Architecture for supporting Hardware Collectives in Output-Queued High-Radix Routers

Sameer Kumar, Laxmikant V. Kalé
Department of Computer Science
University of Illinois at Urbana-Champaign
{skumar2, kale}@cs.uiuc.edu
Craig Stunkel
IBM T.J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598
stunkel@us.ibm.com

Abstract

Collective communication performance is critical for many applications. In this paper, we present an architecture to efficiently support collective operations (like multicasts and reductions) in the switches of parallel computer interconnects. We present an output queuing switch architecture with cross-point buffering. Output queuing architectures have been less popular in the past as they require more internal speedup and buffering. However, with current technology it is straightforward to build output-queued switches. We demonstrate in this paper that output-queued architectures make multicasts and reductions fairly easy to implement efficiently. We show the scalability of our schemes to a large number of switch ports. We present performance of multicasts and reductions on individual switches and networks of switches. We assume a fat-tree topology for the networks of switches. We also present simulation results based on synthetic workloads that emulate a molecular dynamics application.

1 Introduction

Collective communication is a critical communication operation involving all or a large number of processors in the system. Traditional approaches to optimizing collective communication use processor-based optimization algorithms. Processor-based optimization schemes send several point-to-point messages. For example a broadcast can be implemented as point-to-point messages sent along a binomial tree rooted at the source. The root processor sends messages to its children. As the children receive message they send those messages to all their children. This scheme has $log(P)$ phases of point-to-point messages.

Software collective optimization schemes have several problems. For short messages, the broadcast completion time is dominated by the CPU and the network interface controller (NIC) overheads of sending the messages. Large messages being sent by the several children may contend for the same communication channels. Software contention avoidance schemes may have to use barriers to keep messages synchronized [10]. Good collective performance also requires that all intermediate processors immediately process and forward the incoming message. Performance is affected if one of the intermediate processors is running an operating system daemon [4], which can delay the collective operation. Moreover, with message driven execution [8] and asynchronous collectives [9] it is possible that the remote processor is busy doing other work and cannot process the message immediately delaying the broadcast completion.

For the above reasons collective communication support is necessary in the communication hardware. One of the approaches studied in literature implements collectives in the network interface. This can reduce the CPU overhead of sending messages as the processors are less involved in the collective operation. This scheme is also unaffected by operating system daemon issues. Performance improvement of network interface reductions has been presented by Panda et. al. [15]. However, performance of collectives can be limited by the slow NIC hardware. Such collectives exchange several point-to-point messages incurring high NIC software overhead. Large messages from different NICs may also contend with each other.

We believe that collective communication should be supported in the switching network. Both multicasts and reductions should be supported in the network switches. Some current clustering interconnects like Quadrics [20] QsNet [16] and Mellanox [14] Infiniband [7] have multicast support in their switches. But multicast performance is restrictive as these switches have input-queued architectures [12]. For example, in Quadrics QsNet only consecutive ports can be multicast to. Input queuing architectures require complex centralized arbitration to achieve high utilization, and are not a natural match for multicast [19, 13, 12]. Many popular interconnects today also do not have reduction support in their switches.

In this paper, a switch-based solution to optimize multicasts and reductions is presented. We propose an output queuing architecture with crosspoint buffering. Our solution derives from existing literature and further extends it. In the past output queuing architectures have been less popular because they require higher internal bandwidth and more memory. But, with the current ASIC technology it is possible to build crosspoint-buffered output-queued switches. Our switch architecture will support efficient multicasts and reductions. With basic multicast and reduction support in switches other collectives like barrier, all-reduce and all-gather can be easily implemented in the network hardware. For example, all-reduce can be implemented as a reduce followed by a broadcast.

We evaluate our switch architecture with several point-to-point and collective benchmarks. We show the throughput and latency of collective operations on output-queued routers. We simulate independent switches and networks of switches. To support collectives in the network a spanning tree has to be built on the network topology, which is topology specific. In this paper we assume a fat-tree topology [11, 17]. Fat-trees are a popular network topology used by several interconnects like Quadrics QsNet [16], Infiniband [7], IBM SP networks. Fat-tree networks have high bisection bandwidth and can be scaled to thousands of nodes. In this paper, we present schemes to build collective spanning trees on fat-tree networks and the performance of collectives using these spanning trees.

We also present the network throughput and latency when several collectives happen simultaneously. Applications like NAMD [18] (a scalable molecular dynamics application) and CPAIMD [24] (a quantum chemistry application) need multiple such simultaneous collectives. The advantages of hardware collectives is shown through a synthetic benchmark that emulates the collectives in NAMD.

2 Router Architecture

Several input and output queuing architectures have been proposed for high performance interconnect switches. Input queuing (IQ) schemes allow simpler data flow but require centralized arbitration to achieve high utilization. IQ routers also suffer from head of line blocking which restricts their throughput. Using multiple virtual channels and smart buffer management improves the performance of input-queued routers [22, 23, 5]. *Virtual output queuing* [13] (VOQ) can fully utilize the switch. Here each input queue has reserved buffer space for every output queue. Virtual output queuing also has a centralized arbiter and requires $O(K^2)$ buffer space, where K is the number of ports.

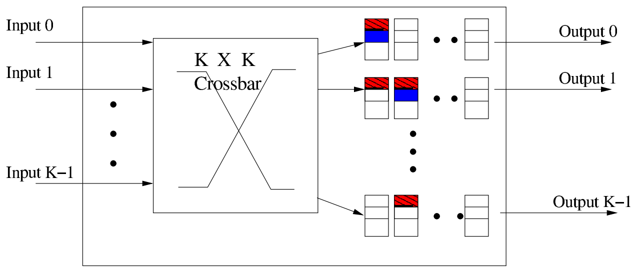
We believe that switch design should have efficient support for multicasts and combines. Input queuing (IQ) and virtual output queuing (VOQ) do not handle multicasts efficiently as they have centralized arbitration [19, 12]. VOQ can achieve full utilization for multicast if every input port has $(2^K - 1)$ queues in a KXK switch, one for every possible subset of output ports. As this requires a tremendous amount of memory, IQ multicast scheduling algorithms use heuristics. Performance can sometimes be severely affected if there is contention for outputs by different multicasts [12].

Two schemes have been proposed to handle multicasts in IQ routers [12, 19, 13], (i) No fanout splitting, and (ii) fanout splitting. Here *fanout* refers to the number of multicast destination ports. In *no-fanout-splitting*, a multicast packet is only sent out if all destination ports are available in that arbitration cycle. The crossbar is used only once, but no-fanout-splitting may require several arbitration cycles to send the packet out and free the input buffer for that packet. No-fanout-splitting is good for multicasts with small fanouts.

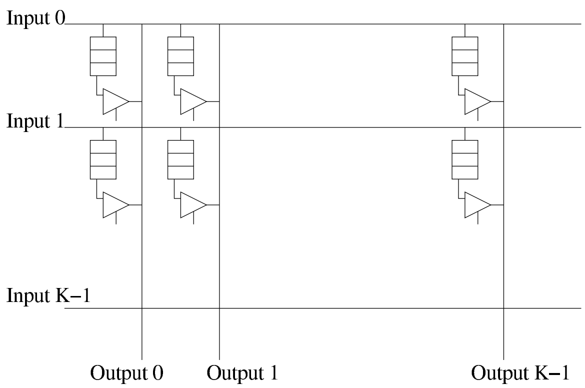
In the *fanout-splitting* scheme a multicast packet is sent to all output ports that are available in that arbitration cycle. In this scheme the multicast packet uses the crossbar bandwidth for several cycles. The maximum achievable utilization for multicast is presented in [12], which is far from full utilization for many traffic patterns. IQ multicast schemes can also have deadlocks in a network of switches.

In this paper, we show the effectiveness of output queuing for hardware collectives. Packets in output queuing are buffered on the output ports of a switch before being sent out. Output queuing has distributed arbitration where each output decides which packet to send independent of other outputs. This architecture is less commonly used as it requires more internal speedup to let input ports talk to several output ports simultaneously. With current ASIC technology it is possible to build output queuing switches. Figure 1(a) shows an output-queued router with buffers at the outputs.

Popular output queuing routers in the past have used shared buffers between output ports [22]. Such shared buffer schemes have limited scalability with respect to link bandwidth and number of ports. We use crosspoint buffering in our router architecture to make the router support high bandwidth links efficiently. Cross-point buffering guarantees that there is a reserved buffer for each pair of



(a) Switch Design



(b) Crosspoint Buffers

Figure 1. Crosspoint Buffering flow control

input and output ports. A graphic description of cross-point buffering is shown by Figure 1(b). Each input port has some reserved memory on every output port. Hence the total buffering required is $O(P^2)$. Packets arriving on input ports are immediately sent to the crosspoint determined by the destination output queue. Our output queuing router with cross point buffering is similar to the SAFC scheme presented in [23]. But [23] only presents the point-to-point performance on one switch. We are mainly concerned with multicast and reduction performance one one switch and networks of switches.

We use virtual cut through routing and credit based flow control [2] between switches. Each switch keeps track of the buffer space available in the next switch. Packets are only sent out if buffer space is guaranteed on every port of the next switch. With crosspoint buffering this implies that all crosspoints for the current input port should have buffer space available for this packet. The flow control is implemented through a credit counter. This counter is initially set to the maximum buffer space at each crosspoint and as packets are sent out it is decremented. When ever the next switch dispatches a packet it sends back the credits to receive more packets.

We show in the next few sections that multicasts and reductions are efficient and easy to implement in such an output-queued architecture. We first show the feasibility of such an architecture.

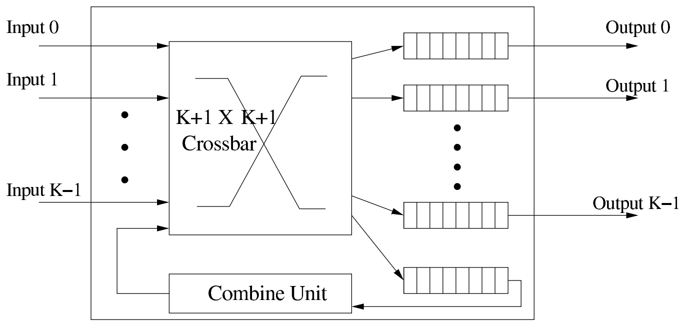
Feasibility of the Output Queuing architecture: Suppose we plan to build an Infiniband 4X switch with a bandwidth of 10Gbps per port. We would also like to support 20m cables or 200ns of round trip time (RTT) . Hence, we would need atleast 250 bytes of memory at each crosspoint. It is usually good to have two or four RTTs for good switch performance. For an 8 port switch the total memory requirement is about 64KB which is easily available in modern ASICs. For a 32 port switch we need 512KB to 1MB of buffer space. With current ASIC technology this should still be possible.

Multicast: Our credit based flow control scheme ensures that when a packet is sent out buffer space is available on all cross-points corresponding to this input port. So for every multicast buffer space will be available on every output port. On arrival, the multicast packet is immediately sent to all the ports determined by the destination address. The multicast packet only uses the crossbar once. Flow-control credits for this multicast packet are only sent back after all multicast packets have been sent out. Hence this scheme can achieve full throughput and also avoid deadlock issues of input queuing schemes.

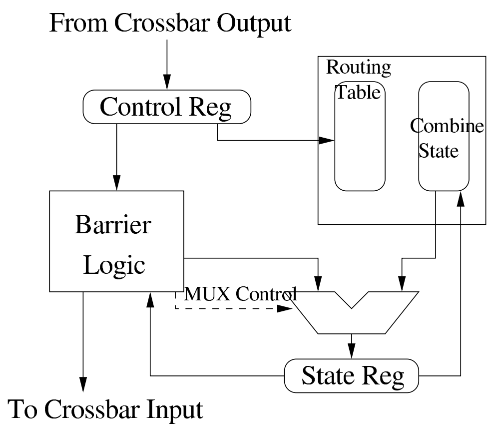
Reduction: Our design also supports the *Combine* operation which can be used to support reductions and barriers in hardware. We extend the barrier combine unit presented in [21] to perform reductions. The combine unit receives packets from the crossbar output and performs reductions. Every reduction has access to local state. For example, in the global sum operation the local state can store the current partial sum. For a global array sum, the local state could be an array of floating point numbers. This local state is updated by the combine unit whenever a reduction packet arrives. After all reduction packets have been processed, the combine unit sends a reduction packet back into the crossbar to be sent to the parent switch in the spanning tree.

The combine unit connects from the output port through a feedback to an input port in the switch, as demonstrated in Figure 2(a). The combine unit behaves like any other output port in the switch. Reduction packets arriving on input ports of the switch are buffered at the output port connected to the combine unit before being processed. The architecture of the combine unit is shown in Figure 2(b).

It can take a few cycles to receive reduction packets, as the entire packet is needed to detect errors. (We do not explicitly simulate errors but we do model the delays.) The header of the packet is stored in the control register. The combine logic uses the address in the packet header to lookup the routing table for the local state of the current re-

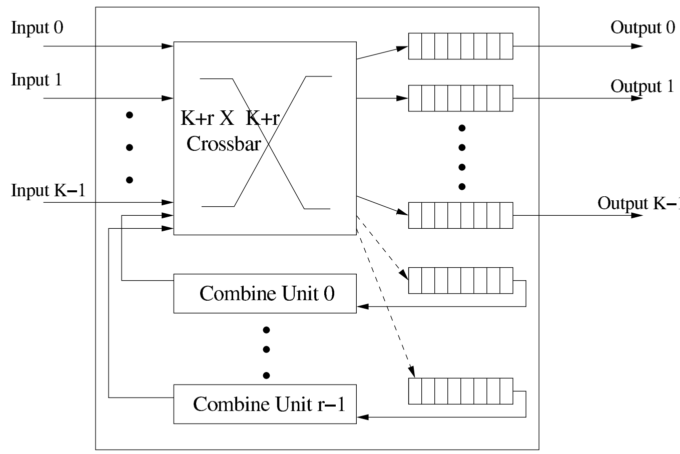


(a) Switch Design with Combine Unit

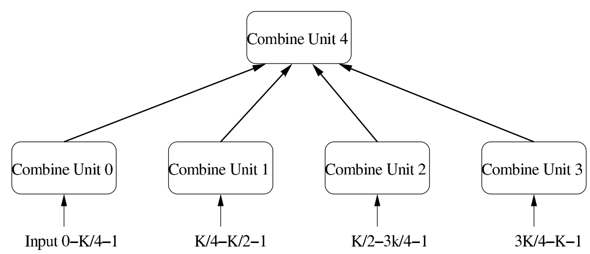


(b) Combine unit Architecture

Figure 2. Combine unit in the Output-Queued Router



(a) Switch with r combine units



(b) Combine units organized in a tree ($r = 5$)

Figure 3. Combine units in the Output-Queued Router

duction. In the following cycles the logic unit computes and updates the local state based on the data from the packet.

For short reductions and barriers it may be possible to pipeline packet arrival and computation to process one packet every cycle [21]. But for larger reductions involving several data points, the combine unit may stall on each combine operation. For switches with a large number of ports a single combine unit may become a point of contention. As ASIC speeds are much slower than custom designed CPU speeds, this may hamper the overall efficiency of the global reduction operation.

Figure 3(a) shows the switch architecture with 'r' combine units. The combine units are organized as a tree with $r-1$ leaves and one parent (Figure 3(b)). The leaves process the reduction packets from a subset of ports and pass their partial result to the root of the tree. Such a hierarchical design scales to more number of ports as several combines at the leaves can happen simultaneously.

2.1 Building a collective spanning tree

Spanning trees are essential to support collectives in the network hardware. These spanning trees can be directed trees where packets only travel in one direction on each hop. A broadcast with one source needs such a tree. If the network time to do a broadcast does not depend on the root of the spanning tree, we can also build undirected spanning trees for broadcasts. Here any leaf of the tree can do a broadcast with the same overhead. It is possible to build such a tree in a fat-tree network. The time to do a broadcast would be $O(log(P))$ independent of which leaf has sent the broadcast message. Our switch design has support for undirected spanning trees. Such undirected trees save routing table memory as any leaf can send messages. With directed trees [21] each sender would require a separate tree.

The routing table has a bit vector of destination ports for each collective address, as opposed to a parent and a list of children. For a multicast operation packets are sent to all ports set except the port on which the packet arrived on.

We implement combines as follows: suppose a routing table destination bit-vector has k outputs set, then the combine manager would process k-1 reduction packets and send the current partial result to the remaining port on which it did not receive a packet.

Both multicasts and combines use the same routing table entries. The tag in the packet determines whether the operation is a multicast, barrier, reduction etc.

2.2 Fat-tree Networks

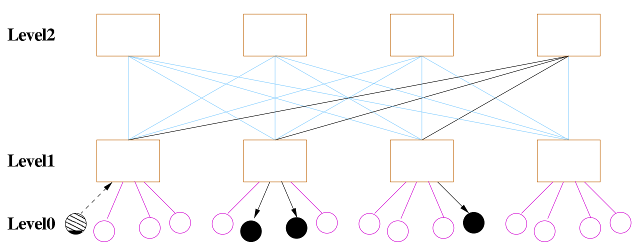


Figure 4. Fat-tree with 16 nodes

In this section, we describe our design to build collective spanning trees on network topologies. We take fat-trees as an example of an interconnect topology. Fat-trees are generalizations of k-ary n-trees [17]. Figure 4 shows a 4-ary 2-tree network. Routing in a k-ary n-tree has two phases, (i) the upward phase where a packet is sent to any of the lowest common ancestors of the source and the destination, (ii) the downward phase where the packet is routed from this ancestor to the destination through a fixed path.

This scheme can be extended to support collectives as follows: a multicast packet is sent to one of the lowest common ancestors of all the nodes from where it is routed to all the destinations in the tree. The advantage of using one common ancestor for all the nodes is that the spanning tree can be used by any leaf to do a multicast.

Collective tree algorithms for the Quadrics QsNet are presented in [3]. Here several trees are built to support hardware multicast on a discontinuous set of nodes. This is because the Quadrics network can only multicast to a contiguous set of nodes. Our switch architecture places no such constraints. We propose schemes to build several spanning trees to support multiple simultaneous multicasts and reductions.

Figure 4 illustrates a simple collective tree building algorithm. In the figure, the port $K-1$ is used to go up to the lowest common ancestor of all the nodes. Routes from this ancestor to all the destinations constitute the spanning tree. This simple scheme would lead to a load-imbalance among the top level switches when several multicast trees need to be built. An more effective tree building algorithm is presented below :

```
buildTree(id, destlist, swlist, tlist, up)
id : the switch id of the current switch
destlist : list of processor destinations
swlist : list of previous switches
tlist : list of treeInfos, where each
treeInfo contains the list of output ports
at that switch
up : boolean flag that shows direction

begin
   swlist.insert(id);
   //Need to go further up
   if(!inHighestLevel(id, destlist) && up) {
      parent = leastLoadedParent(id);
      buildTree(parent,destlist,swlist,
                 tlist, true)
   }

   //Going down in the fat-tree
   for count : 0 -> numPorts/2 - 1
     if(child[count] routesto destlist) {
        tlist[my_pos].insert(count);
        buildTree(child[count],destlist,
                   swlist,tlist, false);
     }
end
```

Here *leastLoadedParent()* gets the least loaded parent for the current switch. The load of the switch is determined by the number of multicast trees passing through that switch. This algorithm minimizes contention on the upward path of the packet. It also load-balances the routing memory required for each collective operation by choosing switches with fewer collective trees passing through them.

3 Network simulation

We simulated switches with the above architecture using POSE [25] which is a parallel event driven simulation language. We simulated 8 port and 32 port routers in a fat-tree topology with adaptive routing. Table 1 shows the parameters of our simulation. These parameters are derived from Infiniband 4X interconnects.

We first present the throughput and packet latency of point-to-point communication using the well known communication patterns *transpose, bit reversal, complement and uniform*. We simulated a 256 node fat-tree network with 8 port and 32 port output queuing switches. We also varied the amount of buffer space at each crosspoint in the switch.

Parameter	Value
Bandwidth	10 Gbps
Packet Size	256 bytes
Channel Delay	20 ns
Switch Delay	90ns
Switch Ports	32
ASIC Speed	250 Mhz
NIC Send Overhead	1300 ns
NIC Recv. Overhead	1300ns

Table 1. Simulation Parameters

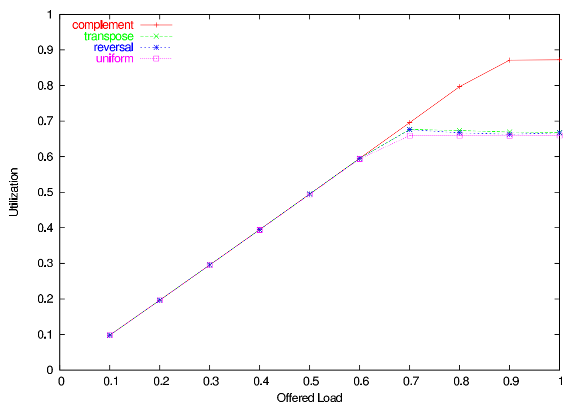


Figure 5. Throughput on a 256 node network with 8 port switches and 2 packet buffers

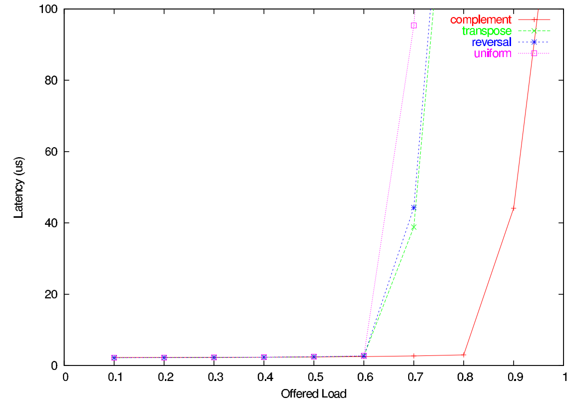


Figure 6. Latency on a 256 node network with 8 port switches and 2 packet buffers

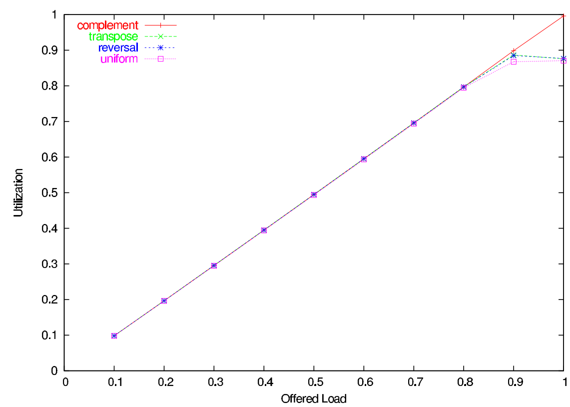


Figure 7. Throughput on a 256 node network with 8 port switches and 4 packet buffers

Figures 5, 6, 7 and 8 show the throughput and response times with 8 port switches. Figures 9, 10, 11 and 12 show the performance of 32 port switches.

Performance is best with 32 ports and 4 packets for each crosspoint. With 32 port switches the fat-tree has 32 switches organized in two levels. Since complement is contention free [17, 6] its throughput is 100% at full load. Uniform, Transpose and Reversal also have good throughput of about 93%. This high throughput is due to output queuing, adaptive routing [1] in fat-trees and the fact that there are only two levels or 3 points of contention in the entire network. Response times are also good for Complement and Uniform and only blow up for Transpose and Reversal for load factors greater than 0.9.

A performance evaluation of 8 port input-queued routers and a 256 node fat-tree networks is presented in [17]. Our output queuing routers perform better for all permutations with close to full throughput.

3.1 Multicast Performance

The performance of multicast on an 8 port switch is presented in Figure 13. Here each port sends to a packet to random destinations and with an average fanout of 4. Packets are generated on each port with a Poisson distribution with a mean inversely proportional to the load factor.

As the mean fanout of the multicast is four, performance saturates at a load factor close to 0.25. With only two nodes sending data the performance of multicast saturates at a load factor of 0.8, as shown in Figure 14. These results are better than the performance of virtual output-queued routers, presented in [19], where for un-correlated traffic with fanout of 4 the performance for a 2X8 switch is 0.65 and for an 8X8 switch it is 0.22.

Figure 15 shows the multicast latency for a 256 node fat-

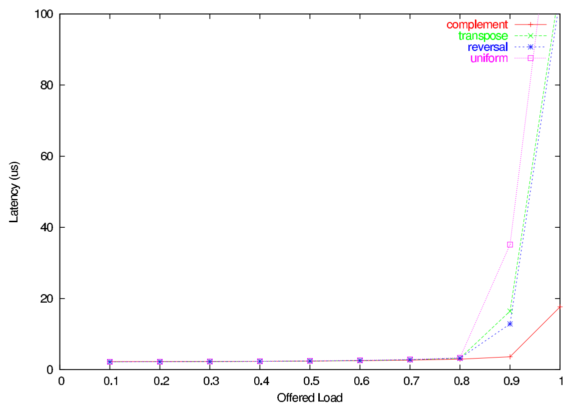


Figure 8. Latency on a 256 node network with 8 port switches and 4 packet buffers

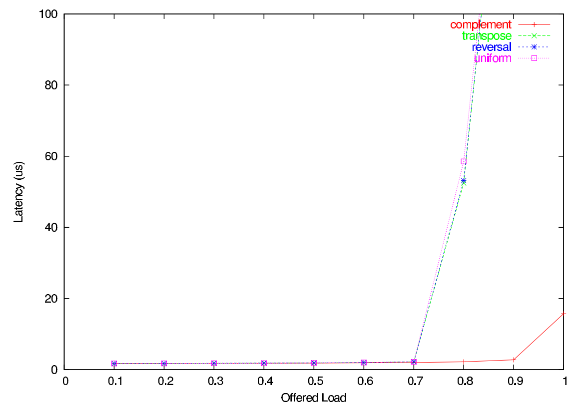


Figure 10. Latency on a 256 node network with 32 port switches and 2 packet buffers

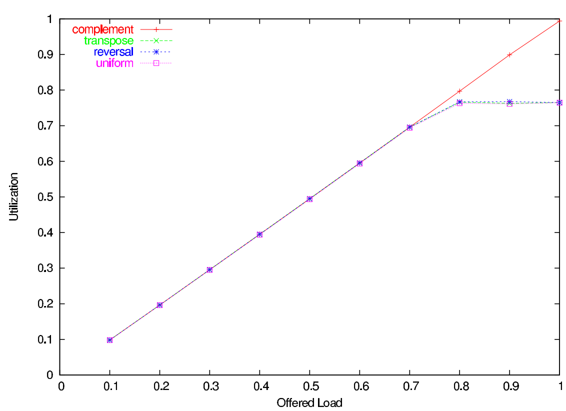


Figure 9. Throughput on a 256 node network with 32 port switches and 2 packet buffers

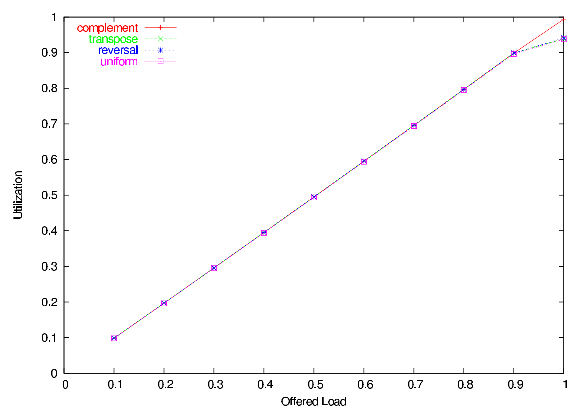


Figure 11. Throughput on a 256 node network with 32 port switches and 4 packet buffers

tree network. Here each node sends a multicast packet to a random set of destinations with an average fanout of 8. It can be seen that the latency is stable for load-factors under 0.125, showing the effectiveness of our scheme on a network of switches.

3.2 Reduction Performance

The simulated performance of a reduction is shown in Figure 16 for a fat-tree network with 256 nodes. With only one reduction a network with 8 port switches performs better than a network with 32 port switches for message sizes greater than 64 bytes. The 32 port performance degrades with increasing message size because of the stalls in the reduction pipeline for large packets (Section 2).

Multiple combine units enhance the performance of 32 port switches. Reduction completion time with 32 port switches and 5 combine units is shown in Figure 16. (Reductions on fat-tree networks use only one port in the upward path and a maximum of K/2 ports. Hence the effective number combine units is actually 3.) Notice this performance is good even with large reductions.

4 Synthetic MD benchmark

In this section, we present the advantages of having hardware collective support in the network. We present the performance of a synthetic benchmark that emulates our molecular dynamics application NAMD. Processors in NAMD multicast coordinates to a small subset of processors which compute forces on those atoms and return results back to the source processor. In the synthetic benchmark,

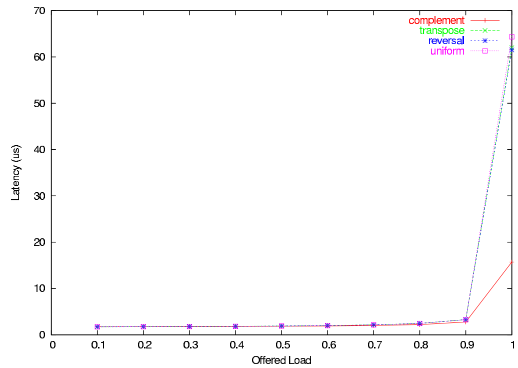


Figure 12. Latency on a 256 node network with 32 port switches and 4 packet buffers

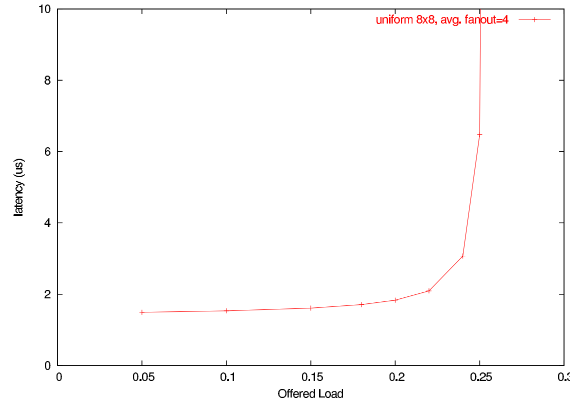


Figure 13. Response time for multicast traffic on an 8X8 switch with an average fanout of 4

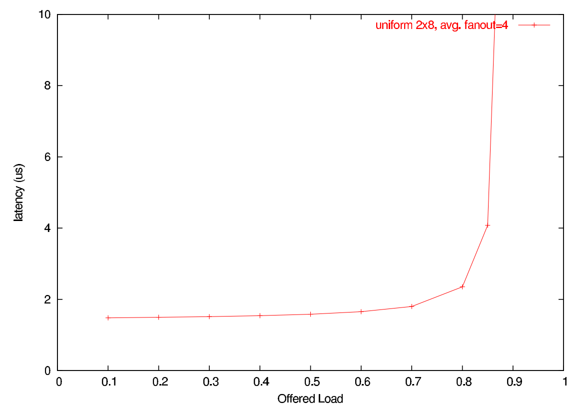


Figure 14. Response time for multicast traffic on a 2X8 switch with an average fanout of 4

$P/16$ processors multicast data to random destinations with an average fanout of 16. In the benchmark on 256 nodes, 16 nodes send multicast messages with an average fanout of 16. Here fanout represents the number of destination nodes of a multicast.

Figures 17 and 18 show the performance of this synthetic benchmark with hardware multicast and multicast with point-to-point messages on 256 nodes. The figures clearly show the advantage of hardware multicast. As the network with 8 ports has more levels and hence more points of contention, hardware multicast has more performance gains. On parallel systems with thousands of nodes even with 32 port switches there will be several levels and more contention for switch outputs. We believe that performance gains of hardware collectives on such large systems are indicated in the 8 port plots.

5 Summary and Future Work

In this paper, we present the advantages of output queuing for supporting hardware collectives in parallel system interconnects. We showed that an output-queued router with crosspoint buffering achieves better performance with multicast than some of the related work presented in literature. Multicast is quite easy to implement in our router and it avoids deadlocks and other problems faced by input queued routers.

We show that output queuing also has impressive performance with permutations on fat-tree networks. We are able to achieve almost full network throughput for common permutations with 32 port switches and 4 packet buffers. Large radix crossbars enable us to build fat-tree networks with fewer levels which minimizes contention. Thus we show the need to build large radix routers.

We also present schemes to support reductions in switches. We extend a barrier scheme presented in literature to support reductions. We show that the scheme with only one barrier unit does not scale for large packet reductions. We then present a hierarchical scheme that scales to high radix routers and large reductions. A performance comparison of the two schemes is also presented in the paper.

In this paper, we present an algorithm to build spanning trees on fat-tree networks. This greedy algorithm uses heuristics that aim at minimizing contention on the upward path of packets and also the routing memory required for the collectives. We also present the performance of several simultaneous multicasts. Such simultaneous collectives are used by NAMD.

We plan to study output queuing switch with input flow-

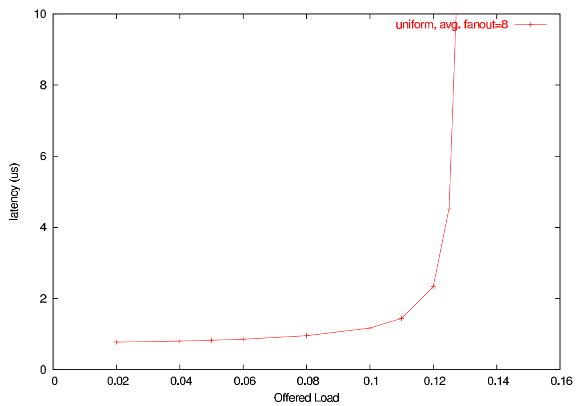


Figure 15. Multicast response time on a 256 node fat-tree network with an average fanout of 8

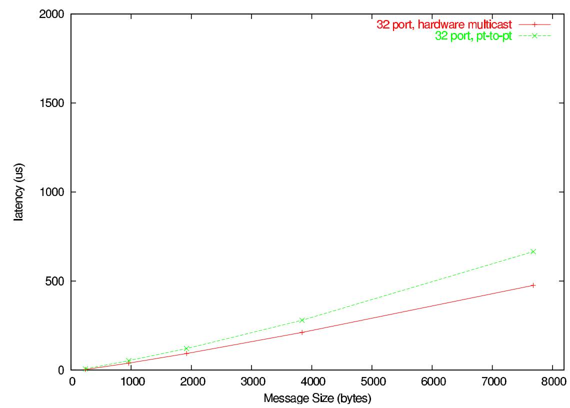


Figure 17. Comparison of hardware multicast and pt-to-pt messages for several small simultaneous multicasts of average fanout 16

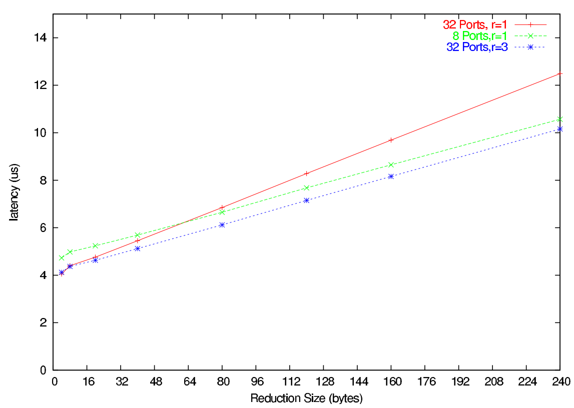


Figure 16. Reduction Time on 256 nodes

control buffers. For Infiniband 4X, buffer space at cross-points may not be a serious issue. However with higher bandwidth networks the RTT would become several packets requiring input flow-control buffers. Having such an input buffer will reduce total buffer space but may increase the hardware logic.

References

[1] Y. Aydogan, C. B. Stunkel, C. Aykanat, and B. Abali. Adaptive source routing in multistage interconnection networks. In *Proceedings of the International Parallel Processing Symposium*, pages 258–267, 1996.

[2] Blackwell, T. Chang, K. Kung, H.T., and L. D. Credit-based flow control for ATM networks. In *Proc. of the First Annual Conference on Telecommunications R&D in Massachusetts*, 1994.

[3] S. Coll, J. Duato, F. Petrini, and F. J. Mora. Scalable hardware-based multicast trees. In *Supercomputing 2003*, Phoenix, AZ, November 2003.

[4] S. P. Darren J. Kerbyson, Fabrizio Petrini. The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8,192 Processors of ASCI Q. In *Supercomputing 2003*, November 2003.

[5] M. Galles. The sgi spider chip. In *Proceedings of Hot Interconnects IV*, pages 141–146, 1996.

[6] S. Heller. Congestion-free routing on the cm-5 data router. *LNCS*, 853:176–184, 1994.

[7] Infiniband Trade Association. Infiniband Architecture Specification, Release 1.0, October 2000.

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

[9] L. V. Kale, S. Kumar, and K. Vardarajan. A Framework for Collective Personalized Communication. In *Proceedings of IPDPS'03*, Nice, France, April 2003.

[10] S. 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.

[11] C. Leiserson. Fat-trees: Universal networks for hardware efficient supercomputing. *IEEE Transactions on Computers*, 35(10):892–901, 1985.

[12] M. A. Marsan, A. Bianco, P. Giaccone, E. Leonardi, and F. Neri. On the throughput of input-queued cell-based switches with multicast traffic. In *Proceedings of IEEE Infocom*, 2001.

[13] N. McKeown, M. Izzard, A. Mekkittikul, W. Ellersick, and M. Horowitz. Tiny Tera: A packet switch core. *IEEE Micro*, 17(1):26–33, /1997.

[14] Mellanox Ltd. http://www.mellanox.com.

[15] A. Moody, J. Fernandez, F. Petrini, and D. K. Panda. Scalable nic-based reduction on large-scale clusters. In *Supercomputing 2003*, Phoenix, AZ, November 2003.

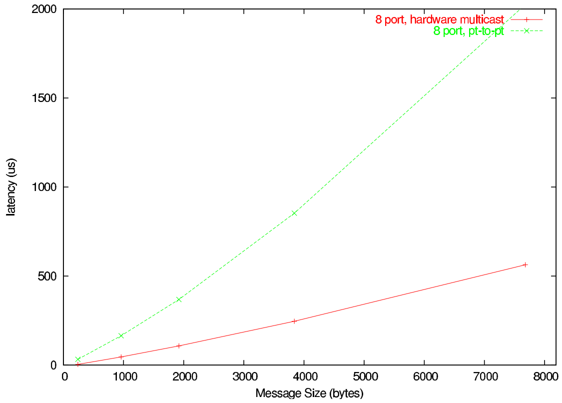


Figure 18. Comparison of hardware multicast and pt-to-pt messages for several small simultaneous multicasts of average fanout 16

[16] F. Petrini, S. Coll, E. Frachtenberg, and A. Hoisie. Performance Evaluation of the Quadrics Interconnection Network. *Cluster Computing*, 6(2):125–142, April 2003.

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

[18] J. C. Phillips, G. Zheng, S. Kumar, and L. V. Kalé. NAMD: Biomolecular simulation on thousands of processors. In *Proceedings of SC 2002*, Baltimore, MD, September 2002.

[19] B. Prabhakar, N. McKeown, and R. Ahuja. Multicast scheduling for input-queued switches. *IEEE Journal of Selected Areas in Communications*, 15(5):855–866, 1997.

[20] Quadrics Ltd. http://www.quadrics.com.

[21] R. Sivaram, C. Stunkel, and D. Panda. A reliable hardware barrier synchronization scheme. In *Proceedings of IPPS 1997*, pages 274–280.

[22] R. Sivaram, C. B. Stunkel, and D. K. Panda. HIPIQS: A high-performance switch architecture using input queuing. *IEEE Transactions on Parallel and Distributed Systems*, 13(3):275–289, 2002.

[23] Y. Tamir and G. L. Frazier. High performance multiqueue buffers for vlsi communication switches. In *Proceedings of 15th International Symposium on Computer Architecture (ISCA)*, pages 343–354, 1988.

[24] R. Vadali, L. V. Kale, G. Martyna, and M. Tuckerman. Scalable parallelization of ab initio molecular dynamics. Technical report, UIUC, Dept. of Computer Science, 2003.

[25] T. Wilmarth and L. V. Kalé. Pose: Getting over grainsize in parallel discrete event simulation. In *2004 International Conference on Parallel Processing*, page (to appear), August 2004.