Collision Graph based Communication Scheduling with Re-routing in Parallel Systems [*]

David Ray Surma
Edwin Hsing-Mean Sha
Dept. of Computer Science & Engineering
University of Notre Dame
Notre Dame, IN 46556

Research Report TR-97-2
January 1997

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. One of the biggest problems in petaflop computers is the routing of the messages and the overhead involved. To solve this problem, we propose a study of the communication involved when nodes transfer information.

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 [3–6] 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, such as the one presented in [3], has been used and deals specifically with the message transmissions. Therefore, the new scheduling technique is not a type of multi-processor scheduling, but rather deals strictly with determining the ordering and routing of these transmissions.

It was found that the compiler can play a significant role in deciding what is best in determining this ordering and routing. 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. Indeed this is the approach that this paper follows. We present compile-time algorithms which work to reduce the run-time communication overhead.

To fully understand this problem, consider as an example a 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 two-dimensional mesh network of six processors. Tasks assigned to the same processor incur no communication overhead. Otherwise, the task assignments indicate that messages must be sent between nodes in the graph to exchange computed data. Thus, node 1

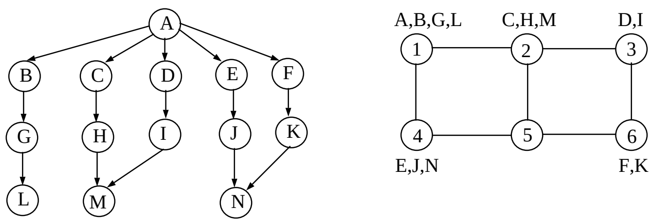

Figure 1: (a) Task Flow DAG. (b) Tasks assigned to processing nodes

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

Table 1: Example Communication Schedules

must send messages to nodes 2, 3, 4, and 6 corresponding to edges $A \to C, A \to D, A \to E$, and $A \to F$ of the DAG. Additionally, node 3 must send a message to node 2 ($I \to M$), and node 6 must send a message to node 4 ($K \to N$). Since there is only a single bidirectional link between each node, it is obvious that collisions in the network occur. The first two columns of Table 1 give possible orderings of the message traffic when *XY-routing* is being used. Messages appearing on the same line may be sent in parallel without collisions occurring. Under *wormhole* [7] 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, these two orderings yield different completion times. Schedule 1 would complete at time $4t$ while schedule 2 provides an ordering which completes at time $3t$. Thus, there is a savings of 25% based simply on the communication schedule.

An even greater amount of improvement can be obtained if message ($A \to F$) is rerouted to traverse in a YX direction. The third column shows the schedule when the routing scheme is relaxed to allow this transfer with the rerouting of this message denoted as $A \to F'$. The completion time of this new orderings is $2t$. Thus, while judiciously selecting an ordering of the messages can improve the overall completion time as the second column indicates, the time can be reduced even further by re-routing some of the messages. It is the combination of both techniques, the ordering or *scheduling* of the messages as well as the re-routing of some of them, done in order to reduce the completion time that is the problem addressed in this paper.

The term used for this research is *communication scheduling*. Not only does it encompasses routing aspects and path selection issues as discussed in [8, 9], it also determines the order and timing that the messages in the system should be sent. Thus, this problem is much different from the traditional multi-processor scheduling problem [10–12]. There have been several studies related to the problem addressed here. Many of them deal with computer networks and the store-and-forward which occurs using routing tables [13]. Recent efforts have focused on developing communication algorithms for interconnection networks. One such work was done by Bianchini and Shen [14]. 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. A traffic smoothing routine was implemented which simulates a fluid-flow pipeline where the network links behave like fluid in pipes and can be split or combined in different ways. Consequently, a communication link can be used as a large number of channels that can be split or combined in different ways. However, they do not perform any scheduling of the individual message transmissions. Rather, they rely on a dynamic $FCFS$ approach. Thus, their objective is different from ours where we consider tasks already mapped to processors and the goal is to select the route and ordering of when the nodes should transmit their messages.

Lee and Kim [8] in their work 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 they need to communicate. They also do not consider the scheduling of the individual messages. Our work deals with these individual message transfers and does not dedicate paths to nodes. Kandlur and Shin [15] present a work similar to [8] in that dedicated paths are found. The problem with these techniques is that seldom used dedicated paths can cause other messages to follow longer paths when the needed links which are dedicated are not being used. Additionally, they do not use any type of scheduling which can improve the overall performance. Recent work by Eberhart and Li [16] 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 [17] 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 [18–20] 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* [21] 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. For clarity, throughout the paper dimension ordered XY routing will be used in the examples and in the experiments. Re-routing will consider routing in the YX direction as well. The techniques presented in this paper are not restricted to these types of routing. In fact, the Collision Graph can be built using any type of routing. XY routing is chosen for its simplicity in presenting this new area of study.

The outline of the work is as follows. The next section introduces the ideas of the Collision Graph and shows how it is used in the communication scheduling problem. Section 3 introduces the idea of restricted re-routing and shows that it can further reduce the communication overhead. Next, the general scheduling technique and the results of experiments performed on this and baseline approaches are presented. Lastly, concluding remarks are given.

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. By schedule, it is meant the ordering of when the messages are to be transmitted and the paths they will traverse in the network. Because determining such a schedule is very difficult, a transformation into a simpler, better defined problem is sought. Such a transformation is presented in this section along with examples showing this transformation and techniques used to solve the new problem. To form the foundation upon which the communication scheduling algorithms are built, this paper considers a restricted case model of network traffic. In this analysis, all messages are assumed to have the same origination time at the sending nodes. Thus, the presented analysis concerns itself with choosing the ordering and paths without attention paid to the arrival times. The handling of a general case message traffic is left for future work.

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. The formal definition of a message in the context of this paper is as follows.

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 start the process of determining a communication schedule, a fixed routing scheme is used. For simplicity strict *XY-routing* is used throughout this paper. However, the techniques presented are not restricted to this type of routing. In section 3, *re-routing* will be considered and this fixed routing constraint will be relaxed. Additionally, when transmitting messages, no preemption is allowed. Consequently, when a message is scheduled to begin transmission it must have a clear path from source to destination.

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

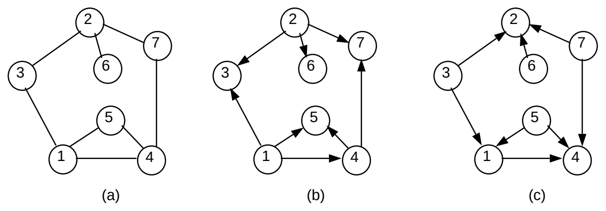

Figure 2: (a) Collision Graph (b) and (c) Different DAGs derived from (a)

With these assumptions, the graph model is introduced which will be the basis for determining the various communications schedules.

2.2 Collision Graph

The central aspect in determining the communication schedule in a network requires considering which messages collide in their routing patterns. Without preemption, two or more messages which collide should not be scheduled at the same time. Therefore, a partial ordering is established among these messages because one must follow the other. Conversely, messages which do not collide may be scheduled concurrently. To aid in the ordering process, 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. Hence, there is a one-to-one correspondence between the size of the graph and the number of messages being sent. The edges in the graph indicate that these messages collide. Using these criteria on the message schedule 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.

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 arrows to the CG. Doing this transforms the graph from an undirected graph into a DAG. Since determining the direction of the edges is the major problem to be accomplished, the task of finding a message schedule has been transformed into a problem of determining the orientation of the edges in the CG model.

2.3 Edge Orientation

Determining the orientation of the edges determines the schedule ordering. 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.

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 in a straightforward manner. The first step is to find the node(s) without any incoming edges. These nodes are then scheduled to be transmitted in parallel. Next, these nodes and their edges are removed from the graph, and the process repeats until all nodes have been scheduled.

Consider again the example shown in Figure 2(a). The undirected graph shows the collision pattern of the messages. One possible orientation of the edges is shown in Figure 2(b). Here the resulting schedule is nodes (v_1, v_2) being transmitted in parallel followed by (v_3, v_4, v_6) in parallel and then (v_5, v_7). It is important to observe that because v_7 does not have any outgoing edges, the graph is acyclical and deadlock free. Now consider the graph shown in Figure 2(c). This edge orientation produces a schedule of nodes (v_3, v_5, v_6, v_7) in parallel followed by (v_2, v_1) and then (v_4). Again note that there is no cycle.

Considering these two schedules, the second one is the better one as will be explained in the next section. The issue, then, is how to derive the best schedule from the CG. Furthermore, a question arises whether there even exists an optimal schedule. If such a schedule does exist, the characteristics need to be determined and the schedule found.

2.4 Optimal Communication Scheduling

The schedule corresponding to Figure 2(c) is better than the one corresponding to Figure 2(b) because it has four messages proceeded in parallel followed by two messages after a delay

Level	Scenario 1	Scenario 2
1 nodes	1, 2	3, 5, 6, 7
2 nodes	3, 4, 6	2, 1
3 nodes	5, 7	4

Table 3: Level assignments for the example problem

and then the last message after another delay. In (b) the schedule will have only two messages processing at the start followed by three messages after a delay and then the final two messages. Thus, if we look at these transfers by grouping together the nodes that can be processed in parallel and putting them into levels we get Table 3. Messages at a particular level can be transmitted in parallel. Each message at the higher levels must wait until one or more messages in a lower level have been transmitted. Thus, using worm-hole routing assumptions and considering that a message takes time t to traverse the network, the sum of the times for all messages to complete their transmissions fir scenario 1 is $14t$. The corresponding time for scenario 2 is $11t$. Throughout this paper the performance metric used will be this overall completion time sum. Note that if the nodes are assigned a value corresponding to the level where they are scheduled (i.e. nodes in level 1 have an assigned level value of 1) then this overall completion time is synonomous with the level sum (e.g. for scenario 2, the completion time is $11t$ and the level sum is 11). Thus, this level sum will be used interchangeably with the overall completion time sum.

To minimize this level sum, or overall processing time, the following equation is used. It is derived from the CG where N and V are as previously defined, R is the resulting summation value, and w is a function which returns the level of a particular node.

$$R = \sum_{1}^{N} w(V_i) \tag{1}$$

The goal is to minimize R, and by doing so the optimal schedule will be obtained. In the next section it will be shown that this problem is NP-complete. The following discussion provides the intuition necessary to understand that claim.

It makes intuitive sense that having as many nodes as possible executing in parallel improves the performance. However, we claim that while this is a necessary criteria for scheduling it is not sufficient. Consider the CG given in Figure 3(a). The maximum number of nodes that can be scheduled in the first level is three. There are several such combinations with two of them being shown in Figures 3(b) and (c). The OCG in part (b) has nodes 1, 2, 3 in level 1 followed by nodes 4 and 6 in level 2 and node 5 in level 3. Using equation 1 we get R=10. Now consider the OCG in part (C). Here level 1 is comprised of nodes 1, 3, 6 and level 2 has nodes 2, 4, 5 yielding an R value of 9. Thus, the second scenario provides a better communication schedule.

What this example showed was that selecting the largest independent set at a particular level (in this case level 1) does not guarantee an optimal solution. Thus, a second criteria is

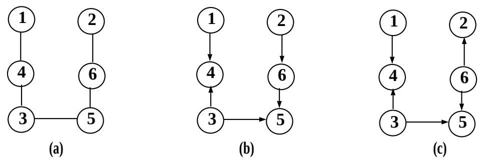

Figure 3: (a) Collision Graph. (b) and (c) Two resulting OCGs.

required. Note that the first scenario had three levels containing nodes while the second scenario only used two levels. Thus, minimizing the *total number of levels* is a necessary condition to determine the 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.

2.5 Minimum Edge Orientation Problem

The problem faced in determining an optimal schedule from the Collision Graph model can be reduced to the following problem called the *Minimum Edge Orientation Problem*, or *MEOP*.

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 $w : V \to Z$ 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 [21].

Theorem 2.1 *The decision problem of the MEOP is NP-complete.*

Since finding an optimal solution is NP-complete, heuristics are used in determining several efficient algorithms. The Collision Graph is the basis for all of the algorithms. Additionally, it is used in the re-routing techniques. These are addressed in the next section.

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

The algorithms designed in this class adhere to the assumptions that all messages have the same arrival time and are of equal length. Additionally, XY routing will be used to determine the initial CG. The CG, it should be remembered, is a general construct and can be constructed from any form of routing. As has been stated, XY routing is used in the examples and experiments due to its simplicity in presenting the ideas and techniques of this research. From the discussion given in section 2, two criteria were given for determining an optimal schedule. The first is to have as many messages transfer in parallel as possible. The second criteria is to have as many nodes start at the earliest possible time. Doing this requires some messages to be delayed to allow a larger set of messages to be processed sooner. Because this problem is NP-Complete, heuristics are used in developing three algorithms. The first algorithm is an implementation of a first-come first-served method of scheduling. The second algorithm uses a greedy technique in determining its independent sets and offers a degree of improvement over the *FCFS* method. The third algorithm implements a technique called the *maximal independent set* method to obtain even better results.

3.1.1 First-come First-served Scheduling Algorithm

This algorithm operates in a way which simply processes the list of messages to be sent in a top-down fashion taking the first message and trying to schedule it at the lowest available level. The algorithm continues processing the message list in this manner until all of the messages have been scheduled. This algorithm is called *FCFS* because it simply schedules the message list in the order that it has been obtained.

3.1.2 ISCOM Algorithm

This algorithm is an improvement over the *FCFS* technique. *ISCOM*, or *Independent Set COMmunication*, strives to exploit the parallelism in the message list thereby improving the scheduling of the messages. It determines a set of messages which can run in parallel in the network and assigns them to the next available level. This determination of an independent set is where a greedy technique is applied. In general, *greedy* techniques make processing choices based on what appears best at the moment.

The *ISCOM* algorithm takes the first message in the message list and determines a set of other messages in the network that can run in parallel with it and with each other. Thus, an independent set of messages is determined. It should be noted that this is not necessarily the largest independent set in the message list, nor is it necessarily the maximum independent set that the first message appears in. 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. Processing continues by finding the next unscheduled message in the list, determining an independent set, scheduling the set and then removing it. This continues until all the messages have been scheduled.

3.1.3 MISCOM Algorithm

The *MISCOM*, or *Maximal Independent Set COMmunication*, algorithm differs from the *ISCOM* in that the latter algorithm started by scheduling an independent set of messages containing the *first* message in the list. The *MISCOM* algorithm drops the requirement that the first message be in the set. Therefore, the algorithm determines an independent set for each message in the message list using the same technique as the *ISCOM* algorithm. The size of each set is noted and is used to determine which set will be scheduled. The set to be scheduled is the largest independent set and if a tie exists, the number of edges of the nodes in the independent set is used as a *tie breaker*. A set with a larger number of edges will be selected over a set with fewer edges. Using this criteria as a tie breaker provides a better solution because when this set is scheduled and removed, more nodes will have dependencies eliminated and will be candidates for scheduling. If a tie still exists, a *FCFS* approach is employed to arbitrarily make the selection.

Upon determining the first set to be scheduled, the algorithm continues similarly to the *ISCOM* algorithm in that independent sets are determined, scheduled, and then deleted from the message list. It should be noted that while this algorithm is an improvement over the *ISCOM* algorithm, it still does not find the maximum independent set of the messages. Rather, it determines one independent set for each message and takes the largest of these sets.

3.1.4 Comparison of Fixed Routing Techniques

Consider the CG shown in Figure 2(a). Table 4 shows the resulting schedules derived from using each of the described techniques. The first column contains the results from performing the $FCFS$ technique. Messages 1 and 2 are in the first level but since they collide with the other messages, they are alone in level 1. The level sum is 14 and is the highest of the three techniques. The second column employs the $ISCOM$ technique. That techniques requires that a maximal independent set be found 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. Next, the $MISCOM$ algorithm is used. It proceeds by finding an independent set for each message and taking the largest of these. The set at level 1 is determined to be 3, 5, 6, and 7. The final schedule is shown in the last column of the table and the resulting sum is 11.

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 4: Example problem level assignments

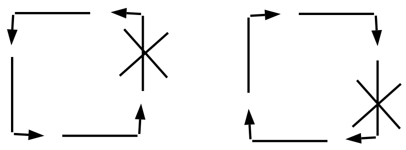

Figure 4: Illustration of allowable routing turns

Thus, the MISCOM algorithm was found to yield the best results. Because of this, in subsequent experiments and analysis, only the $FCFS$ and $MISCOM$ techniques will be used. The $ISCOM$ algorithm will not be used since it is similar but does not perform as well as the $MISCOM$ algorithm.

3.2 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 such as XY routing limits the degree of improvement. In this section, this restriction will be lifted allowing more freedom of path selection. Obviously, if conflict-free paths can be found, the number of levels required will decrease and a better communication schedule results. Three algorithms were developed to address this issue. The first is a form of $FCFS$ rescheduling, the second reschedules the output of the $MISCOM$ algorithm, and the third reschedules the $MISCOM$ algorithm as well but the $MISCOM$ algorithm begins with an improved initial schedule. Each of these will be shown in section 4 to outperform techniques which do not allow any re-routing. Before these techniques are presented, the issue of deadlocks must be considered.

3.2.1 Deadlock Avoidance

In any message passing interconnect, the issue of deadlocks must be addressed. Strict dimension-ordered XY-*routing* is well known to be deadlock free. However, when combining this type of routing 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 [7]. XY routing prohibits 4 of the turns to prevent such a cycle from occurring. In this work, only 2 turns will be prohibited as shown in Figure 4. Thus, our term for the type of routing that will be used is XY and *restricted YX* routing.

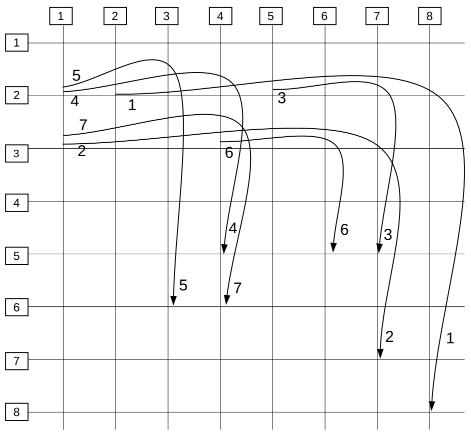

Figure 5: Paths traversed by messages

3.2.2 Rescheduled FCFS

The first technique developed to take advantage of the more relaxed routing was a $FCFS$ approach. Just as was done before, the messages are processed in a top-down fashion. Consider again the message list shown in Table 2. The results of XY routing for $FCFS$ were shown in Table 4. The new technique begins by trying to schedule a message in an XY fashion. If a collision-free path exists, it is assigned the current level value and scheduled. If, however, a collision-free path cannot be found, YX routing will be explored. If an improvement can be achieved, then the message will be scheduled using YX routing. However, if YX routing cannot be done (violates the restricted turn constraint, only travels in one direction) or if it offers no improvement, then the message will be scheduled using the original XY pattern.

Figure 5 shows the paths the messages of Table 2 take in the network. With *rescheduled FCFS*, the processing would be as follows. Messages 1 and 2 would get scheduled using XY routing since they do not collide. They would be assigned to level 1. Trying to use XY routing for message 3 results in a collision with both messages 1 and 2. Thus, YX routing is considered. Since this form of routing can be done (no violation of the constraints), message 3 gets re-routed and will also be scheduled at level 1. Figure 6 shows message 3 in a dotted line to denote that it has been rescheduled to traverse the network in a YX manner at level 1.

Continuing on, message 4 will be re-routed and will also process at level 1. Message 5, however, would have a conflict in either type of routing. Therefore, since no advantage can be gained, it will remain scheduled with XY routing and will be delayed until level 2. In a similar manner, message 6 and 7 have collisions in either direction. Consequently, they will be scheduled using XY routing and will be in level 2.

The fourth column of Table 5 shows the communication schedule obtained from using $FCFS$ with re-routing. Note that the level sum is 10 where before it was 14, a 28.6% improvement.

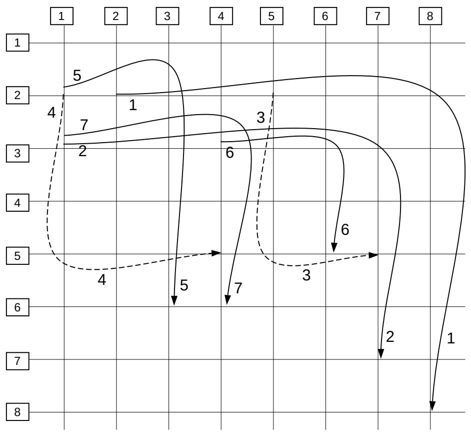

Figure 6: Rescheduled FCFS paths

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 5: Level assignments with rescheduling

3.2.3 Rescheduled MISCOM

The next technique developed reschedules the results obtained from the *MISCOM* technique. This was done by starting with the messages in the highest level and then trying to re-route them to see if any improvement can be obtained. In the example, and also shown in column 3 of Table 5, message 4 is in level 3 and is the first candidate for rescheduling. As previously described, a message must satisfy deadlock avoidance and other criteria before it can be rescheduled. Having satisfied these, message 4 is re-routed to traverse the network in a YX fashion thereby moving it up to level 1. Figure 7 shows this rescheduling with message 4 being denoted with a dashed line.

Next, the messages in level 2 are considered. There are two of them and they are processed in an arbitrary fashion. Message 2 is considered first and found that if it traverses the network in a YX manner, it will collide with message 4 and still be in level 2. Therefore, since no improvement can be achieved, it is not rescheduled. Message 1, however, can be re-routed to move up to level 1 in the schedule. Figure 7 shows message 1 being re-routed.

At this point, this technique stops since the messages in level one obviously cannot be re-routed to move to a lower level. Column 5 of Table 5 shows the communication schedule obtained from using *MISCOM* with re-routing. The level sum has improved to 8 from the previous value of 11. Thus, a 27.3% improvement has been obtained.

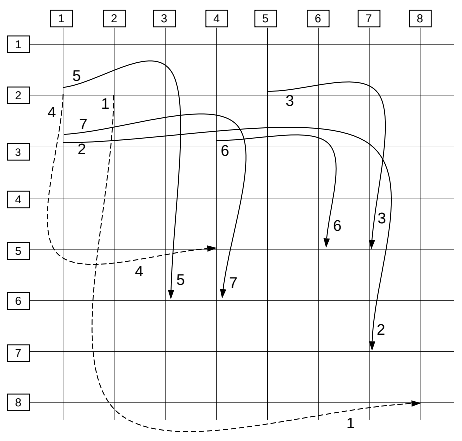

Figure 7: Rescheduled MISCOM paths

3.2.4 Rescheduled MISCOM with Improved Initial Schedule

The last technique explored combines elements of the first two procedures, the rescheduled $FCFS$ and the rescheduled $MISCOM$. For fixed routing it was shown that the $MISCOM$ technique outperforms the $FCFS$ procedure. Likewise, for re-routing, rescheduled $MISCOM$ outperforms the rescheduled $FCFS$ technique. However, $MISCOM$ works on messages that all have the default fixed routing scheme of XY. This new algorithm takes the output of the rescheduled $FCFS$ and uses it as input to the rescheduled $MISCOM$ technique. Thus, a CG is built from the message list corresponding to column 4 of Table 5. The $MISCOM$ technique is applied which yields level 1 nodes of (5, 6, 7, 3', 4'). next, messages 1 and 2 are considered for re-routing. Only message 1 can be re-routed to achieve a better performance. The resultant schedule is shown in the last column of Table 5. 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 6 contains results for experiments performed on message lists of size 20 with the hotspot index ranging from 10-90%. Values for five algorithms ($FCFS$, $MISCOM$, Rescheduled $FCFS$, Rescheduled $MISCOM$, and the Rescheduled $MISCOM$ with an improved initial schedule) are shown with the last column representing the maximum percent improvement that

Hotspot Index	FCFS	MISCOM	Rescheduled FCFS	Rescheduled MISCOM	Re-MISCOM w/ improved init. sched.	Percent Improvement
10%	29.20	28.31	26.46	25.76	25.64	12.19
25%	31.89	31.04	28.04	27.29	27.11	14.99
50%	43.18	42.23	35.75	34.53	34.35	20.45
75%	67.49	66.67	52.30	51.61	51.41	23.83
90%	85.66	85.29	66.30	65.49	65.36	23.70

Table 6: Experiments with 20 messages

Hotspot Index	FCFS	MISCOM	Rescheduled FCFS	Rescheduled MISCOM	Re-MISCOM w/ improved init. sched.	Percent Improvement
10%	49.62	47.86	44.34	42.22	42.10	15.16
25%	55.99	54.13	48.51	46.16	46.01	17.82
50%	87.65	85.80	68.22	66.49	66.00	24.70
75%	137.75	135.97	105.40	104.28	104.07	24.45
90%	178.17	177.72	136.03	135.21	135.05	24.20

Table 7: Experiments with 30 messages

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 that the percent improvement increases from 12% to approximately 24%. This improvement indicates that performing communication scheduling significantly improves the performance in a network.

Table 7 gives the results of experiments performed on message lists of size 30. Again note that significant performance gains of over 20% can be obtained by using re-routing of the $MISCOM$ algorithm. Comparing this table with Table 6 it is found that the percent improvement increases for each hotspot index value. Thus, as the number of messages increase, which causes more collisions to occur in the network, the scheduling of the communication becomes even more valuable.

The question arises, however, will this performance always increase as the number of messages or collisions increases. To answer this, note that in Table 6 when the hotspot index increases from 75% to 90% the performance *improvement* decreases slightly. Also look at Table 7 when the hotspot index increases from 50% to 75% and to 90%. Again, the performance gain begins to diminish. This is due to the fact that as more and more messages have collisions, the corresponding collision graph resembles more of a *clique*[22]. With a clique, a first come first serve algorithm does not perform as poorly, and since it is the baseline from which the algorithms are measured, the gain diminishes. For the 30 message case, the reason why the gains began to diminish after a 50% hotspot index is because the message list is larger. Increased message list size and hotspot index are the two factors which increase the number of collisions which occur.

While the gains do diminish slightly as the amount of collisions becomes large, the overall improvement is still over 20%. This amount of improvement stays in this range as the number of messages increases to 40, 50 and 60. Tables 8, 9, and 10 show this improvement. Additionally,

Hotspot Index	FCFS	MISCOM	Rescheduled FCFS	Rescheduled MISCOM	Re-MISCOM w/ improved init. sched.	Percent Improvement
10%	72.71	70.17	64.91	61.85	61.19	15.84
25%	85.42	80.80	72.87	69.10	68.22	20.14
50%	137.56	132.11	106.67	103.83	103.59	24.69
75%	228.76	222.32	175.44	173.59	173.47	24.17
90%	299.92	299.02	229.92	228.69	228.50	23.81

Table 8: Experiments with 40 messages

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	361.70	23.49

Table 9: Experiments with 50 messages

the effects of increased collisions can be seen a little more clearly as the performance gain from using rescheduling diminishes as the hotspot index increases and the number of messages increases. Thus, from looking at these tables the following conclusions can be drawn. The rescheduled $MISCOM$ technique provides an improvement over a baseline $FCFS$ technique in the range of 20%. The exact amount of improvement starts lower when there are few collisions in the network caused by a low hotspot index or by few messages. As the amount of collisions increases (hotspot index increases and the message list size grows) the performance increases up to a point. This indicates that the rescheduling technique works especially well when the message traffic is moderate. Improvement of around 25% is obtained.

As the number of collisions increases further, the performance gains then begin to diminish. This is due to the collision graph resembling more of a clique. Thus, the shape of a performance graph would loosely resemble a 'bell curve'. Still, even at 60 messages with a 90% hotspot index, the improvement is 23%.

Figure 8 shows the relationship between the performance gain and the hotspot index for the 40 message case. 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

Hotspot Index	FCFS	MISCOM	Rescheduled FCFS	Rescheduled MISCOM	Re-MISCOM w/ improved init. sched.	Percent Improvement
10%	127.55	122.09	115.33	108.89	107.85	15.44
25%	156.79	154.66	135.18	128.50	127.20	18.87
50%	281.51	275.42	219.44	213.72	212.79	24.41
75%	492.84	489.13	381.66	378.58	377.69	23.36
90%	649.13	647.39	502.84	501.17	500.45	22.90

Table 10: Experiments with 60 messages

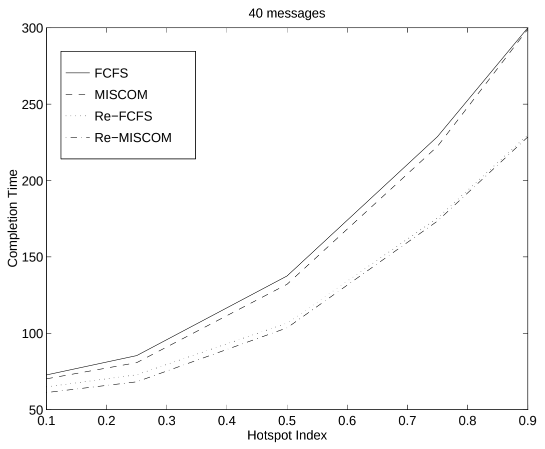

Figure 8: Performance vs Hotspot Index

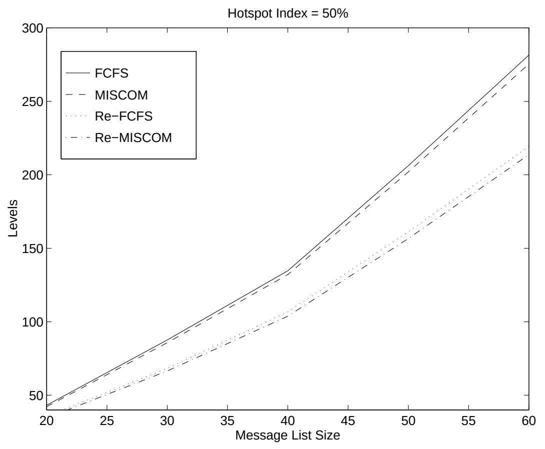

Figure 9: Performance vs Message List size

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 performace is shown in Figure 9. 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.

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 to reduce this overhead by *scheduling* the individual message transmissions. By scheduling, it is meant the ordering and routing of the individual message transmissions. The basis of this technique is the newly developed *Collision Graph* construct. 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] D. I. Moldovan and J. A. B. Fortes, "Partitioning and mapping algorithms into fixed size systolic arrays," *IEEE Transactions on Computers*, vol. c-35, pp. 1–12, January 1986.

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

[5] A. A. Munshi and B. Simons, "Scheduling sequential," *SIAM Journal of Computing*, pp. 728–741, August 1990.

[6] S. Shukla, B. Little, and A. Zaky, "A compile-time technique for controlling real-time execution of task-level data-flow graphs.," in *1992 International Conference on Parallel Processing*, vol. II, pp. 49–56, 1992.

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

[8] 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.

[9] 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.

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

[11] K. R. Baker, *Introduction to Sequencing and Scheduling.* New York, NY: Wiley, 1974.

[12] R. W. Conway, W. L. Maxwell, and L. W. Miller, *Theory of Scheduling.* Reading, MA: Addison Wesley, 1967.

[13] A. S. Tanenbaum, *Computer Networks.* New Jersey: Prentice-Hall, 1996.

[14] 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.

[15] 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.

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

[17] 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.

[18] D. R. Surma, S. Tongsima, and E. Sha, "Optimal communication scheduling based on collision graph model," in *1996 International Conference on Accoustics, Speech and Signal Processing*, vol. VI, pp. 3319–3322, 1996.

[19] 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.

[20] D. R. Surma and E. Sha, "Hybrid static-dynamic communication scheduling for parallel systems," in *ACM Symposium on Applied Computing - to be published*, February 1997.

[21] M. R. Garey and D. S. Johnson, *Computers and Intractability, A Guide to the Theory of NP-Completeness.* New York, NY: W.H.Freeman and Company, 1979.

[22] K. Hwang, *Advanced Computer Architecture, Parallelism, Scalability, Programmability.* New York, NY: McGraw-Hill, 1993.