Efficient Agent-based Multicast on Wormhole Switch-based Irregular Networks

Yi-Fang Lin Jan-Jan Wu
Institute of Information Science, Academia Sinica
Nankang, Taipei, R.O.C.
{ice,wuj}@iis.sinica.edu.tw

Pangfeng Liu
Department of Computer Science and Information Engineering
National Taiwan University
Taipei, Taiwan, R.O.C.
pangfeng@csie.ntu.edu.tw

Abstract

This paper describes an agent-based approach for scheduling multiple multicast on wormhole switch-based networks. Multicast/broadcast is an important communication pattern, with applications in collective communication operations such as barrier synchronization and global combining. Our approach assigns an agent to each subtree of switches such that the agents can exchange information efficiently and independently. The entire multicast problem is then recursively solved with each agent sending message to those switches that it is responsible for. In this way, communication is localized by the assignment of agents to subtrees. This idea can be easily generalized to multiple multicast since the order of message passing among agents can be interleaved for different multicasts. We conduct experiments to demonstrate the efficiency of our approach by comparing the results with SPCCO, a highly efficient multicast algorithm. We found that SPCCO suffers link contention when the number of simultaneous multiple multicast becomes large. On the other hand, our agent-based approach achieves better performance in large cases.

1 Introduction

Multicast/broadcast is commonly used in many scientific, industrial, and commercial applications [1]. Distributed-memory parallel systems require efficient implementations of multicast and broadcast operations in order to support various applications.

In recent years, with the speed of microprocessors increasing and cost decreasing and the availability of high bandwidth, low latency switches (such as Fast Ethernet switches, Myrinet switches, ATM switches, Servernet switches) at a reasonable cost, it is popular to interconnect workstations/PCs together with commodity switches. This makes clusters of workstations/PCs an appealing vehicle for cost-effective parallel computing.

To reduce communication latency and buffer requirement, wormhole switching technique [4, 15] is often used in these switches. Systems with wormhole routing provide a very small buffer space at each hop and divide a message into small flits that travel through the network in a pipeline fashion. The main drawback of wormhole switching is that blocked messages hold up the links, prohibiting other messages from using the occupied links and buffers. In a multicast, the source node sends the same data to an arbitrary number of destination nodes. When multiple multicast operations occur at the same time, it is very likely that some messages may travel through the same link at the same time and thus content with each other, if they are not scheduled properly.

Minimizing contention in collective communication has been extensively studied for systems with regular network topologies, such as mesh, torus and hypercubes [3, 5, 6, 10, 9, 11, 16]. Switch-based networks, on the other hand, typically have irregular topologies to allow the construction of scalable systems with incremental expansion capability. These irregular topologies lack many of the attractive mathematical properties of the regular topologies. This makes routing on such systems quite complicated. In the past few years, several deadlock-free routing algorithms have been proposed in the literature for irregular networks [2, 7, 12, 17]. These routing algorithms are quite complex and thus make implementation of contention-free multicast operations very difficult.

The goal of this paper is to develop efficient (multiple) multicast algorithms for irregular wormhole switch-based networks. In [8], Fan and King proposed an unicast-based implementation of single multicast operation based on Eulerian trail *routing. In this paper, we consider the widely used, commercially available deadlock-free routing strategy called "up-down" routing.*

Kesavan and Panda proposed a series of single and multiple multicast algorithms [13]. The basic idea is to order the destination processors into a sequence, then apply a binomial tree-based multicast [14] on these destina-

tions. The chain concatenation ordering (CCO) algorithm first constructs as many partial order chains (POC) as possible from the network. A partial order chain is a sequence of destinations such that we can apply a binomial multicast on it without any contention. The CCO algorithm then concatenates these POCs into sequence where a binomial multicast is performed [13]. The sequence consists of fragments of processor sequences in which messages within the same fragment can be sent independently, therefore congestion is reduced. Based on the CCO algorithm, the source-partitioned CCO (called SPCCO) performs multiple multicasts simultaneously. Each multicast produces its own sequence (consisting of POCs), and each resulting sequence is shifted until the source appears at the beginning of the sequence. By shifting these sequence, the communication is "interleaved" according to the source, and communication hot-spots are avoided.

Both CCO and SPCCO use the idea of POC to reduce contention. Within a single POC different messages do not interfere with one another as long as they are from different sections within a POC. However, this POC structure may not always be preserved since the later binomial multicast is not aware of it. Our agent-based algorithm deals with this issue by localization and interleaving. For a single multicast, our algorithm uses a recursive construct to localize communication. We then generalize it to multiple multicast by interleaving the communication tasks among different subnetworks.

Figure 1 compares the numbers of contented links per multicast from SPCCO and our recursive agent-based multicast (RAM) under different number of simultaneous multicasts. The system consists of sixteen switches. Each switch has 16 ports, with half of which connected to processors, and the other half connected to other switches. Each multicast has 101 and 408 destinations on the left and the right side of Figure 1 respectively. As indicated in Figure 1, our multicast algorithm incur less contention. More experimental data will be presented in Section 4.

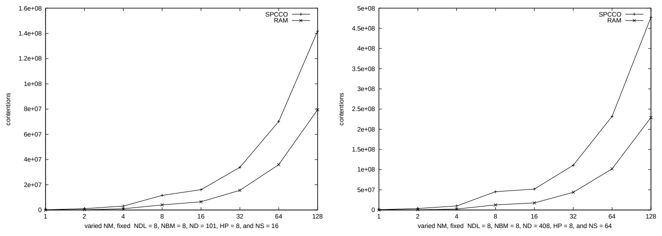

Figure 1. Number of contented links under different numbers of multicasts.

Our agent-based approach starts with a recursive multicast algorithm. An agent for a multicast is chosen for each subtree of the up-down routing tree. An agent is responsible for relaying the multicast messages to all the destinations in that subtree. This task is divided into subtasks for each subtree, where they are performed recursively. We generalize this algorithm to multiple multicast by choosing a primary agent *for each multicast. The primary agent are chosen from the subtrees of the root of the routing tree, and are properly interleaved so that the tasks are distributed evenly. The primary agents for different multicasts exchange messages and then use the multicast algorithm to propagate messages. Depending on how primary agents are chosen and how the information are exchanged among the primary agents, our agent-based multiple multicast algorithm has four variations and will be described in detail in section 3.*

The rest of the paper is organized as follows: Section 2 formally describes the communication model in this paper. Section 3 first describes our multicast algorithm, and then describes the generalization to multiple multicast. Section 4 reports our experimental results, and finally we conclude with Section 5.

2 Model

This section describes our communication model. The connectivity of switches in the network can be represented by a graph $G = (V, E)$, where the set of nodes V represents switches, and the set of edges E represents the bidirectional connection channels among switches. The graph G can be highly irregular. In addition, each processor is connected to a unique switch. Figure 2 illustrates an irregular network consisting of 4 switches (each has 8 ports) and 15 processors, and the corresponding graph.

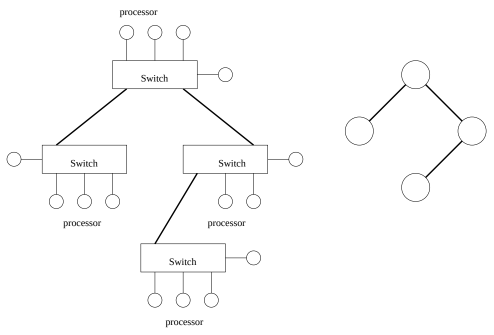

Figure 2. An irregular network of 4 switches and 15 processors and the corresponding connection graph.

2.1 Routing Mechanism

We now describe the up-down routing [7] used in our multiple multicast algorithm. The up-down routing mecha-

nism first uses a breadth-first search to build a spanning tree T for the switch connection graph $G = (V, E)$. Since T is a spanning tree of G, E is partitioned into two subsets – T and $E - T$. Those edges in T are referred to as tree edges *and those in $E - T$ as* cross edges *[13]. Since the tree is built with a BFS, the cross edges can only connect switches whose levels in the T differ by at most 1. A tree edge going up the tree, or a cross edge going from a processor with a higher processor id to a processor with a lower one, are referred to as* up links. *The communication channels going the other direction are* down links. *In up-down routing a message must travel all the up links before it travels any down links. Due to the acyclic nature of how the direction of links are defined, the up-down routing is deadlock-free.*

2.2 Contention

We assume that a switch can deliver multiple messages simultaneously from ports to ports, as long as the messages are delivered from different source and destination ports. This assumption is consistent with current routing hardware technology. As a result, congestion on the communication links becomes the major bottleneck.

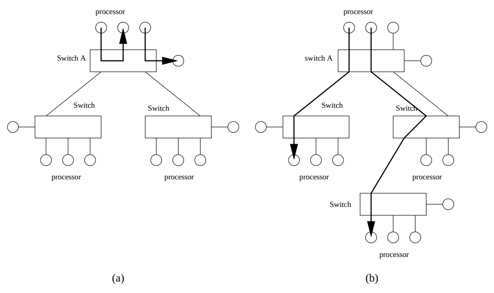

Figure 3. Example cases that avoid contention on the inter-switch channels.

We consider three cases where link contention can be avoided. We will focus on a particular switch A. In the first case, as shown in Figure 3(a), all source/destination processors are connected to the same switch A. In this case, there will be no contention since the messages travel through different paths within the switch. In the second case, as shown in Figure 3(b), both source processors reside on A. In this case, both can send messages to destinations in different subtrees of A simultaneously. Note that a destination node could be any processor in these two subtrees.

We now consider the third case where two messages travel through four subtrees of switch A, as indicated in Figure 4. If the two messages both go through switch A, there will be no link contention between them. Note that the source and destination processors may appear anywhere in the four subtrees.

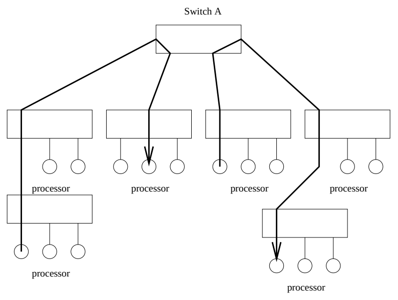

Figure 4. An example case that avoids contention on the inter-switch channels.

3 Agent-Based Algorithms

This section describes our framework of a agent-based multiple multicast algorithm. We first introduce the algorithm for single multicast, then generalize the idea to multiple multicast. Note that our algorithms assume the up-down routing mechanism. The algorithms specify how to perform a single/multiple multicast by determining the source and destination of all the intermediate communications, but the actual route from source to destination is determined by the up-down routing.

3.1 Single Multicast

For a given irregular network, we first construct a routing tree as in up-down routing [7]. The routing tree has all the switches as the tree nodes, and the inter-switch communication channels as the tree edges. Every tree node is the root of a unique subtree in this routing tree, and for ease of notation we will not distinguish a tree node (a switch in the network) from the subtree where it is the root.

For a given multicast message m and a switch v we will define two functions – an agent function $A(m, v)$ that returns a processor within the subtree rooted at v and will be responsible for relaying multicast message m, and a cost function $C(m, v)$ that estimates the total cost of sending m to all of its specified destinations within the subtree rooted at v. Note that all these tree nodes here represent switches, not processors.

We define these agent and cost functions recursively. Let $D(m, v)$ be the set of destination processors of message m that are connected to switch v. First we consider the case where v is a leaf in the routing tree, then $A(m, v)$ is defined to be an arbitrary destination processor in $D(m, v)$, and the cost function $C(m, v)$ is $\log |D(m, v)|$. If $|D(m, v)|$ is 0, that is, m does not have any destination attached to switch v, we define $A(m, v)$ to be an empty set and $C(m, v) = 0$.

We now consider the agent and the cost function for an internal node v in the routing tree. The agent function for v

is defined as follows: If $|D(m,v)| > 0$, we pick an arbitrary destination of m in $D(m,v)$ to be $A(m,v)$. Otherwise we consider all the children of v that m must be sent to, and set $A(m,v)$ to be the agent from these subtrees that has the highest cost. Formally, let $S(m,v)$ be the set of children of v that have destinations of m in their subtrees, then $A(m,v) = w$ such that $w \in S(v)$ and $C(m,w) \geq w'$ for all $w' \in S(v)$. Note that from this definition the agent of a switch is not necessarily connected to the switch itself.

The cost function for an internal node is defined as follows: For the purpose of recursion we assume that the agent of v knows the message m. If $|D(m,v)|$ is 0, the agents of tree nodes from $S(v)$ will first perform a multicast among themselves using a binomial multicast [14], then as soon as an agent a from $S(m,v)$ finishes receiving m, it recursively performs a multicast to all the destinations in the subtree where it is defined as the agent. The total communication cost is then defined as $C(m,v)$.

When $|D(m,v) > 0|$, the situation is more complicated sicne the agent of v can send m to other destinations in $D(m,v)$, or to the agents of $S(m,v)$. We apply a procedure ForwardInSwitch *that determines the order for those in $D(m,v)$ and $S(m,v)$ to receive messages. After the schedule is fixed we compute the total cost $C(m,v)$ for v. The algorithm* ForwardInSwitch *takes $D(m,v)$ and $C(w,m)$ for all $w \in S(m,v)$ as inputs, then computes an optimized schedule and the total cost. The details of* ForwardInSwitch *will be given later. The pseudo code of our recursive agent-based multicast (RAM) is given in Figure 5.*

```
RAM(v, m)
{
  if (|D(m, v)| == 0)
    A(m, v) sends message m to all agnets in S(m, v)
    by a binomial multicast;
  else
    call ForwardInSwitch to determine the order for A(v, m)
    to send messgaes to elements in D(m, v) and
    agents of S(m, v);
  For all switch s in S(m, v)
    call RAM(s, m);
}
```

Figure 5. The pseudo code of RAM, which recursively performs a multicast for each subtree of switch v that has destinations of message m.

When $|D(m,v)| > 0$, v does have some destination processors for message m and one of them is the agent of v. When the agent sends messages to those destinations in $D(m,v)$ (Figure 3 (a)), the messages will not interfere with each other. Also when the agent of v sends messages to those agents in $S(m,v)$ (Figure 3 (b)), no contention is possible if no cross edges are involved. In addition, the message passing from one category (Figure 3 (a)) will not contend with those in the other category (Figure 3 (b)).

When $|D(m,v)| = 0$, we use a single multicast to send the messages among all the agents of $S(m,v)$, with one of them now being assigned as the agent of v. From Figure 4 we conclude that these messages will not contend with each other unless cross edges are involved, since the agents of different subtrees in $S(m,v)$ will not be in the same subtree. After guaranteeing low congestion, the algorithm ForwardInSwitch*, which optimizes the schedule of the message-passing among agents, computes the total cost.*

3.2 *ForwardInSwitch* Scheduling

The inputs of ForwardInSwitch *are the switch v whose cost will be determined, the set of destinations attached to v ($D(m,v)$), and a set of subtrees of v to which the message m must be sent (denoted previously by $S(m,v)$). Let $A = (a_1, a_2, ...a_n)$ be the agents of these subtrees from $S(m,v)$, and $C = (c_1, c_2, ...c_n)$ be their costs respectively. For ease of explanation we assume that the list A and C have been sorted in non-increasing cost, that is, $c_i \geq c_{i+1}$.*

First the algorithm orders the n agent and $|D(m,v)|$ destination processors into a sequence Seq by which they will be scheduled to receive m. We keep two lists of processors that have not yet been scheduled (A, and D) – one for the agents and the other for the destinations attached to v. The algorithm maintains the size of D and estimates the time to complete all of them as $comp(D) = \lceil \log(|D|+1) \rceil$. If the current $comp(D)$ is greater than the cost of the first agent a in A (with the largest cost), we schedule a destination from D by moving it from d to Seq and recompute $comp(D)$. If the current $comp(D)$ is smaller than the cost of a, we schedule a by moving a from A to Seq. We repeatedly do so until all processors in A and D are scheduled.

After knowing the Seq we schedule communication tasks one at a time. Let K be the set of processors that have received message m. During each step the algorithm computes the number of processors in K that were from D, and moves this many processors from the head of the list of Seq into K. Note that we only use destinations in D to forward messages to the agents, since the agents will be busy forwarding messages to the destinations they are responsible for.

3.3 Multiple Multicast

This section describes our generalization of single multicast algorithm for multiple multicast.

1. *For each message m we choose a* primary agent *among the agents of $S(m,r)$ - the set of subtrees of root r. Each source processor then sends its message to its primary agent.*

```
ForwardInSwitch(L, D)
{
  initialize A, D and Seq
  while A or D is not empty, do the following
    if the cost of the first processor from A > comp(D)
      remove the first processor from A
      and append it to Seq
    else
      remove a destination from D and recompute comp(D)
      append this destination to Seq
  K = the agent of v
  while Seq is not empty, do the following
    compute p, the number of processors in K that
    were moved from D
    move the first p processors from Seq to K
}
```

Figure 6. The pseudo code of *ForwardInSwitch*.

2. *The primary agent sends its message m to a destination d in $D(m, r)$ if any, and to the agents of $S(m, v)$.*

3. *Each agent a of $S(m, r)$ sends messages to its destinations by calling* RAM*, and a sends m to $D(m, r)$ with a binomial multicast.*

We consider several alternatives in the first two steps of our multiple multicast algorithm. First we consider two alternatives in choosing the primary agent. It is now clear that if different multicasts select different primary agents, we can "interleave" the traffic in the second step and achieve good performance. On the other hand, we do not want to place the primary agents away from the original multicast source very often, which may cause large traffic through the root of the routing tree. As a result there is a tradeoff between good locality and interleaving. In our implementation we experimented two methods – we either choose the primary agent that is in the same subtree as the multicast source, or any agent of switches in $S(m, v)$ at random. These two approaches will be denoted as SameTree *and* Random *respectively.*

Secondly, we consider alternatives in implementing the second step of our multiple multicast algorithm. After the primary agent is chosen, it has to send the message to a processor in $D(m, r)$ and all the agents of switch in $S(m, r)$. This can be implemented in two different methods – the primary agent can send m to all the others with a binomial multicast, or it can work together with all the other primary agents to propagate information cyclicly. In the second approach, we arrange the chosen processor in $D(m, v)$ and all the primary agents as a ring. Each processor in the ring is responsible for relaying the information to the right side neighbor in the ring. Initially every primary agent places its message into this "circular track" and the message will be relayed to all the primary agents. We refer to these two approaches as Binomial *and* Cyclic *respectively. Combined with the alternatives we have four multiple multicast algorithms as follows –* SameTree-Binomial, SameTree-Cyclic, Random-Binomial *and* Random-Cyclic*. These four algorithms will be denoted as STB, STC, RB and RC, and their performances will be reported in the next section.*

4 Simulation Experiments and Results

In this section, we present results of simulation experiments to compare the algorithms proposed in Section 3 and the two order-chain-based algorithms proposed in prior works (CCO, SPCCO).

4.1 Experiments and Performance Measures

We developed a C++, discrete event-based simulator for our experiments. The simulator can model wormhole routing switches with arbitrary network topologies. We chose system parameters as follows. Communication start-up time was 5.0 microseconds, link transmission time was 10.5 nanoseconds, and routing delay at switch was 200 nanoseconds. The default buffer size at each port was assumed to be 1 flit. The default numbers of input ports and output ports were assumed to be 16. The network topologies were generated randomly. For each data point, the multicast performance was averaged over 100 different network topologies.

For all experiments, we assumed a default system configuration of a 512-processor system interconnected by 64 sixteen-port switches in an irregular topology. 50% of the ports on a switch are connected to processors, and the other 50% of the ports are connected to other switches. Links were not allowed between ports of the same switch. A random number generator was used to decide the port and switch or the processing node to which a given switch port should be connected to.

4.2 Performance Comparison

For our study, we varied each of the following parameters one at a time: the message length (NBM), the number of destinations in each multicast (ND), the number of simultaneous multicast operations (NM), the number of switches (NS), and connectivity of switches (HP). Since message length, number of multicast operations, and system size varied in our experiments, instead of using latency as the measurement of performance, we use throughput*, which is defined by M/T, where M is the total length of the messages and T is the parallel completion time of the (multiple)multicast operation.*

In the following we compare the performance of our proposed algorithms, `RAM`, `STC`, `RC`, `STB`, *and* `RB`*, and the two ordered-chain-based algorithms,* `CCO` *and* `SPCCO`.

4.2.1 Effect of Number of Multicast Operations

First we examined the effect of variation in the number of multicast operations on the performance of the proposed algorithms. Other parameters were assumed to be as follows: number of switches $NS = 64$ *(and thus 512 processors), number of ports connected to processors* $HP = 8$*, and number of destinations in each multicast* $ND = 408$*. We also chose two different distributions for multicast sources: distributing the sources among 2 subtrees (NDL=2) or distributing the sources over the entire network (NDL=all). The destinations were generated randomly. For each data point, the multicast performance was averaged over 50 different sets of destinations.*

As shown in Figure 7, when there are few (less then eight) multicast operations, ordered-chain-based algorithms perform better than our agent-based algorithms. This is because when the number of multicast operations is small, message contention is not significant and thus the importance of reducing number of communication stages outweights that of reducing message contention. However, when the number of multicast operations increases, the impact of message contention becomes more important and therefore the benefit of agent-based optimization becomes more significant.

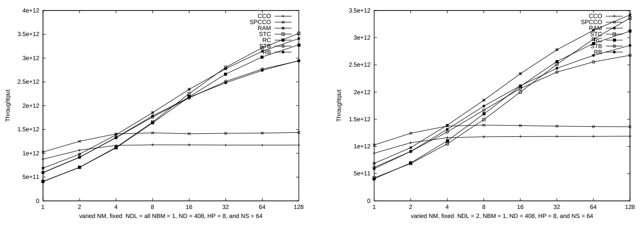

Figure 7. Throughput under different number of multicast operations

4.2.2 Effect of Number of Switches

We studied the scalability of the proposed algorithms on different systems sizes. We varied the number of switches from 16 to 256, with 50% of the ports connected to processors and the other 50% connected to switches, For each switch size, we chose three different numbers of multicast operations, NM=8,32,or 128. Each multicast has 101 randomly generated destinations. For each data point, the multicast performance was averaged over 50 different sets of destinations.

As shown in Figure 8, the throughput of the agent-based algorithms, the throughput of the ordered-chain-based algorithms, and the improvement ratio of the agent-based algorithms over the ordered-chain-based algorithms all increase when the number of switches (and processors) increases. A possible reason is that when number of switches increases, the level of the up-down routing BFS tree also increase, hence the number of hops between the sender and the receiver of a cross-subtree message may increase. Longer path increases the potential of contention. Since our agent-based algorithms guarantee the path of each message be no more than 2 hops, they are scalable with respect to number of switches. Furthermore, Figure 8 shows that when the number of multicast operations increases (NM=8,32,128 respectively), the improvement ratio of our agent-based algorithms over the ordered-chain-based algorithms also increases.

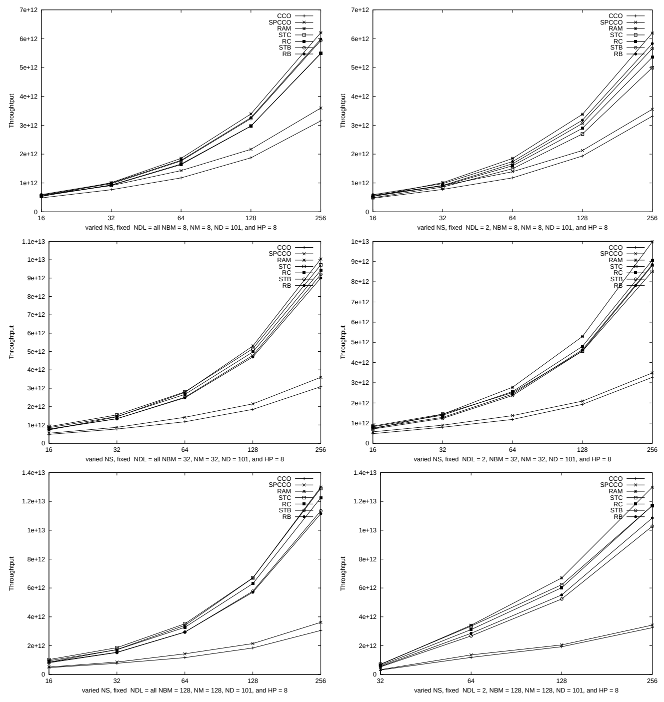

Figure 8. The throughput under different numbers of the switches

4.2.3 Effect of Number of Destinations

In this experiment, number of switches $NS = 64$ *and number of ports connected to processors* $HP = 8$*. We chose three different numbers of multicast operations* $NM = 8, 32, 128$*. We varied the number of destinations for each multicast from 100 to 512. Figure 9 shows the throughput of these algorithms. As we can see, the throughput of these algorithms increases when the number of destinations increases, and the improvement ratio of the agent-based algorithms over the ordered-chain-based algorithms also increases on size increase in destinations.*

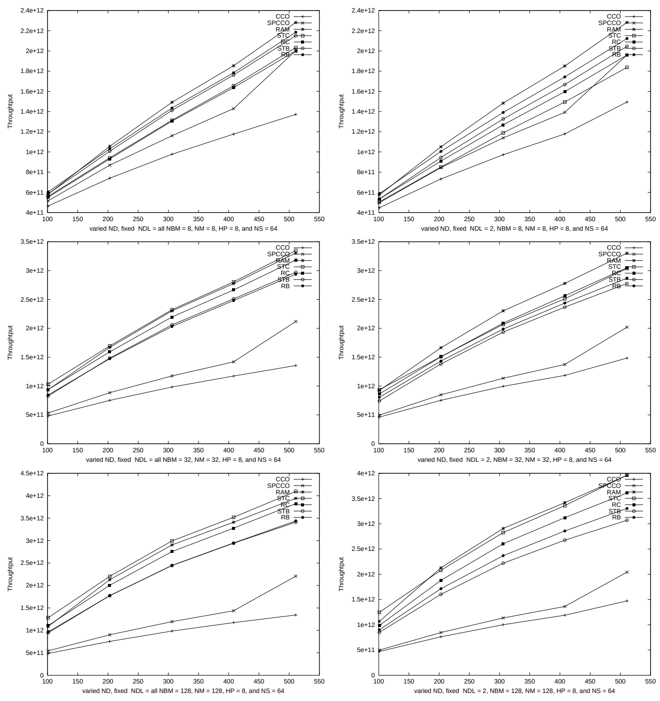

Figure 9. Throughput under different numbers of destinations

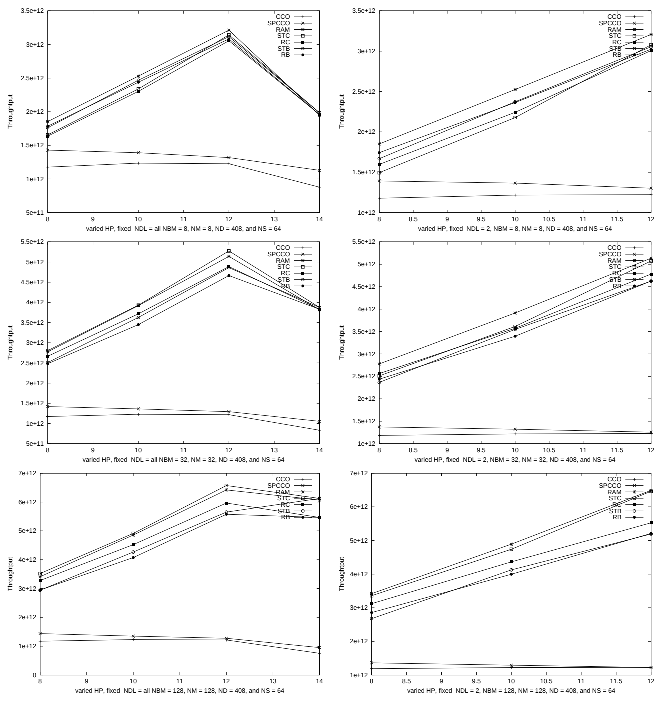

Figure 10. The throughput under different numbers of host ports

4.2.4 Effect of Switch Connectivity

*In this experiment, number of switches $NS = 64$ and number of destinations $ND = 408$. We chose three different numbers of multicast operations $NM = 8, 32, 128$, and varied the number of ports connected to processors. Figure 10 shows the throughput of these algorithms. As we can see, the throughput of our agent-based algorithms increases when the number of ports connected to processors (*HP*) increases, but the throughput drop when the value of* HP *reaches a threshold value (12 in this case). This is because when number of processors increases, the parallelism in relaying messages also increases. However, the number of links connected to other switches decreases as* HP *increases, which in turn increases the chance of contention. When* HP *reaches the threshold value, the impact of contention outweights the benefit of parallelism. The ordered-chain-based algorithms also behave similarly to the agent-based algorithms, although not as evident.*

4.2.5 Effect of Message Length

*We examined the effect of message length on the performance of proposed algorithms. We chose two message lengths, $128KB$ for short messages and $32MB$ for long messages, and varied the number of multicast operations with long messages (*NBM*). The source and destinations of a multicast were generated randomly. We also chose three different numbers of multicast operations, $NM = 8, 32, 128$. As shown in Figure 11, when the number of long-message multicast operations is small, the performances of the agent-based algorithms are worse than those of the ordered-chain-based algorithms. The possible reason is that long messages are likely to increase the chance of contention, and when the number of long-message multicast operations is small, they may not be evenly distributed in the BFS tree and thus may cause hot-spots in communication.*

In summary, the agent-based algorithms and the ordered-chain-based algorithms compliment each other. The ordered-chain-based algorithms are superior to the agent-based algorithms for small number of multicast operations, while the agent-based algorithms perform better than the ordered-chain-based algorithms for larger number of multicast operations (larger than 8 in our experiments). The difference in performance of these algorithms increases with increase in number of switches, number of multicast destinations, and number of processors in the system.

5 Conclusion

This paper describes an agent-based approach for scheduling multiple multicast on wormhole switch-based networks. Our approach assigns an agent to each subtree of

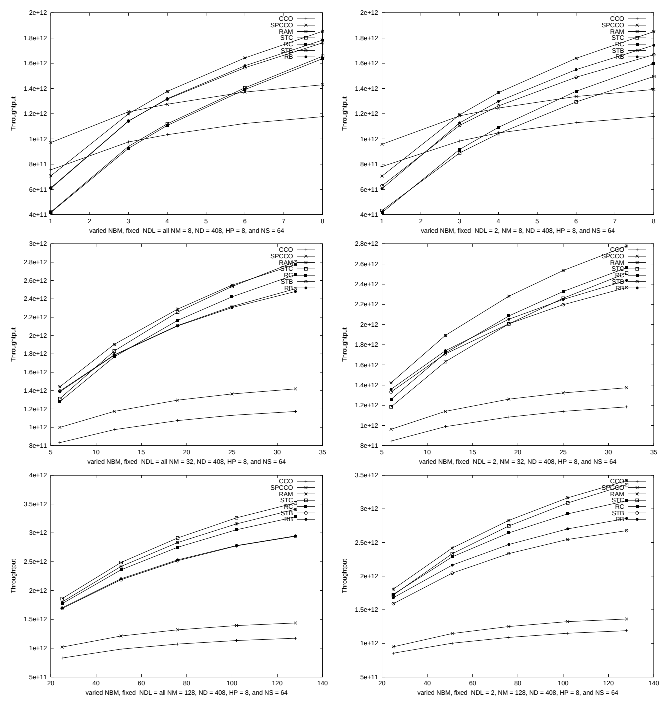

Figure 11. Throughput under different numbers of long-message multicasts

switches such that the agents can exchange information efficiently and independently. The entire multicast problem is recursively solved with each agent sending message to those switches that it is responsible for. Communication is localized by the assignment of agents to subtrees. In addition, the agent mechanism provides an easy mechanism in performing multiple multicasts simultaneously, without high level of congestion. We compare the results with SPCCO [13] and conclude that SPCCO, a highly efficient multicast algorithm, in some large cases may not interleave multiple multicast well enough. In contrast our agent-oriented approach dispatches multiple multicast sufficiently well in large cases.

We will conduct further investigation in the alternative of choosing primary agents and the way to exchange information among them. The two approaches for choosing primary agents, Random *and* SameTree *are two completely different approaches. We may be able to find a balance between communication locality and even workload distribution. Similarly, we will consider other alternatives in information exchange among primary agents. Since the primary agents are very likely to be close to the root of the routing tree, it is possible that we can derive good algorithms to exploit this locality.*

References

[1] *In* Message Passing Interface Forum, MPI: A Message-Passing Interface Standard, *Mar. 1994.*

[2] *N. J. Boden, D. Cohen, R. F. Felderman, A. E. Kulawik, C. L. Seitz, J. Seizovic, and W. Su. Myrinet - a gigabit per second local area network.* IEEE Micro*, pages 29– 36, Feb. 1995.*

[3] *W. Dally. Deadlock-free message routing in multiprocessor interconnection networks.* IEEE Trans. Comput.*, C-36(5):547– 553, May 1987.*

[4] *W. J. Dally and C. L. Seitz. Deadlock-free message routing in multiprocessor interconnection networks.* IEEE Transactions on Computers*, C-36(5):547– 553, May 1987.*

[5] *J. Duato. On the design of deadlock-free adaptive routing algorithms for multicomputers. In* Proceedings of Parallel Architectures and Languages Europe 91*, June 1991.*

[6] *J. Duato. A necessary and suffi cient condition for deadlock-free adaptive routing in wormhole networks. In* Proceedings of the 1994 International Conference on Parallel Proceeding*, August 1994.*

[7] *M. D. S. et. al. Autonet: A high-speed, self-confi guring local area network using point-to-point links. Technical Report SRC research report 59, DEC, April 1990.*

[8] *K.-P. Fan and C.-T. King. Effi cient multicast on wormhole switch-based irregular networks of workstations and processor clusters. In* Proceedings of the Internationl Conference on High Performance Computing Systems*, 1997.*

[9] *P. T. Gaughan and S. Yalamanchili. Adaptive routing protocols for hypercube interconnection networks.* IEEE Computer*, 26(5):12– 23, May 1993.*

[10] *C. Glass and L. Ni. The turn model for adaptive routing.* J. ACM*, 41:847– 902, Sept. 1994.*

[11] *G. Gravano, G. D. Pifarre, P. E. Berman, and J. L. C. Sanz. Adaptive deadlock- and livelock-free routing with all minimal paths in torus networks.* IEEE Trans. Parallel and Distributed Systems*, 5(12):1233– 1251, Dec. 1994.*

[12] *R. Horst. Servernet deadlock avoidance and fractahedral topologies. In* Proceedings of the International Parallel Processing Symposium*, pages 274– 280, April 1996.*

[13] *R. Kesavan and D. K. Panda. Effi cient multicast on irregular switch-based cut-through networks with up-down routing. In* IEEE Trans. Parallel and Distributed Systems*, volume 12, August 2001.*

[14] *F. T. Leighton.* Introduction to Parallel Algorithms and Architectures: Arrays, Trees, hypercubes*. Morgan Kaufmann.*

[15] *L. Ni and P. McKinley. A survey of wormhole routing techniques in direct networks.* IEEE Computer*, 26(2):62– 76, February 1993.*

[16] *A.-H. E. P.K. McKinley, H. Xu and L. Ni. Unicast-based multicast communication in wormhole-routed networks.* IEEE Transactions on Parallel and Distributed Systems*, 5(12):1252– 1265, December 1994.*

[17] *W. Qiao and L. Ni. Adaptive routing in irregular networks using cut-through switches. In* Proceedings of the 1996 International Conference on Parallel Proceeding*, pages I:52– 60, August 1996.*