Efficient Communication Scheduling with Re-routing based on Collision Graphs *

David Ray Surma
Edwin Hsing-Mean Sha
Dept. of Computer Science & Engineering
University of Notre Dame
Notre Dame, IN 46556
E-mail: dsurma@cse.nd.edu, esha@cse.nd.edu

Abstract

Parallel systems are increasingly being used in applications requiring high throughput or which have real-time deadlines because of their potential for computation time savings. However, this savings is often offset by the communication overhead inherent in such systems. In this paper, such a communication overhead was encountered while performing simulations of partial differential equations (representing fluid dynamics problems) by using the multi-dimensional wave filters method. With tightly-coupled architectures as the platform, the static *communication scheduling* of messages in the network is addressed. The compile time determination of when nodes should send their messages to other nodes in the network is what is termed static communication scheduling. Additionally, the *routing* of these messages is also addressed. Although the static scheduling of computational tasks has been studied for some time, our problem is very new. This paper utilizes the newly developed *Collision Graph* model to begin the study of compile-time analysis of the run-time communication overhead. Using this model, the determination of an optimal schedule is proven to be *NP-Complete*. Several efficient algorithms are designed to reduce the overhead by scheduling the message transmissions as well as determining their routing scheme. Experiments show a significant improvement over baseline approaches.

Index Terms - Tightly-coupled networks, communication, parallel systems, graph modeling, scheduling.

*This work was supported in part by NSF MIP 9501006.

1 Introduction

In systems requiring high throughput or which have real-time deadlines, high-performance multi-processor designs are increasingly being used. As one of the point design teams to develop *Petaflop* super-computers sponsored by *NSF, ARPA, DOD* and others, our research group was challenged by the need of obtaining an optimized execution time based on *tightly-coupled* massively parallel architectures such as the *EXECUBE* [1]. One application being studied is the implementation of a parallel solution for simulating partial differential equations, representing fluid dynamics problems, by using the *multi-dimensional wave digital filters* method [2]. In addition to the large amounts of computation time needed, significant communication time between processors was required by the data volume involved in those simulations. While using multiple processors can reduce the computation time, the communication overhead required in the system can be substantial.

We found that to improve the total execution time the communication time between processors for every pair of points should be minimized. The goal was to eventually hide this communication time by overlapping it with the computation time. The creation of a new scheduling technique was required to achieve such a goal, since most of the existing scheduling methods do not consider the communication characteristics of the problem [3, 4] and are unable to achieve an optimal schedule. In multi-processor systems, the performance is directly affected by the way tasks are allocated to the individual processing elements. Then, communication costs stem from the underlying message passing which occurs. This research assumes that a suitable task allocation scheme has been used and therefore is not a type of multiprocessor scheduling[4]. Rather, this work deals strictly with determining the ordering and routing of these message transmissions. It was found that the compiler can play a significant role here. Modest improvement can be obtained by judiciously selecting just the ordering. Moving from a fixed routing scheme to a more flexible one results in more significant overhead reduction. Thus, by determining both the schedule and the routing scheme the compiler can work to reduce the communication overhead substantially. This work presents compile-time algorithms which work to reduce the run-time communication overhead.

To fully understand this problem, consider an example filter section represented by the task *directed acyclical graph*, or *DAG*, of Figure 1. Figure 1(b) shows one possible assignment of this graph to a six processor two-dimensional mesh network. While tasks assigned to the same processor incur no communication overhead, this assignment scheme indicates that messages must be exchanged. Furthermore, since there is only a single bidirectional link between each node network collisions occur. The first two columns of Table 1 give possible orderings of the resulting message traffic when *XY-routing* is used. Messages appearing on the same line may be sent in parallel without collisions occurring. Under *worm-hole* [5] routing assumptions that each message takes the same amount of time, t, to traverse the network, and assuming that the messages are of equal length, schedule 1 gives an ordering which completes at time $4t$ while schedule 2 completes at $3t$. Thus, there is a savings of 25% based on the communication schedule. An even greater amount of improvement can be obtained if message $(A \rightarrow F)$ is rerouted to traverse in a YX direction. The third column shows this new schedule with the rerouted message denoted as $A \rightarrow F'$. The completion time of this new ordering is $2t$. Thus, the work of this paper addresses the ordering or *scheduling* of the messages, as well as the re-routing of some of them, to reduce the overall completion time.

The term used for this research is *communication scheduling*. It not only encompasses routing aspects and path selection issues as discussed in [6, 7], it also determines the order and timing that the messages in the system should be sent. There have been several studies related to the problem addressed here. One

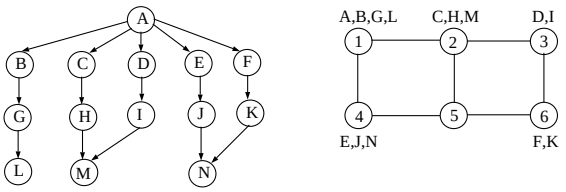


Figure 1: (a) Task Flow DAG. (b) Tasks assigned to processing nodes

Schedule 1	Schedule 2	Re-routed Schedule
$A \rightarrow C;\ A \rightarrow E$	$A \rightarrow F;\ A \rightarrow E$	$A \rightarrow F';\ A \rightarrow D$
$A \rightarrow D$	$A \rightarrow D;\ K \rightarrow N$	$A \rightarrow B;\ A \rightarrow C;\ I \rightarrow M;\ K \rightarrow N$
$A \rightarrow F$	$A \rightarrow C;\ I \rightarrow M$	
$K \rightarrow N;\ I \rightarrow M$		

Table 1: Example Communication Schedules

such effort focuses on developing communication algorithms for interconnection networks[8]. There a `traffic scheduling' algorithm for multi-processor networks was introduced to try and balance and saturate the links of the network based on the fact that a large number of messages must eventually be delivered. Their work, however, uses a $FCFS$ approach and does not perform any *scheduling* of the individual message transmissions. Lee and Kim [6] perform path selection in a wormhole routed network just as our work does. However, they search for unique paths for pairs of communicating nodes so that the tasks can communicate without interference whenever needed. Kandlur and Shin[9] present a work similar to [6] in that dedicated paths are found. The problem with these techniques is that the dedicated paths can cause other messages to follow longer paths even though the dedicated paths may not be currently in use. Additionally, they do not use any type of scheduling which can improve the overall performance. Recent work by Eberhart and Li[10] does perform a type of communication scheduling on two-dimensional mesh architectures. However, they use *dynamic* scheduling and restrict their work to communication patterns that are commonly used in data parallel applications. The work presented here can apply to any type of message-passing activity.

By using a new model presented by Surma and Sha in [11] known as a *Collision Graph*, this paper starts the research on the compile-time analysis of the run-time communication overhead incurred by point-to-point message transmissions. This problem was addressed in a very restricted way in [12] but what is developed here is a new scheduling framework able to deal with real-life problems. Just as multi-processor scheduling problems are *NP-Complete* for most precedence-constrained tasks, the *communication scheduling problem* or *CSP* is shown to be *NP-Complete* as well. Because of this, heuristic methods are employed to arrive at the communication schedules. At compile time, this approach utilizes information about the message traffic to determine a route that the messages should take as well as an ordering of when they should be transmitted. Experiments show significant improvement over baseline techniques in cutting the communication overhead.

2 Foundations and Graph Model

The starting point for this research is a list of N messages which are to be sent by the nodes in the network. The goal is to find an optimal communication schedule which will reduce the overall processing time. Because determining such a schedule is very difficult, this section presents a transformation into a simpler, better defined problem. To form the foundation upon which the communication scheduling algorithms

Message ID	Est. Arrival Time	Num of Packets	Source	Destination
1	1.0	1	(2,2)	(8,8)
2	1.0	1	(3,1)	(7,7)
3	1.0	1	(2,5)	(5,7)
4	1.0	1	(2,1)	(5,4)
5	1.0	1	(2,1)	(6,3)
6	1.0	1	(3,4)	(5,6)
7	1.0	1	(3,1)	(6,4)

Table 2: Example message list

are built, this paper considers a restricted case model of network traffic where all messages are assumed to have the same origination time at the sending nodes. Thus, the concerns are with choosing the ordering and paths without attention paid to the arrival times. Future work will eliminate this constraint.

2.1 Preliminaries

Table 2 shows a sample message list of the type being considered in this work to be executed on a $10X10$ two-dimensional mesh processor network.

Definition 2.1 *A message is defined to be $M = (m_{eat}, m_P, m_S, m_D)$ where m_{eat} is the estimated arrival time of the message; m_P is the number of packets in the message, m_S is the source node of the message, and m_D is the destination node of the message.*

Definition 2.2 *The estimated arrival time, m_{eat}, is the estimated time a message originates at a node. It is determined by an analysis of the network communication times and the processing requirements of the individual tasks.*

To begin determining a communication schedule, a fixed routing scheme is used. For simplicity strict *XY-routing* is used. However, the *techniques* presented are not restricted to this type of routing. In section 3, *re-routing* in the YX direction is considered as this fixed routing constraint is relaxed. Additionally, no preemption of message transmissions is allowed. Consequently, when a message is scheduled to begin transmission it must have a clear path from source to destination.

2.2 Collision Graph

Central to deriving the communication schedules is a consideration of which messages collide in their routing paths. Without preemption, two or more messages which collide should not be scheduled concurrently. Therefore, a partial ordering is established. To aid in the finding this order, an undirected graph called a *Collision Graph*, or *CG* is introduced.

Definition 2.3 *Given a set of messages M, a Collision Graph is defined as $G = (V, E)$ where V is the set of nodes $v_1, v_2, ...v_N$ representing messages $M_1, M_2, ...M_N$; and $E = \{(v_i, v_j)|$ the paths of M_i and M_j intersect.$\}$*

In using such a model, a node corresponds to a route that a particular message traverses in the network, and edges indicate that these routes collide. Using this definition on the messages from Table 2 produced the CG shown in Figure 2(a). Note that to build the CG the routing scheme must be used. However, the CG is not tied to any particular type of routing. In this paper, XY routing is used to construct the initial CG's, but they could easily be adapted to work with any type of routing technique.

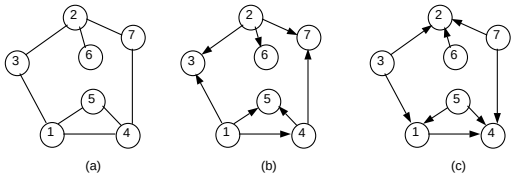


Figure 2: (a) Collision Graph (b) and (c) Different DAGs derived from (a)

2.3 Edge Orientation

In order to determine the communication schedule from this undirected graph some specification of which messages precede others must be made. This is accomplished by adding direction to the edges of the CG. An edge directed from $v_1 \rightarrow v_2$ denotes that the message corresponding to v_1 is to be scheduled before the message corresponding to v_2. If no edge exists between any two nodes then they may be scheduled in parallel. Determining the edge orientation converts the CG into an *Ordered Collision Graph* or OCG and is the major problem to be accomplished. Thus, the task of finding a message schedule has been transformed into a problem of determining the orientation of the edges in the CG model.

Definition 2.4 *An Ordered Collision Graph, $G' = (V, E')$, is derived from the CG, $G = (V, E)$, where V is the set of nodes $v_1, v_2, ...v_N$ representing messages $M_1, M_2, ...M_N$; and is defined as is a directed acyclic graph (DAG) with each undirected edge $e \in E$ corresponding to a directed edge $e' \in E'$ and $|E| = |E'|$.*

From the edge orientation of the OCG, the actual communication schedule can be obtained by first finding the node(s) without any incoming edges, scheduling them to be transmitted in parallel, removing them and their edges from the graph, and repeating the process until all nodes have been scheduled. Consider again the example CG of Figure 2(a). One possible edge orientation is shown in Figure 2(b) and the resulting schedule shown in the first column of Table 3. Observe that because v_7 and v_5 do not have any outgoing edges, the graph is acyclical and deadlock free. Another orientation is shown in Figure 2(c) and the resultant schedule shown in the third column of Table 3. The next section shows that the second schedule is the better of the two. The issue, then, is how to derive the best or optimal schedule from the CG.

2.4 Optimality and Minimum Edge Orientation Problem

Analyzing the schedules in the first and third columns of Table 3 by using worm-hole routing timing assumptions yields times of $14t$ and $11t$ for the *sum* of the transmission times for all messages for scenario 1 and 2 respectively. Throughout this paper the performance metric used will be this overall completion time sum. Note that this sum corresponds directly with the summation of the assigned level values for each node. Thus, these values will be used interchangeably. Reducing this sum can be accomplished in two general ways. The first is to maximize the number of nodes which can be transmitted in each level. The second approach is to minimize the number of levels. As discussed in [12] both of these are necessary conditions in arriving an an optimal schedule. The problem, then, breaks down to finding the maximum independent set at each level which produces a solution with the minimum number of levels. In terms of the Collision Graph model, this problem can be reduced to the following problem called the *Minimum Edge Orientation Problem*, or *MEOP*.

Level	FCFS nodes	ISCOM nodes	MISCOM nodes
1	1, 2	1, 6, 7	3, 5, 6, 7
2	3, 4, 6	2, 4	2, 1
3	5, 7	3, 5	4
Sum	14	13	11

Table 3: Example problem level assignments

Definition 2.5 *Given an undirected graph $G = (V, E)$, determine the orientation of each edge to make G become a DAG (Directed Acyclic Graph). Function $w : V \rightarrow Z$ gives the level of each node according to the partial order of the resultant DAG. The total weight $\sum_1^N w(v)$ is then minimized. Additionally, the fewest number of levels possible is desired so $max_{v \in V}\ w(v)$ is minimized.*

It can be shown from the definition of the problem that to determine the minimum number of levels is equivalent to solving the graph coloring problem which is known to be NP-complete.

3 Scheduling Techniques

In deriving communication scheduling algorithms, two general classes of techniques were used. The first deals with using the CG along with a *fixed* routing scheme to obtain schedules while the second also uses the CG but allows the routing scheme to be altered. With each technique, several algorithms were developed.

3.1 Fixed Routing Algorithms

These algorithms all adhere to the assumptions that all messages have the same arrival time and are of equal length. The first algorithm is an implementation of a *First-come First-served, or FCFS* technique. The message list is processed in a top-down fashion taking each message and trying to schedule it at the lowest available level. From the discussion given in section 2, two criteria were given for determining an optimal schedule. Each of these criteria are addressed by the next two algorithms. The second algorithm, *ISCOM*, or *Independent Set COMmunication*, uses a greedy technique in determining independent sets to be scheduled in parallel at each level and offers a degree of improvement over the $FCFS$ method. It processes by taking the first message in the message list and determines a set of other messages that can run in parallel with it and with each other. It should be noted that this is not necessarily the largest independent set in the message list because the algorithm simply processes the message list searching for *any* set of messages that this first message can execute in parallel with. After determining this independent set, these messages are scheduled and removed from the message list. The process then repeats with the next unscheduled message in the list and continues until all the messages have been scheduled.

The third algorithm, *MISCOM* or *Maximal Independent Set COMmunication*, processes similarly to the $ISCOM$ algorithm but with one difference. The *MISCOM* algorithm determines an independent set for *each* message in the message list and schedules the largest set. The set does not have to contain the first unscheduled message like the *ISCOM* algorithm required. If two or more sets are of the same size, the number of edges in the set is used as the *tie-breaker*.

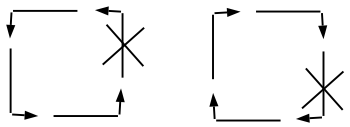


Figure 3: Illustration of allowable routing turns

3.2 Comparison of Fixed Routing Techniques

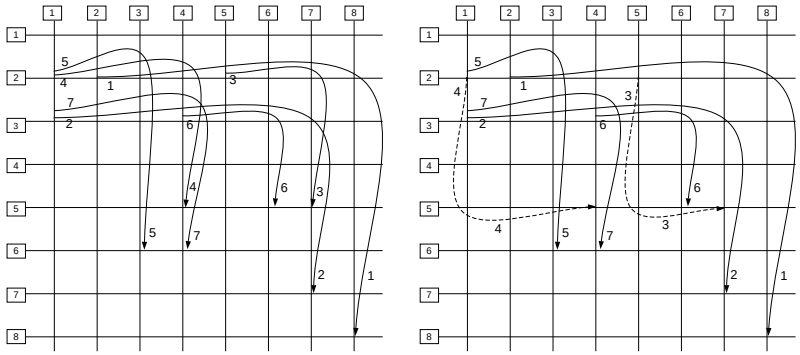
Table 3 shows the resulting schedules derived from using each algorithm on the message list of Table 2. The $FCFS$ technique has messages 1 and 2 in the first level but since they collide with the other messages, they are alone in level 1. The resulting level sum is 14 and is the highest of the three techniques. The $ISCOM$ technique is required to find a maximal independent set with contains the first message. Thus, the nodes at the first level are 1, 6, and 7. For the second level, node 2 must be a part of the set determined since it was not in the set determined for level 1. The resulting sum is 13 showing it to be an improvement over the $FCFS$ approach. The $MISCOM$ algorithm finds an independent set for each message and taking the largest of these produces the level 1 set to be 3, 5, 6, and 7. The resulting sum is 11 making it the best algorithm of the three. Because the $MISCOM$ algorithm is similar but better than the $ISCOM$ algorithm, in subsequent experiments and analysis, the $ISCOM$ algorithm will not be used.

3.3 Communication Scheduling with Re-routing

As shown, judiciously scheduling the communication can reduce the number of levels required. However, being restricted to always using a fixed routing scheme limits the degree of improvement. This restriction is now lifted as re-routing in the *YX* direction is allowed. Before presenting algorithms to do this type of re-routing, the issue of deadlocks must be addressed. *XY-routing* is well known to be deadlock free but when this type of routing is combined with *YX* routing, it is possible for a cycle to occur. In a two-dimensional mesh, there are eight possible turns and two possible cycles [5]. While XY routing prohibits 4 of the turns to prevent a cycle, our work only prohibits the 2 turns shown in Figure 3. Thus, our term for the type of routing that will be used is XY and *restricted YX* routing.

The first technique developed to incorporate re-routing is a form of $FCFS$ rescheduling. Consider again the message list shown in Table 2. The messages are processed in a top-down fashion with *XY* routing tried first. If a collision-free path exists for a message, it is assigned the current level value and scheduled. Otherwise, YX routing will be explored to see if it both is allowed (does not violate the restricted turn constraint, does not travel in only one direction) and if it offers any improvement. If improvement can be obtained, the message is re-routed to traverse the network in a *YX* direction. Figure 4(a) shows the paths the messages of Table 2 take in the network. Figure 4(b) shows the messages after re-routing has been applied with dashed lines denoting that messages 3 and 4 have been re-routed. The fourth column of Table 4 shows the resulting communication schedule with re-routed messages denoted with a ' mark. The level sum is 10 where before it was 14, a 28.6% improvement.

The second algorithm developed reschedules the output of the $MISCOM$ algorithm. This is done by starting with the messages in the highest level and trying to re-route them to see if any improvement can be obtained. In the example, and also shown in column 3 of Table 4, message 4 is in level 3 and is the first candidate for rescheduling. This message is eligible for re-routing and doing so moves it up to level 1. Next, the messages in level 2 are considered and then the technique stops since the messages in level



(a) Original message paths (b) Rescheduled FCFS paths

Figure 4: Paths traversed by messages

Level	FCFS	MISCOM	Rescheduled FCFS	Rescheduled MISCOM	Re-MISCOM w/ improved initial sched.
1	1, 2	3, 5, 6, 7	1, 2, 3', 4'	3, 5, 6, 7, 4', 1'	5, 6, 7, 3', 4', 1'
2	3, 4, 6	2, 1	5, 6, 7	2	2
3	5, 7	4			
Sum	14	11	10	8	8

Table 4: Level assignments with rescheduling

one cannot be moved to a lower level. Column 5 of Table 4 shows the communication schedule obtained and the level sum has improved to 8 from the previous value of 11. Thus, a 27.3% improvement has been obtained.

The last technique developed combines elements of the first two procedures. The rescheduled $FCFS$ approach is performed first. The resulting schedule is then used as input into the rescheduled *MISCOM* algorithm. Thus, in the example a CG is built from the message list corresponding to column 4 of Table 4. The rescheduled $MISCOM$ technique is applied and the resultant schedule is shown in the last column of the table. It is slightly different from the rescheduled $MISCOM$ output, but has the same sum or overall processing time value.

4 Experiments and Results

Simulations were performed on randomly generated lists of 20, 30, 40, 50 and 60 messages with 100 trials for each list size. The values presented are averages of 50 separate trials that were necessary to compensate for the random nature of the message traffic generation algorithm. Additionally, the message traffic was determined randomly but with a *hotspot* parameter incorporated to vary the amount of message collisions. This parameter determines the frequency of message destinations to a particular region of the network.

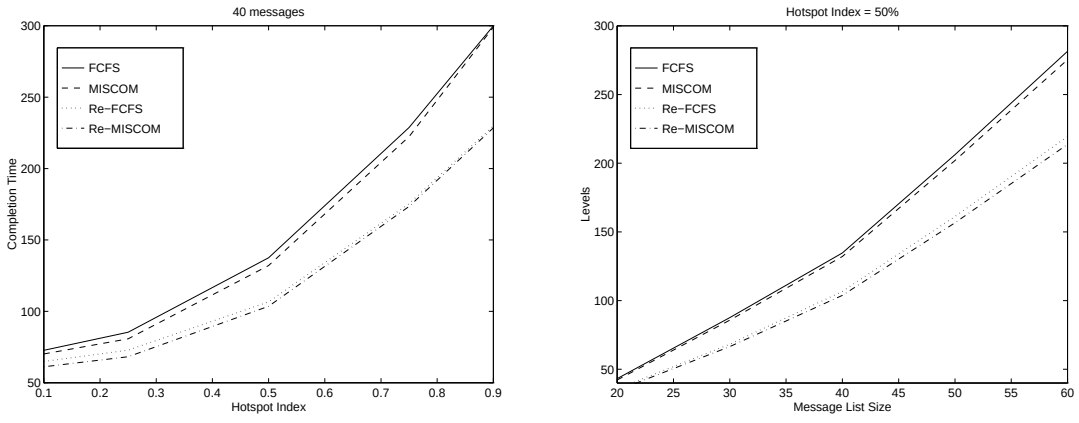
Table 5 contains results for experiments performed on message lists of size 50 with the hotspot index ranging from 10-90%. Values for five algorithms (*FCFS, MISCOM, Rescheduled FCFS, Rescheduled MISCOM, and Rescheduled MISCOM with an improved initial schedule*) are shown with the last column

Hotspot Index	FCFS	MISCOM	Rescheduled FCFS	Rescheduled MISCOM	Re-MISCOM w/ improved init. sched.	Percent Improvement
10%	98.25	94.79	88.92	84.00	83.21	15.65
25%	117.63	113.13	101.06	95.82	94.48	19.68
50%	207.48	201.97	161.29	156.66	156.53	24.56
75%	357.65	355.13	275.53	273.04	272.96	23.68
90%	472.73	471.93	363.24	361.70	361.68	23.49

Table 5: Experiments with 50 messages

representing the maximum percent improvement that can be obtained from using these techniques. Note that for each hotspot index, the Rescheduled $MISCOM$ with an improved initial schedule yields the lowest amount of levels. In addition, note the way the percent improvement behaves as the hotspot index changes. As this index is increased (signifying increased network congestion) the percent improvement increases up to a point (50% hotspot index). Then, the amount of improvement levels off. This leveling off is due to the collision graph resembling more of a *clique*. With a clique, a *FCFS* algorithm does not perform as poorly, and since it is the baseline from which the algorithms are measured, the gain diminishes. Thus, the performance loosely resembles a `bell curve' where the maximum performance gain appears at a moderate amount of collisions.

While the gains do diminish slightly as the amount of collisions becomes large, the overall improvement is still over 20% even when the hotspot index is 90% and 60 messages are being transmitted. Figure 5(a) shows the relationship between the performance gain and the hotspot index for a 40 message experiment. Because the amount of improvement that can be gained from using the newstart $MISCOM$ is so small compared to using the rescheduled $MISCOM$, it is not shown in the graph. In fact, it is our conclusion that to get the additional small amount of improvement it is not worth the extra processing that is required. From the graph, note that the rescheduled MISCOM is always much better than the two techniques which do not use rescheduling, $FCFS$ and $MISCOM$. Considering the effects of increased message list sizes on the performance is shown in Figure 5(b). Here the hotspot index is fixed at 50% and the message list sizes are varied from 20 to 60 messages. Note again that the two rescheduling techniques perform much better than the $FCFS$ and $MISCOM$ techniques.



(a) Performance vs. Hotspot Index

(b) Performance vs. Message List Size

Figure 5: Performance graphs

5 Conclusions

While working on massively parallel systems, it was found that the communication overhead greatly impacted the system performance. A compile-time technique was developed using the newly developed *Collision Graph* to reduce this overhead by *scheduling* the individual message transmissions. It was found that using a re-routing strategy resulted in a very significant improvement over a fixed routing technique. Together, the Collision Graph and this scheduling strategy form a framework for which continued study into communication scheduling can be done. While point-to-point transmissions were studied in this paper, future work is centering on *multi-casts* as well as on a more general case model of the message traffic.

References

[1] P. M. Kogge, "EXECUBE- A New Architecture for Scalable MPPs," in *1994 International Conference on Parallel Processing*, vol. I, pp. 77–84, August 1994.

[2] A. Fettweis and G. Nitsche, "Numerical integration of partial differential equations using principles of multidimensional wave digital filters," *Journal of VLSI Signal Processing*, no. 3, pp. 7–24, 1991.

[3] H. Kasahara and S. Narita, "Practical multiprocessor scheduling algorithms for efficient parallel processing," *IEEE Transactions on Computers*, vol. c-33, November 1984.

[4] H. El-Rewini, T. G. Lewis, and H. H. Ali, *Task Scheduling in Parallel and Distributed Systems*. Englewood Cliffs, NJ: Prentice Hall, 1994.

[5] L. M. Ni and P. McKinley, "A survey of wormhole routing techniques in direct networks," *IEEE Computer*, vol. 26, February 1993.

[6] S. Lee and J. Kim, "Path selection for communicating tasks in a wormhole-routed multicomputer," in *1994 International Conference on Parallel Processing*, vol. 3, pp. 172–175, 1994.

[7] W. J. Dally and C. L. Seitz, "Deadlock-free message routing inmultiprocessor interconnection networks," *IEEE Transactions on Computers*, vol. C-36, pp. 547–553, May 1987.

[8] R. P. Bianchini and J. P. Shen, "Interprocessor traffic scheduling algorithm for multiple-processor networks," *IEEE Transactions on Computers*, vol. C-36, pp. 396–409, April 1987.

[9] D. D. Kandlur and K. G. Shin, "Traffic routing for multicomputer networks with virtual cut-through capability," *IEEE Transactions on Computers*, vol. c-41, pp. 1257–1270, October 1992.

[10] A. Eberhart and J. Li, "Contention-free communication scheduling on 2d meshes," in *1996 International Conference on Parallel Processing*, pp. 44–51, 1996.

[11] D. R. Surma and E. Sha, "Application specific communication scheduling on parallel systems," in *Eigth International Conference on Parallel and Distributed Computing Systems*, pp. 137–139, September 1995.

[12] D. R. Surma and E. Sha, "Static communication scheduling for minimizing collisions in application-specific parallel systems," in *International Conference on Application-Specific Systems, Architectures, and Processors*, pp. 210–219, August 1996.