RC23025 (W0312-022) December 3, 2003
Computer Science

IBM Research Report

Design and Analysis of the BlueGene/L Torus Interconnection Network

M. Blumrich, D. Chen, P. Coteus, A. Gara, M. Giampapa, P. Heidelberger, S. Singh, B. Steinmacher-Burow, T. Takken, P. Vranas
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Design and Analysis of the BlueGene/L Torus Interconnection Network

M. Blumrich, D. Chen, P. Coteus, A. Gara, M. Giampapa, P. Heidelberger,
S. Singh, B. Steinmacher-Burrow, T. Takken, P. Vranas
IBM T.J. Watson Research Center
Yorktown Heights, New York

Abstract

BlueGene/L (BG/L) is a 64K (65,536) node scientific and engineering supercomputer that IBM is developing with partial funding from the United States Department of Energy. This paper describes one of the primary BG/L interconnection networks, a three dimensional torus. We describe a parallel performance simulator that was used extensively to help architect and design the torus network and present sample simulator performance studies that contributed to design decisions. In addition to such studies, the simulator was also used during the logic verification phase of BG/L for performance verification, and its use there uncovered a bug in the VHDL implementation of one of the arbiters.

1. Introduction

BlueGene/L (BG/L) is a scientific and engineering, message-passing, supercomputer that IBM is developing with partial funding from the U.S. Department of Energy Lawrence Livermore National Laboratory. A 64K node system is scheduled to be delivered to Livermore, while a 20K node system will be installed at the IBM T.J. Watson Research Center for use in life sciences computing, primarily protein folding. A more complete overview of BG/L may be found in [1], but we briefly describe the primary features of the machine.

BG/L is built using system-on-a-chip technology in which all functions of a node (except for main memory) are integrated onto a single ASIC. This ASIC includes two 32-bit Power PC cores (the 440); the 440 was developed for embedded applications. Associated with each core is a 64-bit "double" floating-point unit (FPU) that can operate in SIMD mode. Each (single) FPU can execute up to two multiply-adds per cycle, meaning that the peak performance of the chip is 8 floating-point operations per cycle. Each 440 has its own instruction and data caches (each 32KB), a small L2 cache that primarily serves as a pre-fetch buffer, a 4MB shared L3 cache built from embedded DRAM, and a DDR memory controller. In addition, the logic for five different networks is integrated onto the ASIC. These networks include a JTAG control and monitoring network, a Gbit Ethernet macro, a global barrier and alert network, a "tree" network for broadcasts and combining operations such as those used in the MPI collective communications library, and a three dimensional torus network for point-point communications between nodes. This paper will focus on the torus network.

The ASIC can be used as either an I/O node or as a Compute node. I/O nodes have their Ethernet macro connected to an external switch enabling connectivity to hosts, however they do not use the torus network. Compute nodes do not connect their Ethernet, and talk to the I/O nodes over the tree network. The Livermore machine will have 64 Compute nodes for each I/O node. I/O nodes will have at least 512MB and Compute nodes will have at least 256 MB of memory, depending on the cost of memory at the time of delivery.

Because of the high level of integration and relatively low target clock speed (700 MHz target), the system is designed to deliver unprecedented aggregate performance at both low cost and low power consumption. At this clock rate, each node has a peak of 5.6 GFlops, while the 64K node system has a peak of 367 Tera Flops. Each ASIC will consume only 12 watts of power. Because of the low power, a very high density of packaging can be achieved. Two compute ASICs and their associated memory are packaged onto a compute card, 16 compute cards are mounted on a node card, and 16 node cards are packaged in a 512 node midplane. Two midplanes are packaged in a 1024 node rack, which is about the size of a large refrigerator.

Because the 440 core does not contain shared memory support, the L1 caches of the two cores on the same ASIC are not coherent. Memory is consistent from the L2 on out, but software is required to appropriately manage the L1's. The system can operate in one of two modes. In communications coprocessor mode, one core is responsible for computing while the other core handles most messaging functions. Careful software coordination is required in this mode to overcome the lack of L1 coherence. When configured in this mode, the peak performance of the 64K node system is 183 Tera Flops. In the second mode, "virtual node" mode, each core has its own memory space and each core is responsible for both computing and message handling; the system has two sets of network injection and reception FIFOs, so that both cores can simultaneously access the network interfaces.

2. Torus Network

We now describe the torus network in some detail. Many of the design decisions were driven by simulation performance studies, as will be described in Section 4.

The torus network uses dynamic routing with virtual cut-through buffering [8]. A torus was chosen because it provides high bandwidth nearest neighbor connectivity, which is common in scientific applications, but also for its scalability, cost and packaging considerations. A torus requires no long cables and, because the network is integrated onto the same chip that does computing, no separate switch is required. Previous supercomputers such as the Cray T3E [12] have also used torus networks.

Torus packets are variable in size – from 32 to 256 bytes in increments of 32 byte chunks. The first eight bytes of each packet contain link level protocol information (e.g.,

sequence number) and routing information including destination, virtual channel and size. A 24-bit CRC is appended to each packet, along with a one byte valid indicator. The CRC permits link level checking of each packet received, and a timeout mechanism is used for retransmission of corrupted packets. The error detection and recovery protocol is similar to that used in IBM SP interconnection networks as well as in the HIPPI standard.

For routing, the header includes six "hint" bits, which indicate in which directions the packet may be routed. For example, hint bits of 100100 means that the packet can be routed in the x+ and y- directions. Either the x+ or 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 registers that contain the coordinates of its neighbors, and 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. These hint bits appear early in the header, so that arbitration may be efficiently pipelined. The hint bits can be initialized either by software or hardware; if done by hardware, a set of two registers per dimension is used to determine the appropriate directions. 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 statically (xyz) routed. Besides point-to-point packets, a bit in the header may be set that causes a packet to be broadcast down any dimension. 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 faulty nodes, provided they are not co-linear.

The torus logic consists of three major units, a processor interface, a send unit and a receive unit, as shown in Figure 1. The processor interface consists of network injection and reception FIFOs. Access to these FIFOs is via the double 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 8 injection FIFOs organized into two groups: two high priority (for inter-node OS messages) and six normal priority FIFOs, which are sufficient for nearest neighbor connectivity. Packets in all FIFOs can go out in any direction. Each group of reception FIFOs contains 7 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. Up to six injection and six reception FIFOs may be simultaneously active.

Each of the six receivers, as shown in Figure 1, has four virtual channels (VCs). Multiple VCs help reduce head-of-line blocking [4], but in addition, mesh networks including tori with dynamic routing, can deadlock unless appropriate additional "escape" VCs are provided [5,7]. We use a recent, elegant solution to this problem, the "bubble" escape VC as proposed in [10,11]. BG/L has two dynamic VCs, one bubble escape VC that can be used both for deadlock prevention and static routing, and one high priority bubble VC. Each VC has 1 KB of buffering, enough for four full-sized packets. In addition to the VCs, the receivers include a "bypass" channel so that packets can flow through a node without entering the VC buffers, under appropriate circumstances. Dynamic packets can only enter the bubble escape VC if no valid dynamic VCs are available.

A token flow control algorithm is used to prevent over-flowing the VC buffers. Each token represents a 32B chunk. For simplicity in the arbiters, a VC is marked as unavailable unless 8 tokens (a full-sized packet) are available. 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 [10,11] require that tokens for one full-sized packet are required for a packet already on the bubble VC to advance, but that tokens for two full-sized packets are required for a packet to enter the bubble VC, upon either injection, a turn into a new direction, or when a dynamic VC packet enters the bubble. This rule ensures that buffer space for one packet is always available after an insertion and thus some packet can always, eventually move. However, we discovered that this rule is incomplete for variable-sized packets when our simulator deadlocked using this rule. 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 simply modified so that each packet on the bubble is accounted for as if it were a full-sized (8 chunk) packet.

Eight byte acknowledgement (ack-only) or combined token-acknowledgement (token-ack) packets are returned when packets are either successfully received, or when space has freed up in a VC. Acknowledgements 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.

Due to the density of packaging and pin constraints, each link is bit serial. The torus is internally clocked at one-fourth the rate of the processor, so at the target 700 MHz clock rate, each torus link is 175 MB/sec. There are sufficient internal busses so that each of the 6 outgoing and 6 incoming links can be simultaneously busy; thus each node can be sending and receiving 1.05 GB/sec. In addition, there are two transfer busses (paths) coming out of each receiver that connect with the senders. Thus, a single receiver can have up to 4 simultaneous transfers, e.g., one to its normal reception FIFO, one to the high priority reception FIFO, and two to two different senders.

Arbitration is distributed and pipelined, but occurs in three basic phases. It generalizes an approach used in [3] and represents tradeoffs between complexity, performance, and ability to meet timing constraints. First, each packet at the head of the injection or VC FIFOs decides in which direction and on what VC it prefers to move. For statically routed packets, there is only one valid choice, but dynamically routed packets may have many choices. 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 indicating both link and token availability for each VC in each direction. This bit vector is and-ed with a bit vector of possible moves constructed from the packet's hint bits and VC. This defines the set of possible and available arbitration requests. In addition, the sender provides 2 bits for each VC indicting one of four ranges of available

downstream tokens. Of all the possible and available dynamic direction/VC pairs, the packet selects the one with the most available downstream tokens. Ties are randomly broken. If no dynamic direction/VC combination is available, the packet will request its bubble escape direction/VC pair (if available), and if that is also unavailable, the packet makes no arbitration request. This is a somewhat simplified description since 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.

Second, since each receiver has multiple VC FIFOs (plus the bypass) an arbitration phase is required to determine which of the requesting packets in the receiver wins the right to request. If a high priority packet is requesting, it wins. Barring that, a modified "Serve the Longest Queue" (SLQ) is used, based on 2 bit (4 ranges) FIFO Fullness indicators, i.e., the packet from the most full VC (as measured to within the 2 bits of granularity) wins. However, this cannot always be used since 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).

Third, the receivers and injection FIFOs present their requests to the senders. Note that on a given cycle a receiver will present at most one request to the senders. Thus each sender arbiter can operate independently. The sender gives 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 1.0) 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 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.

3. 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. Due to the potential size of such a model, simulation speed was a significant concern and a proven shared memory parallel simulation approach was selected. In particular, parallel simulation on shared memory machines has been shown to be very effective in simulating interconnection networks (see, e.g., [13]) whereas success with message passing parallel interconnection network simulators is harder to come by (see, e.g., [2]). We also recognized the difficulties in developing an execution-driven simulator such as that in [6] for a system with up to 64K processes, and therefore decided upon a simulator that would primarily be driven by application pseudo-codes, 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, etc. This pseudo-code included a subset of the MPI point to point messaging calls as a workload driver for the simulator. We also extended the IBM UTE trace capture utility that runs on IBM SP machines and were able to use such traces as simulator inputs (for up to several hundreds of nodes).

The basic unit of simulation time is a network cycle, which is defined to be the time it takes to transfer one byte. As BG/L is organized around 512 node (8x8x8) 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 need to be properly synchronized. To deal with this problem, we use a simple but effective "conservative" parallel simulation protocol known as "YAWNS" [9]. 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. Consider an event that is executed during the window (starting at time t) on processor i that is destined to arrive on processor j in the future; such an event represents the arrival of the first byte of a packet. Since the minimum transit time is w, the arrival cannot occur during the current window, represented by the interval [t, t+w-1]. Processor i simply puts a pointer to the event on an i-to-j linked list. When each processor reaches the end of the window, it enters a barrier synchronization. Upon leaving the barrier, each processor is sure that every other processor has executed all events up to time t+w-1 and that all inter-processor events are on the appropriate inter-processor linked lists. Processor j can therefore go through all its i-to-j linked lists, remove events from them, and put the events on its own future event list. Once this is done, the processors can simulate the next window [t+w, t+2w-1]. If w=1, then this protocol requires a barrier synchronization every cycle, however, on BG/L, the minimum inter-midplane delay will be approximately w=10 network cycles. When a large number of BG/L nodes are being simulated, each processor will execute many events during a window, i.e., between barriers, and thus the simulator should obtain good speedups.

The simulator runs on a 16-way IBM "nighthawk" SMP with 64 GB of memory. The model of the torus hardware contains close to 100 resources per node (links, VC token counters, busses, FIFOs, etc), so that a full 64K node system can be thought of as a large queuing network with approximately 6 million resources. It consumes a large amount of memory and runs slowly; a 32K node simulation of 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 nodes, and sometimes achieves superlinear speedup due to the private 8MB L3 caches on the SMP and the smaller per node memory footprint of the parallel simulator.

The model, which was written before the VHDL, 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, a gross simplification was the assumption that the injection FIFOs were of infinite size, and that packets are placed in these FIFOs as early as possible rather than as space frees up in the FIFOs. This has little effect on network response time and throughput measurements during the middle of a run, but can affect the dynamics particularly near the end of runs. The simulator also 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 are modeled to a high level of detail.

4. Sample Performance Studies

In this section, we present some examples of use of the simulator to study design trade-offs 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.

Response Time in Light Traffic: Figure 2 plots 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 so that the average link utilization is less than one. This Figure compares static routing to dynamic routing with one or more dynamic VCs and one or more busses (paths) connecting receivers to senders. Simpler, random, arbitration rules than SLQ and JSQ were used and the plot was generated early in our studies when the target link bandwidth was 350 MB/sec. (The 350 MB/sec. assumption essentially only affects results by a rescaling of the y-axis.) The figure shows the clear benefit of dynamic over static 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 2 VC/2 path to 4 VC/4 path configuration.

Throughput Under Non-Uniform Traffic: Figure 3 plots the throughput, as a function of time, for a 4K node BG/L system under a highly non-uniform traffic pattern. In this pattern, the destinations of 25% of the packets are chosen randomly within a small "hot" contiguous sub-mesh region consisting of 12.5% of the machine. The remaining 75% of the packets chose their destinations uniformly over the entire machine. Again, random arbitration and a 350 MB/sec link speed were used. The figure considers three different buffer sizes for the VC FIFOs: 512B, 1KB, and 2KB. At the beginning of the run throughput (as measured over 10,000 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 the hot region. Measurements during the simulations indicated that those links generally have a mean utilization of around 95%.

Arbitration Policies: Figure 4 plots 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-mention random policy. In light traffic, SLQ provides little benefit (since queues aren't that full) but JSQ does reduce response time in moderate traffic; at 95% link utilization, the average response time is reduced by about 20%. Figure 5 plots the throughput for a 4K node BG/L under the hot region model for the different arbitration policies. While the throughputs of all policies stabilize near the same value, the decline is slowest for the SLQ policy (75% of the cycles 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, non-uniform traffic.

All-to-All: 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. The Figure again shows the benefit of dynamic over static routing. For this pattern, there is marginal benefit in going from 1 to 2 dynamic VCs, but what is important is that the average link utilization is, at approximately 98%, close to the theoretical peak. This peak includes the overhead for the token-ack packets, the packet headers and the 4 byte CRC trailers. A reasonable assumption for the BG/L software is that each packet carries 240 bytes of payload, and with this assumption the plot shows that the payload occupies 87% of the links. Not shown in these plots is the fact that a very low percentage of the traffic flows on the escape bubble VC and that statistics collected during the run showed that few of the VC buffers are full. Three-dimensional FFT algorithms often require the equivalent of an All-to-All, but on a subset of the nodes consisting of either a plane or a line in the torus. Simulations of these communications 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 64x32x32 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%. 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 All-to-All communications pattern on a 32x16x16 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 due to 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 closer to peak performance.

Nevertheless, the above study points out a disadvantage of the torus architecture for asymmetric machines in which the application cannot be easily mapped so as to result in a close proximity communications pattern.

Virtual Channel Architecture: Here we consider several different deadlock prevention escape VC architectures. The first, originally proposed in [5] 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 dimension ordered, the escape VC is x first, then y if no x hops remain, then z if no x or y hops remain. In direction ordered, 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. We again use the hot region workload where the hot region starts at coordinates (0,0,0) and the datelines are set at the maximum coordinate value in each dimension. Figure 7 plots the throughput as a function of time. The dimension ordered dateline pair shows particularly poor and wild behavior, with a steep decline in throughput, followed by a rise and then another steep decline. Figure 8 plots the throughput on a per VC basis for a longer period of time. The decreasing and increasing 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.

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 could 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 a token-ack is returned that frees at least one previously blocked packet to move. Figure 9 plots this unblocking probability along with the throughput as a function of time. 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, the throughput increases and vice-versa.

Performance Verification: To verify the VHDL logic of the torus, we built a multi-node verification testbench. This testbench, which runs on the Cadence VHDL simulator, consisted of workload drivers that inject packets into the injection FIFOs, links between nodes on which bits could 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 could 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 All-to-All. The VHDL simulator was limited (by memory) to a maximum of 64 nodes, so we simulated both a 4x4x4 torus and an 8x8x1 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/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.

5. Current Status

We received the first pass chips back from manufacturing in mid-June, 2003 and have been testing them extensively for correctness (and performance). Second pass chips will be released to manufacturing in mid-November, 2003 and the first batch of these is expected back in March 2004. Build and test of the Livermore machine will occur in stages during 2004, and we expect the full 64K node system to be operational in early 2005.

Acknowledgements

We are grateful to Jose Duato, Peter Hochschild, and Craig Stunkel for helpful conversations.

References

1. Adiga et al., (2002). An Overview of the BG/L Supercomputer. *Proceedings of the 2002 Supercomputing Conference* www.sc-conference.org/sc2002/
2. Benveniste, C. and Heidelberger, P. (1995). Parallel Simulation of the IBM Interconnection Network. In *Proceedings of the 1995 Winter Simulation Conference*. IEEE Computer Society Press, 584 – 589.
3. Dally, W.J., Dennison, L.R., Harris, D., Kan, K. and Xanthoppulos, T. (1994). Architecture and Implementation of the Reliable Router, In *Proceedings of HOT Interconnects II*, 122-133.
4. Dally, W.J. (1992). Virtual-Channel Flow Control. *IEEE Transactions on Parallel and Distributed Systems* 3, No. 2, 194-205.
5. Dally, W.J. and Seitz, C.L. (1987). Deadlock-Free Message Routing in Multiprocessor Interconnection Networks. *IEEE Transactions on Computers* C-36, No. 5, 547-553.
6. Dickens, P.M., Heidelberger, P., and Nicol, D.M. (1996). Parallelized Direct Execution Simulation of Message-Passing Parallel Programs. *IEEE Transactions on Parallel and Distributed Systems* 7, No. 10, 1090-1105.
7. Duato, J. and Pinkston, T.M. (2001). A General Theory for Deadlock-Free Adaptive Routing Using a Mixed Set of Resources. *IEEE Transactions on Parallel and Distributed Systems* 12 No. 12, 1219-1235.
8. Kermani, K. and Kleinrock, L. (1979) Virtual Cut-Through: A New Computer Communication Switching Technique, *Computer Networks*, Vol. 3, pp267-286.
9. Nicol, D. Micheal, C. and Inouye, P. (1989). Efficient Aggregation of Multiple LP's in Distributed Memory

Parallel Simulations, In *Proceedings of the 1989 Winter Simulation Conference*. IEEE Computer Society Press, 680-685.

10. Puente, V., Beivide, R., Gregorio, J.A., Prellezo, J.M., Duato, J., and Izu, C. (1999). Adaptive Bubble Router: A Design to Improve Performance in Torus Networks. In *Proceedings of the 1999 International Conference on Parallel Processing,* 58-67.
11. Puente, V., Izu,C., Beivide, R., Gregorio, J.A., Vallejo, F. and Prellezo, J.M. (2001). The Adaptive Bubble Router. *Journal of Parallel and Distributed Computing* 61, No. 9, 1180—1208.
12. Scott,S. and Thorson, G. The Cray T3E network: Adaptive routing in a high performance 3D torus. In *Proceedings of HOT Interconnects IV*. 1996
13. Yu, Q., Towsley, D. and Heidelberger, P. (1989). Time-Driven Parallel Simulation of Multistage Interconnection Networks. In *Distributed Simulation, 1989*. The Society for Computer Simulation International, 191-196.

Figures

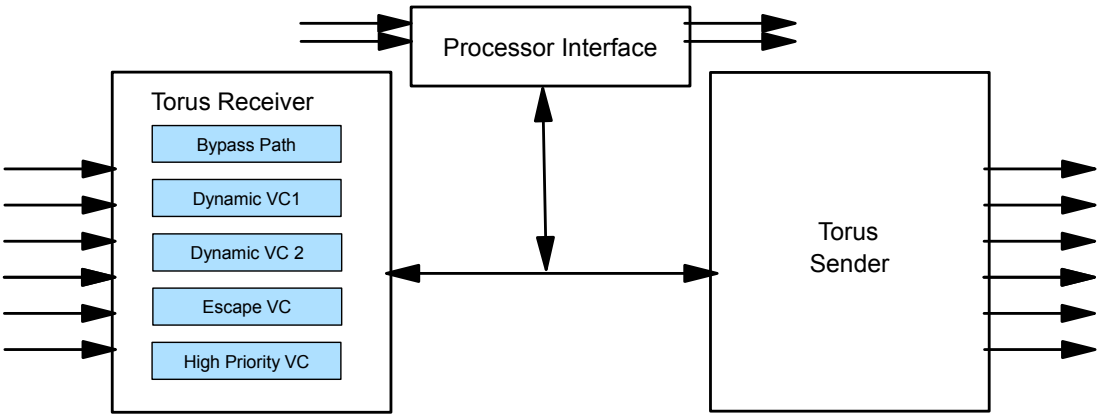

Figure 1: General Structure of Torus Router

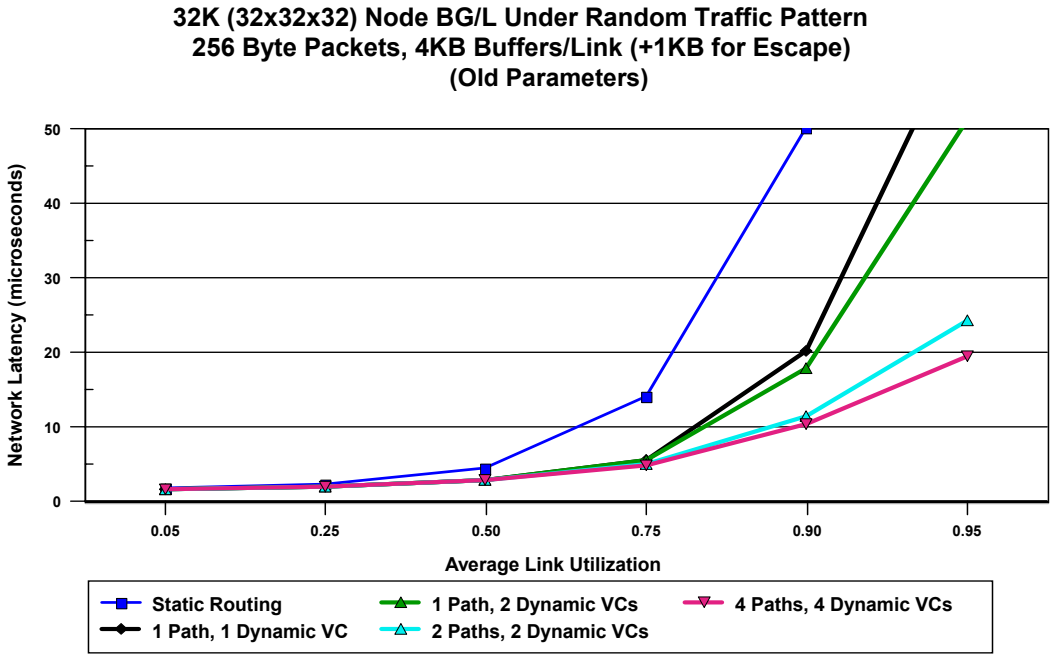

Figure 2: Sample Response Time in Light Traffic

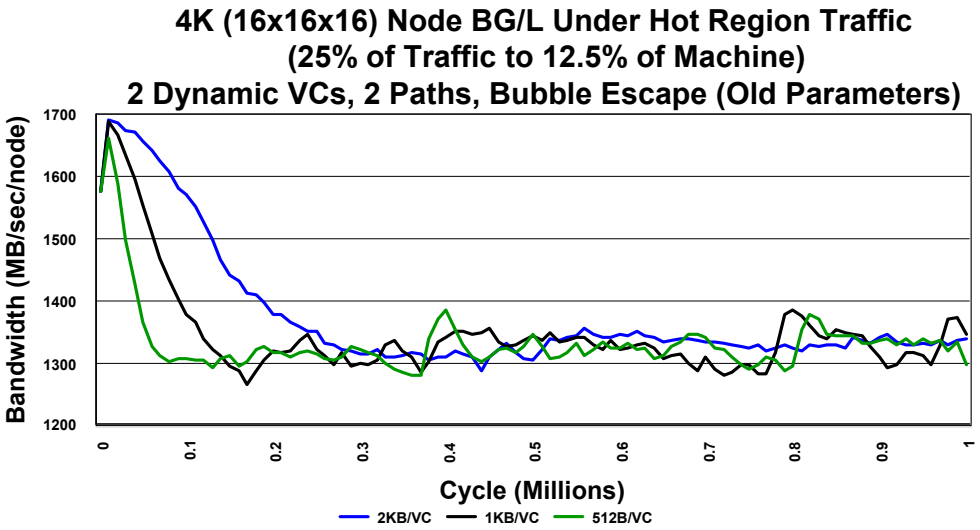

Figure 3: Throughput Under Non-Uniform Load

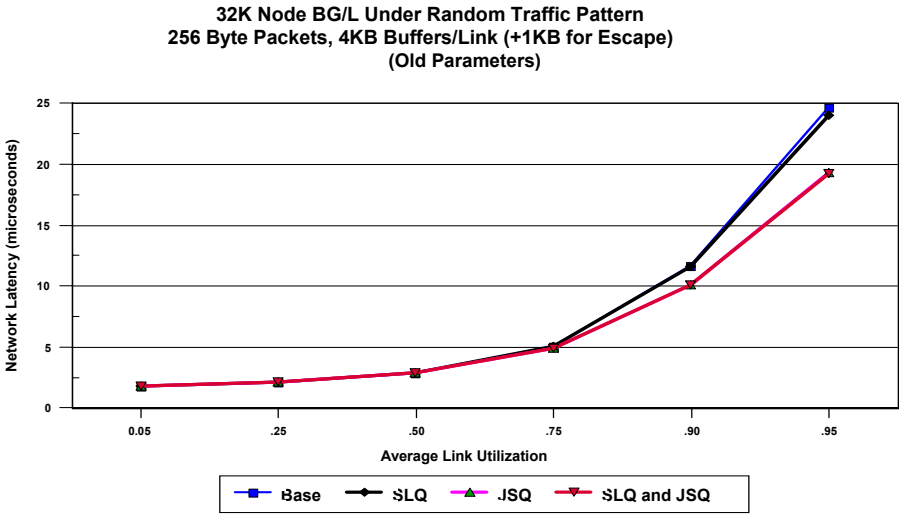

Figure 4: Response Time in Light Traffic for Different Arbitration Policies

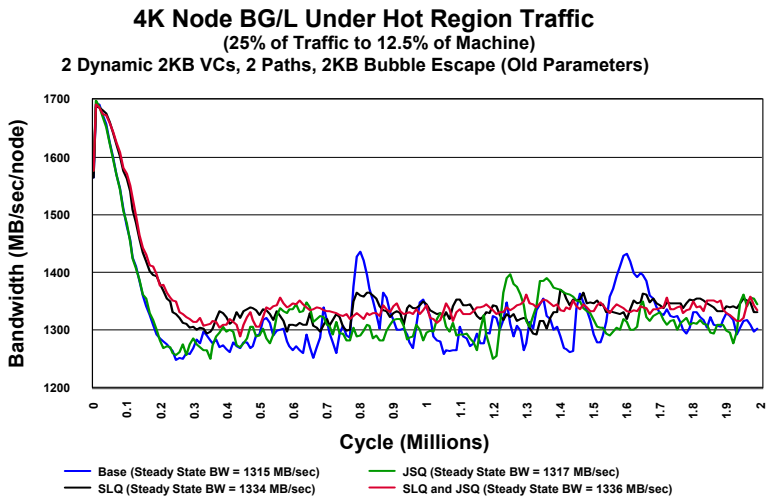

Figure 5: Throughput Under Non-Uniform Traffic for Different Arbitration Policies

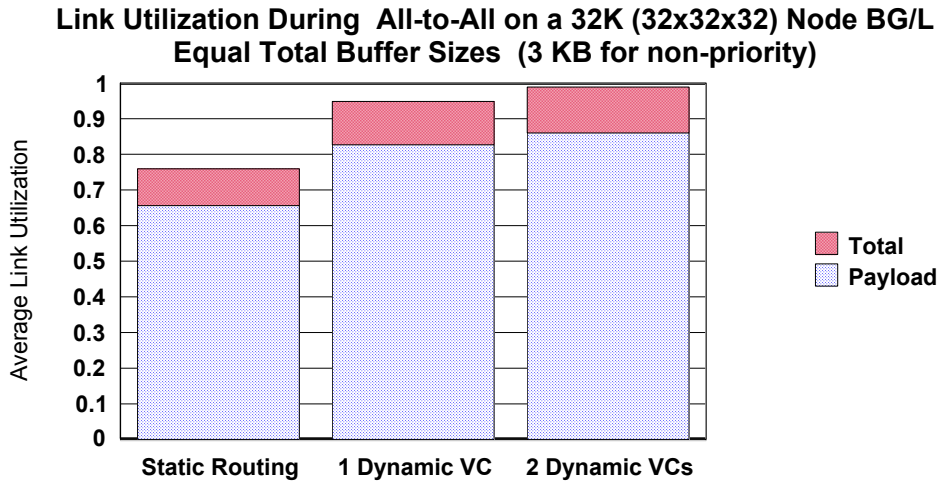

Figure 6: Average Link Utilization During All-to-All

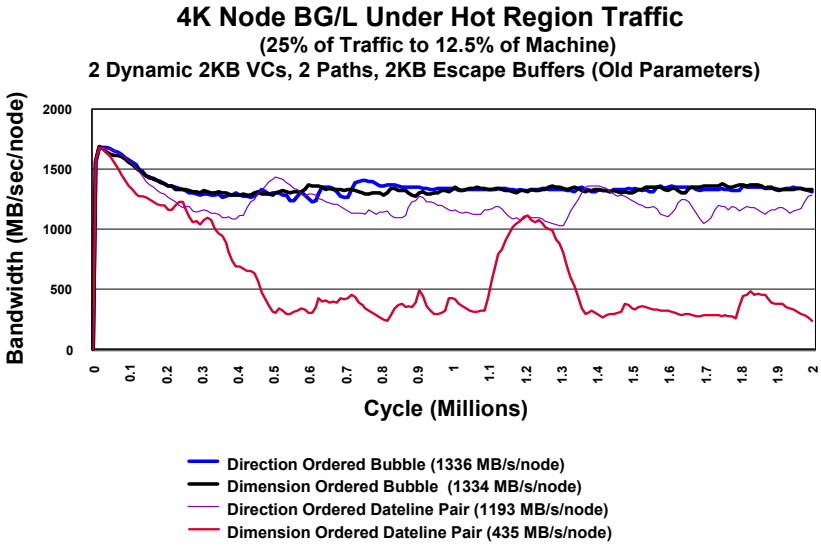

Figure 7: Throughput Under Hot Region Traffic for Different Escape VC Architectures

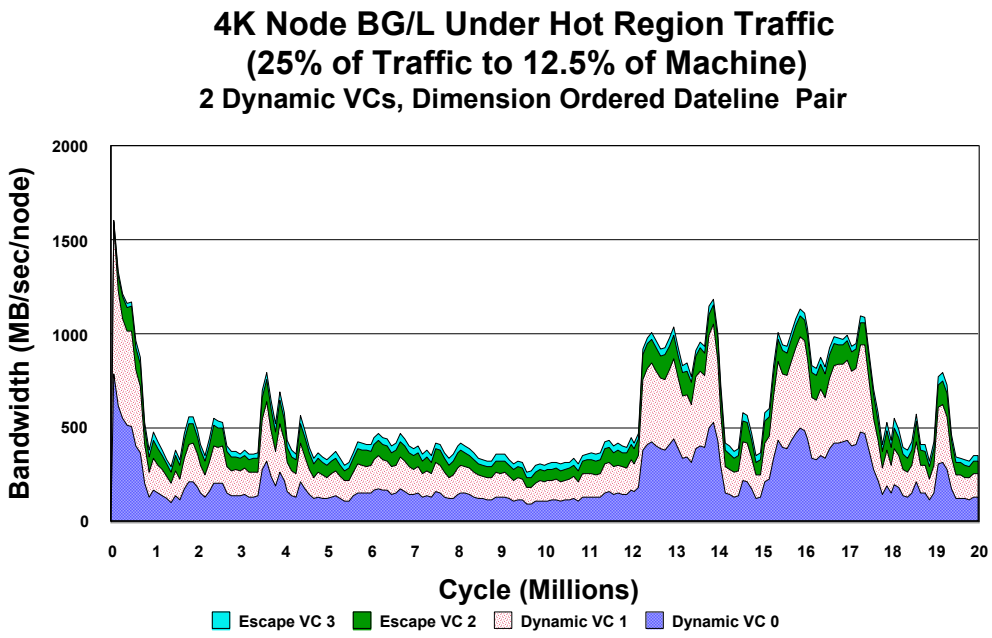

Figure 8: Throughput on Each VC for the Dimension Ordered Dateline Pair Escape VC Architecture

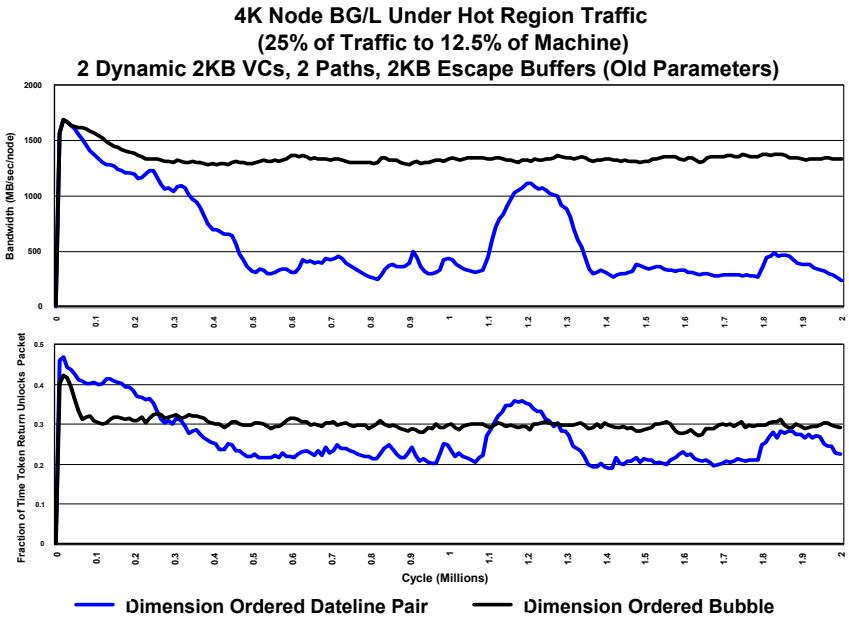

Figure 9: Explanation of the Bandwidth Waves