$ directions. Either the $x+$ or the $x-$ hint bits, but not both, may be set. If no x hops are required, the x hint bits are set to 0. Each node maintains a set of software-configurable registers that control the torus functions. For example, a set of registers contains the coordinates of its neighbors. Hint bits are set to 0 when a packet leaves a node in a direction such that it will arrive at its destination in that dimension, as determined by the neighbor coordinate registers. These hint bits appear early in the header so that arbitration may be efficiently pipelined. The hint bits can be initialized by either software or hardware; if done by hardware, a set of two registers per dimension is used to determine the appropriate directions. Once the hint bits are set, they cannot be subsequently changed by software. These registers can be configured to provide minimal hop routing. The routing is accomplished entirely by examining the hint bits and virtual channels; i.e., there are no routing tables. Packets may be either dynamically or deterministically dimension-ordered (xyz) routed. That is, they can follow a path of least congestion based on other traffic, or they can be routed on a fixed path. Besides point-to-point packets, a bit in the header may be set that causes a packet to be broadcast down any Cartesian dimension and deposited at each node. Hardware broadcasting in more than one dimension would have greatly complicated the logic, although deadlocks could have been prevented by broadcasting in the same xyz order, as described in [5]. The hardware does not have the capability to route around "dead" nodes or links. However, software can set the hint bits appropriately so that such nodes are avoided; full connectivity can be maintained when there are up to three noncolinear faulty nodes.

The torus logic consists of three major units—a processor interface, a send unit, and a receive unit (**Figure 1**). The processor interface consists of network injection and reception FIFOs (queues in which access is according to the *first in*, *first out* rule). Access to these FIFOs is via the double floating-point unit (FPU) registers; i.e., data is loaded into the FIFOs via 128-bit memory-mapped stores from a pair of FPU registers, and data is read from the FIFOs via 128-bit loads to the FPU registers. There are a total of eight injection FIFOs organized into two groups: two high-priority (for internode operating system messages) and six normal-priority FIFOs, which are sufficient for nearest-neighbor connectivity. Packets in all FIFOs can go out in any direction.

On the reception side, there are again two groups of FIFOs. Each group contains seven FIFOs, one high-priority and one dedicated to each of the incoming directions. More specifically, there is a dedicated bus between each receiver and its corresponding reception FIFO. As in mechanisms in the BG/L collective network, there is a checksum associated with each injection FIFO; these can be used for fault-isolation and debugging purposes to see whether a node sends the same data onto the network in two different runs. Watermarks can also be set by software for the injection and reception FIFOs so that interrupts fire when the FIFO contents cross the corresponding threshold. For storage, all torus FIFOs use static random access memory chips (SRAMs) protected by error checking and correction (ECC), and all internal data paths are checked for parity. There is a total of 58 KB of SRAMs.

The datapath for each of the six receivers, as shown in Figure 1, is composed of an eight-stage input pipeline, four virtual channels (VCs), and a bypass channel. The input pipeline processes the packet header on the fly. Multiple VCs help reduce head-of-line blocking [6], but—since mesh networks, including tori with dynamic routing, can deadlock—appropriate additional escape

VCs are provided [7, 8]. The approach we use to this problem is similar to the recent, elegant solution proposed in [9, 10]: the *bubble escape* VC. BG/L has two dynamic VCs, one a bubble escape VC that can be used both for deadlock prevention and deterministic routing, and the other a high-priority bubble VC. Dynamic packets can enter the bubble escape VC only if no valid dynamic VCs are available. Each VC has 1 KB of buffering, enough for four full-sized packets. The bypass channel allows packets, under appropriate circumstances, to flow through a node without entering the VC buffers. This places stringent timing requirements both on the processing of the header routing information by the input pipeline and on the routing arbitration by the receiver arbiters.

A token flow-control algorithm is used to prevent the VC buffers from overflowing. Each token represents a 32-byte chunk. For simplicity in the arbiters, a VC is marked as unavailable unless space is available to handle a full-sized packet. However, token counts for packets on dynamic VCs are incremented and decremented according to the size of the packet. The bubble rules, as outlined in [9, 10], include the rule that tokens for one full-sized packet are required for a packet already on the bubble VC to advance. Tokens for two full-sized packets are required for a packet to enter the bubble VC upon injection, upon a turn to a new direction, or when a dynamic VC packet enters the bubble. This rule ensures deadlock-free operation, since buffer space for one packet is always available after an insertion, and thus some packet can always move eventually. However, when our simulator deadlocked using this rule, we discovered that the rule is incorrect for variable-sized packets. With this rule, the remaining free space for one full-sized packet can become fragmented, resulting in a potential deadlock. To prevent this, the bubble rules are modified so that each packet on the bubble is accounted for as if it were a full-sized packet. BG/L uses dimension-ordered routing on the bubble VC.

Eight-byte acknowledgment (*ack-only*) packets or combined token–acknowledgment (*token–ack*) packets are returned either when packets are successfully received or when space has freed up in a VC. Acknowledgments permit the torus send units to delete packets from their retransmission FIFOs, which are used in the error-recovery protocol. The send units also arbitrate between requests from the receiver and injection units.

Because of the density of packaging and ASIC-module pin constraints, each of the 12 BG/L torus links (six in, six out) is bit-serial. The torus is internally clocked at a quarter of the rate of the processor. At the target 700-MHz clock rate, each torus link is 1.4 Gb/s, which gives 175 MB/s. There are enough internal buses so that each of the six outgoing and six incoming links can be busy simultaneously; thus, each node can be sending and receiving 1.05 GB/s. In addition, there are two transfer buses (paths) coming out of each receiver that connect with the senders. As a result, a single receiver can have up to four simultaneous transfers, e.g., one to its normal processor reception FIFO, one to the high-priority processor reception FIFO, and two to two different senders. The torus network input and output are byte-wide and are serialized and deserialized by the high-speed signaling units outside the torus module. The torus internal buses are all 11 bits wide (eight data, one parity, two control).

Arbitration is distributed and pipelined, but occurs in three basic phases. It generalizes an approach described in [11] and represents tradeoffs among complexity, performance, and the ability to meet timing constraints. In the first phase, a decision is made for each packet at the head of the injection or VC FIFOs about the direction in which it should preferably move and which VC it should use. There is only one valid choice for deterministically routed packets, but there may be many choices for dynamically routed packets. The preferred direction and VC are selected using a modified join-the-shortest-queue (JSQ) algorithm, as follows. The senders provide the receivers and injection FIFOs with a bit that indicates both link and token availability for each VC in each direction. This bit vector is ANDed with a bit vector of possible moves constructed from the packet hint bits and VC. This defines the set of possible and available arbitration requests. In addition, for each VC the sender provides two bits that indicate one of four ranges of available downstream tokens. Of all the possible and available dynamic direction and VC pairs, the packet selects the one with the most available downstream tokens. Ties are randomly broken. If no combination of dynamic direction and VC is available, the packet requests its bubble escape direction and VC pair (if available). If that is also unavailable, no arbitration request is made for the packet. This is a somewhat simplified description, since data bus availability must also be taken into account. In addition, when a packet reaches its destination, the "direction" requested is simply the corresponding reception FIFO.

In the second phase, arbitration is required to determine which of the requesting packets in the receiver wins the right to request, since each receiver has multiple VC FIFOs in addition to the bypass channel. If a high-priority packet is requesting, it wins. Barring that, a modified *serve the longest queue* (SLQ) is used, based on two-bit (four ranges) FIFO fullness indicators; i.e., the packet from the VC that is fullest as measured to within the two bits of granularity wins. However, this algorithm cannot always be used, because doing so may completely block out a VC. Therefore, a certain (programmable)

fraction of the arbitration cycles are designated SLQ cycles, in which the above algorithm is used, while the remaining cycles select the winner randomly. A packet on the bypass channel always receives the lowest priority unless it is a high-priority packet.

In the third phase, the requests from the receivers and injection FIFOs are presented to the senders. Note that on a given cycle a receiver presents at most one request to the senders; thus, each sender arbiter can operate independently. The senders give highest priority to token–ack or ack-only packets, if any. Barring that, the senders tend to favor packets already in the network and use a similar modified SLQ algorithm in which there are SLQ cycles and random cycles. In particular, a certain programmable fraction of cycles (typically 100%) give priority to packets already in the network, unless the only high-priority packet requesting is in an injection FIFO. On such cycles, the modified SLQ algorithm is used. Higher priority can be given to injection packets by lowering the above in-network priority fraction. On cycles in which injection packets receive priority, barring in-network high-priority packets, the modified SLQ algorithm is also used.

This design is complex; the torus unit requires slightly more area (10.2 mm^2 in CU-11) than a PowerPC* 440 processor core and its associated double FPU. The entire torus arbitration is carried out by 44 arbiters on the receiver/processor-injection side and six arbiters on the sender side. To manage this level of complexity, the design is highly modular. For example, a single arbiter module was designed that could serve all VC FIFOs and processor-injection and bypass channels. This module was then instantiated 38 times.

Simulator overview

Given the complexity and scale of the BG/L interconnection network, having an accurate performance simulator was essential during the design phase of the project. Because of the potential size of such a model, simulation speed was a significant concern. Thus, we selected a proven shared-memory parallel simulation approach. In particular, parallel simulation on shared-memory machines has been shown to be very effective in simulating interconnection networks (see [12], for example), whereas success with message-passing parallel interconnection network simulators is more difficult to achieve (see [13], for example). We also recognized the difficulties in developing an execution-driven simulator, such as that in [14], for a system with up to 64K nodes. We therefore decided upon a simulator that would be driven primarily by application pseudocodes, in which message-passing calls could be easily passed to the simulator; such calls include the time since the last call (the execution burst time), the destination and size of the message, and so on. This pseudocode included a subset of the message passing interface (MPI) point-to-point messaging calls as a workload driver for the simulator. We also extended the IBM unified trace environment trace capture utility that runs on IBM SP* supercomputer machines and were able to use such traces as simulator input for up to several hundreds of nodes.

The basic unit of simulation time is a *network cycle*, defined to be the time it takes to transfer one byte. Because the BG/L is organized around 512-node ($8\times8\times8$) midplanes, the simulator partitions its work on a midplane basis; i.e., all nodes on the same midplane are simulated by the same processor thread, and midplanes are assigned to threads in as even a manner as possible.

Because different threads are concurrently executing, the local simulation clocks of the threads must be properly synchronized. To deal with this problem, we use a simple but effective "conservative" parallel simulation protocol known as YAWNS (yet another windowing network simulator) [15]. In particular, we take advantage of the fact that the minimum transit time between midplanes is known and is at least some constant $w \geq 1$ cycles. In this protocol, time "windows" of length w are simulated in parallel by each of the threads. At the end of each window, a synchronization phase takes place in which each thread picks up the information about packets that will arrive in future windows. If $w = 1$, this protocol requires a barrier synchronization every cycle. On BG/L, however, the minimum inter-midplane delay is approximately $w = 10$ network cycles. When a large number of BG/L nodes are being simulated, each processor executes many events during a window, i.e., between barriers; thus, the simulator should obtain good speedups.

The simulator runs on a 16-way IBM POWER3+* 375-MHz symmetric multiprocessor (SMP) node with 64 GB of memory. The model of the torus hardware contains close to 100 resources per node (links, VC token counters, buses, FIFOs, etc.), so that a full 64K-node system can be thought of as a large queuing network with approximately six million resources. The model consumes a large amount of memory and runs slowly; a 32K-node simulation of a fully loaded network advances at about 0.25 microseconds of BG/L time per second of wall clock time. However, it obtains excellent speedup—typically, more than 12 on 16 processors—and sometimes achieves superlinear speedup because of the private 8-MB L3 caches on the SMP and the smaller per-node memory footprint of the parallel simulator.

The model, which was written before the hardware design in Very high-speed integrated circuit Hardware Description Language (VHDL) was implemented, is thought to be a quite accurate representation of the BG/L

hardware, although a number of simplifications were made. For example, in BG/L, the arbitration is pipelined and occurs over several cycles. In the simulator, this is modeled as a delay of several cycles followed by presentation of the arbitration request. Because the simulator focuses on what happens once packets are inside the network, the assumptions that the injection FIFOs were of infinite size and that packets were placed in these FIFOs as early as possible, rather than as space became available in the FIFOs, are both gross simplifications. This has little effect on either network response time or throughput measurements during the middle of a run, but it can affect the dynamics, particularly near the end of runs. Also, the simulator did not model the error-recovery protocol; i.e., no link errors were simulated, and the ack-only packets that are occasionally sent if a link is idle for a long time were not modeled. However, the arbitration algorithms and token flow control were modeled to a high level of detail.

Sample simulation performance studies

In this section, we present some examples of use of the simulator to study design tradeoffs in BG/L. The studies presented are illustrative and sometimes use assumptions and corresponding parameters about the system that do not reflect the final BG/L design.

Figure 2 shows the response time for various 32K-node BG/L configurations when the workload driver generates packets for random destinations and the packet generation rate is low enough that the average link utilization is less than 1. This figure compares deterministic routing to dynamic routing with one or more dynamic VCs and one or more buses (paths) connecting receivers to senders. Random arbitration rules simpler than SLQ and JSQ were used, and the plot was generated early in our studies when the target link bandwidth was 350 MB/s. (The 350-MB/s assumption essentially affects the results by only a rescaling of the y-axis.) The figure shows the clear benefit of dynamic over deterministic routing. It also shows that there is little benefit in increasing the number of dynamic VCs unless the number of paths is also increased. Finally, it shows only marginal benefit in going from a two-VC, two-path configuration to a four-VC, four-path configuration.

Throughput under nonuniform traffic

Figure 3 plots the throughput as a function of time for a 4K-node BG/L system under a highly nonuniform traffic pattern. In this pattern, the destinations of 25% of the packets are chosen randomly within a small "hot" contiguous submesh region consisting of 12.5% of the machine. The destinations for the remaining 75% of the packets were chosen uniformly over the entire machine. Again, random arbitration and a 350-MB/s link speed were used. The figure reflects three different buffer sizes for the VC FIFOs: 2 KB, 1 KB, and 512 B.

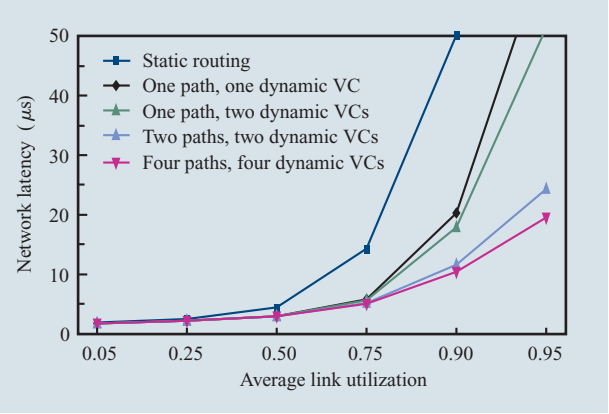

Figure 2

Sample response time in light traffic. [32K-node BG/L ($32 \times 32 \times 32$) under random traffic pattern. 256-B packets. 4 KB of buffers per link (+1 KB for escape). Early assumptions were used (see text).]

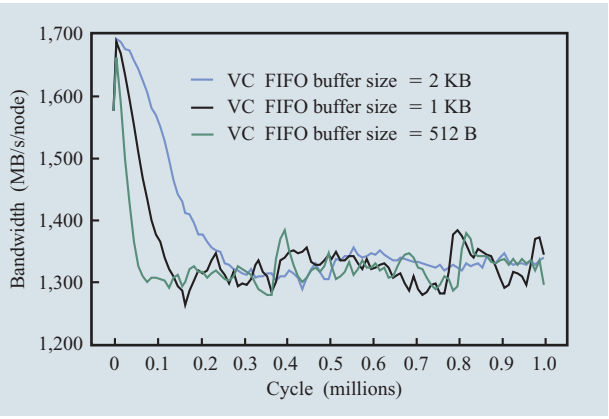

Figure 3

Throughput under a highly nonuniform load. [4K-node ($16 \times 16 \times 16$) BG/L under hot-region traffic (25% of traffic to 12.5% of machine); two dynamic VCs, two paths, bubble escape. Early assumptions were used (see text).]

At the beginning of the run, throughput (as measured over 10K-cycle intervals) increases as packets enter the network, but then declines as the buffers fill up. Eventually, the throughput levels off at a value that is approximately equal for the three buffer sizes. The decline happens more quickly for smaller buffer sizes. It is worth noting that the steady-state throughput is close to the peak possible throughput for this workload; the throughput is limited by the number of links entering

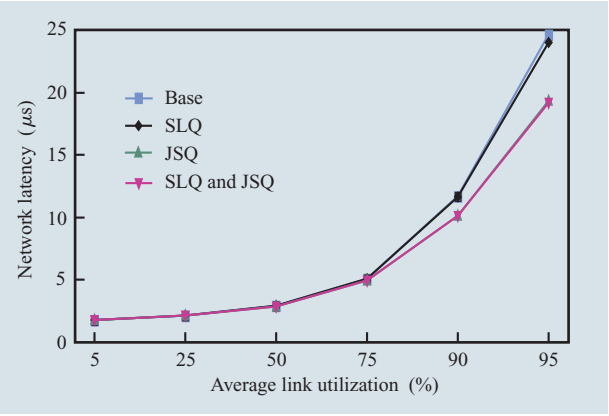

Figure 4

Response time in light traffic for different arbitration policies. [32K-node BG/L under random traffic pattern; 256-B packets, 4 KB of buffers per link (+1 KB for escape). Early assumptions were used (see text).]

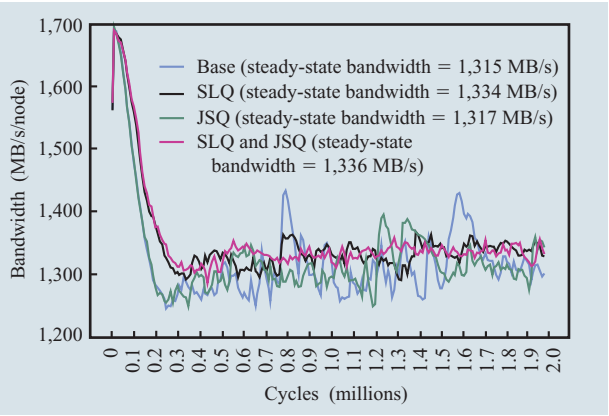

Figure 5

Throughput under nonuniform traffic for different arbitration policies. [4K-node BG/L under hot-region traffic (25% of traffic to 12.5% of machine); two dynamic 2-KB VCs, two paths, 2-KB bubble escape. Early assumptions were used (see text).]

the hot region. Measurements during the simulations indicated that those links generally have a mean utilization of around 95%.

Arbitration policies

Figure 4 shows the response time for the light-traffic, random-destination workload on a 32K-node BG/L system using different arbitration policies. The "base" policy is the above-mentioned random policy. In light traffic, SLQ provides little benefit (since queues are not that full), but JSQ does reduce response time in moderate

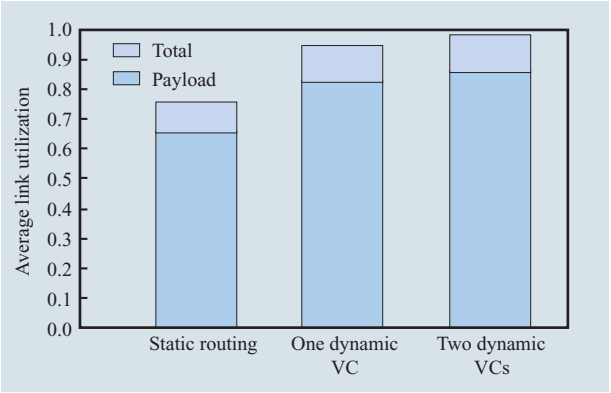

Figure 6

Average link utilization during alltoall on a 32K-node (32×32×32) BG/L with equal total buffer sizes (3 KB for nonpriority.)

traffic; at 95% link utilization, the average response time is reduced by about 20%.

The throughput for a 4K-node BG/L under the hot-region model for the different arbitration policies is shown in **Figure 5**. While the throughputs of all policies stabilize near the same value, the decline is slowest for the SLQ policy (75% are SLQ cycles). For this traffic pattern, JSQ provides little benefit. Thus, the two policies are complementary; JSQ helps reduce response time in moderate traffic and SLQ helps defer throughput degradation under heavy, nonuniform traffic.

Alltoall

MPI alltoall is an important MPI collective communications operation in which every node sends a different message to every other node. **Figure 6** plots the average link utilization during the communications pattern implied by this collective. Again, the data shows the benefit of dynamic over deterministic routing. For this pattern, there is marginal benefit in going from one to two dynamic VCs, but what is important is that the average link utilization is approximately 98%, close to the theoretical peak. This peak includes the overhead for the 8-B ack or token–ack packets, the packet headers, and the 4-B hardware trailers. There is also a 2-B gap between packets to help keep the links trained. Thus, a 256-B packet actually implies that the link is occupied for 270 (= 256 + 4 + 8 + 2) byte times. A reasonable assumption for the BG/L software is that each packet carries 240 bytes of payload and, with this assumption, the figure shows that the payload occupies 87% of the links. The fact that a very low percentage of the traffic flows on the escape bubble VC is not shown in these figures. Also not shown, but indicated by statistics collected during the run, is the fact that few of the VC

buffers are full. Three-dimensional fast Fourier transform algorithms often require the equivalent of an alltoall, but on a subset of the nodes consisting of either a plane or a line in the torus. Simulations of these communication patterns also resulted in near-peak performance.

The above simulation was for a symmetric BG/L. However, the situation is not so optimistic for an asymmetric BG/L. For example, the 64K-node system will be a $64 \times 32 \times 32$-node torus. In such a system, the average number of hops in the x dimension is twice that of the y and z dimensions, so that even if every x link is 100% busy, the y and z links can be, at most, 50% busy. Thus, the peak link utilization is, at most, 66.7%. In other words, the bisection bandwidth is the same as in a $32 \times 32 \times 32$-node machine. Since 12% of that is overhead, the best possible payload utilization is 59%. However, we expect significantly more blocking and throughput degradation due to full VC buffers. Indeed, a simulation of the alltoall communications pattern on a $32 \times 16 \times 16$ torus resulted in an average link utilization of 49% and payload utilization of 44%, corresponding to 74% of the peak. This figure is probably somewhat pessimistic because of the simulator artifact of infinite-sized injection FIFOs, which distorts the effects at the end of the simulation. We also believe that appropriate injection flow-control software algorithms can reduce VC buffer blocking and achieve a performance that is closer to peak. Nevertheless, the above study points out a disadvantage of the torus architecture for asymmetric machines in which the application cannot easily be mapped such that the result is a close-proximity communications pattern.

Virtual channel architecture

Here we consider several different deadlock-prevention escape VC architectures. The first, originally proposed in [7], has two escape VCs per direction. Each dimension has a "dateline." Before crossing the dateline, the escape VC is the lower-numbered of the pair, but after crossing the dateline, the escape VC is the higher-numbered of the pair. In addition, we consider dimension-ordered or direction-ordered escape VCs. In the dimension-ordered version, the escape VC is x first, then y if no x hops remain, then z if no x or y hops remain. In the direction-ordered version introduced by Cray Inc. [2], the escape VCs are ordered by $x+$, $y+$, $z+$, $x-$, $y-$, $z-$ (other orderings are possible).

We also consider dimension- and direction-ordered escape VCs for the bubble escape and again use the hot-region workload, in which the hot region starts at coordinates (0, 0, 0) and the datelines are set at the maximum coordinate value in each dimension. **Figure 7** shows the throughput as a function of time. The dimension-ordered dateline pair shows particularly poor

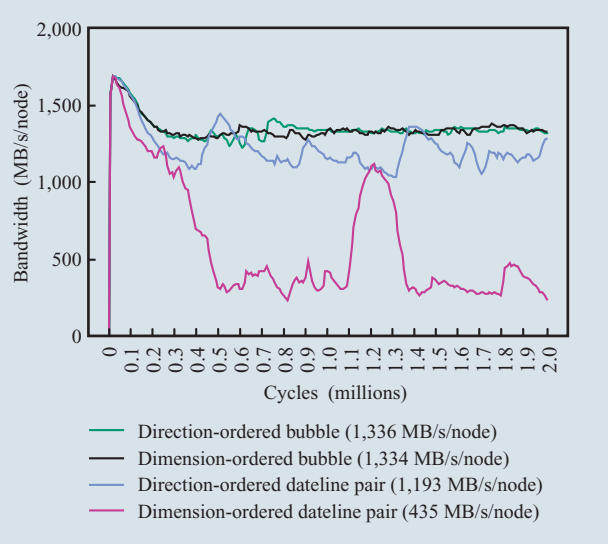

Figure 7

Throughput under hot-region traffic for different escape VC architectures. [4K-node BG/L under hot-region traffic (25% of traffic to 12.5% of machine); two dynamic 2-KB VCs, two paths, 2-KB escape buffers. Early assumptions were used (see text).]

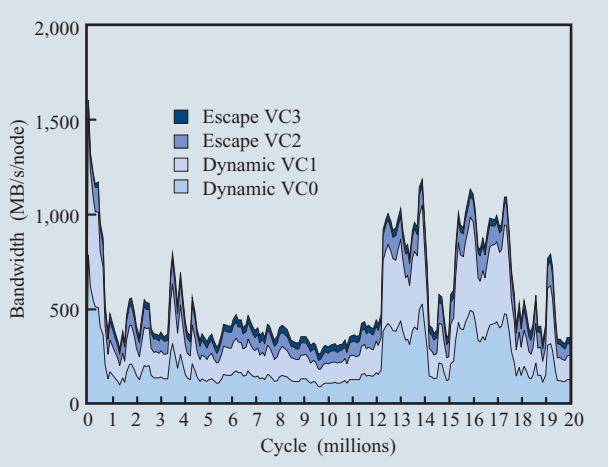

Figure 8

Throughput on each VC for the dimension-ordered dateline pair escape VC architecture. [4K-node BG/L under hot-region traffic (25% of traffic to 12.5% of machines; two dynamic VCs, dimension-ordered dateline pair). Early assumptions were used (see text).]

and wild behavior, with a steep decline in throughput, followed by a rise and then another steep decline.

Figure 8 shows the throughput on a per-VC basis for a longer period of time. The decreasing and increasing

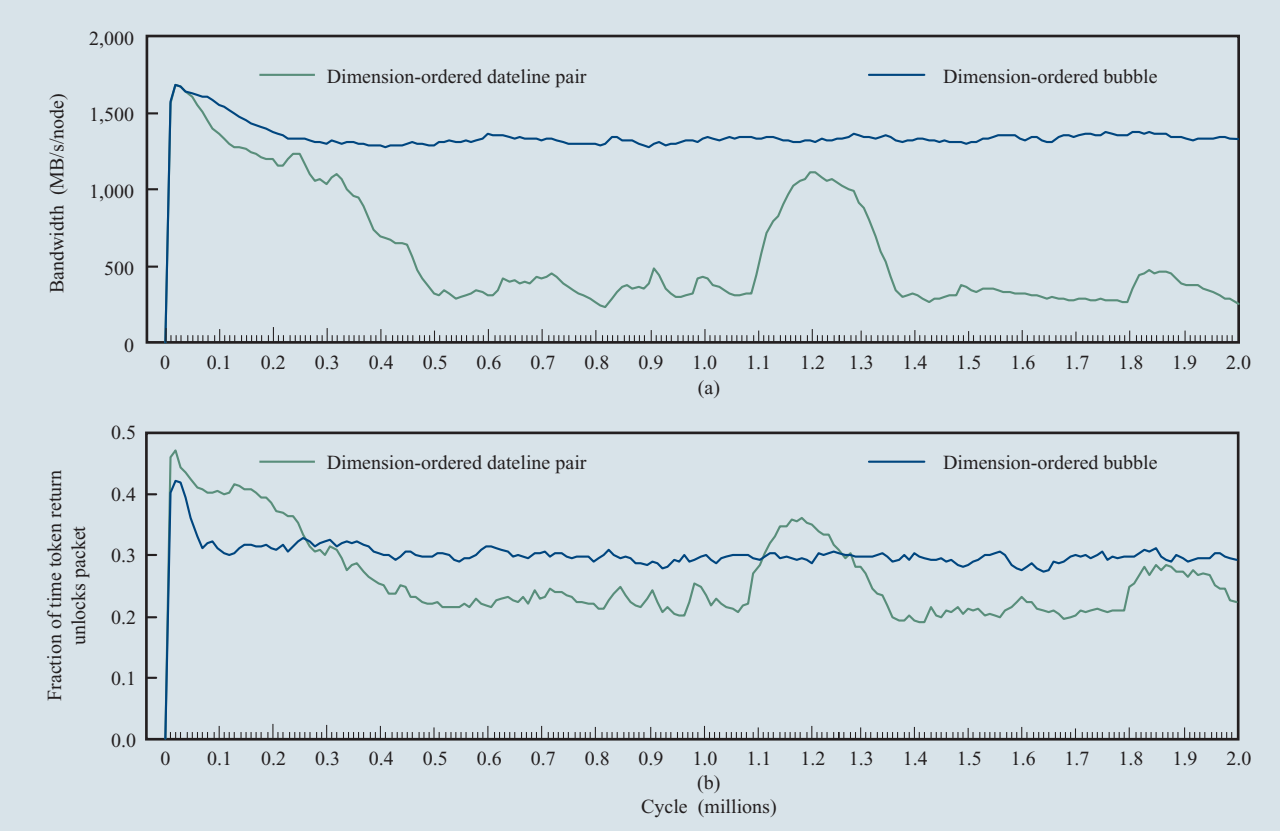

Figure 9

(a) Bandwidth as a function of time. (b) Unblocking probability as a function of time. [4K-node BG/L under hot-region traffic (25% of traffic to 12.5% of machine); two dynamic 2-KB VCs, two paths, 2-KB escape buffers. Early assumptions were used (see text).]

bandwidth waves persist, even over this much longer time scale. An appreciable fraction of the traffic flows on the escape VCs, indicating a high level of VC buffer occupation.

One may ask, *What causes these waves?* First, the placement of the dateline causes an asymmetry in the torus, whereas the bubble escape is perfectly symmetrical in each dimension. Since there are two escape VCs, we thought it likely that packets at the head of the VC buffers could be waiting for one of the escape VCs, but tokens are returned for the other escape VC. In such a situation, no packets can move, even though the link may be available and downstream buffer space is available. To confirm this, the simulator was instrumented to collect additional statistics. In particular, we measured the fraction of time in which a token–ack is returned that frees at least one previously blocked packet to move. **Figure 9** shows this unblocking probability, along with the throughput, as a function of time, confirming our hypothesis. The unblocking probability is relatively constant for the bubble (after the initial decline), but varies directly with the throughput for the dateline pair; when the unblocking probability increases (or decreases), the throughput increases (or decreases).

Performance verification

To verify the VHDL logic of the torus, we built a multinode verification testbench. This testbench, which runs on the Cadence** VHDL simulator, consisted of the following: workload drivers that inject packets into the injection FIFOs; links between nodes on which bits can be corrupted to test the error-recovery protocol; and packet checkers that pull packets out of the reception FIFOs and check them for a variety of conditions, such as whether the packet arrived at the correct destination and whether its contents were received correctly. The workload drivers can be flexibly configured to simulate a number of different traffic patterns.

As we neared the end of the logic verification process, we wanted to ensure that network performance was as

intended. One of the benchmarks we tested was the alltoall. The VHDL simulator was limited by memory to a maximum of 64 nodes, so we simulated both a $4 \times 4 \times 4$ torus and an $8 \times 8 \times 1$ torus and compared the average link utilizations to those predicted by the performance simulator. While these agreed to within 2%, the VHDL, corresponding to the actual network hardware, indicated that VC buffers were fuller than that predicted by the performance simulator. A close inspection of the arbitration logic revealed that a one-cycle gap in the arbitration pipeline of the receivers could occur when all possible outgoing links and VCs were busy. This gap was sufficient to permit packets from the injection FIFOs to sneak into the network, leading to fuller VCs than intended. A simple fix to eliminate this possibility was implemented, and subsequent VHDL simulations indicated greatly reduced levels of VC buffer occupation.

Sample hardware performance measurements

We now describe some performance measurements of a number of important communications patterns done on the 512-node prototype. These measurements were made using the Blue Gene/L advanced diagnostics environment (BLADE) kernel [16]. Since BLADE is a lightweight kernel permitting direct access to the BG/L hardware, and since these measurements do not involve the MPI library, the measurements are intended to stress and demonstrate raw torus hardware performance. Also, given our understanding of the network hardware and protocols, the packet format, and the communications pattern, we are able to express results as a percentage of peak performance. For example, as described earlier, a 256-byte packet actually implies that the link is occupied for 270 byte-times, and this provides a limit on the throughput.

Ping-pong latency

Ping-pong latency is measured by sending a message from one node, which we call *ping*, to another node, *pong*. Once the message is received and stored by pong, pong sends it back to ping. Ping receives and stores the full message. The one-way latency (half the total time from the point of the first injection to the final reception by ping) is measured in units of processor cycles and is plotted against message size for one, two, and three network hops (**Figure 10**).

The curves marked *total* are plots of the measured one-way latency, as described above. The curves marked *network bound* are calculated theoretically on the basis of the expected bandwidth and latency of the network. They do not include the bandwidth and latency incurred in transferring the message from memory to the torus injection FIFOs or from the torus reception FIFOs to memory. In this sense, they represent a lower bound for the ping-pong latency. Note that these measurements were made using a thin packet layer interface and do not include MPI overhead.

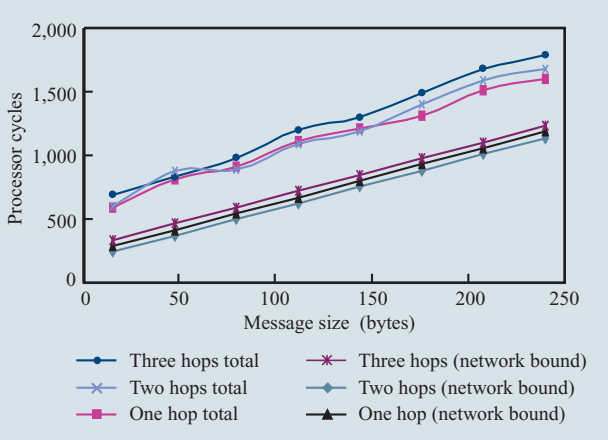

Figure 10

BG/L torus network ping-pong latency as measured on the hardware ($2 \times 2 \times 2$ mesh).

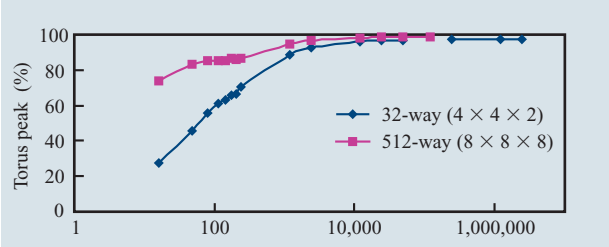

Figure 11

BG/L torus all-to-all communication pattern efficiency as measured on the hardware.

Alltoall

Figure 11 plots the percentage of peak performance obtained on both a 32-way $4 \times 4 \times 2$ torus and a 512-way $8 \times 8 \times 8$ torus for the all-to-all pattern described earlier. The x-axis in this plot is the number of payload bytes (assuming that 16 bytes of software overhead are carried in each packet) between each source–destination pair. On the 512-way—even when the messages are extremely short, involving only one 32-byte packet between each pair of nodes—the torus delivers 71% of theoretical peak, and for long messages, it delivers more than 98% of peak. The simulator predicts the throughput quite accurately. For example, with ten full-sized packets, the simulator predicted 94% of peak (i.e., 94% link utilization), whereas

the hardware measurement achieved 96% of peak. This is a difference of only 2%, with most of that discrepancy due to different software start-up sequences. Thus, the hardware measurement is in close agreement with the simulation. Again, this measurement involves a thin packet layer and is not an actual MPI alltoall measurement.

Hot region

In this pattern, a subcube of nodes are receivers, and all nodes outside this subcube are senders. Each sender sends a certain large number of 256-byte packets to each receiver. The amount of time required to complete this test is determined by the number of links into the hot-region subcube. For a hot spot ($1 \times 1 \times 1$ subcube), 92% of peak is achieved, while 95% of peak is achieved for both $2 \times 2 \times 2$ and $4 \times 4 \times 4$ hot regions, again in accordance with simulations. Note that the hot-spot communications pattern corresponds to that of the MPI allgather collective communications call.

Line and plane fill

Line and plane fill communication patterns are those obtained when broadcasting in either a row or a plane of the torus. This pattern can take advantage of the row-broadcast capability in BG/L. Depending on the mapping, the major communication patterns in the Linpack benchmark correspond to row or plane broadcasts. By appropriately dividing the data to be broadcast into different sections (*colors*) and broadcasting the different colors in different directions, one can attempt to keep all links busy transmitting different data at all times. For the line broadcast of large messages, more than 99% of peak is achieved. For the plane broadcast, we employed both cores, with one core servicing the plus directions and another servicing the minus directions. In this case, certain nodes have to make "corner turns" in which, for example, packets traveling in the $x+$ direction are received and reinjected in the $y+$ direction. Our prototype software accomplished this by loading the data from the torus reception FIFOs to the FPU registers and then storing the register contents to both memory and the torus injection FIFOs. For production software, some additional care must be taken to prevent deadlocks in case the injection FIFOs are full, but the prototype software did not encounter a deadlock situation. In this case, more than 96% of peak is achieved for the plane fill. With the links running at 175 MB/s and assuming 87% payload efficiency, this translates to the ability to broadcast in a plane at 585 MB/s ($= 175 \times 4 \times 0.87 \times 0.96$).

Conclusions

We have described the details of the three-dimensional torus network which forms the principal interconnect of the Blue Gene/L supercomputer. By combining adaptive cut-through and deterministic routing, we have formed a robust and deadlock-free network. By comparing simulation with measured results, we have demonstrated the ability to realistically model a massively parallel network, to precisely predict network performance under load, and to accurately assess the impact of architecture choices such as arbitration codes, buffer sizes, and number of virtual channels. Using a thin packet layer interface, we have measured the performance of the BG/L torus for all-to-all, plane fill, and line fill, independently of the MPI overhead. For most application codes, we believe that the BG/L torus will be more than adequate in terms of reliability, bandwidth, and latency.

Acknowledgments

We are grateful to Jose Duato, Peter Hochschild, and Craig Stunkel for helpful conversations concerning interconnection networks. Barry Trager helped us choose the CRC polynomials. Ralph Bellofatto, Ruud Haring, and Gerard Kopcsay provided invaluable assistance during hardware bring-up. George Chiu provided support throughout the entire project.

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

References

1. N. R. Adiga et al., "An Overview of the Blue Gene/L Supercomputer," *Proceedings of the ACM/IEEE Conference on Supercomputing*, November 2002, pp. 1–22; see *www.sc-conference.org/sc2002/*.
2. S. Scott and G. Thorson, "The Cray T3E Network: Adaptive Routing in a High Performance 3D Torus," *Proceedings of HOT Interconnects IV*, August 1996, pp. 147–156.
3. P. Kermani and L. Kleinrock, "Virtual Cut-Through: A New Computer Communication Switching Technique," *Computer Networks* **3**, 267–286 (1979).
4. High-Performance Parallel Interface (HIPPI) Standards Activities; see *http://www.hippi.org*.
5. D. K. Panda, S. Singal, and R. Kesavan, "Multidestination Message Passing in Wormhole *k*-ary *n*-cube Networks with Base Routing Conformed Paths," *IEEE Trans. Parallel & Distr. Syst.* **10**, No. 1, 76–96 (January 1999).
6. W. J. Dally, "Virtual-Channel Flow Control," *IEEE Trans. Parallel & Distr. Syst.* **3**, No. 2, 194–205 (March 1992).
7. W. J. Dally and C. L. Seitz, "Deadlock Free Message Routing in Multiprocessor Interconnection Networks," *IEEE Trans. Computers* **C-36**, No. 5, 547–553 (May 1987).
8. J. Duato and T. M. Pinkston, "A General Theory for Deadlock-Free Adaptive Routing Using a Mixed Set of Resources," *IEEE Trans. Parallel & Distr. Syst.* **12**, No. 12, 1219–1235 (December 2001).
9. V. Puente, R. Beivide, J. A. Gregorio, J. M. Prellezo, J. Duato, and C. Izu, "Adaptive Bubble Router: A Design to Improve Performance in Torus Networks," *Proceedings of the IEEE*

International Conference on Parallel Processing, September 1999, pp. 58–67.
10. V. Puente, C. Izu, R. Beivide, J. A. Gregorio, F. Vallejo, and J. M. Prellezo, "The Adaptive Bubble Router," *J. Parallel & Distr. Computing* **61**, No. 9, 1180–1208 (September 2001).
11. W. J. Dally, L. R. Dennison, D. Harris, K. Kan, and T. Xanthoppulos, "Architecture and Implementation of the Reliable Router," *Proceedings of HOT Interconnects II*, August 1994, pp. 122–133.
12. Q. Yu, D. Towsley, and P. Heidelberger, "Time-Driven Parallel Simulation of Multistage Interconnection Networks," *Proceedings of the Society for Computer Simulation (SCS) Multiconference on Distributed Simulation*, March 1989, pp. 191–196.
13. C. Benveniste and P. Heidelberger, "Parallel Simulation of the IBM SP2 Interconnection Network," *Proceedings of the IEEE Winter Simulation Conference*, December 1995, pp. 584–589.
14. P. M. Dickens, P. Heidelberger, and D. M. Nicol, "Parallelized Direct Execution Simulation of Message-Passing Parallel Programs," *IEEE Trans. Parallel & Distr. Syst.* **7**, No. 10, 1090–1105 (October 1996).
15. D. Nicol, C. Micheal, and P. Inouye, "Efficient Aggregation of Multiple LPs in Distributed Memory Parallel Simulations," *Proceedings of the Winter Simulation Conference*, December 1989, pp. 680–685.
16. M. E. Giampapa, R. Bellofatto, M. A. Blumrich, D. Chen, M. B. Dombrowa, A. Gara, R. A. Haring, P. Heidelberger, D. Hoenicke, G. V. Kopcsay, B. J. Nathanson, B. D. Steinmacher-Burow, M. Ohmacht, V. Salapura, and P. Vranas, "Blue Gene/L Advanced Diagnostics Environment," *IBM J. Res. & Dev.* **49**, No. 2/3, 319–331 (2005, this issue).

Received May 6, 2004; accepted for publication June 15, 2004; Internet publication April 29, 2005

Narasimha R. Adiga *IBM Engineering and Technology Services, Golden Enclave, Airport Road, Bangalore 560 017, India (anarasim@in.ibm.com)*. Mr. Adiga is a Staff Research and Development Engineer. In 1998 he received a B.E. degree from Karnataka Regional Engineering College, India. He subsequently joined IBM, where he has worked on the development of the PCI-PCIX exerciser/analyzer card and verification of the torus with a single-node BG/L compute chip for Blue Gene/L. Mr. Adiga is currently working on memory interface controller designs for future systems.

Matthias A. Blumrich *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (blumrich@us.ibm.com)*. Dr. Blumrich is a Research Staff Member in the Server Technology Department. He received a B.E.E. degree from the State University of New York at Stony Brook in 1986, and M.A. and Ph.D. degrees in computer science from Princeton University in 1991 and 1996, respectively. In 1998 he joined the IBM Research Division, where he has worked on scalable networking for servers and the Blue Gene supercomputing project. Dr. Blumrich is an author or coauthor of two patents and 12 technical papers.

Dong Chen *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (chendong@us.ibm.com)*. Dr. Chen is a Research Staff Member in the Exploratory Server Systems Department. He received his B.S. degree in physics from Peking University in 1990, and M.A., M.Phil., and Ph.D. degrees in theoretical physics from Columbia University in 1991, 1992, and 1996, respectively. He continued as a postdoctoral researcher at the Massachusetts Institute of Technology from 1996 to 1998. In 1999 he joined the IBM Server Group, where he worked on optimizing applications for IBM RS/6000* SP systems. In 2000 he moved to the IBM Thomas J. Watson Research Center, where he has been working on many areas of the Blue Gene/L supercomputer and collaborating on the QCDOC project. Dr. Chen is an author or coauthor of more than 30 technical journal papers.

Paul Coteus *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (coteus@us.ibm.com)*. Dr. Coteus received his Ph.D. degree in physics from Columbia University in 1981. He continued working at Columbia to design an electron–proton collider, and from 1982 to 1988 was an Assistant Professor of Physics at the University of Colorado at Boulder, studying neutron production of charmed baryons. In 1988, he joined the IBM Thomas J. Watson Research Center as a Research Staff Member. Since 1994 he has managed the Systems Packaging Group, where he directs and designs advanced packaging and tools for high-speed electronics, including I/O circuits, memory system design and standardization of high-speed DRAM, and high-performance system packaging. His most recent work is in the system design and packaging of the Blue Gene/L supercomputer, where he served as packaging leader and program development manager. Dr. Coteus has coauthored numerous papers in the field of electronic packaging and holds 38 U.S. patents.

Alan Gara *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (alangara@us.ibm.com)*. Dr. Gara is a Research Staff Member at the IBM Thomas J. Watson Research Center. He received his Ph.D. degree in physics from the University of Wisconsin at Madison in 1986. In 1998 Dr. Gara received the Gordon Bell

Award for the QCDSP supercomputer in the most cost-effective category. He is the chief architect of the Blue Gene/L supercomputer. Dr. Gara also led the design and verification of the Blue Gene/L compute ASIC as well as the bring-up of the Blue Gene/L prototype system.

Mark E. Giampapa *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (giampapa@us.ibm.com)*. Mr. Giampapa is a Senior Engineer in the Exploratory Server Systems Department. He received a B.A. degree in computer science from Columbia University. He joined the IBM Research Division in 1984 to work in the areas of parallel and distributed processing, and has focused his research on distributed memory and shared memory parallel architectures and operating systems. Mr. Giampapa has received three IBM Outstanding Technical Achievement Awards for his work in distributed processing, simulation, and parallel operating systems. He holds 15 patents, with several more pending, and has published ten papers.

Philip Heidelberger *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (philiph@us.ibm.com)*. Dr. Heidelberger received a B.A. degree in mathematics from Oberlin College in 1974 and a Ph.D. degree in operations research from Stanford University in 1978. He has been a Research Staff Member at the IBM Thomas J. Watson Research Center since 1978. His research interests include modeling and analysis of computer performance, probabilistic aspects of discrete event simulations, parallel simulation, and parallel computer architectures. He has authored more than 100 papers in these areas. Dr. Heidelberger has served as Editor-in-Chief of the *ACM Transactions on Modeling and Computer Simulation*. He was the general chairman of the ACM Special Interest Group on Measurement and Evaluation (SIGMETRICS) Performance 2001 Conference, the program co-chairman of the ACM SIGMETRICS Performance 1992 Conference, and the program chairman of the 1989 Winter Simulation Conference. Dr. Heidelberger is currently the vice president of ACM SIGMETRICS; he is a Fellow of the ACM and the IEEE.

Sarabjeet Singh *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (sarabj@us.ibm.com)*. Mr. Singh is a Senior Research and Development Engineer with the Engineering and Technology Services Division of IBM, currently on assignment at the IBM Thomas J. Watson Research Center. He received a B.Tech. degree in electrical engineering from the Indian Institute of Technology in 1996 and subsequently joined IBM, where he has worked on various research projects involving all aspects of ASIC and system-on-a-chip (SoC) design. Over the past seven years he has worked on many CMOS technologies (Blue Gene/L in Cu-11 technology), up to 700-MHz clock designs, asynchronous logic design, and Small Computer System Interface (SCSI) drive controllers to HPC systems. Mr. Singh is currently working on memory subsystem microarchitecture for an HPC system based on the STI cell.

Burkhard D. Steinmacher-Burow *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (steinmac@us.ibm.com)*. Dr. Steinmacher-Burow is a Research Staff Member in the Exploratory Server Systems Department. He received a B.S. degree in physics from the University of Waterloo in 1988, and M.S. and Ph.D. degrees from the University of Toronto in 1990 and 1994, respectively. He subsequently joined the Universitaet Hamburg and then the Deutsches Elektronen-Synchrotron to work in experimental particle physics. In 2001, he joined IBM at the Thomas J. Watson Research Center and has since worked in many hardware and software areas of the Blue Gene research program. Dr. Steinmacher-Burow is an author or coauthor of more than 80 technical papers.

Todd Takken *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (taken@us.ibm.com)*. Dr. Takken is a Research Staff Member at the IBM Thomas J. Watson Research Center. He received a B.A. degree from the University of Virginia and an M.A. degree from Middlebury College; he finished his Ph.D. degree in electrical engineering at Stanford University in 1997. He then joined the IBM Research Division, where he has worked in the areas of signal integrity analysis, decoupling and power system design, microelectronic packaging, parallel system architecture, packet routing, and network design. Dr. Takken holds more than a dozen U.S. patents.

Michael Tsao *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (mtsao@us.ibm.com)*. Dr. Tsao is a Research Staff Member at the IBM Thomas J. Watson Research Center. He received B.S.E.E, M.S.E.C.E, and Ph.D. degrees from the Electrical and Computer Engineering Department of the Carnegie-Mellon University in 1977, 1979, and 1983, respectively. He joined IBM at the Thomas J. Watson Research Center in 1983 and has worked on various multiprocessor projects including RP3, GF11, Vulcan, SP1, SP2, MXT, and BG/L. Dr. Tsao is currently working on cache chips for future processors.

Pavlos Vranas *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (vranasp@us.ibm.com)*. Dr. Vranas is a Research Staff Member in the Deep Computing Systems Department at the IBM Thomas J. Watson Research Center. He received his B.S. degree in physics from the University of Athens in 1985, and his M.S. and Ph.D. degrees in theoretical physics from the University of California at Davis in 1987 and 1990, respectively. He continued research in theoretical physics as a postdoctoral researcher at the Supercomputer Computations Research Institute, Florida State University (1990–1994), at Columbia University (1994–1998), and at the University of Illinois at Urbana–Champaign (1998–2000). In 2000 he joined IBM at the Thomas J. Watson Research Center, where he has worked on the architecture, design, verification, and bring-up of the Blue Gene/L supercomputer and is continuing his research in theoretical physics. Dr. Vranas is an author or coauthor of 59 papers in supercomputing and theoretical physics.