PARALLEL SIMULATION OF LARGE SCALE INTERCONNECTION NETWORKS
USED IN HIGH PERFORMANCE COMPUTING

BY

PRAVEEN KUMAR JAGADISHPRASAD

B.E., Bangalore University, 2001

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2004

Urbana, Illinois

Abstract

Interprocessor communication time is one of the major performance bottlenecks in parallel applications . Consequently performance prediction of parallel applications requires an accurate prediction of communication time. Accurate prediction of communication time involves detailed modeling of entities in the interconnection network. Such a detailed modeling also helps in designing message passing layers for current and future systems . This masters thesis describes a detailed parallel discrete event based simulator for the interconnection networks used in high performance computing machines. We have simulated an example each in the direct and indirect class of interconnection networks . Fat Tree network, which is employed in most of the current top 10 clusters will be the candidate for simulation of an indirect network. More specifically QSNET will be used for fat tree simulation. IBM Bluegene/L which is a next generation supercomputer will be used as an example for simulating a direct network.

To my parents.

Acknowledgements

First of all I would like to thank my advisor, Prof. Laxmikant Kalé . He gave me full freedom to choose my thesis topic and has always treated me with lot of patience . I also thank him for his encouragement and for allowing me to work with highly talented people at the parallel programming lab.

I would also like to thank Terry Wilmarth, author of POSE on which my network simulators are based, for helping me in various stages of the project. Sameer Kumar for providing valuable insights into QSNET with his indepth knowledge. Eric Bohm whose BigNetSim helped my understanding of POSE and network simulation. Gengbin Zheng, with his trademark patience in helping me with Bluegene model and a host of other questions . Yogesh Mehta, member of Bluegene group along with rest of friendly folks at PPL. Avneesh Pant for providing access to infinite cluster.

Also lot of thanks to Fabrizio Petrini,David O'Neal,David Addison and Eitan Frachtenberg for help on QSNET, and Philip Heidelberger for help with Bluegene/L network. Most of the credit for any preciseness in the simulator model should go to them.

NCSA Consultants for providing me with a wonderful work environment . I am indebted to Galen Arnold,David McWilliams, Michelle Gower, Susan John , Ian Brooks,Weddie Jackson . NCSA Tungsten cluster in friendly user mode has been a great help for my simulations. Thanks to Susan John and Karen Fernsler .

Finally I reserve my best to thank my parents for being strong and understanding during this prolonged period.

Table of Contents

List of Figures

Chapter 1

Introduction

Computational science has aided a lot of current advances in scientific and engineering disciplines like molecular dynamics, rocket simulation, atmospheric sciences etc . High Performance Computing machines have enabled these applications by providing significant computing power,memory and mass storage. Parallel applications generally tend to have lot of interprocessor communication. As a result optimizing communication time is important [16] to get a decent speedup .

Communication time can be optimized at low level software layers such as Converse [4] or MPI [12] or in the application itself. Detailed modeling of entities in an interconnection network is needed to support any useful communication optimization. Quantitative methods make a lot of simplifications such as ignoring network contention which has a significant impact on communication time. Flit level network simulation tracks changes in the network every clock cycle and gives much more accurate predictions. Currently available switched network solutions such as Quadrics [14], Myrinet [2] Infiniband [1] have sophosticated network adapters connected by a switched network with redundant links . So we need to have a detailed model of physical entities such as node,network adapters,switches,channels and non-physical entities such as topology,protocol stacks to simulate the whole setup.

Bluegene/L being developed by IBM with partial funding from U.S Department of Energy is a next generation supercomputer with 65536 nodes .It is essential to evaluate the performance of such machines for different target applications before they are built or before they are actually purchased.

During the design stage of interconnection network, it is useful to know the performance for various parameters such as number of virtual channels,routing strategies, flow control, buffering etc. A simulator would be useful in making appropriate design choices. Another uitility of an simulator for existing machines is it helps reduce the peformance tuning iterations . Otherwise each tuning iteration may take days depending on queue wait times at supercomputing sites.

1.1 Thesis Contribution

- Model QsNet components and develop a PDES Network Simulator. Observe effects of contention in different parts of the network like network adapter,pci channel and switches . Validate contention effects in the simulator.

- Model a 3D torus network using the actual Bluegene/L network specification. Use actual application emulation traces to run a parallel simulation.

BgSim [19] is an simulator used for performance prediction of large parallel machines based on actual execution of applications . It builds on an already existing system that can emulate large parallel machines [17]. Based on the CHARM++ [5] parallel programming system, the emulator has successfully simulated several million threads (one for each target machine processor) on clusters with only hundreds of processors. But it's current network model is not based on the actual bluegene network. This thesis will describe an network simulator which models a 3D torus network which is one of the primary Bluegene/L networks.

1.2 Thesis Organization

Parallel discrete event simulation and Parallel Object Simulation Environment (POSE) in general will be explained first. We will then describe a fat tree network and use quadrics network as a case study . The design components will be modeled at the host level and switch level . Wherever possible, a brief description of QsNet components are provided followed by the corresponding simulator model. Contention effects in different parts of the network will be shown through simulation runs. An overview of Bluegene simulator will be given. Actual Bluegene torus network design along with the corresponding POSE based simulation model will be described. Some results from based on application traces will be discussed. Finally we have conclusion and future work.

Chapter 2

Parallel Object Simulation Environment

Parallel Object Simulation Environment (POSE) [18] is a parallel discrete event simulation framework built using Charm++. It is used to simulate complex systems in which events can occur at discrete points of simulated time at irregular intervals.

2.1 CHARM++

CHARM++ [6] is a object based portable parallel programming language that embodies message driven execution. It has been successfully used to simulate challenging applications like Molecular dynamics,Cosmology and Rocket Simulation. It encourages virtualization, wherein the programmer is encouraged to develop fine grained execution components (known as *chares*) . The number of chares are independent of the number of physical processors. This freedom allows applications to run millions of chares on finite set of processors, a feature critical for success of an large scale simulator.

2.2 POSE

POSE is a Parallel Discrete Event Simulation framework developed by Terry Wilmarth at the Parallel Programming Lab, UIUC. It maps simulation entities known as *posers* to Charm++

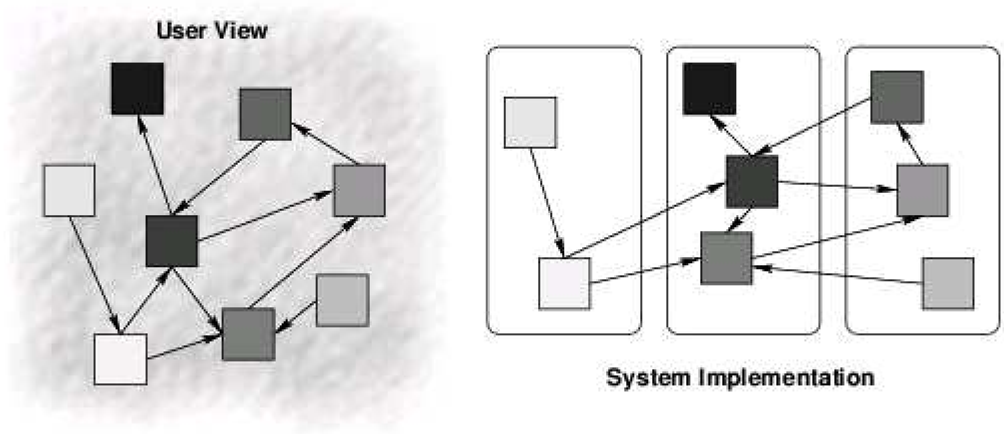

Figure 2.1: CHARM++ Virtualization

chares. Each poser maintains current simulation time (simulation time elapsed since the creation of object) in *object virtual time* (OVT) variable. Each poser has a set of *event methods* which are executed atomically on being invoked by messages from other posers . Messages intended for a poser are placed in an event queue and selected for execution based on timestamp .Posers can advance simulated time in two ways. First is by the *elapse function*. This adds the given time offset to the poser's OVT. The second is on event invocations by remote posers sending messages. Depending on current time, an *offset* is added to OVT.

Speculative Synchronization in POSE makes use of optimistic concurrency control which originated as Time-Warp. When an object receives an event, it gets control of the processor. The object's synchronization strategy is then invoked and checks for any synchronization error corrections (rollbacks, cancellations) before it performs forward execution steps (events).

Here the opportunity to perform *speculative computation* arises. All optimistic strategies perform some amount of speculative computation. In more traditional approaches, an event arrives and is sorted into the event list and the earliest event is executed. POSE knows the event is the earliest available on the processor, but it does not know if it is the earliest in the entire simulation, thus executing it is speculative.

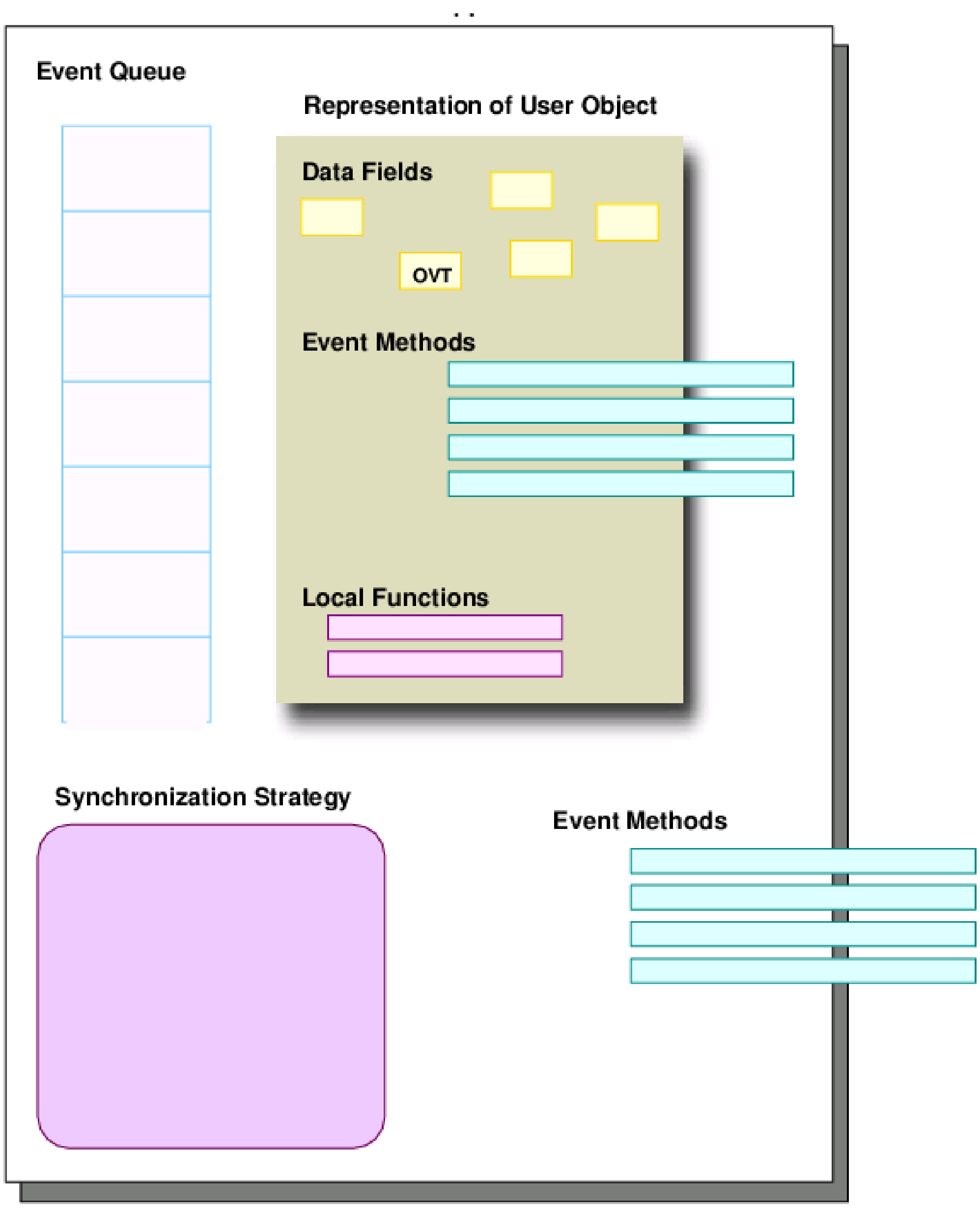

Figure 2.2: POSE : Parallel Object Simulation Environment

In POSE, the behavior is similar with some exceptions. First, it has a *speculative window* that governs how far into the future beyond the global virtual time (GVT) estimate an object may proceed. Speculative windows are similar to the time windows of other optimistic variants, except in how events within the window are processed.

In POSE, events are inserted into the event queue on the object for which they are destined. When the object gets control, it invokes the synchronization strategy to process events. Typically, the event just received will be the event processed, and no other events will exist on that object. However, some events may arrive slightly ahead of schedule and may be worthy candidates for speculative execution. Others may arrive with timestamps very far in the future and it may be unwise to execute them speculatively.

The synchronization strategy determines how to process the events within a speculative window. To avoid rescheduling overhead, POSE allow objects freedom to speculate once they get control. Event processing on the object follows this pattern: if there are events with timestamp $t \geq$GVT but within the speculative window, do *all* of them. The later events are probably not the earliest in the simulation, and it is very likely that they are not even the earliest on that processor. POSE allows the strategy to speculate that those events are the earliest that the *object* will receive. By handling events in bunches, POSE reduces scheduling and context switching overhead and benefits from a warmed cache, but risks additional rollback overhead.

Chapter 3

Indirect Switched Networks

Indirect Networks [13] are a major class of networks wherein instead of providing a direct connection among some nodes, the communication between any two nodes has to be carried through some switches. Each node has a network interface card which connects to an network switch. So while a subset of switches connect to nodes and possibly other switches, another subset interconnect just switches. The topology of the network is defined by the way switches are interconnected.

3.1 Fat Tree Description

Fat Tree networks [9] are used extensively in current generation high performance networks such as Quadrics and Infiniband. Some of the key advantages include high bisection bandwidth, scalable topology, compact switches and simple routing . A Complete 64 node fat tree is shown in 3.1

QsNet uses a *fat-tree* (more specifically, 4-ary n-tree) interconnection topology. The graph k-ary n-tree has been defined in [15] [8]. It is a type of fat-tree which can be defined as follows:

Definition: *A k-ary n-tree is a fat-tree that has two types of vertices: $P = k^n$ processing nodes and nk^{n-1} switches. The switches are organized hierarchically with n levels that have k^{n-1} switches at each level. Each node can be represented by the n-tuple* $\{0, 1, ..., k-1\}^n$,

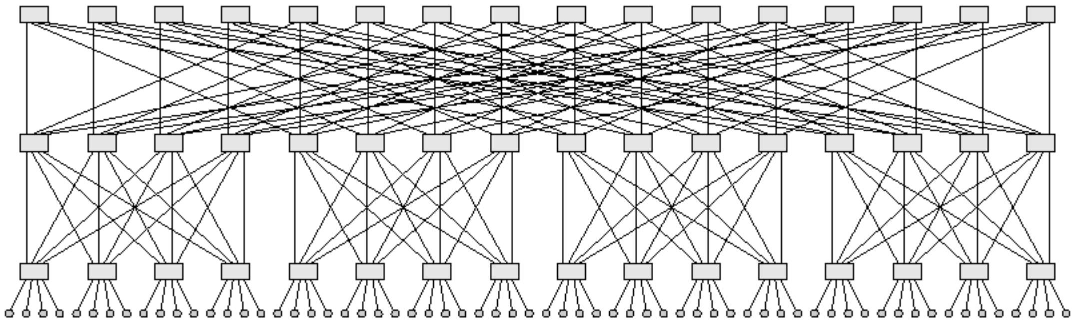

Figure 3.1: 64 Node Fat Tree

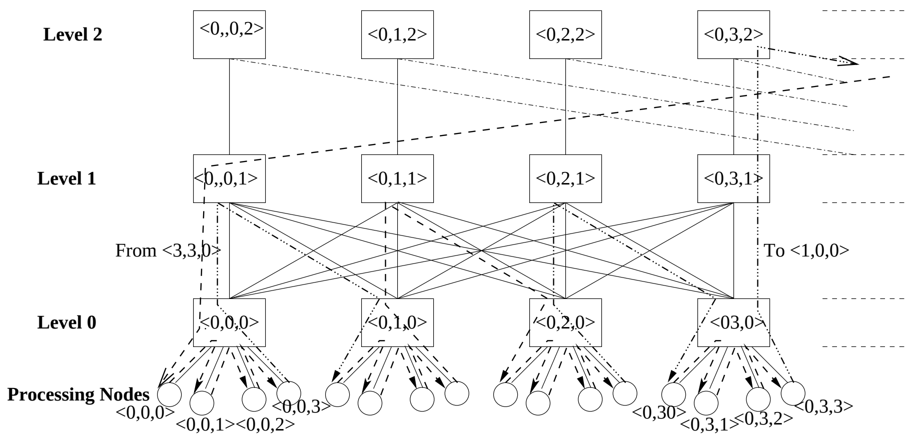

Figure 3.2: Partial Fat Tree

while each switch is defined as an ordered pair $\langle w, l \rangle$ *where* $w \;\epsilon\; \{0, 1, ..., k-1\}^{n-1}$ *and* $l \;\epsilon\; \{0, 1, ..., n-1\}$. Here the parameter l represents the level of each switch and w identifies a switch at that level. The root switches are at level $l = n-1$, while the switches connected to the processing nodes are at level 0. A partial fat tree is show in 3.1.

- *Two switches,* $\langle w_0, w_1, ..., w_{n-2}, l \rangle$ *and* $\langle w'_0, w'_1, ..., w'_{n-2}, l' \rangle$ *are connected by an edge iff* $l' = l + 1$ *and* $w_i = w_{i'}$ *for all* $i \neq n-2-l$

- *There is an edge between the switch* $\langle w_0, w_1, ..., w_{n-2}, 0 \rangle$ *and the processing node* $\{p_0, p_1, ..., p_{n-1}\}$ *iff*

$$w_i = p_i \;\; for\;all\;\; i \;\epsilon\; \{0, 1, ..., n-2\}$$

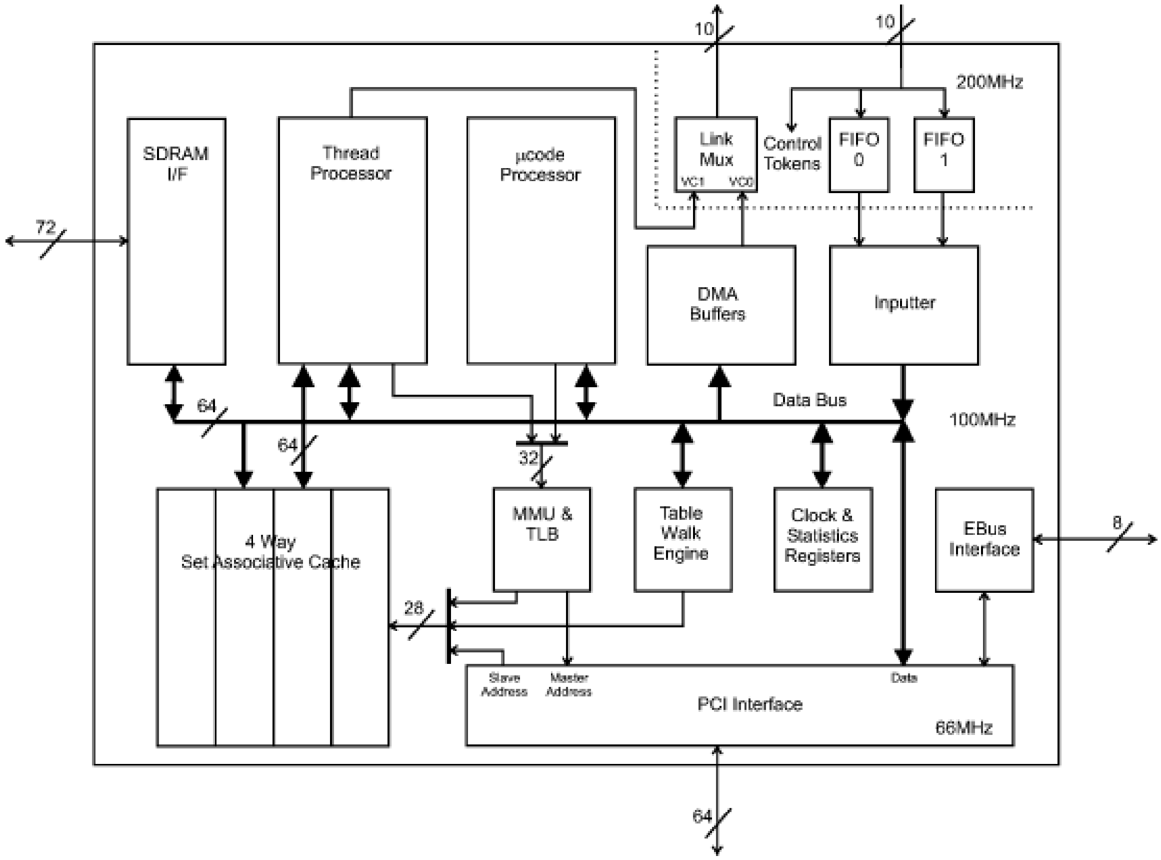

Figure 3.3: Elan Network Adapter Architecture

3.2 Network Interface Card

The Elan Adapter [10] (Elan3 in specific) is a sophosticated communications unit present in the Node (or Host). It enables processors in the node to communicate with other nodes through an fat tree network. It generates and accepts packets to and from the network. It also helps offload communication tasks from the processor [7], thereby allowing the processor to do useful computation .

3.3 Functional Units

The functional units are explained in detail in [10] . We will have a look at some units which affect simulator accuracy.

Microcode Engine

This is a 32 bit microcode processor that supports four seperate threads of execution. All the threads together are allowed to have upto 8 pipelined requests to the memory system. The four microcode threads of execution are as follows

- DMA thread (Dproc) generates the DMA packets to be written to the network . It also maintains a high and low priority system queue of outstanding DMAs. Large DMAs are timesliced to allow small DMAs a chance to complete quickly.
- Inputter thread (Iproc) is responsible for processing the incoming packets from the network.
- Thread processor scheduler controls the scheduling and descheduling of the thread processor. The μcode processor maintains two system queues of low and high priority threads
- The main processor schedules operations on the elan by writing to a command port . When a write to a command port is detected, the command processor is woken up to process the operation.

PCI Interface

PCI Interface is one of the major bottlenecks as it lies on the critical path of data transfer. PCI bus latency and bandwidth limitations play a very important role in main processor to main processor or elan transfers.

- The master interface can be used by the thread processor or by any of the μcode processor threads . Adjacent accesses are merged to form long burst accesses. It can accept upto 4 memory accesses and has a 64 byte read FIFO and 64 byte write FIFO to stream data to and from the Elan

- The major uses of slave interface are by the main processor to issue commands to the command port, to read and write to Elan memory, and to access the 8 bit wide general purpose I/O interface.

Microcode Engine Implementation

The implementation is grouped in the NetInterface poser.

- The DMA engine divides the message received from the node into packets. The payload size per packet is set to 320 for elan3. Extra overhead including routing header and transaction overhead such as memory address,transaction type, context value and crc are accounted for each transaction . The final outgoing packet size might be between 374-394 bytes for full data payload. Additional bytes might be needed for routing in larger networks. The packet is further broken down into flits and sent over the dedicated send virtual channel.

- The Inputter implementation overlaps with DMA thread implementation, correctly serializing the μcode engine but preserving all the functionality . Also in overlapped ack mode, as soon as a transaction is assembled an Ack is sent out. On receiving an EOP token, the packet/message state is cleaned up.

- An active table of partially received packets and messages is maintained. It is indexed by a unique message id constructed from the message source and original message id. It also performs flow control and low level protocol handling. There will be a penalty when elan is simultaneously sending and receiving packets. This swap penalty is due to contention between Iproc and Dproc threads . We do not model a thread processor since it cannot be exploited by a user . But we have a flexible number of processing engines, which could potentially be used to implement the thread processor functionality. Each processing engine can have multiple ports, which in turn can have multiple virtual channels. Writes to the command port are modeled by a message sent from Node poser to NetInterface poser.

PCI Interface

PCI bandwidth and latency are specified in a configuration file. If data transfer occurs from main memory, the best achievable bandwidth is limited by the PCI bandwidth. These are slightly different depending on uni-directional or bi-directional access. The PCI latency and Bandwidth bottleneck implementation is shared between PCI channel posers and NetInterface Poser.

Channels

Channel posers represent the physical channels connecting the nodes and switches together. They also have additional function of simulating bandwidth sharing in flit level multiplexing.

3.4 Packet Protocol and Flow Control

Messages from the processor are split into packets at Elan . For flow control purposes, the packets can be subdivided into flits. Ultimately the lowest level of useful physical units are a byte wide. But the protocol used to enable successful transmission of data occurs at the packet level. Figure 3.4 shows the internal structure of a packet.

- Route - This part of packet header has some data used by switches for routing.
- Transactions - There are two types of transactions, Write Block and non-Write Block. Remote DMA generally uses one or more Write Block transactions. Each Write Block has 64 byte of data, five of which together form the 320 byte payload limit.
- Eop token - This will be discussed in the Flow Control Protocol section.

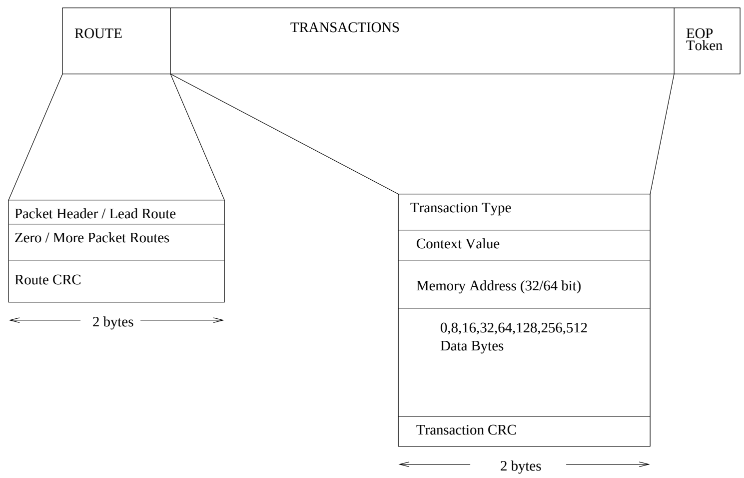

Figure 3.4: Packet Format

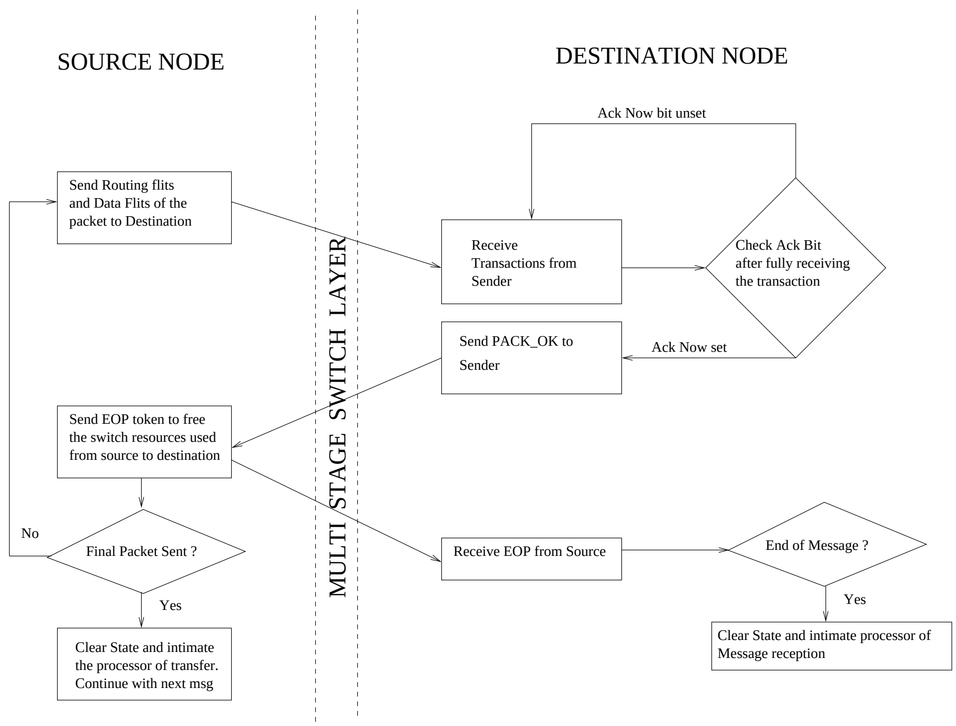

Figure 3.5: Packet Protocol

Packet Protocol

Figure 3.5 shows a simplified version of the actual protocol. The switch network is assumed to be fault free.

- The sender sends the routing and data flits to the switch it is connected to, at the first level of the multilevel switch network.
- The flits traverse the switch hierarchy and reach the destination. There is a seperate flow control protocol in place for contention in the network.
- The receiver stores the incoming flits in the inputter FIFO. Once an transaction is assembled, the transaction decoder checks to see if Ack Now bit is set. If it is set, then an acknowledgement token PACK_OK is sent to the source. For an Overlapped Packet mode (which is used by DMA), the first transaction itself will have the bit set. For an Non-Overlapped mode , the bit will be set only in the last transaction.
- After receiving PACK_OK, the sender clear buffers and sends a End of Packet Token. This clears the route reserved by the packet. Also it sends out flits from the next packet and the cycle continues until the entire message is sent out.

Flow Control

Each Switch/Node has a finite input buffer. When a receiver node is busy and unable to process incoming flits, it has to have enough buffers to store them. When it realizes that it's buffer is filling up, it sends a STOPx token to the switch, asking it to stop. When the buffer is relatively free, it sends STARTx token asking the switch to resume transmission. The simulator assumes that the receiver node consumes the transactions before they fill the FIFO. In the switch network, when a packet header is unable to progress due to a busy output port, the packet has to block in place. Flow Control tokens have to be sent to all the

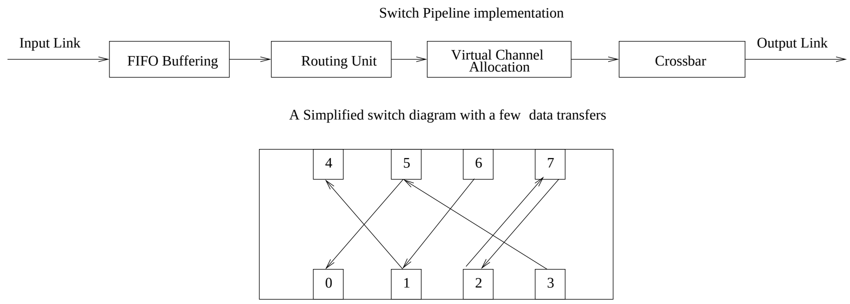

Figure 3.6: Simulator Switch Implementation

intermediate switches and source node to stop/start sending flits. The simulator implements this by having flow control start/stop tokens take the reverse route from the busy switch .

3.5 Elite Switch

Elite3 Switch consists of 8 bi-directional links, each one byte wide. There are two virtual channels per link. Each hop takes about 35ns through the switch. It supports a bandwidth of upto 400Mbytes/s in each direction and used adaptive wormhole routing.

The simulator switch model is shown in 3.6 . The pipeline design might not exactly match the real implementation. But the 400 Mbytes/s bandwidth indicates that the largest pipeline for a flit will be 5ns wide. Also the overall sum of the pipeline delays is equal to 35 ns for non-contention cases. So as long as these conditions are adhered to, the flit latencies shall be right. The route flit spends some extra time than other flits due to the route and virtual channel selection overhead.

3.5.1 Routing Strategies

Elite supports both source routing and adaptive routing. Simulator design also supports both. There is a simple routing function in the switch. For source routing, the routing logic decides the route, without taking into consideration any load characteristics . Though the decision occurs in switch, the logic used is will be the same as the one used in calculating routes at source. Adaptive routing proceeds as follows :

1. For downward links, only the port corresponding to the destination quarter should be chosen. There is no other choice.

2. For upward links, the least loaded link is chosen as follows

   (a) If there are one or more totally unconnected links it will randomise over one of these. A six bit pseudo random counter is used and the bottom two bits are used to select the output. If more than one input is trying to adaptively select at the same time, then the logic ensures that all the inputs will be directed to different random outputs. Simulator implementation uses a 6 bit Linear Feedback Shift Register , with the feedback coming from XOR of bottom two bits.

   (b) If all the output links are connected on at least one virtual channel, then the output with the lowest priority and age is selected. If there is more than one at the same lowest priority and age then a random selection is used between them. The simulator implementation gives equal priority to all packets. So just age is considered for selection.

Wormhole routing ensures that routing overhead is endured only by the route flits. All the data flits have a free passage once a mapping exists between input and output links. So in effect communication time is decided by latency constraints for route flit and bandwidth limitations for the data flits.

3.5.2 Multiplexing Schemes

We have implemented two different multiplexing schemes based on the resource unit for multiplexing.

Flit Level Multiplexing

The output port can support two virtual channels (VCs) simultaneously. When both are transmitting data, multiplexing is done flit by flit. If we have a third packet wanting to go through the same port , it's virtual channel identifier (vcid) in the switch is added to a first come first serve (fcfs) queue. Then the switch sends flow control tokens back to the packet's source through the same route that it arrived , forcing the switches and node to block the packet in place. The packet remains blocked until EOP arrives for one of the two output VCs. The old input-output mapping is deleted and the first entry of fcfs queue is mapped to that output VC. The EOP goes on it's course to the next switch. A flow control token is back to the previous switches till the restarted packet's source node , signalling an restart of flow. The buffered route and data flits are then sent to the next switch . Each Switch on receiving the flow start token, sends all the buffered flits onto to the next switch.

Packet Level Multiplexing

Only one virtual channel can transmit at a time. So multiplexing is now done packet by packet. A single packet occupies the output port. Other packets have to wait in fcfs queue. If the packet is blocked in the following switches STOPx token is propagated. Then the other virtual channel can be used. As in flit level multiplexing, EOP will cause the output VC to be free. The selection of input VC when EOP or STOPx arrive, is the same as explained in flit level multiplexing. With some changes in design, simulation time speedup is seen if we increase the data flit size. There are no appreciable simulated time changes when varying flit sizes. All through, the header flit size is kept constant at 2 bytes. Since data flits occupy an exclusive channel , they don't affect other data flows as in flit level multiplexing. When a path is released at the head port, but blocked at the intermediate ports, flit level multiplexing is used at the blocked ports. In this case varying data flit sizes might have some impact on the simulated time.

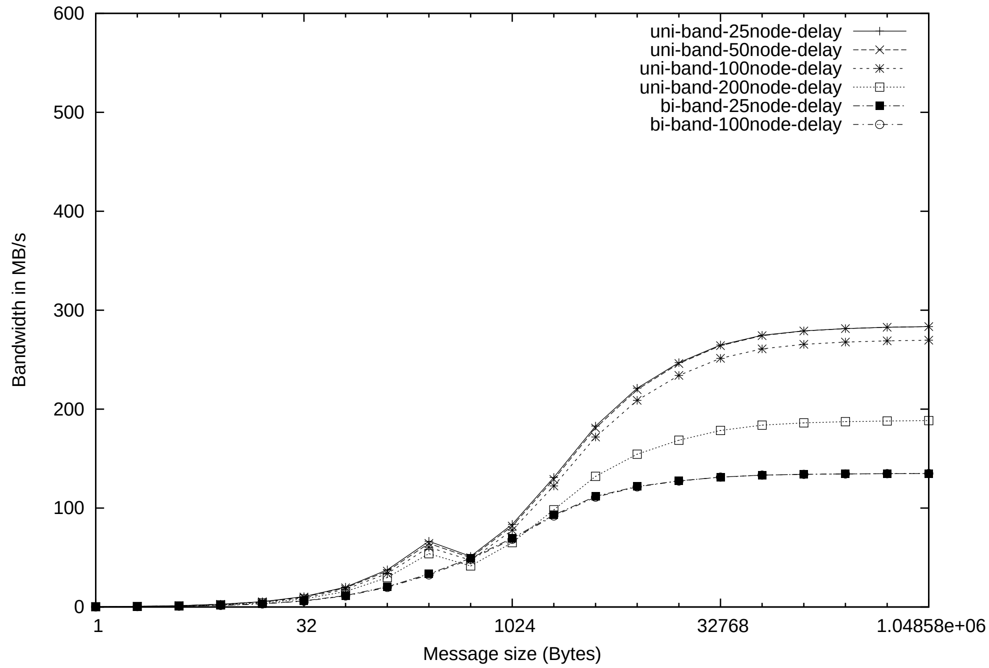

Figure 3.7: Uni and Bidirectional Bandwidths

3.6 Simulation Results

Figure 3.7 shows the effect of wire-delays on bandwidth. Also contention due to bi-directional traffic is shown, wherein the PCI bandwidth plays a major role in determining the peak bandwidth .

Figure 3.8 shows bi-directional charm latency measured on PSC Lemieux with simulated measurements. The simulation results can be divided into two categories. One is based on an arbitrary delay to account for contention among DMA and Inputter threads for the microcode processor. The other has a zero value for it. Receives are assumed to be posted when the envelope arrives. There is a slight jump at 288 bytes to account for switch from eager to rendezevous protocol. CPU overhead is ignored. But as observed from the no-overhead graph, this can be accounted for easily in the simulator. There is a provision for elapsing extra sender and receiver delay in the configuration files . They have been set to zero by default for all the simulations here. Figure 3.9 plots the results for a larger range of

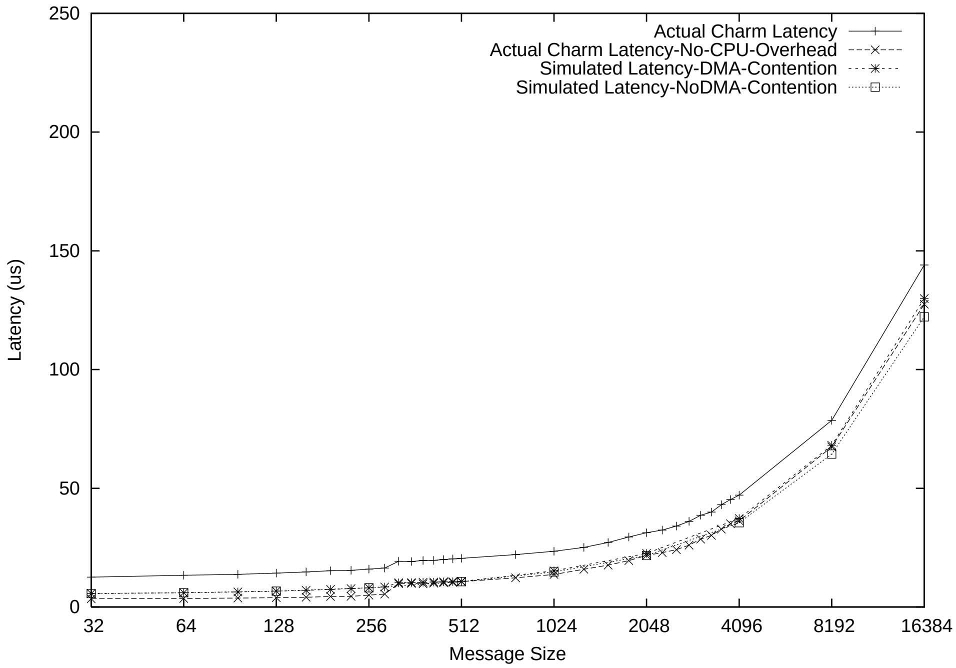

Figure 3.8: Bidirectional Charm Latency on Lemieux - Small Range

values.

Figures 3.10 and 3.11 show the effects of wire delay on bandwidth of prefix and k-shift multicast strategies (using flit level multiplexing). In k^{th} step each node P_i sends a message to its neighbor at distance k. For k-shift this neighbor is $P_{(i+k) \bmod P}$, while it is P_{iXORk} for prefix-send. The graph is not entirely symmetric as some randomization is present in the network.

Figure 3.12 shows an averaging of 34 values of bandwidths for each k in k-shift (using packet level multiplexing) . Each value had a different starting state which was pseudorandom. The results are similar to the actual values on Lemieux [8].

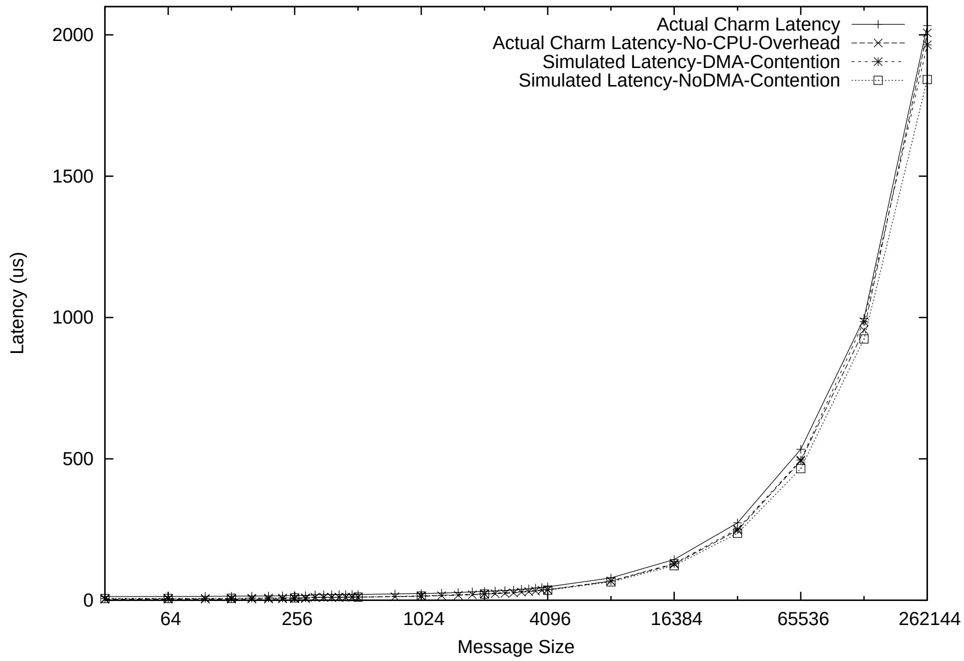

Figure 3.9: Bidirectional Charm Latency on Lemieux - Large Range

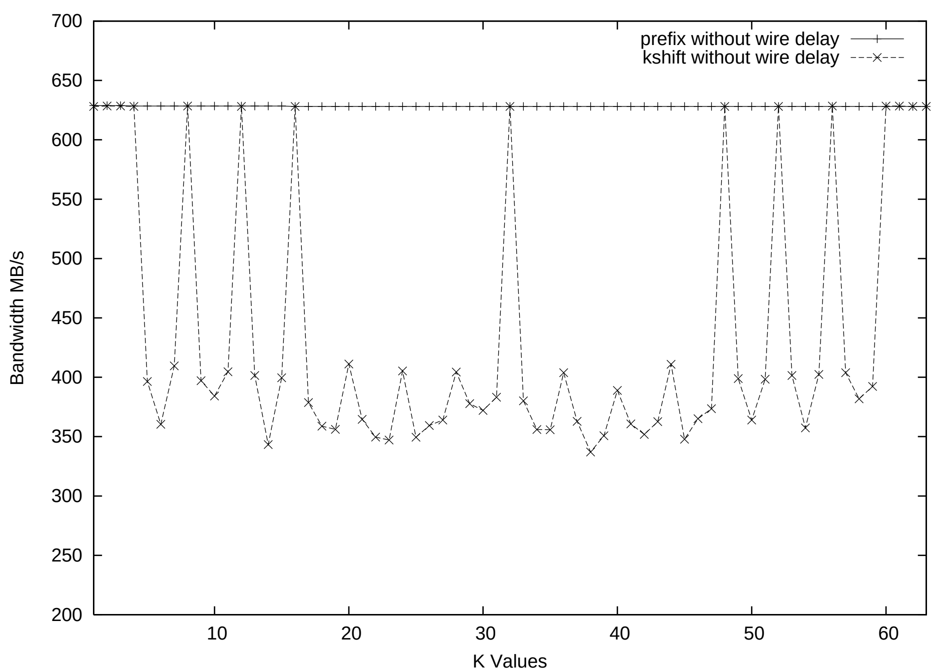

Figure 3.10: Kshift and Prefix with minimal wire delay

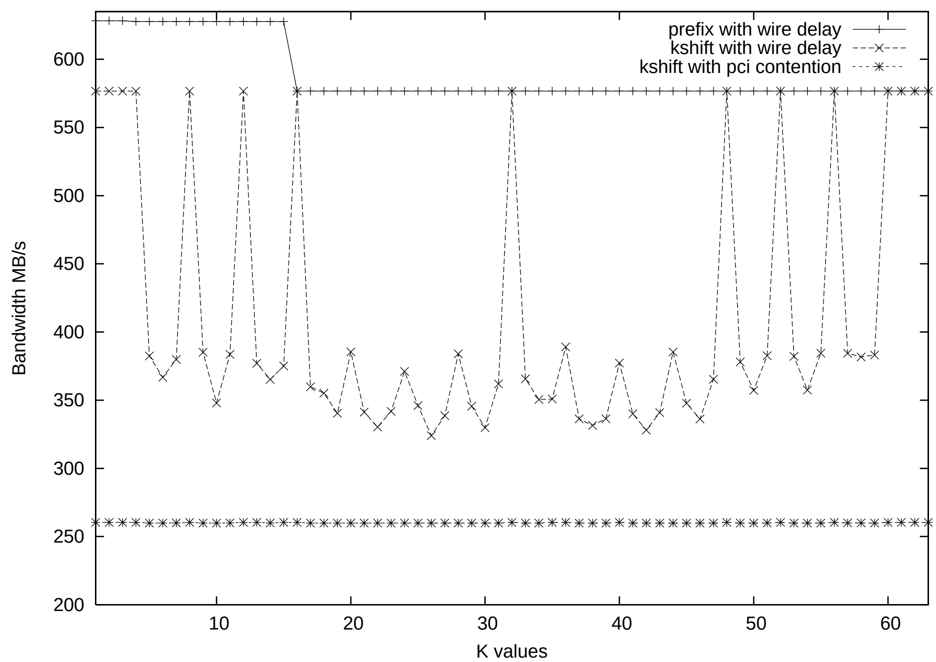

Figure 3.11: Kshift and Prefix with significant wire delay

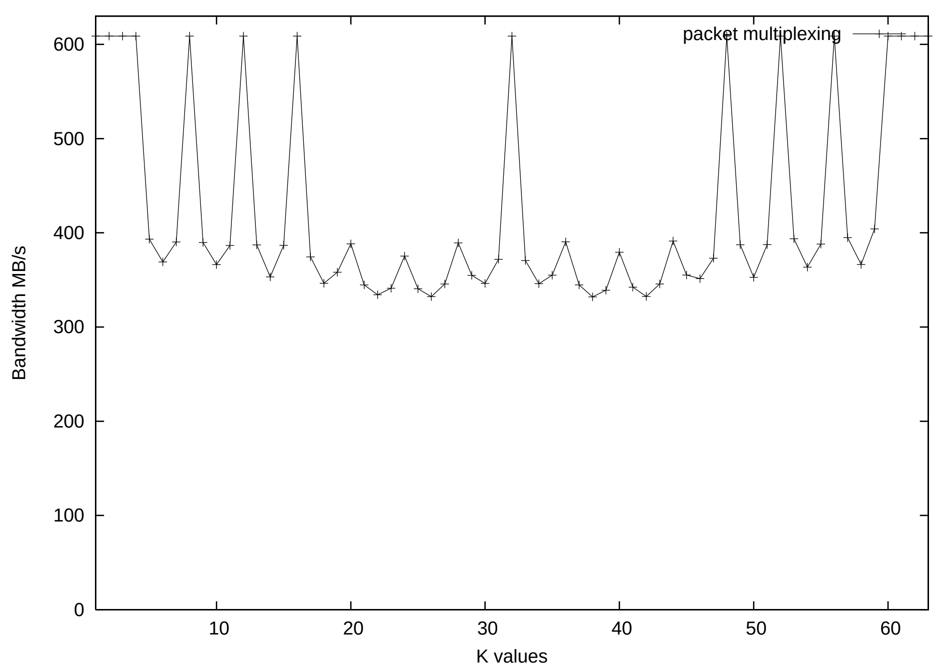

Figure 3.12: Kshift with parameters from PSC Lemieux

Chapter 4

Direct Networks

4.1 Bluegene Emulator

The emulator [17] supports a low-level, but fairly general API. The API supports multiple instruction streams(threads) that can share memory within a single node. Each thread has its own stack and a work queue. An emulated node is shown in 4.1 . The communication threads check for incoming messages from the network and put the messages in either a worker's queue or a node global queue. In order to make use of locality of data for performance improvement, affinity queues are created for each worker thread. A worker thread will process work from it's own queue before checking the global queue. Access to a reliable communication layer is provided via calls to send short (limited-length) messages to other nodes, along with an index of the handler to be invoked on the destination.

4.2 Bluegene Simulator

BigSim[19] is a Bluegene Simulator which builds on the emulator to enable performance prediction . It uses structured-dagger [3], a coordination language built on top of CHARM++ to store event dependencies. This enables it to simulate a broad range of applications, where out-of-order messaging is possible. Needless to say, this can simulate applications with in-order messages also. Network Contention was ignored in BigSim and a simple hop-based

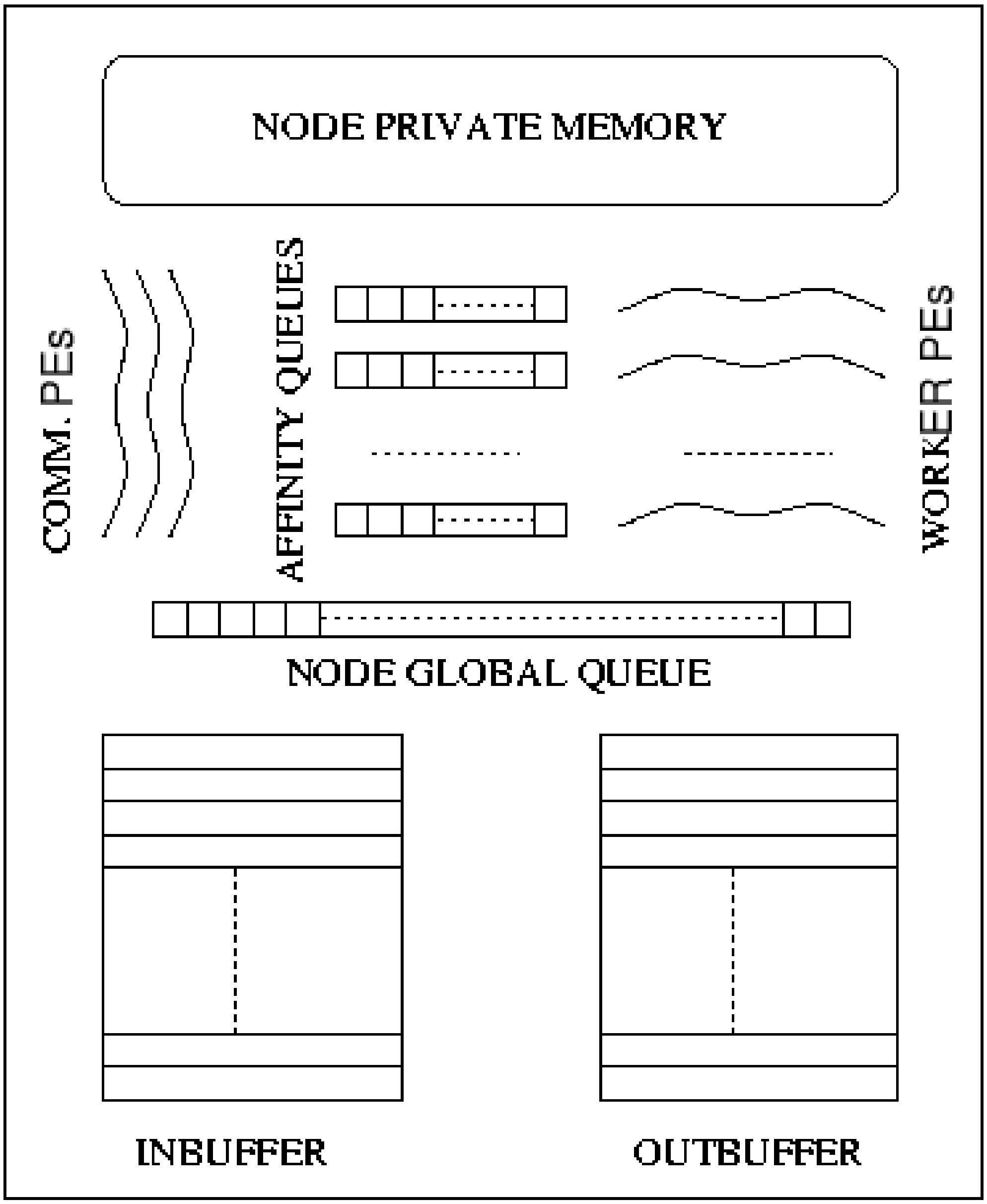

Figure 4.1: Functional view of an emulated node

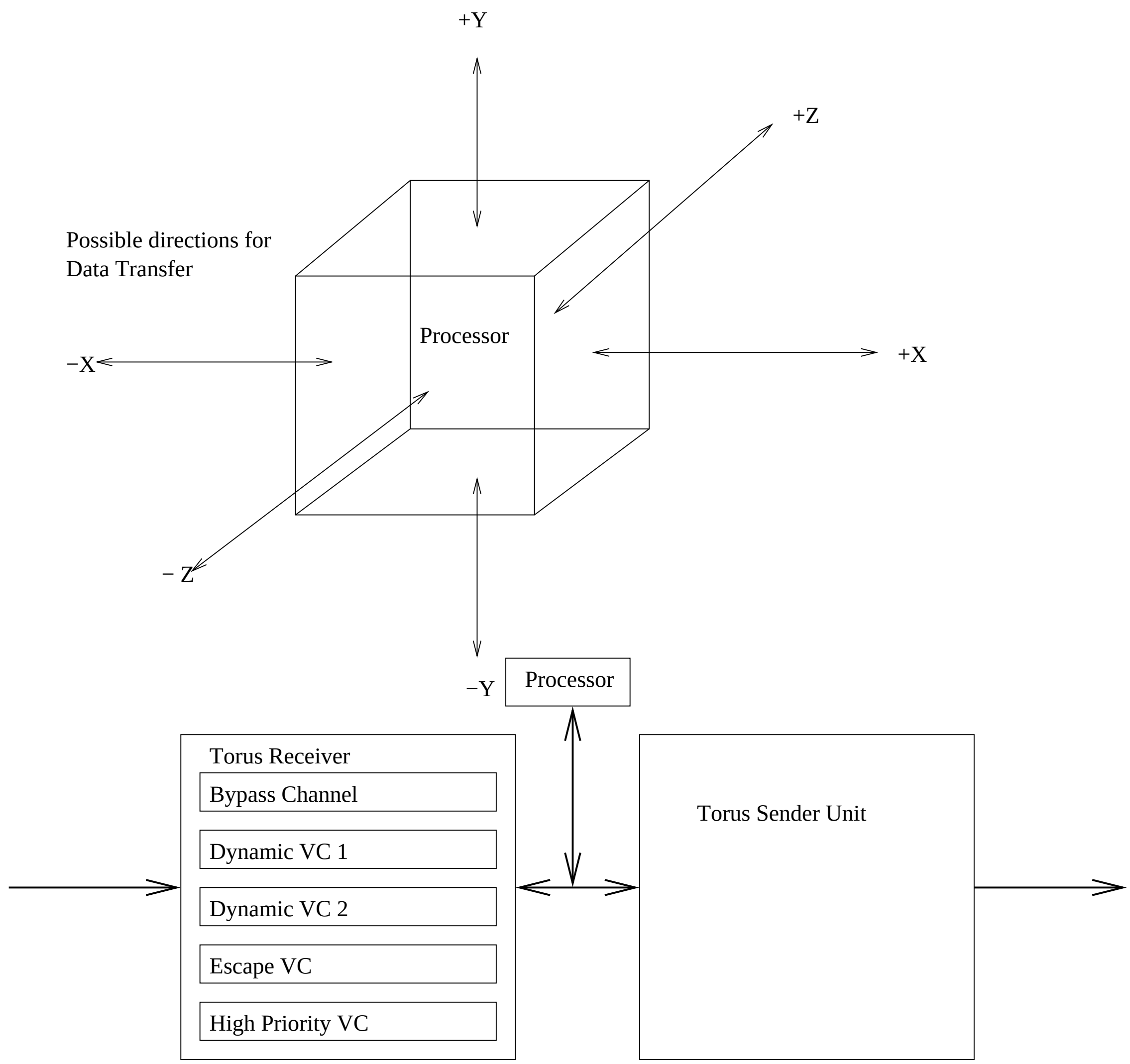

Figure 4.2: Processor sending and receiving messages in all 6 directions through sender and receiver units

latency was used for data transfer time. This is fine for loosely coupled parallel applications, but for tightly coupled fine grained applications, communication time might have significant effect on runtime. BigNetSim [20] is a step forward in that direction. It has a provision for modeling contention in a 3D torus network.

As part of this thesis, we have developed a Bluegene/L Network Simulator which does offline POSE based simulations, using application traces. It models the actual Bluegene/L torus network architecture [11] more closely.

4.3 Host Model

The processor interface consists of network injection and reception FIFOs. There are a total of 8 injection FIFOs organized into two groups: two high-priority and six normal priority FIFOs. Packets in all FIFOs can go out in any direction. Each group of reception FIFOs contain 7 FIFOs, one high priority one and one dedicated to each of the incoming directions. The simulator can be split into two parts. The first part reads the logfiles and elapses CPU time for events. The messages generated by the events are checked to see if destination is in the current node. If so, the message is sent directly or else it is sent on the network. The second part is the network simulator which plugs into the application module, sending and receiving messages. Once a message is received by the Network Interface poser, it splits it into packets. Each packet can hold multiples of 32 bytes, upto 256 bytes and has a 4 byte trailer. The poser then sends out a arbitration request to the sender units seeking to transfer data.

4.4 Receiver and Sender Units

Each receiver has four virtual channels (VCs) as shown in 4.2. Escape VC helps in preventing deadlocks. Bypass channel can be used to flow through a node without any buffering. Each buffer has 1KB of memory, enough to hold four full sized packets. Escape "bubble" VC can be used only when dynamic VCs are unavailable. A token flow control algorithm is used to prevent overflowing the VC buffers. A VC is marked unavailable unless 8 tokens are available. But token counts are updated based on the packet length . Escape VC can be used only if we can guarantee that space required for a full sized packet i.e 256B is available, even after reserving space for the current packet. The simulator implementation is designed to include all the issues mentioned above.

4.5 Routing and Arbitration

Routing can be done in any of the six directions. The header includes six hint bits which indicate the direction to be taken. The simulator implementation selects upto one each in x,y, and z directions, by calculating shortest distances based on the torus topology.

Arbitration is distributed and pipelined, but occurs in three phases

1. "Join the Shortest Queue" algorithm is used to choose the preferred direction and VC. Each sender virtual channel provides the available downstream tokens and the VC with most available token is chosen. In the simulator implementation, we choose upto three shortest directions. Then each direction is scanned for available tokens . Escape VC is used only when the other three VCs don't have available tokens. High priority VC is treated as a dynamic VC. The VC with highest available tokens is selected.

2. "Serve the Longest Queue" is used to determine which receiver virtual channel wins the right to request senders. If a high priority packet is requesting, it wins. Other than that, the FIFO with the least number of available tokens it is selected. A programmable fraction of cycles are used to select winners in random. In the simulator implementation, selection is based on a first come first serve basis with random selections for requests at the same time.

3. Sender Arbiter gives top priority to the acks. Next in order are packets in the network. Finally come the packets from the processor in the node (except the high priority FIFO). The fraction of bias can be controlled to give higher priority to either of the transfer modes. In the simulator implementation, we have a Switch poser which asynchronously collects data from the senders about available downstream tokens. All arbitration and routing requests are handled by the switch poser.

Figure 4.3 shows a simplified flow of control in the simulator .

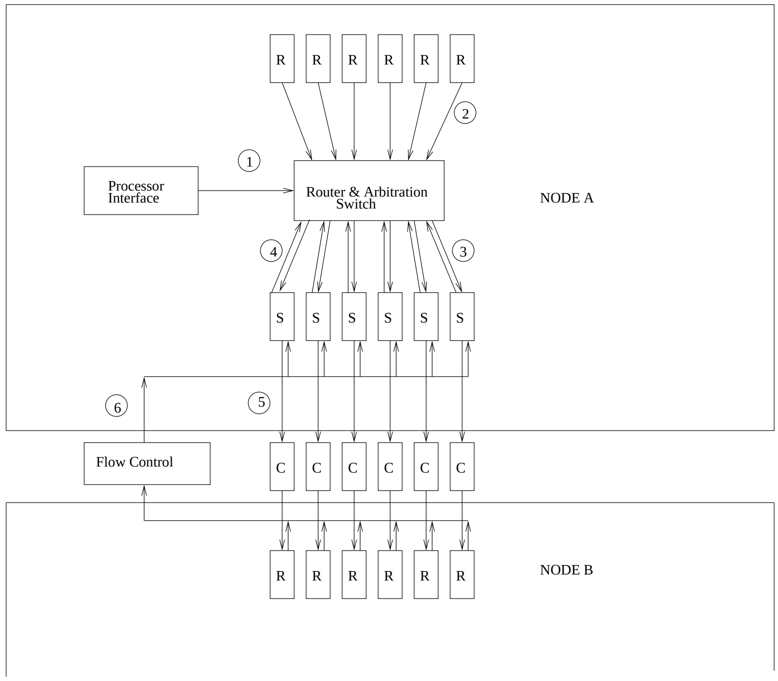

Figure 4.3: Flow of Control in Network Simulator

1. Processor Interface injects packets into the network. The Switch poser looks up the latest data and tries to select the optimal VC as described in the routing and arbitration section. If such a VC is available it dispatches it to Sender. Otherwise it puts it into a queue of requests that it maintains.

2. Receiver sends arbitration requests to the Switch poser, which does similar routing and arbitration as explained previously.

3. Packets are dispatched to sender. Channel posers are used to dispatch them. Available token data inside the Switch poser is updated.

4. On the successful reception and forward by the receiver in the next Node, Sender updates it's data about available tokens in the Switch Poser.

5. Bandwidth Sharing (packet multiplexing) between receiver VCs going to same sender is done by channel posers . This is a simple way to implement locking of a sender unit

when a receiver VC is sending a packet.

6. Receiver VC updates the Sender when a packet is received by sending a token ack. The sender updates the Switch unit, which in turn searches it's requests queue to determine if a new connection can be made between the sender, and receiver VC corresponding to the stored request.

Virtual Cut-Through buffering ensures that latency is reduced by pipelining routing and arbitration . Only the route flits endure these delays and the data flits have a free flow. When there is head of line blocking in the VCs, all of the packet comes and blocks in the same virtual channel.

4.6 Simulation Results

We use a Jacobi1D application with 1D decomposition to generate bluegene traces. The parallel Jacobi1D program uses a one-dimensional blocked distribution of data which is divided into chunks. Each Chunk can be implemented as a Charm++ parallel object. The life cycle of this object `jacobiLifeCycle` can be written in structured-dagger program as shown in Fig 4.4. Entry function jacobiLifeCycle() is defined as special remotely invocable function of the Chunk object. chares concurrent objects in Charm++ whose methods can be remotely invoked. When the messages from chares possessing the neighboring strips on the left and right arrive, the methods `getStripFromLeft` and `getStripFromRight` are triggered respectively. Only when both these messages have arrived, the computation can be performed by the `doWork` function.

Figure 4.5 contains timings for the simulation on a GigE based cluster. There is a decent speedup upto 32 processors. Since network simulators are very sensitive to small messages, a better interconnect might lead to better speedups for over 64 processors.

Figure 4.6 shows the huge impact of bandwidth on the application. Apart from the single user broadcast, there are a number of broadcasts during startup for CHARM++. So the

```
entry void jacobiLifeCycle()
{
 for (i=0; i<MAX_ITER; i++)
 {
  atomic{sendStripToLeft();sendStripToRight();}
  overlap
   {
    when getStripFromLeft(Msg *leftMsg) {
      atomic { copyStripFromLeft(leftMsg); }
    }
    when getStripFromRight(Msg *rightMsg){
      atomic { copyStripFromRight(rightMsg); }
    }
   }
  atomic{ doWork(); /* Jacobi Relaxation */ }
 }
}
```

Figure 4.4: Sample structured dagger code

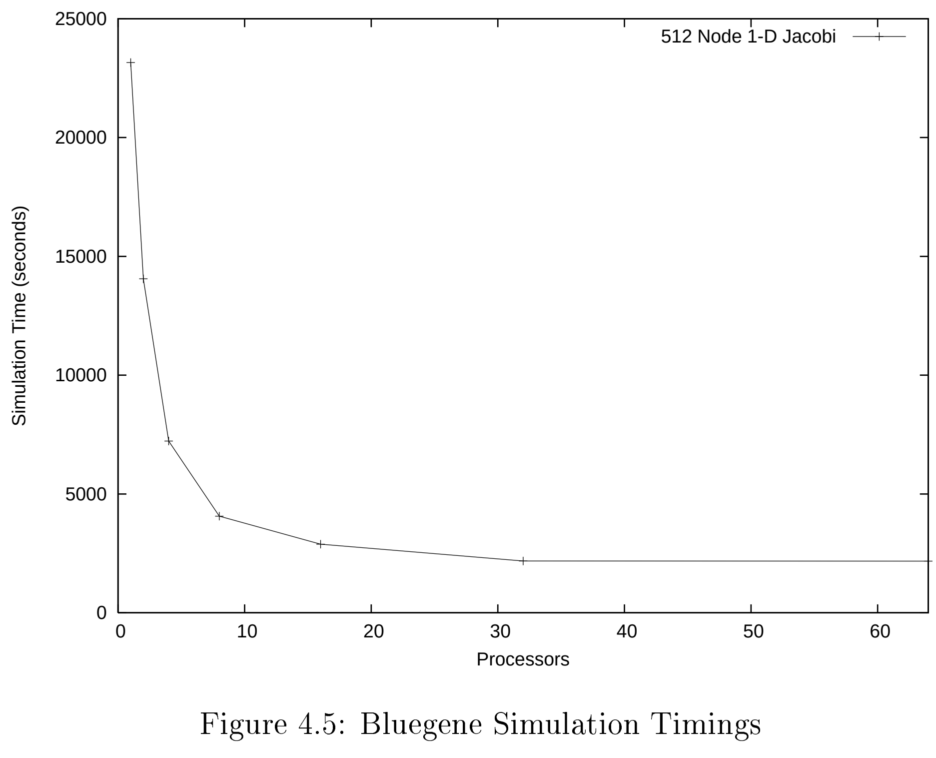

Figure 4.5: Bluegene Simulation Timings

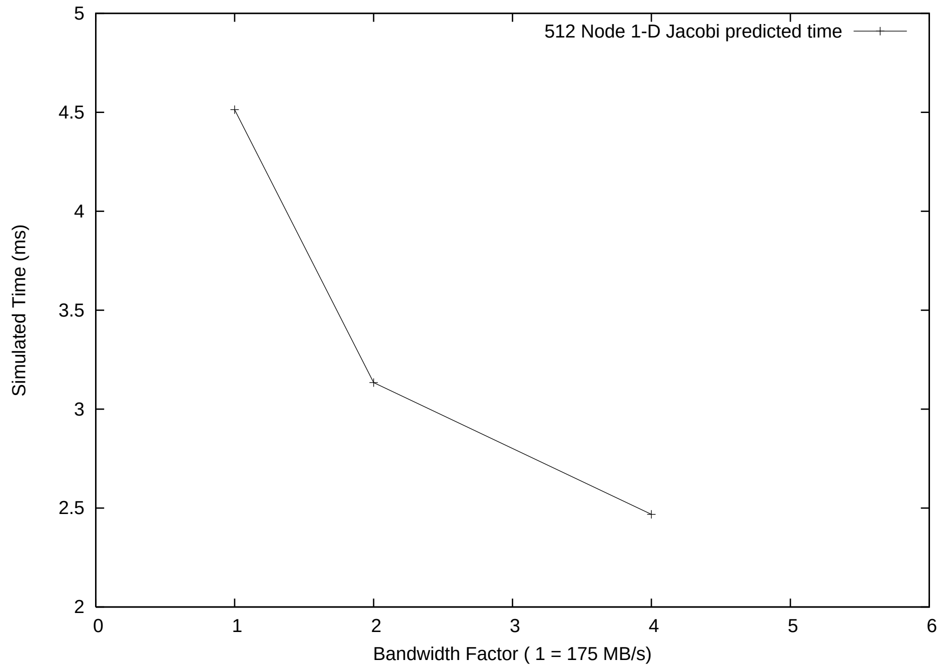

Figure 4.6: Impact of bandwidth on simulation

launch node bandwidth becomes a bottleneck, as it has to send messages to all the nodes (in the present naive broadcast scheme).

Figure 4.7 shows that latency doesn't impact the simulation as much as bandwidth . This has more to do with the small number of messages received by the chunk currently. Also even though broadcast operations experience magnitudes increase in latency, congestion effects (related to bandwidth) far outstrip them for the values plotted .

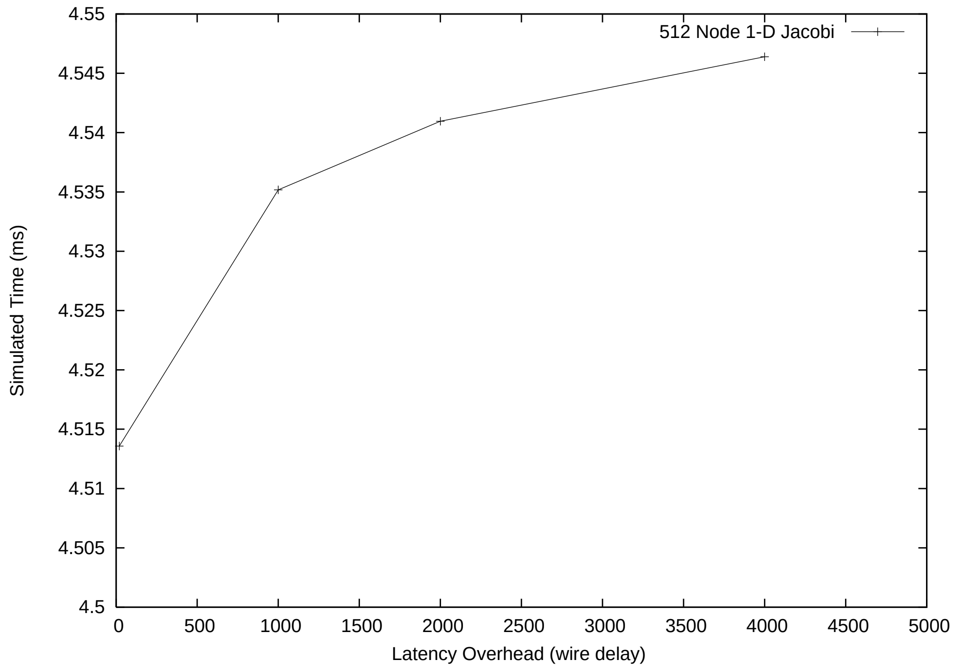

Figure 4.7: Impact of Latency on simulation

Chapter 5

Conclusion

Interconnection networks play a critical role in the performance of parallel applications. We have developed a network simulator which is based on POSE, a parallel discrete event simulation framework. This enables us to simulate large networks in detail . The network might have hundreds of thousands of components including switch,network adapters,virtual channels,node,switch and pci channels. Serial execution is limited by memory of current machines. Even otherwise, for small sized simulations, the runtime is huge.

We have simulated QsNet in detail and compared simulation results with actual runs on Pittsburgh Supercomputing Centre's Lemieux machine. Contention effects were modeled to a good accuracy . This simulator can be extended to involve a lot of other popular interconnects which use indirect switching technology. Bluegene/L afforded us a opportunity to simulate a large direct network in detail. We built on previous work to do application based performance prediction using our bluegene network simulator. Thus we modeled two different networks which represent most of the existing and future network architectures.

5.1 Future work

A tree network for collective operations should be modeled for Bluegene/L. Networks with large number of components take quite a bit of time to simulate even in parallel . A more tight coupling of the parallel discrete event simulation framework and the network simulator could lead to potential savings in time.

The network simulator could have more in depth modeling at the electrical engineering level. A gate level hardware model would also be a good aid for interconnection network designers.

References

[1] Infiniband Trade Association. Infiniband Architecture Specification Vol 1, Release 1.0.a, June 2001.

[2] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W. Su. Myrinet—A Gigabit-per-Second Local-Area Network. *IEEE Micro*, 15(1):29–38, February 1995.

[3] L. V. Kale and Milind Bhandarkar. Structured Dagger: A Coordination Language for Message-Driven Programming. In *Proceedings of 2nd International Euro-Par Conference*, volume 1123-1124 of *Lecture Notes in Computer Science*, pages 646–653, September 1996.

[4] L. V. Kalé, Milind Bhandarkar, Narain Jagathesan, Sanjeev Krishnan, and Joshua Yelon. Converse: An Interoperable Framework for Parallel Programming. In *Proceedings of the 10th International Parallel Processing Symposium*, pages 212–217, Honolulu, Hawaii, April 1996.

[5] L. V. Kale and Sanjeev Krishnan. Charm++: Parallel Programming with Message-Driven Objects. In Gregory V. Wilson and Paul Lu, editors, *Parallel Programming using C++*, pages 175–213. MIT Press, 1996.

[6] L.V. Kale and S. Krishnan. Charm++: A portable concurrent object oriented system based on C++. In *Proceedings of the Conference on Object Oriented Programming Systems, Languages and Applications*, September 1993.

[7] Sameer Kumar and L. V. Kale. Opportunities and Challenges of Modern Communication Architectures: Case Study with QsNet. Technical Report 03-15, Parallel Programming Laboratory, Department of Computer Science, University of Illinois at Urbana-Champaign, 2003.

[8] Sameer Kumar and L. V. Kale. Scaling collective multicast on fat-tree networks. Technical Report 03-11, Parallel Programming Laboratory, Department of Computer Science, University of Illinois at Urbana-Champaign, 2003.

[9] C.E. Leiserson. Fat-trees: Universal Networks for Hardware-Efficient Supercomputing. *IEEE Transactions on Computers*, 34(10), October 1985.

[10] Quadrics Supercomputers World Ltd. Elan reference manual, November 1999.

[11] M.Blumrich, D.Chen, P.Coteus, A.Gara, M.Giampapa, P.Heidelberger, S.Singh, B.Steinmacher-Burow, T.Takken, and P.Vranas. Design and analysis of the bluegene/l torus interconnection network. Technical report, IBM Research Division, T.J.Watson Research Center,P.O.Box 218, Yorktown Heights, NY 10598, 2003.

[12] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard*, May 1994.

[13] José Duato Sudhakar Yalamanchili Lionel Ni. *Interconnection Networks, An Engineering Approach*. 2003.

[14] Fabrizio Petrini, Wu chun Feng, Adolfy Hoisie, Salvador Coll, and Eitan Frachtenberg. The Quadrics Network: High Performance Clustering Technology. *IEEE Micro*, 22(1):46–57, January-February 2002.

[15] Fabrizio Petrini and Marco Vanneschi. K-ary N-trees: High performance networks for massively parallel architectures. Technical Report TR-95-18, 15, 1995.

[16] David E.Culler Richard P.Martin, Amin M.Vahdat and Thomas E. Anderson. Effects of communication latency, overhead, and bandwidth in a cluster architecture. In *Proc. 24th International Symposium on Computer Architecture(ISCA-24),ACM Press,pp 85-97*, 1997.

[17] Neelam Saboo, Arun Kumar Singla, Joshua Mostkoff Unger, and L. V. Kalé. Emulating petaflops machines and blue gene. In *Workshop on Massively Parallel Processing (IPDPS'01)*, San Francisco, CA, April 2001.

[18] Terry Wilmarth and L. V. Kalé. POSE: A study in scalable parallel discrete event simulation. Technical Report 03-01, Parallel Programming Laboratory, Department of Computer Science, University of Illinois at Urbana-Champaign, June 2003.

[19] Gengbin Zheng, Gunavardhan Kakulapati, and L. V. Kale. A parallel simulator for performance prediction of large parallel machines. Technical Report 03-02, Parallel Programming Laboratory, Department of Computer Science, University of Illinois at Urbana-Champaign, June 2003.

[20] Gengbin Zheng, Terry Wilmarth, Orion Sky Lawlor, Laxmikant V. Kalé, Sarita Adve, and David Padua. Performance modeling and programming environments for petaflops computers and the blue gene machine. In *NSF Next Generation Systems Program Workshop, 18th International Parallel and Distributed Processing Symposium(IPDPS)*, Santa Fe, New Mexico, April 2004.