

An Optimal Scheduling Algorithm for an Agent-Based Multicast Strategy on Irregular Networks

Yi-Fang Lin^{1,2}, Zhe-Hao Kang¹, Pangfeng Liu¹, and Jan-Jan Wu²

¹ Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan.

² Institute of Information Science, Academia Sinica, Taipei, Taiwan.

Abstract. This paper describes an agent-based approach for scheduling multiple multicast on switch-based networks with irregular topologies. 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. The key to the performance of this agent-based approach is the message-passing scheduling between agents and the destination processors. We propose an optimal scheduling algorithm, called *ForwardInSwitch* to solve this problem.

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. Distributed-memory parallel systems, such as cluster systems, require efficient implementations of multicast and broadcast operations in order to support various applications. 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 network 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 [1–7]. Cluster networks, especially switch-based clusters, 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 routing algorithms have been proposed in the literature for irregular networks [8–11]. 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 switch-based networks. In [12], 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 routing strategy called “up-down” routing. The best known results on multicast on irregular networks are the *Partial-Order-Chain*-based algorithms proposed by Kesavan and Panda [13]. The basic idea is to order the destination processors into a sequence, then apply a binomial tree-based multicast [14] on these destinations. 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. However, 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.

To solve this problem, in [15] we proposed an agent-based multicast algorithm, which avoids network contention by localizing and interleaving message passings in multicast. Our agent-based approach starts with a recursive multicast algorithm. An agent for a multicast is chosen for each subtree of the routing tree. An agent is responsible for relaying (forwarding) 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 forward messages.

The key to the performance of the agent-based multicast strategy is the scheduling of message forwarding between agents as well as between an agent and the destination processors within each subtree. In our previous work we use a rudimentary scheduling for this purpose. The focus of this

paper is an optimal scheduling algorithm, called *ForwardInSwitch*, for message forwarding. We provide theoretical analysis for the optimality and time complexity of *ForwardInSwitch*. Our experimental results also demonstrate significant performance improvement of our multicast algorithms in comparison with the CCO and SPCCO multicast algorithms. 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 presents the *ForwardInSwitch* optimal scheduling algorithm. Section 5 reports our experimental results, and finally we conclude with Section 6.

2 Model

We now describe the up-down routing [9] used in our multiple multicast algorithm. The up-down routing mechanism 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.

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.

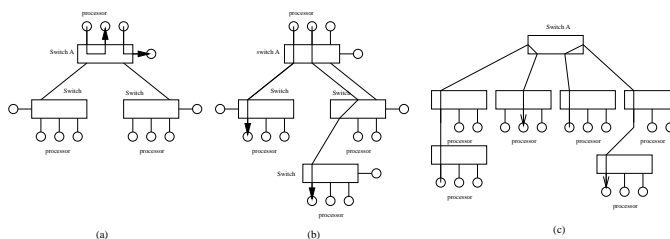


Fig. 1. Example cases that avoid contention on the inter-switch channels.

We consider three cases where link contention can be avoided. In the first case, as shown in Figure 1(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 1(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. In the third case two messages travel through four subtrees of switch A , as indicated in Figure 1 (c). If the two messages both go through switch A , there will be no link contention between them.

3 Agent-Based Algorithms

3.1 Single Multicast

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 .

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 . For a leaf v , $A(m, v)$ is defined to be an arbitrary destination processor in $D(m, v)$, and $C(m, v)$ is $\log |D(m, v)|$. For an internal node v , 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 $w' \in S(v)$. For the cost function part, 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 since 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. The algorithm *ForwardInSwitch* takes $D(m, v)$ and $C(w, m)$ for all $w \in S(m, v)$ as inputs, then computes an optimal schedule and the total cost. The details of *ForwardInSwitch* will be given later.

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 1 (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 1 (b)), no contention is possible if no cross edges are involved. In addition, the message passing from one category (Figure 1 (a)) will not contend with those in the other category (Figure 1 (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 1 (c) 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.

3.2 Multiple Multicast

Let r be the root of the up-down routing tree. The agent-based multiple multicast is carried out in three steps as described below. First 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. Second, the primary agent sends its message m to a destination d in $D(m, r)$ if any, and to the agents of $S(m, v)$. Finally, 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.

4 ForwardInSwitch

We have two kinds of nodes in our *ForwardInSwitch* scheduling. The first is called *local nodes*, which are processors within a switch (or a local cluster). Local nodes can send and receive data among themselves. The second is *remote nodes*. Each remote node represents a remote agent that we need to send the message to. Once a remote agent receives the data from one of the local nodes, it will be responsible for distributing the data among the processors within that subtree.

Initially we have the agent local node as the source of the broadcast. The agent needs to send the data to all the other nodes (local and remote nodes) in the system. We define the *finishing time* as the time for all nodes in the system to receive the data, and, we would like to find a broadcast schedule with the minimum finishing time.

We assume that the local nodes are homogeneous, so that it takes one unit time for any local node to send a data to any other nodes. However, it takes very different amount of time for a remote node to receive a data, and this time is at least 1 time unit. To be more specific, when a local node sends data to another local node at time t , both local nodes can start sending data to another node at time $t+1$. However, if a local node sends data to a remote node at time t , the local node can start sending data to another node at time $t+1$, but the remote node will not complete its operation until $C(m, r)$, which will be determined recursively from bottom to the top of the routing tree. Recall that $C(m, r)$ is the cost for an agent r to send messages m to all the destination processors located in the subtree rooted at r . As a result we define the *finishing time* of a remote node r to be the $t + C(m, r)$, where t is the time the parent of r starts sending the data to r . The total time of *ForwardInSwitch* is then determined by the maximum of all nodes

4.1 Scheduling Algorithm

Let n and m be the number of local and remote nodes. The remote nodes are r_1, r_2, \dots, r_m with costs c_1, c_2, \dots, c_m . Without lose of generality

we assume that $c_i \geq c_{i+1}$, for $1 \leq i \leq m - 1$. We use $l(n)$ to denote the level number of a node n . We first observe that the remote nodes should be scheduled according to non-decreasing order according to their costs. That is, there exists an optimal *ForwardInSwitch* schedule in which $l(r_i) \leq l(r_{i+1})$, for $1 \leq i \leq m - 1$. If we assume that there exists an optimal *ForwardInSwitch* schedule in which $l(r_i) > l(r_{i+1})$ for some i , it is easy to see that by switching r_i and r_{i+1} the finishing time will not increase.

We use a binary search to determine the optimal *ForwardInSwitch* finishing time. If we could determine that, given a target finishing time T , whether all tree nodes can finish, we could use at most $O(\log C)$ round of testings to determine the optimal *ForwardInSwitch* finishing time, where C is maximum possible finish time. As a result, the key point of our algorithm is to determine, given a time constraint T , whether all nodes can finish in time.

We divide the remote nodes into two groups – *critical* and *non-critical*. A remote node r_i is critical at time t if $t + c_i$ is at least T , where T is the target finishing time constraint. If a remote node is critical, it should be scheduled immediately otherwise it will miss the deadline T . If the node is non-critical, then it can wait.

We now describe our testing algorithm which determines whether it is possible to obtain a *ForwardInSwitch* scheduling within time T . At every time step, all the local nodes that have already received the message, select the destinations according to the following priority. (1) critical remote nodes, (2) local nodes, (3) non-critical remote nodes

Theorem 1. *There exists an optimal ForwardInSwitch schedule that obeys the priority.*

Proof. Since a critical node must be scheduled immediate to avoid missing its deadline, it has the highest priority. We only need to show that there exists an optimal *ForwardInSwitch* schedule that will schedule non-critical remote nodes only when there is *no* local node to send messages to.

We assume that there is an optimal schedule in which a non-critical remote node r is scheduled at time t and a local node b is scheduled at a later time $t' > t$. Let a be the local node that sends data to r in the optimal schedule. Now we will do the following changes. We will make a to send data to b instead of r at time t , and make b to send data to r at time $t + 1$, then make b to send data to c at time $t + 2$, where c is the node b sends data to at time $t' + 1$ in the optimal schedule. Since r is not critical at time t , delaying it to $t + 1$ will not miss the deadline T . The subtree rooted at c started at time $t' + 1$ in the original optimal schedule, and now starts at time $t + 2$. Since that $t' > t$, or $t' + 1 \geq t + 2$. The subtree of c will not be delayed either. ■

We use this checking algorithm to verify whether a finishing time T is feasible. From Theorem 1 we know that if there exists an optimal schedule for *ForwardInSwitch*, the checking algorithm will find it. Now with a binary search on T , we can easily determine the optimal T , hence

the optimal *ForwardInSwitch* schedule. It is easy to see that the finish time will not be more than $n + \sum_{i=1}^m c_i$, so at most $O(\log(n + C))$ rounds of checking, where C is the summation of costs from remote nodes, suffice to find the optimal finish time.

5 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).

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.

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 the number of ports on a switch (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.

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 $HP = 8, 12$, and number of destinations in each multicast $ND = 153, 204, 537, 716$. The destinations were generated randomly. For each data point, the multicast performance was averaged over 50 different sets of destinations.

As shown in Figure 2, when there are few (less than 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 outweighs 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.

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 128, with 50% of the ports connected to processors and the other 50% connected to switches. For each switch size, number of multicast operations $NM=32$. Number of multicast destinations $ND=134, 179$. For each data point, the multicast performance was averaged over 50 different sets of destinations.

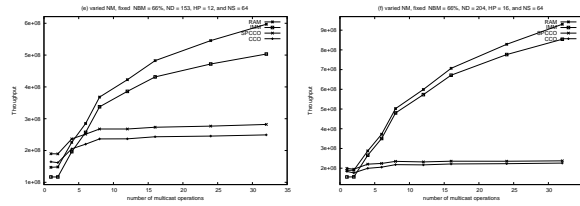
As shown in Figure 2, the throughput of the agent-based algorithms, the throughput of the ordered-chain-based algorithms, and the improvement ratio of the agent-based algorithms over the ordered-chain-based algorithms all increase when the number of switches (and processors) increases. A possible reason 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.

Effect of Number of Destinations In this experiment, number of switches $NS = 64$ and number of ports connected to processors $HP = 8$. We chose two different numbers of multicast operations $NM = 4, 32$. We varied the number of destinations for each multicast from 100 to 900. Figure 2 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.

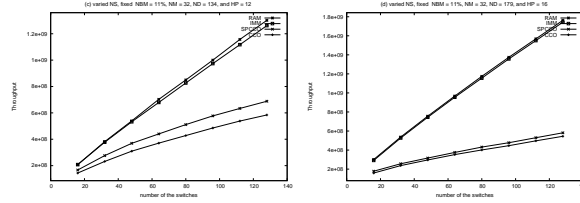
Effect of Message Length We examined the effect of message length on the performance of proposed algorithms. We chose two message lengths, $128KB$ for short messages and $32MB$ for long messages, and varied the number of multicast operations with long messages (NBM). The source and destinations of a multicast were generated randomly. As shown in Figure 2, when the number of long-message multicast operations is small, the performance discrepancy between agent-based algorithms and the ordered-chain-based algorithms is small. 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.

6 Conclusion

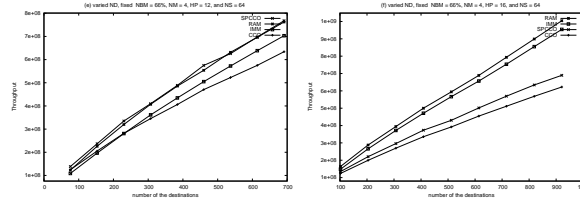
This paper describes an agent-based approach for scheduling multiple multicast on switch-based networks. Our approach assigns an agent to



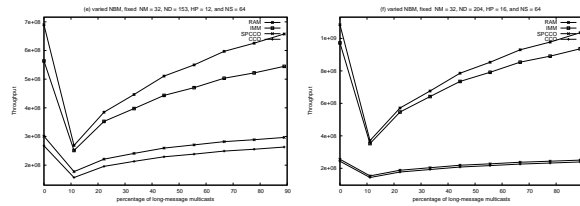
Throughput under different number of multicast operations.



The throughput under different numbers of switches.



Throughput under different numbers of destinations.



Throughput under different numbers of long-message multicasts.

Fig. 2. Simulation result comparison by varying different parameters.

each subtree of 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, with very low chances of network contention.

We compare the results with SPCCO [13] and found that SPCCO, a highly efficient multicast algorithm based on *Partial Ordered Chains*, incurs high contention in large cases. Our agent-based approach minimizes contention by properly interleaving multiple multicast and optimally scheduling message passings between agents and destination processors to avoid hot spots.

References

1. Dally, W.: Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Trans. Comput.* **C-36**(5) (1987) 547–553
2. Duato, J.: On the design of deadlock-free adaptive routing algorithms for multicomputers. In: *Proceedings of Parallel Architectures and Languages Europe 91*. (1991)
3. Duato, J.: A necessary and sufficient condition for deadlock-free adaptive routing in wormhole networks. In: *Proceedings of the 1994 International Conference on Parallel Processing*. (1994)
4. Glass, C., Ni, L.: The turn model for adaptive routing. *J. ACM* **41** (1994) 847–902
5. Gaughan, P.T., Yalamanchili, S.: Adaptive routing protocols for hypercube interconnection networks. *IEEE Computer* **26**(5) (1993) 12–23
6. Gravano, G., Pifarre, G.D., Berman, P.E., Sanz, J.L.C.: Adaptive deadlock- and livelock-free routing with all minimal paths in torus networks. *IEEE Trans. Parallel and Distributed Systems* **5**(12) (1994) 1233–1251
7. P.K. McKinley, H. Xu, A.H.E., Ni, L.: Unicast-based multicast communication in wormhole-routed networks. *IEEE Transactions on Parallel and Distributed Systems* **5**(12) (1994) 1252–1265
8. Boden, N.J., Cohen, D., Felderman, R.F., Kulawik, A.E., Seitz, C.L., Seizovic, J., Su, W.: Myrinet - a gigabit per second local area network. *IEEE Micro* (1995) 29–36
9. et. al., M.D.S.: Autonet: A high-speed, self-configuring local area network using point-to-point links. Technical Report SRC research report 59, DEC (1990)
10. Horst, R.: Servernet deadlock avoidance and fractahedral topologies. In: *Proceedings of the International Parallel Processing Symposium*. (1996) 274–280
11. Qiao, W., Ni, L.: Adaptive routing in irregular networks using cut-through switches. In: *Proceedings of the 1996 International Conference on Parallel Processing*. (1996) I:52–60
12. Fan, K.P., King, C.T.: Efficient multicast on wormhole switch-based irregular networks of workstations and processor clusters. In: *Proceedings of the International Conference on High Performance Computing Systems*. (1997)
13. Kesavan, R., Panda, D.K.: Efficient multicast on irregular switch-based cut-through networks with up-down routing. In: *IEEE Trans. Parallel and Distributed Systems*. Volume 12. (2001)
14. Leighton, F.T.: *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, hypercubes*. (Morgan Kaufmann)
15. Lin, Y.F., Liu, P., Wu, J.J.: Efficient agent-based multicast on wormhole switch-based irregular networks. In: *International Parallel and Distributed Processing Symposium*. (2003)