# The IBM Blue Gene/Q Interconnection Network and Message Unit

Dong Chen, Noel A. Eisley, Philip Heidelberger,
Robert M. Senger, Yutaka Sugawara, Sameer Kumar,
Valentina Salapura, David L. Satterfield
IBM T.J. Watson Research Center
Yorktown Heights, NY 10598
{chendong, naeisley, philiph, rmsenger, ysugawa, sameerk,
salapura, davidsat}@us.ibm.com

Burkhard Steinmacher-Burow
IBM Deutschland Research & Development GmbH
71032 Böblingen, Germany
steinmac@de.ibm.com

Jeffrey J. Parker
IBM Systems &Technology Group
Systems Hardware Development
Rochester, MN 55901
jjparker@us.ibm.com

***Abstract*- This is the first paper describing the IBM Blue Gene/Q interconnection network and message unit. The Blue Gene/Q system is the third generation in the IBM Blue Gene line of massively parallel supercomputers. The Blue Gene/Q architecture can be scaled to 20 PF/s and beyond. The network and the highly parallel message unit, which provides the functionality of a network interface, are integrated onto the same chip as the processors and cache memory, and consume 8% of the chip's area. For better application scalability and performance, we describe new routing algorithms and new techniques to parallelize the injection and reception of packets in the network interface. Measured hardware performance results are also presented.**

***Keywords- parallel computer architecture, interconnect technologies, router architecture, routing algorithms and techniques, network interface architecture***

## I. INTRODUCTION

The IBM Blue Gene/Q (BG/Q) system is the third generation in the IBM Blue Gene line of massively parallel supercomputers. BG/Q can be scaled to 20 PF/s and beyond. An overview of BG/Q is given in [1], while the first generation Blue Gene/L and second generation Blue Gene/P are described in [2] and [3], respectively. This paper is the first detailed description of the BG/Q network and message unit. The highly parallel message unit (MU) provides the functionality of a network interface. Both the network logic and MU are integrated onto the same chip as the processors and cache memory and consume 8% of the chip's area, including IO cells. The network, which is generally configured as a five dimensional torus, is described in Section II. Section III gives an overview of the MU. Software interfaces to the network and MU are described in Section IV and initial performance measurements are reported in Section V.



*SC11*, November 12–18, 2011, Seattle, Washington, USA.


## II. INTERCONNECTION NETWORK

### A. Overview

BG/Q systems consist of compute nodes and IO nodes. Applications run on the compute nodes while file IO is shipped from a compute to an IO node, where it is then sent over a PCIe interface to a file system. Compute nodes are interconnected via a five dimensional (5D) torus. To support a 5D torus, 10 bidirectional ports, or links, are required. The logic implements an additional 11$^{th}$ link, the IO link, which is used to connect the compute and I/O nodes together. Each torus link, including the IO link, operates at 2 GB/s; each link/port can simultaneously send at 2 GB/s and receive at 2 GB/s. To match IO bandwidth to the external file system, only some compute nodes have the IO link attached to an IO node. Typically, an IO node is connected to two compute nodes. In addition, to save pins on the chip, IO nodes use the pins of one of the torus dimensions for the PCIe. The PCIe operates at 4 GB/s; the bandwidth of the two IO links from two compute nodes thus matches the PCIe bandwidth of the IO node.

We now focus on the compute node torus and compare BG/Q to BG/L and BG/P. The network logic on BG/P is essentially identical to that of BG/L, the major difference being an increase in link bandwidth. BG/L has a 3D torus with 175 MB/s per link. Thus each BG/Q link at 2 GB/s is 11.4 times

faster than a BG/L link and the total compute node torus bandwidth on BG/Q is 19 times that of BG/L. Similarly BG/P has a 3D torus with 425 MB/s links, so BG/Q links are 4.7 times faster than BG/P links and the total BG/Q compute node torus bandwidth is 7.8 times that of BG/P.

A 5D torus was chosen for three primary reasons. First, from a performance perspective, it achieves high nearest-neighbor bandwidth while increasing bisection bandwidth and reducing the maximum number of hops (and latency) compared to a lower dimensional torus. For example, a 20 PF 16x16x16x12x2 BG/Q has about 46 (19) times the bisection bandwidth than a 64x48x32 BG/L (BG/P) with the same number of nodes. Compared to BG/L, 11.4x of the 46x comes from increasing the link bandwidth from 175 MB/s to 2 GB/s and 4x comes from reducing the length of the maximum dimension, which for a fixed number of nodes is made possible by increasing the dimensionality of the network. Second, the torus permits partitioning a large machine into independent sub-machines; applications running in different partitions do not affect one another at all, except possibly for file IO. Third, from a packaging perspective, the torus permits most links, those within a midplane, to be electrical rather than optical, reducing cost. The links internal to a midplane (4x4x4x4x2) are through circuit cards. The links that are on the surface of this 5-D cube connect through a link chip to an optical transceiver. The midplane is built from 2x2x2x2x2 boards; there are 32 cards, each with one compute node, attached to the board. There are link chips on each board to connect via optics to boards in other midplanes. The dimensions are labeled A,B,C,D,E, with the opposing directions signified by, *e.g.*, A- and A+. The last dimension E is constrained to always be of length two, thereby keeping its links entirely within a single board and reducing inter-board wiring within a midplane. The pins for the E dimension are used for PCIe on IO nodes.

In addition to the torus network, BG/L and BG/P have a global barrier network and a collective network. To reduce cost, simplify inter-midplane cable connections and to maintain partitionability, since many applications do not use point-to-point and collective messaging at the same time, BG/Q integrates barrier and collective functionality onto the torus network.

Data packets on BG/Q include a 32 B header; 12 B for the network and 20 B for the MU. The data portion of the packet is from 0 to 512 B, in increments of 32 B chunks. With 8 trailing link level packet check bytes and protocol packet overhead, at most 90% of the raw link bandwidth, 1.8 GB/s, can be used for user data.

On BG/Q, the 64 bit PowerPC cores operate at 1.6 GHz while the rest of the memory system, including the MU, operates at half that rate, 800 MHz. The internal network logic operates at 500 MHz; the network handles 4 B per network cycle thereby matching the 2 GB/s link bandwidth.

The on-chip per hop latency for point-to-point packets on BG/Q is approximately 40 ns compared to approximately 97 ns on BG/L and 46 ns on BG/P. Of the 40 ns, which is 20 network cycles, 8 cycles are in the network logic (compared to 12 network cycles on BG/L) with the rest being in the SerDes and high speed signaling. The worst case hardware one way point-to-point latency on a large 16x16x16x12x2 system is expected to be about 2.6 µs, including cable delays.

An overview of the network, showing its major units, is shown in Figure 1. There are 11 send units and 11 receive units; one for each of the links in the 5D torus and one for the IO link. All packets are injected into a network injection FIFO by the MU and all packets are pulled from a network reception FIFO by the MU. There are multiple injection and reception FIFOs divided up for normal priority user point-to-point data on the 5D torus, as well as intra-node local transfers, user high priority and system point-to-point data, and user and system collective data. The number of FIFOs is sufficient to ensure that all links can be kept busy simultaneously. Packets injected into any of the point-to-point injection FIFOs may go out any link, *i.e.*, injection FIFOs are not tied to torus links. When a normal priority point-to-point packet arrives at its destination, it is placed into the reception FIFO associated with the receive unit on which the packet arrived. For example, packets arriving on the A- receiver are always placed in the A- reception FIFO. Upon reception, user high priority, system and collective packets are placed in their corresponding reception FIFOs.

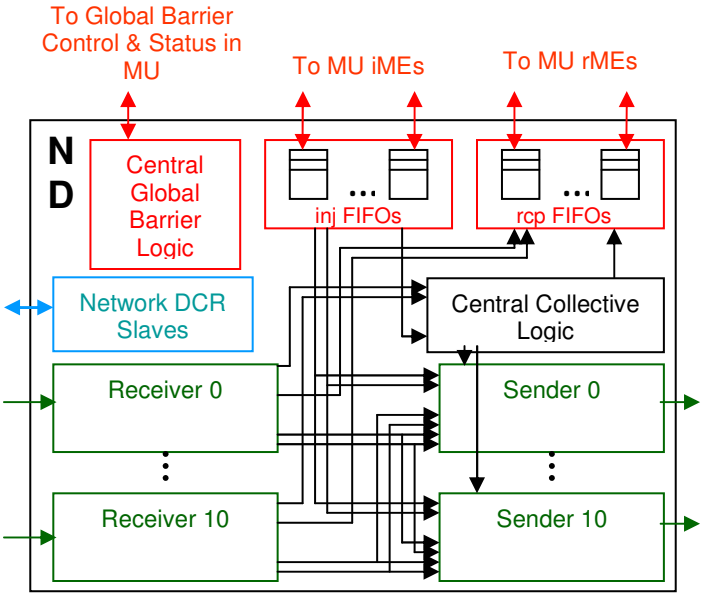


Figure 1. The BG/Q Network Device (ND) Router Logic

### B. *Virtual Channels and Point-to-Point Routing*

To support an integrated network with user, system and collective traffic, BG/Q has more virtual channels (VCs) than BG/L. There are virtual channels (VCs) for point-to-point (user dynamic, user deterministic, user high priority, system) and for collective (user and system) traffic. Each receiver has separate buffers for each VC (with the exception that to reduce internal network VC storage, user commworld and system collective use the same physical VC; software ensures that these two logical VCs never use the same physical links) and there is virtual cut-through logic and token flow control similar to that in BG/L and described more fully in [4] and [5]. To improve BG/Q performance, we increased the number of packets that can be stored in each VC. To reduce head-of-line blocking,

BG/L and BG/P have two VCs for dynamic routing, but the deterministic VC is managed as a single queue FIFO. Improving upon that, a form of virtual output queuing is employed for BG/Q. Each point-to-point VC maintains multiple queues, each holding one or more packets. Packets in different queues requiring different links do not block one another, but deterministically routed packets in different queues that do require the same link are served in first-come-first-served order, thus maintaining in-order delivery. When a dynamic packet is placed in a VC buffer, it selects the queue with the fewest number of packets in it. Packets in different dynamic VC queues do not block one another. Like BG/L, there are multiple data paths from each receiver so that multiple packets can be transferred simultaneously.

Point-to-point routing is similar to BG/L in that "hint" bits specify which of the links can be used (at most one per dimension). However, there are several important improvements to boost performance, especially for asymmetric tori. First, the dimension order for deterministically routed packets is programmable whereas it was fixed on BG/L. This permits routing longest dimension first, which is typically most efficient. Second, for dynamically routed packets, "zone" routing is introduced in which several packet header bits indicate which set of programmable zone masks (stored in network control registers) are to be used. These permit dynamic routing but constrain the order in which dimensions are routed. For example, in a 16x16x12x12x2 torus, they can be programmed to route longest dimensions first; A or B first, then C or D, then E. As BG/L routing is very efficient on symmetric tori, this effectively breaks the routing into symmetric zones of decreasing size, thus packets tend to move from the busy links to either equally busy links, or more lightly loaded links [6]. This reduces pressure on internal network buffers, thereby improving performance. For example, detailed near cycle accurate (parallel) simulations of an all-to-all pattern of a 16x8x8x8 torus showed that performance improved from 66% of network peak without zone routing to 93% of peak with zone routing. Simulations of zone routing on a 16x16x16x8 torus achieved 99% of network peak.

Routing from a compute node to an IO node is handled as follows. The packet destination specifies the coordinates of an "exit" node, that compute node whose IO link is attached to an IO node. The packet routes deterministically to the exit node. If a "toIO" bit in the packet header is set, the packet then routes to the attached IO node over the IO link, where it is received. When routing from an IO node to a compute node, the packet destination specifies the coordinates of the compute node destination. The packet routes over the IO link to the attached compute node, and then to its final compute node destination. Only system packets can route over the IO links.

Like BG/L, BG/Q supports broadcasts down a line of a torus dimension using the point-to-point virtual channels.

### C. *Collective Support*

BG/Q implements support for collective operations within the network. BG/Q support for collectives improves over BG/L in two primary ways. First, whereas BG/L required two passes over the collective network for floating point reductions, BG/Q incorporates logic for one pass double precision floating point sums. Second, BG/Q supports collective operations over MPI sub-communicators, provided they are contiguous sub-rectangles of the torus (e.g., over lines, planes, 3D sub-cubes, etc).

The collective unit supports floating point add, min and max as well as the following fixed point operations: signed and unsigned add, signed and unsigned min and max, bitwise AND, OR and XOR. Header-only packets (0 B payload) are valid collective packets as well, and can be used for short broadcasts or barriers. Broadcast, reduce to a single node and all reduce to all nodes are supported. In addition, a single node can broadcast a short remote get packet (rDMA read, see Section III) onto the network thereby causing a long message reduce or all reduce on the network.

The nodes participating in a collective are defined by class routes, are programmed into control registers on each node and specify a contiguous (non-binary) tree which is embedded within the torus network. Packets go up the tree, being reduced on each hop and are turned around at the tree's root and broadcast back down the tree. The class routes specify which links are inputs on the uptree path, which link is the uptree output (there is none at the root) and whether or not there is a contribution from the local node. Each node can participate in up to 16 different class routes, but there can be many more than 16 class routes in the machine. For example, there could be a class route for each two dimensional AB plane and all of these could be active simultaneously without any interference between them. Packets can be flowing uptree and downtree at the same time, but at any given time the uptree (downtree) collective logic can be active processing packets from only one class route. For reductions, uptree packets are stored in the receiver's VC buffers until packets from all of the class route's inputs have been received. There can be up to 12 inputs: 11 from each of the links and one local contribution. When the output link is free, the packets proceed through parallel, multilevel 2-input, 1-output ALUs with the combined packet being sent on the output link.

Floating point additions are bit-reproducible; if each node inputs the same floating point numbers on two different runs on the same machine geometry and with the same class route, the results of the floating point additions will be the same in both runs. The collective logic has a floating point front end unit that computes the maximum exponent over all inputs. The ALUs operate at link speed, 4 B per network cycle. As the combined packet emerges from the ALUs it enters a floating point back end unit that formats the maximum exponent and combined mantissa into an IEEE compliant floating point number. When NaNs or integer overflows are generated by the collective logic, a maskable interrupt bit is set on the node and a flag bit that trails the packet is set so that all nodes in the class route can be informed of the exception.

Reductions occur at near link bandwidth, up to 86% of the raw link bandwidth. Floating point reductions add an average of 6 network cycles (12 ns) to the per hop point-to-point latency (9 cycles uptree and 3 downtree). The hardware latency for a short allreduce on a 16x16x16x12x2 BG/Q is expected to be about 6.5 μs.

### D. Barrier Support

To further reduce the latency of barriers, BG/Q implements special barrier packets and logic. 16 barrier classes can be set up, similar to the collective classes. A global OR of the inputs of each class is performed on each node and when that changes, a barrier packet is sent up (or back down) the tree. The hardware latency for a barrier on a 16x16x16x12x2 BG/Q is expected to be about 6.3 µs. Barriers are initiated via writes to a memory mapped IO register in the MU and completion is detected by an MU memory mapped IO read. This logic can also be used to support global interrupts across the network.

### E. Network Router Arbitration

The network router logic implements a distributed arbitration mechanism. Each sender broadcasts its link available and token (free buffer space in the neighboring node's receiver) available signals for each virtual channel to all receivers and injection FIFOs. Each point-to-point VC in a receiver then selects a packet from the queues to serve and sends an arbitration request to the receiver's main arbiter when the outgoing sender's link and tokens are available. The receiver main arbiter picks a winner from all requesters and forwards the arbitration request to a sender. A point-to-point injection FIFO arbitrates similarly to a receiver, but only services one packet at a time.

Collective arbitrations are handled by the central collective logic. When a collective packet arrives at a receiver's collective VC or at a collective injection FIFO, a request is raised to the central collective logic until it is granted. The central collective logic has separate up-tree and down-tree arbitration logic and gives priority to system collective operations.

The sender arbitration logic gives grants in the following priority order. Firstly, grants are given to collective or system point-to-point requests. Within collective requests, priority is given to down-tree broadcasts. Secondly, grants are given to user high priority point-to-point requests. Thirdly, the lowest priority is given to normal user (dynamic and deterministic) traffic. For point-to-point traffic within each priority class, the ratio of grants for cut-through traffic from receivers to grants for injections can be programmed via a control register. Short packets for barriers and link level protocols can be sent in between data packets when needed.

### F. Performance Counters, Protocols, RAS and Physical Design

The network maintains a number of programmable performance counters: four per sender and two per receiver. The sender counters can be used to measure link utilization, aggregating a number of VCs onto different performance counters. For example, link utilization due to user point-to-point, system point-to-point, user collective and system collective traffic can be simultaneously and separately counted. On the receiver side, the number of packets (programmable over VCs) and the time integral of the number of packets in queue can be counted. This permits measuring the mean packet queue length, and by Little's Law, $L=\lambda W$, the mean waiting time in a receiver can be inferred (*e.g.* [7]).

A standard, similar to BG/L, link level protocol is used. As packets are sent on a link a copy is stored in a retransmission FIFO for later retransmission if an appropriate acknowledgement is not received within a programmable timeout. Because of 8/10 encoding that occurs when packets go over optical links (between midplanes), a single bit error on a fiber can result in an 8 bit error burst. Two such errors in the same packet can exceed the guaranteed detection properties of most CRCs. As a result, symbol oriented Reed-Solomon (RS) codes are used instead of CRCs and appended to the end of a packet. There is a fixed 10 bit RS code word that covers the unchanging part of a packet (e.g., excludes the link level sequence number) and follows the packet all the way through the network. In addition there are five 10 bit per hop RS code words that cover the entire packet. This code is capable of detecting any five symbol errors and has an escape rate of $2^{-50}$. There are also extensive error consistency checks on the packet's network header, including an 8 bit Hamming code that detects any three bits of error in the header. Like BG/L, there are CRCs that cover all packets sent and all packets received over a link. At the end of a run the paired sender and receiver CRCs must be equal; else there was a packet level RS code escape.

All data paths and internal buffers (VC, injection, reception, retransmission) are covered by single bit correct, double bit detect ECC codes. Most other critical latches, such as state machine latches, use hardened latches and have parity detection.

The network logic comprises just 3% of the chip's area and is 85% clock-gated to save power. The SerDes consumes an additional 4% of the chip area.

### G. Link Chip and Partitioning

As mentioned earlier, each board has a number of link chips (9 per 32-node board). These link chips provide support for the optical modules, including encode/decode, fiber sparing and on-the-fly single bit error correction using an IBM 8B/10B-P code [8]. This code is a modification of the standard 8B/10B code with Hamming distance two. The parity of several data lanes is also encoded and sent over a spare fiber. If a single bit error occurs on a fiber an invalid symbol is detected upon reception. If one and only one single bit error occurs over the parallel 8 bit words, the parity can be used to correct it. Error counters on the link chip can detect when fibers are going bad; the bad fiber can be replaced by a spare with only minimal application impact.

For partitioning, the link chips can be programmed to loop a midplane's links back into the midplane, to the next midplane, or individual links can be held in reset. This makes it impossible for packets to cross from one partition to another. In a dimension, if the full machine is of length n nodes, then a torus is obtained when the partition size is either n or 4 nodes. Otherwise, the partitioned dimension is a mesh.

## III. MESSAGE UNIT

### A. *Overview*

The message unit, as shown in Figure 2. provides the interface between the network and the BG/Q memory system. It is designed to provide low latency and high throughput, enough to keep all the links busy. The MU provides similar functionality to the BG/P DMA, supporting direct puts (RDMA write), remote gets (RDMA read) and memory FIFO messages. The MU maintains pointers to memory for up to 544 injection memory FIFOs and 272 reception memory FIFOs. Cores initiate messages by placing a 64 B descriptor into a slot of an injection memory FIFO and updating that FIFO's tail pointer. A message may target one or more network injection FIFOs; each packet of the message is placed into one of the specified network injection FIFOs. The MU packetizes messages and provides for simple address translation on the reception side. Messages can have arbitrary byte alignment and incoming packets can optionally cause processor interrupts. The MU exploits the BG/Q L2 atomic functionality, so that message byte counters are updated in memory via atomic increments. In comparison, BG/P is limited to only 256 (memory mapped) message byte counters that are stored in the DMA. The MU can also transfer data to/from memory, performing an atomic operation during the transfer. This may be useful for locks, work queues, packet flow control, etc. Whereas the BG/P DMA has two engines, one for injecting packets and one for receiving packets, the MU has a multitude of engines; one injection Messaging Engine (iME) for each network injection FIFO and one reception Messaging Engine (rME) for each network reception FIFO. The iMEs and rMEs share multiple master ports on the BG/Q memory system crossbar switch. The master port bandwidth is sufficient to support all 10 (user mode) torus links of the network when the messages fit in L2. Early VHDL logic simulation measurements show that up to 97.8% of the peak effective data utilization of the links is obtained for a full 5D nearest neighbor exchange. These simulations have been confirmed by hardware measurements. The MU also has one slave port to the crossbar switch and thus provides memory-mapped addresses for software to update FIFO pointers, set up address translation, and handle certain interrupt conditions.

The MU has extensive logic and checks to separate user from system traffic, and to prevent user-space errors from interfering with system messaging. All internal buffers and data paths in the MU are ECC protected, providing very high resistance to soft errors.

### B. *Interface to Local Memory System*

#### 1) *Master ports*

The MU contains multiple master ports that communicate with 16 L2 blocks (L2 slices) via a crossbar switch. All MU reads and writes pass through the globally shared L2 cache and the MU depends on the L2 to manage coherency across the node's memory system. The master port bandwidth is sufficient to keep all network links simultaneously busy. The master ports are shared by requestors consisting of the iMEs, the rMEs, and the message descriptor fetch logic. Although the switch and master ports have separate read and write data paths (thus read data can return while write data is being sent), the crossbar can only schedule one read or one write request per cycle per master port. Each of the requestors is fixed to one of the master ports in an attempt to load balance evenly across the master ports. Furthermore, each requestor assigns a priority to its load/store request, which can be either system (highest), user high-priority (medium), or user normal-priority (lowest). The master port pseudo-randomly selects one requestor per cycle while obeying the priorities. So it is possible for system requests to block user requests but not vice-versa. System message descriptor fetches are assigned system priority while user message descriptor fetches are given a higher priority than normal injection and reception traffic so that messages are always queued up to inject and iMEs are not idle waiting for message descriptors to return from memory.

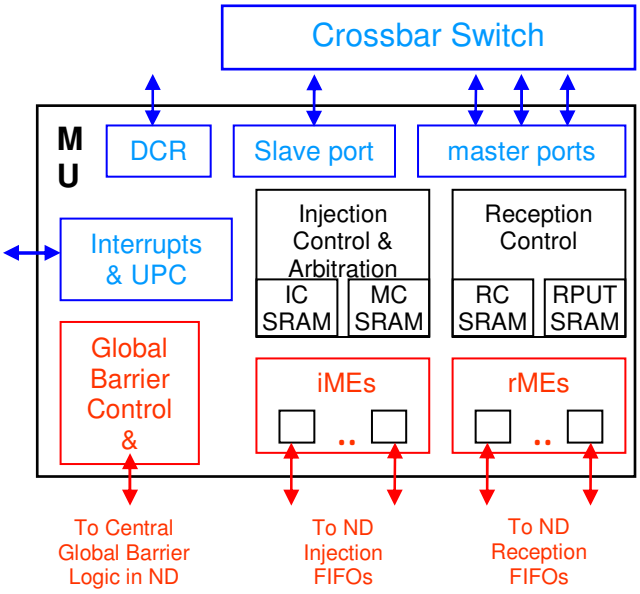


Figure 2. The BG/Q Message Unit (MU) Logic

The master ports were optimized to reduce messaging latency. There is a fast look-up-table (LUT) that responds to requests that have been scheduled by the switch. This LUT contains 16 slots, one slot per L2 slice, and is typically fed by a slower request FIFO that preserves request ordering for a given L2 slice target. Requests to different L2 slices are allowed to proceed out-of-order. In the case that an empty slot exists in the LUT and a new request arrives targeting that slot, it is allowed to immediately populate that slot and bypass the FIFO provided that the FIFO does not also have a queued request targeting that same slot. The master port can buffer multiple requests for a given destination L2 slice, but crossbar scheduling and latency is best when striding accesses across the 16 different L2 slices.

Read data returns from the switch at a rate of 32 B per cycle in either 32, 64, or 128 B chunks. The data is immediately passed to the appropriate network injection FIFO. This implementation provides low-overhead and high bandwidth message injection.

#### 2) *Slave Port*

The MU contains one slave port that communicates exclusively with the cores at 800 MHz via the crossbar switch. Software uses the slave to write and read the MU's memory-mapped IO (MMIO) registers and SRAM locations. All MMIO accesses are 8 B wide. MMIO includes injection and reception FIFO pointers, RDMA write base addresses (RDMA write base addresses are used for memory translation as will be described below), and various FIFO configuration registers. Unlike the master port, slave operations are required to complete in-order of arrival to ensure proper order of operations, *e.g.* one does not want an MMIO store to enable an injection FIFO prior to the preceding MMIO stores that set up the FIFO pointers. Bandwidth is less of a concern for the slave port, and thus only one slave port is necessary for the MU. The MU slave communicates directly with the memory-mapped MU subunits, such as the injection, reception, and RDMA write control logic, as well as various memory mapped registers. Only one subunit can be accessed via the slave at a time, however, those subunits can be simultaneously processing messaging traffic.

### *C. Message Injection*

On the sending side, software injects "descriptor" data structures, defining the messages to be sent, into one or more in-memory injection FIFOs (injection memory FIFOs, imFIFOs). The MU hardware processes each descriptor by splitting and packaging the message into network packets, and injecting those packets into the network injection FIFOs. A descriptor contains 64 B, including a 32 B packet header template used by the MU to construct the header of each packet of the message, a pointer to the payload of the message in the local memory, the number of bytes in the message, interrupt indicators, and an injection map indicating which network injection FIFOs can be used to send the packets. The MU will route each packet to one of the network injection FIFOs with sufficient free space. Thus if the map permits multiple network injection FIFOs, packets from the same message may be placed into different network injection FIFOs. The MU supports local memory copy, where the MU copies a message to another area in the local memory. This feature is implemented using local loopback network injection FIFOs. The MU supports prefetch into L2, where the MU reads message data via a master port to load it into L2, but does not send it to the network.

The injection control logic arbitrates the next message descriptor to fetch for message injection into the network. The injection control logic is connected to the MU slave and one of the MU master ports. It contains the injection control SRAM (ICSRAM) which stores start, size, head, and tail pointers for each of the 544 injection memory FIFOs (imFIFOs). An imFIFO is a circular buffer in memory. The ICSRAM also stores the free space remaining in each imFIFO and a count of the descriptors that have been injected from the corresponding imFIFO. When free space in any imFIFO exceeds a programmable threshold level, software can be notified by a maskable threshold crossing interrupt that fires on a per-imFIFO basis.

To send a message, software copies a descriptor into an imFIFO at the location pointed to by the tail, and then moves the tail past the descriptor. The MU sees that the imFIFO is non-empty, and fetches descriptors starting at the location pointed to by the FIFO's head pointer. Once the message associated with a descriptor has been sent, the MU moves the head past that descriptor and begins to process the next descriptor, until the head and tail pointers are equal and the imFIFO is empty

While the descriptors in a given imFIFO are processed sequentially, descriptors that are injected into different imFIFOs are processed in parallel, hence the benefit of using multiple imFIFOs. There are enough imFIFOs so that each processor thread can have its own set of FIFOs, eliminating the need to acquire locks.

Arbitration of descriptor fetches is performed by pseudo-randomly selecting one winner from the set of non-empty imFIFOs, while observing the necessary priorities. System imFIFOs will always be selected first, followed by user high-priority imFIFOs, followed by normal priority imFIFOs. Descriptor fetch requests are queued prior to entering the master port. This allows for descriptor fetch arbitration to continue if the master port is busy servicing another requestor. It is expected that at times the corresponding master port will be very busy servicing descriptor fetches (e.g. software could enable up to 32 imFIFOs in one cycle), so this master port is purposely lightly loaded with other requestors.

The message control SRAM (MCSRAM) holds descriptors fetched by the injection control logic for further processing by iMEs. It can store up to one descriptor for each imFIFO, and up to 544 descriptors in total. Subsequent message arbitration logic assigns one of the descriptors available in MCSRAM to an idle iME for further processing. It selects a descriptor according to the three-level priority (system, high priority user, normal user) associated with the originating imFIFO. When there are multiple descriptors that have the same highest priority, one of them is selected randomly. This arbitration process is performed for each packet.

An iME builds a packet for the assigned descriptor and stores it in the paired network injection FIFO for transmission. It issues a request to a master port to fetch a portion of the message data and puts it into the packet payload. It selects an optimal master port request size out of the available sizes (32 B, 64 B, and 128 B), subject to the address alignment requirements for each size. To support arbitrary alignment of messages being sent, the iME issues an extra 32 B read for non 32 B aligned payload data to cover the packet's entire payload. As the packet flows out of the network injection FIFO, the network logic packs the payload contiguously, dropping all leading bytes not in the payload and zero filling the packet's trailing bytes. The iMEs can process different message descriptors independently in parallel. Thus the MU can inject packets into multiple network injection FIFOs in parallel, keeping the network links busy.

### *D. Message Reception*

On the reception side, the packets arrive in the network reception FIFOs. The MU receives the packets and writes them into the appropriate location in the memory system. The MU distinguishes between three different types of packets, and accordingly performs different operations:

- Memory FIFO - The MU hardware copies the packets from the network reception FIFOs into one of the in-memory reception FIFOs (reception memory FIFOs, rmFIFOs) as specified in the packet. Software polls the rmFIFOs and processes the received packets. Optionally the MU can raise an interrupt when the last packet of a memory FIFO message has been received and stored to the local memory.
- RDMA write - The MU hardware copies the payload from the network reception FIFOs directly into the predetermined location. After storing the payload, the MU updates a reception byte counter in the local memory system, via an L2 atomic operation. The write locations for the data and byte counter are specified in the packet. Software polls this byte counter to detect message completion.
- RDMA read - The MU hardware treats the packet payload as one or more descriptors and injects those descriptors into a designated imFIFO so they can be processed by the injection MU hardware. The imFIFO is specified in the packet. In a typical usage, these descriptors generate RDMA write packets to transmit the desired data back to the sender node of the RDMA read packet.

An rME reads out a received packet from the paired network reception FIFO and stores it in the local memory, subject to the semantics described above for each packet type. The rMEs can operate independently in parallel. Thus the MU can read out multiple network reception FIFOs in parallel, keeping the network links busy. The rMEs pull data from the network reception FIFOs at 4 B per 800 MHz cycle, which is faster than the network can fill the reception FIFOs (4 B per 500 MHz network cycle). The rMEs buffer and align these data before writing to the master ports.

An rmFIFO is a circular buffer in memory with head and tail pointers held and managed by Reception Control SRAM (RCSRAM). RCSRAM supports up to 272 rmFIFOs. When a memory FIFO packet arrives, the MU copies the packet at the tail of the rmFIFO designated in the packet, and moves the tail past the received packet. Optionally RCSRAM raises an interrupt when the free space in an rmFIFO falls below a specified threshold as a result of receiving the new packet. When software sees that the tail has moved (via polling), software processes the packet (*e.g.* copies the payload to a user buffer), moves the head past the packet, and continues processing packets until the head and tail pointers are equal and the rmFIFO is empty.

There are enough rmFIFOs so that each hardware thread can have its own set of FIFOs, eliminating the need to acquire locks for FIFOs. Note that dynamically routed packets from the same message may arrive in a different order from which they were sent and the MU makes no attempt to re-order such packets upon reception.

To allow multiple rMEs to process packets targeting the same rmFIFO at the same time, RCSRAM maintains a shadow tail, besides the regular tail visible from the processors. This shadow tail points to the location up to where the FIFO space has been reserved for some rMEs to store packets. It is different from the regular tail pointer which points to the location up to where packets have been completely stored. When an rME starts to receive a new memory FIFO packet, it reads the shadow tail, and atomically increments it by the size of the packet for the next rME. Then the rME stores the packet to the location pointed to by the original shadow tail. After storing the packet, the rME increments the regular tail past the received packet, if the following condition is satisfied. When an rME atomically gets and updates the shadow tail, a unique sequence number is assigned to maintain ordering among the rMEs. Using this sequence number, the RCSRAM enforces that an rME updates the regular tail in the same order as it obtained the shadow tail. This algorithm ensures that the regular tail always contains the correct value.

For virtual to physical memory translation on RDMA write transfers, the receiver uses a base address table (BAT) stored in the MU to determine the target address. The BAT holds up to 544 base addresses. The packet header contains one BAT ID to indicate which base address is being used to locate the memory to receive the message, and another BAT ID to indicate which base address is being used to locate the reception counter. These base addresses are added with the "put offset" and "counter offset" in the packet header to determine the memory location for storing payload data and updating the counter, respectively. This is similar to the translation mechanism on BG/P and avoids the complexity of page table translations. These offsets are computed by the MU as packets are injected into the network, based on initial offsets specified in the message descriptor. Software protocols are used to set up the correct translation base address tables.

An RDMA write transfer uses a reception counter to track how much of the message has been received. The reception counter is located anywhere in the local memory. In a typical usage, its value is initialized by software to the number of bytes in the message. As the message data is received, the MU decrements the counter's value via an L2 atomic operation. When the counter's value hits zero, the entire message has been received. The memory location can be occupied by the reception counter during the message transfer, and can then be reused for any other purpose afterwards.

### *E. DCR & UPC*

The MU's Device Control Register (DCR) unit provides a convenient way to access the configuration and interrupt registers. The DCR unit connects via a DCR slave interface to a DCR ring that circumnavigates the BG/Q compute chip and is clocked at 100 MHz to save power.

The MU also contains a universal performance counter (UPC) module that connects directly to a UPC ring. The UPC module provides useful performance counts of packets and messages injected and received, master port throughput (can be calculated using Little's Law), and slave port read and write accesses. Other counts of interest, such as message descriptors fetched per imFIFO are accessible via MMIO.

### F. Global Barrier

The global barrier (and interrupt) network is embedded in the BG/Q torus and is accessed via the MU. A barrier is initiated by writing a memory mapped control register with the MU slave. The network notifies the MU of barrier completion by writing a MMIO barrier status register in the MU that can be polled by software through the MU slave port. The barrier operation does not require injection of descriptors to imFIFOs.

### G. Software error, SER, and Physical Design

The MU has extensive logic to protect the system from user program errors. For example, the master ports have a set of range registers to protect the local memory system. These registers specify which address ranges in the local memory space are accessible by the MU. Each request to a master port is checked and rejected if it does not fall in a valid address range. The MU also checks system/user permission where applicable, to prevent a user program from affecting system software.

All internal buffers and data paths in the MU are ECC protected, providing very high resistance to soft errors. Most other latches are protected with parity for single bit error detection. Important state machines and control flags are implemented using hardened latches. Upon detecting an uncorrectable bit error, each submodule will stop operating to prevent the bit error from propagating to the network or local memory system.

The MU reports error conditions via DCR interrupt registers. They are individually maskable and are consolidated (programmably) into three groups, critical, non-critical, and machine check interrupts.

The MU is only 1% of the total chip area and is over 92% clock gated to save power.

## IV. Software Support

The BG/Q system's software provides highly optimized C inlines via the System's Programming Interface (SPI) for applications, message layer libraries, kernels and system's software to program the MU and the Torus interconnect via the MU. The SPIs are a thin software abstraction layer of the BG/Q network and MU hardware. They are flexible and a very large number of message passing and resource allocation algorithms can be implemented using the SPIs. For example, a message layer library can implement point-to-point and collective algorithms optimized for the semantics of MPI and PGAS runtimes, while an application that requires different resource management or different point-to-point or collective algorithms can directly program to the SPIs for optimized performance. In addition, kernels such as the Blue Gene Compute Node Kernel can program the MU for function shipping I/O system calls to the I/O nodes.

The BG/Q low level messaging software is itself layered as shown in Figure 3.

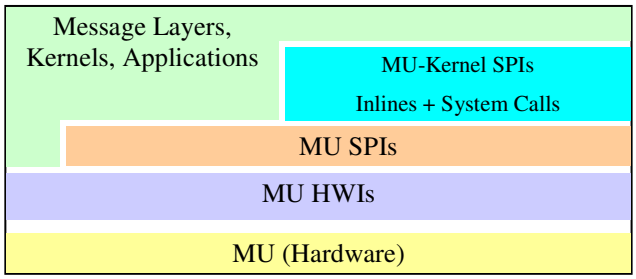


Figure 3. SPI and HWI components.

In the top layer are the various message layers, kernels, and applications that wish to send and receive messages using the low-level messaging software, either using SPI or directly programming through the Hardware Interface (HWI). The message layer software typically supports initialization and management of network resources and implements point-to-point and collective algorithms. High level libraries (not shown in figure) such as MPI are being built on top of message layers.

An MU HWI (Message Unit Hardware Interface) layer defines the structures and bits used by the MU SPIs to manipulate the MU hardware. Low level messaging users use some of these definitions to describe the messages being sent.

There are two SPI layers providing the majority of the interfaces:

1. MU-Kernel SPIs: This layer provides kernel neutrality to the SPIs. These interfaces allocate, initialize, configure, and free MU resources such as injection FIFOs, reception FIFOs, base address table entries and collective class routes. The implementation of these functions is kernel-dependent, but the interfaces are standard across kernels. The goal behind standardizing on the SPIs across kernels is so a user, to the extent possible, does not have to modify his or her application to run on the different kernels. Release 1 of the BG/Q software stack will contain a Compute Node Kernel (CNK) implementation of the kernel SPI. As the MU descriptors require physical addresses of source buffers, the Kernel SPIs also define interfaces to translate virtual addresses to physical addresses.
2. Message Unit (MU) SPIs: These interfaces build message descriptors for the message being sent, inject descriptors to initiate message send operations, receive message packets, and check for message completion. They can be used by all low-level messaging users regardless of whether they are user-mode applications or kernel-mode system code.

Applications or users of the SPIs must use the kernel interfaces to allocate resources and then build descriptors to define the communication patterns and then inject descriptors to initiate communication. There are several descriptors to define FIFO, remote put (RDMA write) and remote get (RDMA read) operations for point-to-point and collective communication.

Completion of sent messages is detected by observing the descriptor counter in the MU injection FIFO. The MU

hardware increments a 64 bit counter after the all the data for a descriptor has been injected on the network. On the receiver, reception FIFOs can be polled to receive packets or counters reaching zero can notify completion of RDMA reads to the application.

For collective communication operations, a class-route that describes the nodes in the collective must be configured via the class route management SPIs on all the participating nodes before the collective operation is initiated. For example, the MPI library can configure class routes in the MPI_Comm_split call to use network acceleration on the sub-communicator. Similarly to point-to-point messages, broadcast and allreduce messages are initiated by injecting collective descriptors into an injection FIFO. The different target buffers in a broadcast or an allreduce can have different alignments. A base address table can be setup to store the buffer address and the descriptor can have a single offset of 0 B on all the participating nodes.

## V. Performance Measurements

We present performance results on up to 512 BG/Q pass 2 prototype nodes. We developed simple C language benchmarks such as ping-pong, nearest neighbor, broadcast and allreduce using the SPI calls and ran them on the prototype hardware.

### A. *Short Message Latency*

Ping-Pong SPI benchmark results using direct put point-to-point transfers are presented in Table 1. These results show increased latency as the network hops increase. It also shows a breakdown of where the time is being spent. The hardware delay is time spent by the network and MU performing the transfer (after injecting the descriptor until an indication of the arrival of the message occurs). The software delay is time spent by software injecting the descriptor, monitoring for completion, and minor loop control.

Performance results show that the hardware latency varies from 0.62 µs at 1 network hop to 1.17 µs at 13 network hops. Software adds about 95 ns to 100 ns overhead. The delta hop delay should be the expected 40 ns average delay through a node plus card trace length, cable and linkchip delays. Thus the actual measured delay depends on the path taken between the nodes. In the measurements of Table 1, the ping node is always (0,0,0,0,0), and pong node coordinate increases in the order of A, B, C, D and E. There are no linkchips involved in any path. The average latency (the 13 hop minus 1 hop latency divided by 12 hops) is 45.3 ns. In a large 20 PFlops peak BG/Q system, the average number of hops is 15.5 for random or all-to-all communication patterns. In a multiple midplane system there will be additional delays through linkchips and optical fibers.

Table 1 Ping-Pong Latency on a 512 Node Mesh

| Network Hops | Hardware Latency (ns) | Delta Hop Delay (ns) | Hardware+Software Latency (ns) |
|---|---|---|---|
| 1 | 622 | | 718 |
| 2 | 671 | 49 | 769 |
| 3 | 713 | 41 | 813 |
| 4 | 760 | 47 | 856 |
| 5 | 808 | 48 | 906 |
| 6 | 849 | 41 | 950 |
| 7 | 891 | 42 | 989 |
| 8 | 940 | 49 | 1038 |
| 9 | 981 | 41 | 1080 |
| 10 | 1022 | 41 | 1122 |
| 11 | 1069 | 47 | 1166 |
| 12 | 1118 | 47 | 1216 |
| 13 | 1166 | 48 | 1264 |

### B. *Nearest Neighbor Throughput*

We ran an SPI benchmark to measure the achievable throughput to nearest neighbors on the 5D torus on a single node, where the links are looped back. For example the A+ direction is looped back to the A- direction, so all out-going data on the A+ direction arrives on the A- links and vice versa. The benchmark allocated 10 injection FIFOs for the 10 different directions on a 5D torus. A RDMA write descriptor is injected to initiate data movement on the looped back links.

Table 2 Nearest Neighbor Link Send+Receive Throughput

| Message Size | Throughput (GB/sec) | % of Raw Link Bandwidth | % of Effective Peak Data Utilization |
|---|---|---|---|
| 4 KB | 17.0 | 42.5 | 47.2 |
| 8 KB | 23.0 | 57.5 | 63.9 |
| 16 KB | 27.9 | 69.8 | 77.5 |
| 32 KB | 31.3 | 78.3 | 86.9 |
| 64 KB | 33.3 | 83.3 | 92.5 |
| 128 KB | 34.2 | 85.4 | 94.9 |
| 256 KB | 34.9 | 87.1 | 96.8 |
| 512 KB | 35.2 | 88.0 | 97.8 |
| 1 MB | 35.4 | 88.4 | 98.3 |
| 2 MB | 26.4 | 66.0 | 73.3 |
| 4 MB | 24.7 | 61.8 | 68.6 |
| 256 MB | 25.0 | 62.5 | 69.4 |

Table 2 shows the cumulative nearest neighbor link throughput; send plus receive bandwidth for all 10 links. At 1 MB, the cumulative throughput achieved on the 10 links in both directions is 35.4 GB/s which is an efficiency of 88.4% of the raw link throughput and 98.3% of the peak 90% effective data utilization of the links. This demonstrates the effectiveness of the highly parallel network and MU design. When message sizes are less than or equal to 1 MB, all 20 send plus receive streams fit in the L2 and there is sufficient memory bandwidth between the L2 and the MU. 2 MB and larger message sizes spill L2 and degradation in throughput is observed. The achievable bandwidth is limited by the DDR memory interface instead of the MU and the network.

### C. *All-to-All performance*

We ran an SPI-based all-to-all performance test where each node sends a total of (n-1) messages, one to each of other (n-1) nodes in the system. On a 512-node prototype, with 4 KB message size and dynamic routing, we obtained all-to-all performance result at 95% of the theoretical peak. The performance improved to 97% of peak for 32 KB message sizes.

### D. *Collective communication performance*

We ran collective latency and throughput benchmarks on up to 512 nodes for floating point add reduction using direct put messages.

Table 3 Collective Latency

| Number of Nodes | Total Hops (Round Trip) | Latency (ns) | Per Hop Delay (ns) |
|---|---|---|---|
| 2 | 2 | 641 | |
| 4 | 4 | 742 | 50.5 |
| 8 | 6 | 876 | 67 |
| 16 | 8 | 984 | 54 |
| 32 | 10 | 1099 | 57.5 |
| 64 | 12 | 1203 | 52 |
| 128 | 14 | 1321 | 59 |
| 256 | 16 | 1443 | 61 |
| 512 | 18 | 1558 | 57.5 |

Table 3 shows the latency of 8 B collective floating point add operations. The latency varies from 0.64 µs on 2 nodes to 1.56 µs on 512 nodes. The average per hop latency (16 hop minus 2 hop latency divided by 14 hops) is 57.2 ns, which is just over 12 ns (6 network cycles) greater than the average ping-pong latency, in close agreement with earlier VHDL simulation measurements. Due to measurement variation, different paths, and somewhat different software used in the ping-pong and collective tests, direct comparisons between the collective and ping-pong latencies are not meaningful for a small number of hops.

Table 4 Collective Throughput

| Message Size (KB) | Throughput (GB/sec) | % of Raw Link Bandwidth | % of Effective Peak Data Utilization |
|---|---|---|---|
| 0.5 | 0.26 | 13.0 | 14.4 |
| 1 | 0.45 | 22.5 | 25.0 |
| 2 | 0.72 | 36.0 | 40.0 |
| 4 | 1.01 | 50.5 | 56.1 |
| 8 | 1.27 | 63.5 | 70.6 |
| 16 | 1.45 | 72.5 | 80.6 |
| 32 | 1.57 | 78.5 | 87.2 |
| 64 | 1.64 | 81.7 | 90.8 |
| 128 | 1.67 | 83.5 | 92.8 |
| 256 | 1.69 | 84.4 | 93.7 |
| 512 | 1.70 | 84.9 | 94.3 |
| 1024 | 1.70 | 85.1 | 94.6 |

Table 4 shows the throughput of long collective floating point add operations measured on 512 nodes. Due to more complex logic for collective sums the percent of peak is slightly lower for collective communication than point-to-point communication.

## VI. CONCLUSION

This paper is the first description of the IBM Blue Gene/Q interconnection network and message unit. The 5D torus network with associated SerDes, and the highly parallel message unit, which provides the functionality of a network interface, are integrated onto the same chip as the processors and cache memory, and consume 8% of the chip's area. BG/Q integrates both collective and global barrier functions into the torus network. A thin software layer enables applications to achieve low-latency and high-throughput. For nearest neighbor exchange using an SPI benchmark measured on real hardware, best case throughput achieved on the 10 links in both directions is 35.4 GB/s which is an efficiency of 88.4% of the raw link throughput and 98.3% of the peak 90% effective data utilization of the links. This demonstrates the effectiveness of the highly parallel network and message unit design.

## ACKNOWLEDGMENTS

The Blue Gene project is a team effort. We especially thank the following people, whose contributions have touched the network and MU in a variety of ways: R. Bellofatto, P. Boyle, T. Bright, G. Chiu, P. Coteus, M. Giampapa, T. Gooding, R. Haring, M. Kaufmann, G. Kopcsay, K.H. Kim, L. Lastras, J. Marcella, M. McManus, T. Musta, B. Nathanson, M. Ohmacht, B. Rosenburg, K. Sugavanam, T. Takken, J. Van Oosten, A. Mamidala, J. Brunheroto, G. Dozsa, D. Miller, M. Blocksome and B. Smith.

The Blue Gene/Q 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. We acknowledge the collaboration and support of Columbia University and the University of Edinburgh.

## REFERENCES

[1] The IBM Blue Gene Team. "The Blue Gene/Q Compute Chip," Presented at the Hot Chips Conference 23, August 17-19, 2011.

[2] A. Gara et. al, "Overview of the Blue Gene/L system architecture," IBM Jorunal of Research and Development, vol 49, no. 2/3, pp. 195 – 212, March/May 2005.

[3] The Blue Gene/P Team. "An overview of the BlueGene/P project," IBM Journal of Research and Development, vol. 52, no. 1/2, pp. 199 – 220, January/March 2008.

[4] M.A. Adiga et. al, "Blue Gene/L torus interconnection network," IBM Jorunal of Research and Development, vol 49, no. 2/3, pp. 265 – 276, March/May 2005.

[5] 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 274 International Conference on Parallel Processing, September 1999, pp. 58 – 67.

[6] S. Kumar, Y. Sabharwal, R. Garg and P. Heidelberger, Optimization of All-to-all communication on the Blue Gene/L supercomputer, In Proceedings of International Conference on Parallel Processing (ICPP), Portland, Oregon, 2008.

[7] L. Kleinrock, Queueing Systems Volume II: Computer Applications. New York: Wiley, 1976.

[8] A.X. Widmer, "Transmission code having local parity", IBM US Patent 5,699,062, December 1997.