

# Communication Scheduling with Re-routing based on Static and Hybrid Techniques <sup>\*</sup>

David R. Surma <sup>†</sup> Edwin H-M. Sha <sup>‡</sup> Nelson Passos <sup>§</sup>

## Abstract

In massively parallel systems, the performance gains are often significantly diminished by the inherent communication overhead. This overhead is caused by the required message passing resulting from the task allocation scheme. In this paper, techniques to reduce this communication overhead by both *scheduling* the communication and determining the routing that the messages should take within a tightly-coupled processor network are presented. Using the recently developed *Collision Graph* model, static scheduling algorithms are derived which work at compile-time to determine the ordering and routing of the individual message transmissions. Since *a priori* knowledge about the network traffic required by static scheduling may not be available or accurate, this work also considers dynamic scheduling. A novel hybrid technique is presented which operates in a dynamic environment yet uses known information obtained by analyzing the communication patterns. Experiments performed show significant improvement over baseline techniques.

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

---

<sup>\*</sup>This work was supported in part by NSF MIP 9501006 and NSF ACS 96-12028.

<sup>†</sup>Dept. of Computer & Information Sciences, Indiana University South Bend, South Bend, IN 46634.

<sup>‡</sup>Dept. of Computer Science, University of Texas at Dallas, Dallas, TX 75083.

<sup>§</sup>Dept. of Computer Science, Midwestern State University, Wichita Falls, TX 76308.

# 1 Introduction

Techniques to implement massively parallel processing have advanced rapidly in recent years as evidenced by the development of systems such as the Cray T3D and T3E, the IBM SP-2 and others. While considerable improvements have been made in the hardware technology, there remains a wide gap between the CPU speed and the communication times. This communication overhead presents a problem as it can severely limit the overall processing performance. To reduce this overhead the creation of a new scheduling technique was required since most traditional scheduling methods do not consider the communication characteristics of the problem in the detail required to achieve an optimal schedule [1–3]. Furthermore, neither do most techniques developed for parallel compilers [1, 4]. In multi-processor systems, the task allocation scheme directly determines the communication requirements. This research assumes that a suitable scheme, such as the one presented in [5], has been used and deals specifically with the ordering, timing and routing of the message transmissions. Therefore, the new scheduling technique is much different from traditional multi-processor scheduling [3, 6, 7] and instead does scheduling at a lower level. Thus, this work presents a novel scheduling technique, called *SCORE*, that works at compile-time to reduce the run-time communication overhead. Since information required by a static technique is often unavailable or inexact, this work expands by considering dynamic scheduling. The presented *hybrid* static-dynamic *HYCORE* technique involves both compile-time analysis and run-time scheduling to reduce the communication overhead. Experiments with both techniques show significant improvements over baseline techniques.

To fully understand the problem of this research, consider the *task directed acyclical graph* 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 require no inter-node communication, this assignment scheme indicates that messages must be exchanged. For example, node 1 sends messages to nodes 2, 3, 4, and 6 corresponding to edges  $A \rightarrow E$ ,  $A \rightarrow D$ ,  $A \rightarrow C$ , and  $A \rightarrow B$  of the *DAG*. Assuming a single bidirectional link between each node, network collisions will occur. The first two columns of Table 1 give possible schedule orderings of the resulting message traffic when *XY-routing* and *all-port* routers [8] are used. These orderings are derived such that messages appearing on the same line do not collide with each other but messages in subsequent lines do collide with at least one of the messages in this group. This research utilizes *wormhole* routing [9] where due to pipelining it is commonly assumed that the distance, as measured in the number of links a message must traverse, is negligible [10]. Therefore, since the messages considered are of equal length, we assume that each message takes the same amount of

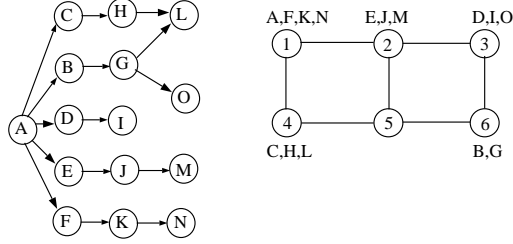


Figure 1: (a) Task Flow Graph (b) Tasks assigned to processing nodes

Schedule 1	Schedule 2	Re-routed Schedule
$A \rightarrow E; A \rightarrow C$	$A \rightarrow B; A \rightarrow C$	$A \rightarrow B'; A \rightarrow D$
$A \rightarrow D$	$A \rightarrow D; G \rightarrow L; G \rightarrow O$	$A \rightarrow C; A \rightarrow E; I \rightarrow M; G \rightarrow L; G \rightarrow O$
$A \rightarrow B$	$A \rightarrow E; I \rightarrow M$	
$G \rightarrow L; I \rightarrow M; G \rightarrow O$		

Table 1: Example Communication Schedules

time,  $t$ , to traverse the network. Thus, schedule 1 gives an ordering which completes at time  $4t$  while schedule 2 completes at time  $3t$  resulting in a savings of 25% based simply on the communication schedule. An even greater amount of improvement can be obtained if message ( $A \rightarrow B$ ) is *re-routed* to traverse in a  $YX$  direction. The third column shows this new schedule with the re-routed message denoted as  $A \rightarrow B'$ . The completion time of this new ordering is  $2t$ . Thus, this work 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*. Not only does it encompass routing aspects and path selection issues as discussed in [11, 12], it also determines the order, timing and routing that the messages in the system should be sent. In static scheduling, this information is used in the determination of a fixed communication schedule. In a dynamic environment, *priorities* are determined statically that are used at run-time to arbitrate the ordering of messages transmissions. Thus, the priorities are determined statically and used dynamically resulting in a *hybrid* approach. There have been several studies related to the problem addressed here. Some deal with computer networks and the store-and-forward which occurs when using routing tables [13]. One effort focusing on developing communication algorithms for interconnection networks is the work of Bianchini and Shen[14]. There a ‘traffic scheduling’ algorithm for multi-processor networks was introduced to balance the links of the network. Their work, however, uses a First-Come First-Served, or *FCFS*, approach and does not perform any *scheduling* of the individual message transmissions. Lee and Kim [11] perform path selection in a worm-hole 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. A similar work by Kandlur and Shin [15] also finds dedicated paths. The problem with these techniques is that the dedicated paths not in use can cause other messages to follow longer paths. Additionally, they do not use any type of scheduling which can improve the overall performance. Work by Eberhart and Li [16] does perform a type of communication scheduling on two-dimensional mesh architectures. However, they 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.

This paper uses a model presented by Surma and Sha [17], called a *Collision Graph*, to begin the research on the compile-time analysis and run-time scheduling of point-to-point message transmissions done to reduce the communication overhead. Work done in [17–20] addresses this problem in a restricted way by performing static scheduling using fixed routing. In [21, 22] a dynamic scheduling technique was introduced but it also dealt only with fixed routing. This research expands and improves those efforts by studying not only the scheduling but the routing scheme of the message transfers. Work done in [23–25] begins to consider static scheduling with re-routing for a special case of message traffic. This work considers two models of traffic along with both static and hybrid scheduling techniques. First, static scheduling techniques utilizing re-routing are developed to determine message orderings for these two models of message traffic. The first model is a restricted case where the messages being considered have the same departure time from the nodes in the system while the second model relaxes this constraint and is a general case model. Static techniques, however, use worst-case estimates for computation times and cannot take advantage of the actual performance which could be better than this estimate. Additionally, it is often difficult to obtain exact information *a priori* about the network traffic. Therefore, this paper builds a new scheduling framework by presenting a hybrid approach that works schedules dynamically but utilizes the known information. Just as multi-processor scheduling problems are *NP-Complete* [26] 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.

The organization of this paper is as follows. Section 2 contains background information necessary for the study. Section 3 presents the collision graph model used to develop both static and hybrid scheduling algorithms. Next the static scheduling of message transmissions utilizing re-routing are discussed. Section 5 contains the hybrid model and its algorithms while section 6 contains experiments, analysis, and results. Finally, concluding remarks are presented.

## 2 Background

The starting point for this research is a list of  $N$  messages to be sent by the network nodes. This list is obtained from an analyzing the point-to-point message exchanges required by the assigned tasks. The objective is to find an optimal ordering, timing and routing of the transmissions that reduces the overall processing time.

**Definition 2.1** *A message is defined to be  $M = (m_{edt}, m_S, m_D)$  where  $m_{edt}$  is the estimated departure time of the message;  $m_S$  is the source node of the message, and  $m_D$  is the destination node of the message.*

This work considers single packet messages composed of an arbitrary number of *flow control digits* or *flits* [27]. The multiprocessor architecture used in the experiments is a 10X10 two dimensional mesh [27] with two uni-directional links between processors. A mesh architecture is used because of its widespread popularity in constructing parallel machines. Companies such as Intel [28], Cray, and Symult have constructed parallel processing machines based on the mesh architecture due mainly to its simplicity and scalability [27]. In addition, research machines such as the Stanford DASH [29], Caltech Mosaic [30], MIT J-Machine and others have been created using the mesh. Nodes are attached to all-port routers, and messages are transmitted using a *XY-routing*. A proposed form of *re-routing* is discussed in section 5. Central to the problem is the notion of a network *collision*. Collisions occur when the *paths* of two concurrent message transmissions intersect.

**Definition 2.2** *A path  $P$  is a subset of  $N$  and  $L$ , where  $N$  is the set of processors in the network, and  $L$  is the set of links connecting these processors.  $P$  contains the nodes,  $N_i$  and links  $L_j$  which connect  $m_S$  and  $m_D$  according to some assigned routing scheme.*

**Definition 2.3** *The interval  $T_{m_i} = [a, b]$  is the time interval required by message  $m_i$  to traverse its path  $P_i$ .*

**Lemma 2.1** *Message  $m_i$  has a collision with message  $m_j$  if and only if  $P_{m_i} \cap P_{m_j}$  is not  $\emptyset$  and  $T_{m_i} \cap T_{m_j}$  is not  $\emptyset$ .*

**Proof:** Proof by contradiction. Assume  $P_{m_i} \cap P_{m_j}$  is  $\emptyset$  or  $T_{m_i} \cap T_{m_j}$  is  $\emptyset$ , then the messages traverse different paths resulting in no collision, or, the messages traverse paths at different times. Again no collision would occur. Now, assume that a collision does occur. Both  $P_{m_i} \cap P_{m_j}$  not  $\emptyset$  and  $T_{m_i} \cap T_{m_j}$  not  $\emptyset$  so that the messages collide and are coincident.  $\square$

Two types of message traffic are considered, a *message burst* model and a general case model.

Message ID	Est. Departure Time	Source	Destination
1	$t_1$	(2,1)	(8,8)
2	$t_2$	(3,1)	(7,7)
3	$t_3$	(2,5)	(5,7)
4	$t_4$	(2,1)	(6,3)
5	$t_5$	(2,1)	(5,4)
6	$t_6$	(3,4)	(5,6)
7	$t_7$	(3,1)	(6,4)

Table 2: Example message list

**Definition 2.4** *The Message Burst model is a set of messages,  $M$  such that  $\forall m_j, m_i \in M, m_{i_{edt}} = m_{j_{edt}}$ .*

In addition to being a common case in message passing situations such as process control and data sharing, the burst model is analyzed first due to its less complicated nature. Then, the general case which relaxes this constraint is studied. The work done to obtain *static* communication schedules requires two phases. The *hybrid* technique, presented in section 5, only uses the first phase. To begin, a partial ordering of the messages is determined along with a routing scheme. Then, a scheduler program determines the proper departure time for each message. The timing assumptions are that it takes 1.0 unit of time for a single flit to transfer one hop in the network and message packets are composed of 10 flits. The actual time corresponding to this *unit* is dependent on the size of the flits and the transmission speeds in the interconnect. The hybrid technique uses this ordering and routing information in the determination of priorities to be assigned to each message. The result of the static technique is an actual schedule with appropriate departure times and routing paths.

### 3 Communication Scheduling and Collision Graph

A transformation into a simpler, better defined problem is presented as the first step in solving the *CSP*. This section describes this transformation and provides examples showing the techniques used to solve the new problem. The message burst model of network traffic is analyzed first and is used to present the graph model. This model will be used in subsequent sections to process the general case traffic model.

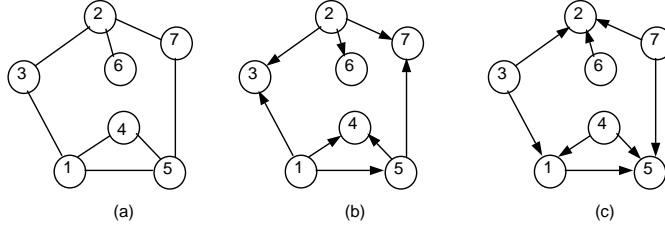


Figure 2: (a) Collision Graph (b) and (c) Two orientations derived from (a)

### 3.1 Collision Graph

Table 2 shows a sample list of  $N$  messages of the type being considered in this research. Central to deriving the communication schedules is a consideration of which messages have a collision. For the burst model, consider all  $t_i$  values to equal some value  $k$ . Thus, two or more messages which collide should not be scheduled simultaneously and a partial ordering is established. To aid in the ordering process, an undirected graph called *Collision Graph*, or  $CG$  is introduced.

**Definition 3.1** *Given a set of messages  $M$ , a Collision Graph is defined as  $G = (V, E)$  where  $V$  is the set of nodes  $v_1, v_2, \dots, v_N$  representing messages  $M_1, M_2, \dots, M_N$ ; and  $E = \{(v_i, v_j) \mid \text{the paths of } M_i \text{ and } M_j \text{ intersect.}\}$*

Using this definition on the messages from Table 2, the  $CG$  shown in Figure 2(a) is produced. Note that while the construction of a particular  $CG$  depends on the routing scheme, in general a  $CG$  is not tied to any type of routing. In this paper,  $XY$  routing is used to construct the initial  $CG$ 's, but the techniques presented could easily be adapted to work with any type of fixed routing scheme. In sections 4 and 5, *re-routing* will be considered as this fixed routing constraint is relaxed.

### 3.2 Edge Orientation

In order to determine the communication schedule from this undirected  $CG$  some specification of which messages precede others must be made. This is accomplished by adding direction to the graph edges. 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  $OCG$ . From the edge orientation of the  $OCG$ , the actual communication schedule can be obtained by first finding the node(s) without any incoming edges. These nodes will be scheduled for transmission in parallel in the first schedule *level*. Next, these nodes and their corresponding edges are removed from the graph, and the next set of nodes without incoming

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

Table 3: Level assignments for the example problem

edges are determined and scheduled in level 2. These messages and edges are then eliminated from the graph and the process continues until all messages have been scheduled. Consider again the example shown in Figure 2(a). Possible edge orientations are shown in Figures 2(b) and (c) with their resulting schedules shown as *Scenario 1* and *Scenario 2* of Table 3. The next section shows that the second schedule is the better of the two. This issue, then, is how to assign edge directions to the *CG* so as to obtain the best or optimal schedule.

### 3.3 Message Traffic Models and Optimal Scheduling

As stated, a key concept in determining the communication schedules is the determination of which messages have path collisions. A collision is caused not only by two messages traversing one or more common links, but they must traverse it at the same time. The previous section built a *CG* based simply on the actual paths without concern for the departure times. Another way of looking at that is if all the messages had the *same* departure time,  $k$ . This is what is referred to as a *message burst* model of message traffic and many applications exhibit this regular type of traffic. Taking into account the differences in departure times or considering these times to be non-uniform is what is called the *general case* model of message traffic. This section considers each case and discusses the optimal way of determining a communication schedule for the general case model.

#### 3.3.1 Burst Model and Optimal Scheduling

The burst model was introduced in the previous section as a *CG* was constructed and two orientations were determined. It was stated that the schedule corresponding to Figure 2(c) is better than the one corresponding to Figure 2(b). Analyzing the Table 3 schedules by again considering each message to take an equal amount of time,  $t$ , to traverse the network yields times of  $14t$  and  $11t$  for the *sum* of the transmission times for all messages for scenario 1 and 2 respectively. Reducing this sum can be accomplished in two general ways. The first is to maximize the number of nodes which can be trans-

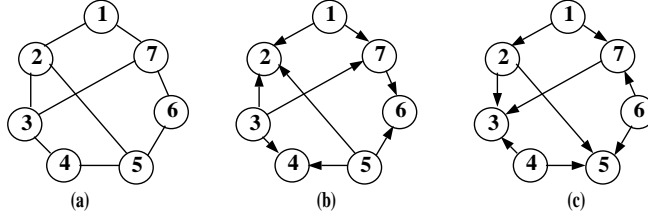


Figure 3: (a) Collision Graph. (b) and (c) Two possible OCGs.

mitted in each level. However doing just this is not enough. Consider the  $CG$  shown in Figure 3(a) and the two orientations shown in (b) and (c). Note that at the first level both scenarios have three messages. However, at the second level scenario 1 has three nodes while scenario 2 only has only two nodes. Thus, a second condition is needed to get an optimal schedule.

This additional condition is to minimize the number of levels. The problem, therefore, requires satisfying both conditions, finding the maximum independent set at each level which produces a solution with the minimum number of levels. 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. Hence, it follows that determining an optimal communication schedule is NP-complete as well.

### 3.3.2 General Case Model and Optimal Scheduling

Consider now the messages listed in Table 2 and let the departure times be 1.0, 1.0, 9.0, 8.0, 4.0, 5.0, 6.0 for  $t_1-t_7$  respectively. For the general case model, the departure times must be considered in addition to the path collisions when determining any communication schedule. Now, just using the value  $t$  for each message transmission is insufficient. A simulation program was written to determine  $C(M)$ , the actual time that a message reaches its destination in the network. The performance metric used throughout the work is the average completion time, or  $ACT$ , as defined in equation 1. This is used in place of the overall schedule length or final completion time because our focus is on the individual message transfers. While this work is interested in having the shortest final completion time it also strives to have as many messages transmitted as soon as possible. Thus, by using the average completion time we can distinguish between two schedules which have the same number of levels or equivalently, the same final schedule completion time.

$$ACT = \frac{\sum_1^N C(M)}{N} \quad (1)$$

Using the schedule from scenario 1 as input into the simulation yielded an *ACT* of 28.29 while the value for scenario 2 is 28.86 making the first scenario a better schedule and a switch from the burst analysis. What this shows is that simply determining a schedule by only considering the routing paths (as was done previously) is insufficient and in this case will not provide the best schedule. (Running the simulation with each scenario with uniform departure times of 1.0 yields *ACT* values of 26.71 and 22.86 for scenarios 1 and 2 respectively). As will be shown in the next section, to solve this problem the message list must be processed in segments which have true collisions (time and path). With each group, *orientation* must be added to the edges in the corresponding collision graph. This problem is called the *Minimum Graph Edge Orientation Problem*, or *MGEOP*.

**Definition 3.2** *The orientation of the edges in the graph  $G = (V, E)$  is an assignment of direction and is computed so as to maximize the number of nodes in each level which can be scheduled concurrently. The level is determined by the function  $w : V \rightarrow Z$  which gives the schedule level of each node according to the partial order of the resultant DAG (Directed Acyclical Graph). In the process the total weight  $\sum_1^N w(v)$  is minimized. Additionally, the average completion time is to be minimized.*

Since the operations of finding a maximum independent set and minimizing the number of levels are both NP-Complete, the *MGEOP* is NP-complete as well.

**Theorem 3.1** *The decision problem of the MGEOP is NP-complete.*

## 4 Static Scheduling Techniques and Re-routing

When developing static communication scheduling algorithms the *burst* model is considered first. While restricted, it is the necessary first step in leading to a general case model of message traffic and its analysis. Additionally, all algorithms presented utilize the concepts that are developed when addressing this model. For the burst model, and later for the general model, algorithms are designed which improve upon baseline approaches.

### 4.1 Message Burst Model Algorithms

The first algorithm developed for this model of message traffic is an implementation of a *FCFS* scheduling strategy and is presented as the baseline algorithm from which all other techniques are measured. Each succeeding algorithm offers varying degrees of improvement over this technique. To get the best possible performance a type of *re-routing* of messages, called *restricted re-routing*, will be performed.

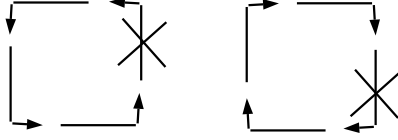


Figure 4: Illustration of allowable routing turns

#### 4.1.1 Restricted Re-routing

The restriction of having all messages travel in the  $XY$  direction is now lifted to allow some message to be routed in a  $YX$  fashion. While  $XY$  routing is well known to be deadlock free, mixing these two types of routing poses the potential for a cycle. In a two-dimensional mesh, there are eight possible turns and two possible cycles [10]. While  $XY$  routing prohibits four of the turns to prevent a cycle, our work only prohibits the two turns shown in Figure 4. Thus, our term for the type of routing that will be used is  $XY$  and *restricted*  $YX$  routing.

#### 4.1.2 FCFS and Re-routed FCFS Scheduling

The  $FCFS$  algorithm when applied to a message burst model simply processes the message list in a top to bottom fashion trying to schedule each message as close to its departure time as possible. The resultant schedule consists of the *time* that the message transmission should be started in the network for best performance. The second column of Table 4 shows the determined schedule and the resulting  $ACT$  for the example messages of Table 2.

The  $FCFS$  algorithm is now improved by implementing this restricted  $YX$  routing and becomes the *Rescheduled FCFS* algorithm. Consider again the message list shown in Table 2. The messages are processed one at a time in a top to bottom fashion with  $XY$  routing being used for each message. If a collision-free path exists for the message, it is scheduled with an  $XY$  route. Otherwise, restricted  $YX$  routing will be considered to see if any improvement can be obtained. If so, the message is re-routed to traverse with a  $YX$  path. Figure 5(a) shows the paths the messages of Table 2 take in the network. Figure 5(b) shows the messages after re-routing has been applied with dashed lines denoting that messages 3, 4 and 7 have been re-routed. The third column of Table 4 shows the resulting communication schedule with re-routed messages denoted with a ' mark. The  $ACT$  value is 22.00 where before it was 26.71, a 17.65% improvement.

Schedule Level	FCFS	Rescheduled FCFS	PS	PSR
1	1, 2	1, 2, 3', 4'	3, 4, 6, 7	3, 4, 6, 7, 5'
2	3, 4, 6, 7	5, 6, 7'	1, 2	1, 2
3	5		5	
<i>ACT</i>	26.71	22.00	22.86	20.00

Table 4: Burst model algorithm results

### 4.1.3 Path Scheduling Algorithms

The next algorithm developed is called the *Path Scheduling*, or *PS*, algorithm. Processing is based on the two criteria for determining an optimal schedule which were laid out in section 3. The first criteria was 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. To accomplish this, the algorithm utilizes a *greedy* [27] technique to determine a set of messages which can be transmitted in parallel. In general, *greedy* techniques make processing choices based on what appears best at the moment. Thus, the *PS* algorithm determines an independent set for each message in the message list and schedules the largest of these sets. Once scheduled, the corresponding nodes of the *CG* are removed along with their edges. If a tie exists among the independent sets determined, the number of edges is used as the *tie-breaker*. Additional ties are broken by an arbitrary process. The number of edges is used because the more edges that can be removed from the *CG* allow additional nodes to have their dependencies removed and can become candidates for scheduling.

Processing continues by finding independent sets for each of the remaining unscheduled messages, scheduling the largest of these sets, removing the nodes and edges from the *CG* and so on until all the messages have been scheduled. The fourth column of Table 4 gives the determined schedule and the *ACT* time. Note that this is an improvement over the *FCFS* technique but it is not as good as the *rescheduled FCFS* technique. The complexity of this algorithm is  $\mathcal{O}(n^4)$  but since it is done at compile-time, the processing requirement is not prohibitive.

In order to outperform the rescheduled *FCFS* technique, the *PS* algorithm is improved by using re-routing. Thus, it becomes the *Path Scheduling with Re-routing*, or *PSR*, algorithm. This technique reschedules the final *PS* schedule by trying to re-route the message(s) starting in the highest level scheduled (the last independent set scheduled). In the example and also shown in column 4 of Table 4, message 5 is in level three and is the first candidate for re-routing. Since it satisfies the requirements, through

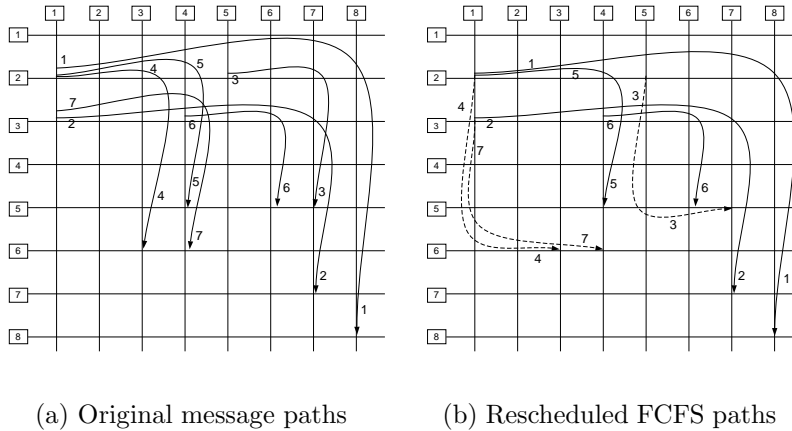


Figure 5: Paths traversed by messages

re-routing it moves up to level 1. Processing continues by looking at the messages in level 2 and attempting to re-route them. In this case no improvement can be obtained so neither message gets re-routed. The algorithm stops after level 2 since the messages in level 1 cannot be moved to a lower level. Column 5 of Table 4 shows the communication schedule and the resulting  $ACT$ . This value is 20.00, an improvement of 12.5% from the  $PS$  technique and a substantial 25.1% over the  $FCFS$  approach.

## 4.2 General Case Model and Algorithms

At this point, the uniform departure time restriction is lifted and the traffic model becomes a general case model. As before, a baseline approach will be analyzed first. Then, algorithms will be developed which repeatedly determine a  $CG$  and apply the techniques used for the restricted case. The basic idea is to apply the techniques of finding a maximal independent set on a subset of the total message set. Because finding a maximum independent set is NP-Complete and this technique processes the schedule incrementally by repeatedly finding such a set, this problem is NP-Complete as well.

### 4.2.1 Baseline Approaches

The baseline algorithms used in studying the general case model are an *As Soon As Possible*,  $ASAP$  approach, a *Rescheduled ASAP* approach, and the  $PS$  and  $PSR$  algorithms from the previous restricted case model study. The  $ASAP$  approach simply sorts the message list by the departure time and then attempts to transmit the messages in the resulting order. The *Rescheduled ASAP* algorithm works exactly the way the rescheduled

*FCFS* approach for the burst model does except that the message list is first sorted. The *PS* and *PSR* algorithms operate exactly as before without concern for the departure times.

#### 4.2.2 Scheduling Communication with Re-routing, SCORE

The algorithm presented to best Schedule the COmmunication with RE-routing is the *SCORE* algorithm. This algorithm improves upon the *PS* and *PSR* techniques by taking into account not only potential path collisions, but also the departure times. Since two messages can share the same path and not have a collision if their departure times are sufficiently far apart, this scheduling algorithm must account for this. This algorithm does so by repeatedly implementing a sorting and an *aging* process. Additionally, message will be re-routed further improvement.

The *SCORE* algorithm works by first sorting the message list by estimated departure time. The schedule of Table 2 has been redrawn in Figure 6(a). Next, a user-input parameter called the *window* is used to select a set of messages,  $S$ , to be processed. In the figure, this window size is 4 and the selected group is boxed. Next, a *CG* is constructed for these messages and the first independent set selected. Then, the remaining nodes in  $S$  which do not get scheduled will either be *aged* or re-routed. Consider node 5. It did not get selected to be scheduled but is eligible to be re-routed since it does not violate the constraints of the restricted  $YX$  routing. Hence, it will be routed in the  $YX$  direction and can process concurrently with nodes 1 and 2. If, however, it could not be re-routed, it would be aged as is the case with node 6.

Aging is a process whereby the estimated departure time gets updated to a later time. For example, if a node  $A$  collides with node  $B$  and  $B$  gets scheduled first,  $A$  cannot begin its transmission until after  $B$  has completed its transmission. The value used for aging is also a user-input parameter but best results occur when this value is similar in value to the length of a standard message. Once this first group of messages have been scheduled, it is eliminated from the message list. Next, the remaining messages are re-sorted and the process repeats using the same window value. This continues until all messages have been scheduled.

### 4.3 Comparison of Scheduling Techniques

Table 5 shows the schedules determined by the baseline algorithms and the *SCORE* algorithm for the example message list of Table 2 with values for  $t_1 - t_7$  as given in

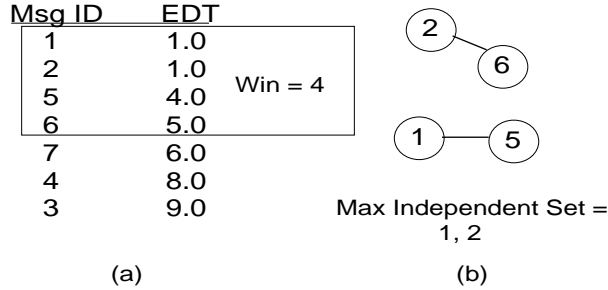


Figure 6: SCORE algorithm (a) applying window (b) building a CG

Schedule Level	ASAP	Rescheduled ASAP	PS	PSR	SCORE
1	1, 2	1, 2, 5, 3	3, 4, 6, 7	3, 4, 6, 7, 5	1, 2, 5
2	5, 3, 6	7, 4, 6	1, 2	1, 2	3, 4, 6, 7
3	7, 4		5		
<i>ACT</i>	28.29	23.14	28.86	25.57	22.86

Table 5: Algorithm results

Figure 6(a). It can be seen that the *SCORE* technique provides the lowest *ACT* of all the algorithms with the percent improvement over the *ASAP* schedule being 28.4%. Thus, the proposed *SCORE* technique performs well in reducing the communication time. Section 6 will provide more complete experiments on all the scheduling techniques presented.

## 5 Hybrid Scheduling

Exact information about the message traffic was used in the previous section to statically determine a schedule which outperforms an *ASAP* approach. However, due to network traffic congestion, computation times differing from the estimates or other fluctuations, such information may not always be known in advance. Therefore, a *hybrid* scheduling technique is proposed which utilizes attributes of both static and dynamic approaches. This new approach will use known information that can be ascertained in advance to determine *priorities* to be assigned to the individual messages. This priority is different from time-based priorities determined for each message as discussed in [31]. Rather, the priorities are based on producing the best overall processing time for the entire message list. At run-time a node will act dynamically as its first requirement by trying to transmit its messages which have the earliest departure time. However, when several messages are available to be scheduled on a particular channel, instead of picking the message with the earliest departure time, the *priority* will be used to do the arbitration.

Message List				ASAP		SCORE		HYCORE	
Msg ID	Est. Dept. Time	Src.	Dest	Msg ID	Compl. Time	Msg ID	Compl. Time	Msg ID	Compl. Time
1	4.0	(3,1)	(8,7)	1	30.0	5	22.0	2'	23.0
2	4.0	(1,1)	(8,4)	2	32.0	7	23.0	4	16.0
3	5.0	(3,1)	(7,7)	3	59.0	8	19.0	8	19.0
4	5.0	(1,3)	(1,5)	4	16.0	2'	23.0	1	30.0
5	6.0	(3,1)	(7,4)	5	66.0	1	37.0	5	30.0
6	6.0	(2,1)	(8,7)	6	40.0	4	28.0	7	36.0
7	6.0	(1,2)	(4,7)	7	46.0	6'	36.0	6'	36.0
8	6.0	(5,5)	(7,7)	8	19.0	3	46.0	3	49.0
ACT values				= 38.50		= 29.25		= 29.88	

Table 6: Comparison of three scheduling techniques

In addition to a priority being assigned, the routing *path* will be determined. This will either be an *XY* or a restricted *YX* path. Consider the example message list shown in the first four columns of Table 6. The *ASAP* and *SCORE* algorithms from the previous section were used to determine routing paths and communication schedules for these messages. Next, a hybrid static-dynamic scheme, called *HYCORE*, was used and its results are given in the last two columns. Note that this technique outperforms the *ASAP* technique. This indicates that having some knowledge of the message traffic leads to an improved schedule. Of course if all information is known *a priori* then a static approach performs the best.

## 5.1 Effect of Variances

Where the real advantage of the hybrid approach occurs is when the actual information about the network deviates from the expected. In this scenario, the statically determined schedule may not be the best schedule because it must account for the worst case estimates. Consider again the message list of Table 6 but with some slight deviations in the departure times. The largest deviation, or *variance*, is 3 time units. Table 7 shows this list and the results for the three scheduling techniques being considered. The abbreviations ‘Act. Dept.’ and ‘Compl’ denote actual departure and completion respectively. The *actual* departure time is when the node actually would be ready to send its messages into the network. These *ACT* values indicate that while the *SCORE* technique is still better than the *ASAP* approach, but not by a great a percentage. It is still better because a variance of only 3 is quite small. The effects of larger variances will be studied in the next section. Applying the *HYCORE* technique yields the best schedule with the results shown in Table 7 and an *ACT* 28.63.

Message List			ASAP		SCORE		HYCORE	
Msg ID	Est. Dept. Time	Act. Dept. Time	Msg ID	Compl. Time	Msg ID	Compl. Time	Msg ID	Compl. Time
1	4.0	2.0	1	22.0	5	25.0	4	16.0
2	4.0	3.0	2	22.0	7	26.0	1	22.0
3	5.0	5.0	3	51.0	8	22.0	2'	22.0
4	5.0	5.0	4	43.0	2'	26.0	5	28.0
5	6.0	6.0	5	58.0	1	40.0	7	31.0
6	6.0	6.0	6	32.0	4	31.0	8	31.0
7	6.0	8.0	7	38.0	6'	39.0	6'	35.0
8	6.0	9.0	8	41.0	3	49.0	3	44.0
ACT values			ACT = 38.38		ACT = 32.25		ACT = 28.63	

Table 7: Effect of variance on scheduling techniques

## 5.2 PRIMAR Algorithm

In this section the application of the *CG* model is used to produce the *PRIority Mapping And Re-routing*, or *PRIMAR*, algorithm. The algorithm works very similar to the *SCORE* algorithm in that both techniques determine the routing paths for the individual messages. The main difference is that while the *SCORE* algorithm determined a schedule, this algorithm determines priorities. Thus, instead of a schedule output with a specified order and start times, the output will be the original messages with a priority field. The first step in the algorithm is to sort the message traffic by estimated departure time. Then a subset,  $S$ , is determined via the same window parameter that the *SCORE* algorithm employed. Next a *CG* is constructed for this set of nodes, a maximal independent set determined, and a priority of 0 assigned to each message in the set. Finding this maximal independent set is done in the same manner as the *SCORE* algorithm.

The messages with priority 0 (highest) and are said to be in  $S'$ . The other nodes in  $S - S'$  have collisions with the nodes in  $S'$ . Therefore, in an attempt to enlarge  $S'$  *re-routing* of the messages in  $S - S'$  is considered. If any of these messages can be successfully re-routed (not violate turn restrictions and not collide with messages in  $S'$ ), they will be assigned priority 0 and will have their *routing flag* changed to  $YX$ . This flag is part of the flit header and each router must be able to interpret it for correct routing. After the nodes with top priority have been determined, they will be eliminated from the graph, and the remaining nodes in the window will be *aged*. Next, the entire list of remaining messages are resorted, another set of messages is selected by using the same window parameter, re-routing is explored, and this set of nodes get priority 1. This continues until all nodes have been assigned a priority. The *PRIMAR* algorithm is shown below.

### Algorithm 5.1 *PRIMAR*

*Input:*  $G=(V,E)$ , and  $M$   
*Output:*  $M(v)_{pri} \forall v \in V$

```

begin
  pri = 0;
  Input window from user;
  I =  $\emptyset$ ;
  repeat until V = 0;
    sort V by estimated departure time, edt;
    limit1 = earliest estimated departure time of a node  $v \in V$ ;
    limit2 = limit1 + window;
    Build  $G_t = (V_t, E_t)$  such that  $V_t = \{v \mid \text{limit1} \leq M(v)_{edt} \leq \text{limit2}\}$ 
      and  $E_t = \{e \mid u \xrightarrow{e} v \text{ and } u, v \in V_t\}$ ;
    Determine the maximum independent set,  $I \subset G_t$ ;
     $\forall v \in I, M(v)_{pri} = \text{pri}$ ;
     $\forall v \in G_t \notin I$ , Explore re-routing for each v
    If re-routing can be done,  $M(v)_{pri} = \text{pri}$  path direction = YX, and add  $M(v)$  to I;
     $\forall v \in (V_t - I) M(v)_{edt} = M(v)_{edt} + \text{age}$ ;
    pri = pri + 1;
    V = V - I;
  end loop;
end algorithm PRIMAR

```

## 6 Experiments

This section presents the experiments performed on both static and hybrid scheduling techniques. Descriptions of the experiments are given followed by the actual results. For the hybrid technique, description of two baseline approaches used for comparison purposes are also given.

### 6.1 Baseline Hybrid Scheduling Techniques

The first baseline technique is a dynamic *ASAP* technique which allows each node to try and start the transmission of its messages at the departure time. If blocked, it will try again once the path is open. If the blocking time is long and several messages are eligible to be scheduled, the one with the earliest departure time gets transmitted first. It also is possible that a message may get blocked somewhere along its path and will only resume transmission once the blocking message completes. The second baseline technique is the *SCORE* static scheduling algorithm presented in the previous section. Simulations in a dynamic environment use the estimated departure time so the static scheduling algorithm determines the ordering and modifies the estimated departure time.

## 6.2 HYCORE Scheduling Technique

The *Hybrid Communication Scheduling with Re-routing, or HYCORE*, technique utilizes the results of the *PRIMAR* algorithm to implement the advantages of both static and dynamic approaches. The *PRIMAR* algorithm is performed first to compute priorities and routing directions for the individual messages. This constitutes the static phase of the technique. At run-time each node selects a message to transmit based on several factors. If a node has only one message ready to transmit, it checks the routing flag and if the appropriate link is available the message is transmitted. However, if the node has several messages that are ready to be transmitted, the priority is used as the arbiter. Thus, the true hybrid nature of the algorithm is exposed. The actual transmission of messages is done dynamically but arbitration is done by using statically determined priorities.

The run-time overhead is minimal in that the priorities are all computed at compile time. Thus, the only run-time cost results from a node checking the priorities of the messages in its queue and routers checking a one-bit routing flag to determine the routing direction.

## 6.3 Results

Experiments were performed on two general sources of message traffic. The first source was derived from message traffic patterns resulting from common applications. LU factorization, matrix multiplication and bitonic sorting are applications which when mapped to multi-processor systems generate specific message passing patterns. Next, randomly generated traffic patterns were analyzed. In both cases, congestion was studied to see how it affected the performance of the various algorithms.

Congestion increases in a system when the *number* of messages increases and when the messages *collide* more frequently in their routing paths. To increase the latter, when studying randomly generated message patterns, a *hotspot index* was used. This parameter gives the frequency that a message destination will be in a particular region. Increasing the number of messages is done by running simulations of lists of various sizes. Because of the randomness of the traffic, 100 simulations of the various list sizes were performed and the results presented are averages of these runs. The performance metric used is the *ACT* for all experiments except for the message burst case. There, the total number of levels is used.

Operation	Msgs Sent	SCORE	ASAP	HYCORE	% Better	SCORE	ASAP	HYCORE	% Better
LU Factorization	42	24.45	28.95	26.02	10.12	38.36	29.55	26.84	9.17
Matrix Multiply	96	30.52	37.96	32.64	14.01	46.41	39.15	34.75	11.24
Bitonic Sorting	200	63.29	81.04	72.72	10.27	94.25	84.71	76.43	7.90
		Variance = 0				Variance = 6			

Table 8: Comparison of scheduling techniques

Hotspot Index	FCFS	PS	Rescheduled FCFS	PSR	Percent Improvement
10%	49.62	47.86	44.34	42.22	15.16
25%	55.99	54.13	48.51	46.16	17.82
50%	87.65	85.80	68.22	66.49	24.70
75%	137.75	135.97	105.40	104.28	24.45
90%	178.17	177.72	136.03	135.21	24.20

Table 9: Burst Model experiments with 30 messages

### 6.3.1 Application-based message traffic

Three common numeric operations, LU factorization, matrix multiplication, and bitonic sorting, were analyzed to determine the message passing that occurs when they are mapped to a two-dimensional mesh architecture. Three scheduling algorithms were used on these traffic patterns and the *ACT* values are given in Table 8. Note that when the variance is 0, a static scheduling approach (the *SCORE* algorithm) performed the best. Further note that the *HYCORE* technique outperforms the *ASAP* approach by up to 14%. Note that the best results occur when the number of messages is moderate. As will be shown when randomly generated message traffic is studied, at few or very large amounts of collisions, the *HYCORE* technique does not perform as well compared to the *ASAP* approach as it does at moderate amounts of collisions. When there is variance of 6, static scheduling no longer works best as it must compensate for worst case times. *HYCORE* still works better than *ASAP* but the percentage is not as great. This is due to the fact that the information used to determine the priorities is not as accurate.

### 6.3.2 Static Scheduling with Randomly Generated Message Traffic

Experiments were performed on randomly generated messages lists for both the static and hybrid scheduling techniques. Message sources and destinations were determined randomly with a normal distribution. For the general case model and for the hybrid experiments, the arrival times were non-uniform and constrained to be in the range of 1.0 to 20.0 to ensure a substantial amount of collisions. Tables 9 and 10 show the results of the experiments performed on the burst traffic model for message lists of size 30 and 50 respectively with the hotspot index varying from 10-90%. Values for four

Hotspot Index	FCFS	PS	Rescheduled FCFS	PSR	Percent Improvement
10%	98.25	94.79	88.92	84.00	15.65
25%	117.63	113.13	101.06	95.82	19.68
50%	207.48	201.97	161.29	156.66	24.56
75%	357.65	355.13	275.53	273.04	23.68
90%	472.73	471.93	363.24	361.70	23.49

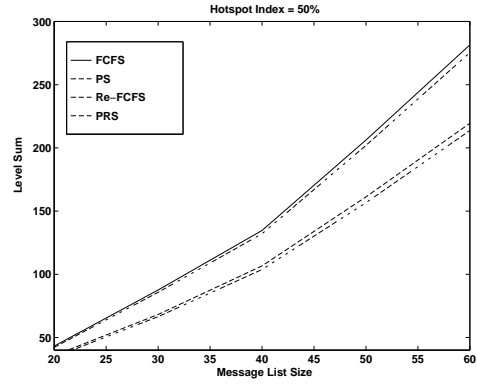
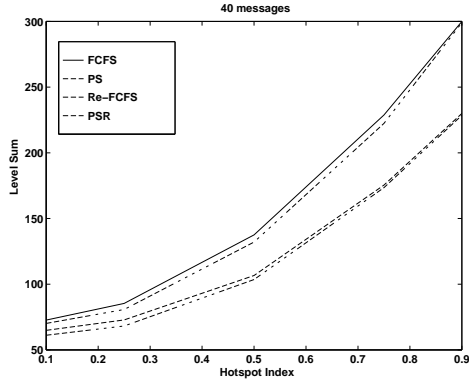
Table 10: Burst Model experiments with 50 messages

algorithms (*FCFS*, Rescheduled *FCFS*, *PS*, and *PSR*) are shown with the last column representing the maximum percent improvement that can be obtained from using these techniques. Note that for each hotspot index in Table 9, the *PSR* performs the best of the four techniques. In addition, note that the percent improvement increases from 15% to over 24%. This improvement indicates that performing communication scheduling significantly improves the performance in a network. Note, however, that as the hotspot index increases past 50%, the *amount* of improvement that can be gained by scheduling the communication begins to diminish. This occurs because as the amount of collisions increases the corresponding *CG* resembles a *clique*[32] with most messages colliding with each other. The effects of increased collisions can also be seen in Table 10. Note again the diminishing of the performance gain after 50% hotspot index.

While the gains do diminish slightly as the collisions increase, the overall improvement is still over 20% even when the hotspot index is 90%. Figure 7(a) shows the relationship between the performance gain and the hotspot index for a 40 message experiment. Figure 7(b) gives the relationship of increased message list size for a fixed 50% hotspot index. The conclusion that can be drawn is that when there are few collisions, there is not much improvement that can be gained as compared with a *FCFS* approach. However, as the amount of collisions increases, improvements in the range of 20% can be obtained.

When performing experiments on the general case model, comparison values for the *SCORE* technique were obtained by executing the *ASAP*, rescheduled *ASAP* and *PSR* algorithms. Table 11 shows the results of these experiments with message list sizes of 40. Note that the improvement ranges from 7.6% to over 22% percent. Table 12 shows the results of experiments performed with message list sizes of 60. Improvements over 20% can again be obtained. As was the case with the message burst model, the amount of improvement does diminish as the collisions gets sufficiently high. This is due to the *CG* beginning to resemble a clique.

To further see the effects of the hotspot index and the number of messages, consider Figure 8. There it can be seen that the *SCORE* technique outperforms the other techniques for all message sizes and hotspot indices studied. However, the amount of



(a) Performance vs. Hotspot Index

(b) Performance vs. Message List Size

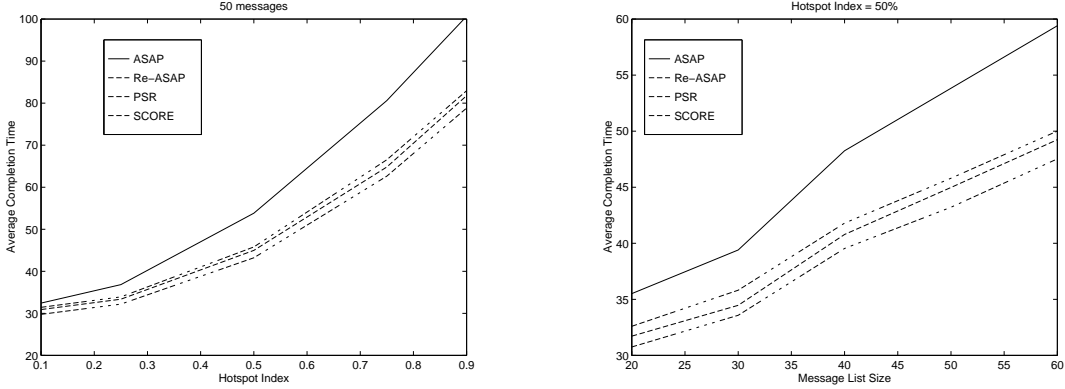
Figure 7: Performance graphs for Burst Model Analysis

Hotspot Index	ASAP	Rescheduled ASAP	PSR	SCORE	% Improvement Over ASAP
10%	30.95	29.39	29.97	28.58	7.66
25%	34.58	31.46	32.15	30.54	11.69
50%	48.25	40.79	41.80	39.54	18.06
75%	69.00	55.42	57.27	53.78	22.05
90%	84.86	68.06	69.66	65.84	22.42

Table 11: General Case model experiments with 40 messages

Hotspot Index	ASAP	Rescheduled ASAP	PSR	SCORE	% Improvement Over ASAP
10%	35.01	33.08	33.35	31.52	9.98
25%	40.09	36.12	36.58	34.68	13.48
50%	59.40	49.24	50.01	47.53	19.99
75%	92.53	73.75	74.53	70.97	23.30
90%	115.04	93.85	94.85	90.42	21.40

Table 12: General Case model experiments with 60 messages



(a) Performance vs. Hotspot Index

(b) Performance vs. Message List Size

Figure 8: Performance graphs for General Case Model Analysis

improvement that can be obtained is greatest at moderate amounts of messages and hotspot indices. The amount of improvements is over 20% when the amount of collisions is of a moderate amount.

### 6.3.3 Hybrid Scheduling with Randomly Generated Message Traffic

Results of experiments utilizing the *HYCORE* technique for randomly generated messages lists of size 20-60 are shown in Table 13. The first 5 columns show the results of various algorithms with zero variance. The % Better column reflects the amount of improvement that the *HYCORE* technique provides over a straight *ASAP* approach. From this table, note that the static *SCORE* algorithm performs best followed by the *HYCORE* technique. The amount of improvement that the *HYCORE* provides over the *ASAP* approach reaches over 20%. What is also interesting is that the *HYCORE* technique compares somewhat favorably to the static technique. The influence of increasing the amount of variance can be seen by comparing the results of the first 5 columns with the last 4 columns shown in Table 13. While static scheduling severely deteriorates, note

Num Msgs.	SCORE Time	ASAP Time	HYCORE Time	% Better	SCORE Time	ASAP Time	HYCORE Time	% Better
20	33.80	39.09	35.24	9.84	51.25	39.42	36.25	8.04
30	35.57	46.18	37.43	18.46	53.30	46.18	39.70	14.03
40	41.55	54.10	43.52	19.56	61.68	53.90	45.41	15.75
50	46.93	62.71	49.07	21.75	68.87	62.16	51.10	17.79
60	52.41	69.91	55.60	20.47	77.28	70.20	58.56	16.58
Variance = 0					Variance = 6			

Table 13: Results for 100 iterations of messages with with 50% hotspot index

Hotspot Index	SCORE Time	ASAP Time	HYCORE Time	% Better	SCORE Time	ASAP Time	HYCORE Time	% Better
10%	33.66	34.90	38.01	8.18	49.75	35.75	38.25	6.54
25%	36.47	37.54	43.61	13.92	52.56	39.15	44.31	11.65
50%	46.91	49.28	61.79	20.25	69.69	52.50	63.04	16.72
75%	65.92	68.86	90.14	23.61	102.23	74.17	92.47	19.79
90%	81.17	87.50	109.49	20.08	129.76	91.83	112.58	18.43
Variance = 0					Variance = 6			

Table 14: Results for 100 iterations of message list size = 50

that the *HYCORE* and *ASAP* techniques are not overly effected by the increase in variance. This is due to the variance being evenly distributed over the message list causing some messages to be transmitted sooner and some later. What this variance does do is reduce the amount of improvement of the *HYCORE* approach over the *ASAP* approach.

The effect of increased congestion can also be seen from Table 13. The amount of improvement that can be obtained increases as the message list size increases. Since the more messages being transmitted causes increased collisions, it is apparent that the *HYCORE* technique performs better in the presence of congestion. However, note that the improvement begins to diminish after 50 messages. Again this is due to the *CG* resembling a *clique*.

To further see the effect of congestion, consider Table 14. This table shows that the *HYSTAD* technique performs moderately better than the *ASAP* approach at low hotspot indices. However, this improvement quickly increases to over 20%. Again, the improvement does diminish at very high amounts of congestion. Considering the effect of variance, it can be seen that again the *SCORE* algorithm performs poorly while the difference between the *HYCORE* and *ASAP* begins to moderate.

## 7 Conclusions

While working on massively parallel systems, it was found that the communication overhead greatly impacted the system performance. To reduce this overhead, techniques were

derived to perform scheduling at a very low level by considering individual message transfers. In a static environment, the *SCORE* algorithm was developed and shown to provide improvement of over 20% as compared to a baseline approach. This algorithm uses the newly developed *Collision Graph* to determine both a schedule and the routing scheme for the message transmissions. Since the information required by static techniques is not always readily available or accurate, the *HYCORE* technique was developed to operate dynamically but with knowledge obtained by a compile-time analysis. Again, improvements of approximately 20% were obtained. Together, the Collision Graph and this scheduling strategy form a framework for which continued study into communication scheduling can be done.

## References

- [1] H. Kasahara and S. Narita, "Practical multiprocessor scheduling algorithms for efficient parallel processing," *IEEE Transactions on Computers*, vol. c-33, pp. 1023–1029, November 1984.
- [2] A. A. Munshi and B. Simons, "Scheduling sequential loops on parallel processors," *SIAM Journal of Computing*, vol. 19, pp. 728–741, August 1990.
- [3] H. El-Rewini, T. G. Lewis, and H. H. Ali, *Task Scheduling in Parallel and Distributed Systems*. Englewood Cliffs, NJ: Prentice Hall, 1994.
- [4] 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.
- [5] 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.
- [6] K. R. Baker, *Introduction to Sequencing and Scheduling*. New York, NY: Wiley, 1974.
- [7] R. W. Conway, W. L. Maxwell, and L. W. Miller, *Theory of Scheduling*. Reading, MA: Addison Wesley, 1967.
- [8] P. K. McKinley, Y. Tsai, and D. F. Robinson, "Collective Communication in Wormhole-Routed Massively Parallel Computers," *IEEE Computer*, vol. 28, pp. 39–50, December 1995.

- [9] W. J. Dally and C. L. Seitz, “The torus routing chip,” *Journal of Distributed Computing*, vol. 1, no. 3, pp. 187–196, 1986.
- [10] L. M. Ni and P. McKinley, “A survey of wormhole routing techniques in direct networks,” *IEEE Computer*, vol. 26, pp. 62–76, February 1993.
- [11] 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.
- [12] W. J. Dally and C. L. Seitz, “Deadlock-free message routing in multiprocessor interconnection networks,” *IEEE Transactions on Computers*, vol. C-36, pp. 547–553, May 1987.
- [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, August 1996.
- [17] D. R. Surma and E. Sha, “Application specific communication scheduling on parallel systems,” in *Eighth International Conference on Parallel and Distributed Computing Systems*, pp. 137–139, September 1995.
- [18] D. R. Surma, S. Tongshima, and E. Sha, “Optimal Communication Scheduling based on Collision Graph Model,” in *1996 International Conference on Acoustics, Speech and Signal Processing*, vol. VI, pp. 3319–3322, May 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, “Collision Graph based Communication Reduction Techniques for Mesh Networks,” in *14th International Conference on Computer Applications in Industry and Engineering*, (Las Vegas, NV), pp. 9–12, November 2001.

- [21] D. R. Surma and E. Sha, "Hybrid static-dynamic communication scheduling for parallel systems," in *ACM Symposium on Applied Computing*, pp. 374–379, February 1997.
- [22] D. R. Surma and E. Sha, "Collision Graph based Communication Scheduling for Parallel Systems," *International Journal of Computers and their Applications*, vol. 5, pp. 11–22, March 1998.
- [23] D. R. Surma and E. Sha, "Efficient Communication Scheduling with Re-routing based on Collision Graphs," in *International Symposium on High Performance Computing Systems*, pp. 483–492, July 1997.
- [24] D. R. Surma and E. Sha, "SCORE: An Efficient Technique to Reduce Congestion in Parallel Systems," in *Tenth International Conference on Parallel and Distributed Computing Systems*, pp. 198–203, September 1997.
- [25] D. R. Surma, "Communication Reduction Techniques for Torus Networks based on Collision Graphs," in *13th International Conference on Parallel and Distributed Computing Systems*, (Las Vegas, NV), pp. 262–267, August 2000.
- [26] 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.
- [27] K. Hwang, *Advanced Computer Architecture, Parallelism, Scalability, Programmability*. New York, NY: McGraw-Hill, 1993.
- [28] I. Corporation, *Paragon XP/S Product Overview*. 1991.
- [29] D. Lenoski and et al, "The Stanford DASH multiprocessor," *IEEE Computer*, vol. 25, pp. 63–79, March 1992.
- [30] C. Lutz and et. al, "Design of the Mosaic Element," in *Proceedings of the MIT Conference on Advanced Research in VLSI*, (Dedham, MA), pp. 1–10, Artech Books, 1984.
- [31] J. Li and M. W. Mutka, "Priority based real-time communication for large scale wormhole networks," in *International Conference on Parallel Processing Symposium*, pp. 433–438, April 1994.
- [32] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. New York, NY: McGraw-Hill, 1990.