ABSTRACT

While research on the static scheduling of computational tasks for parallel systems has been ongoing for years, most work does not consider the communication costs nor does it consider the network congestion. A new static scheduling technique is presented which focuses on the communication overhead inherent in parallel processing systems. This paper builds a framework based on a newly developed graph model called a Collision Graph to study this problem. Using this model, algorithms are developed which can be embedded into existing static scheduling methods to improve their performance. The scheduling of cyclic data flow graphs was shown to be improved significantly as this technique was applied to the recently developed cyclo-compaction scheduling algorithm.

1. INTRODUCTION

In DSP systems requiring high throughput and having real-time deadlines, application specific multi-processor designs are increasingly being used. For example in spacecraft image acquisition, filtering and compression can be done before transmission to ground stations. However, the data volume associated with such images requires a significant communication time between processors. While using multiple processors can reduce the computation time, the communication overhead required in the system can be substantial. Therefore, when considering the overall processing time both components, computation time and communication overhead, must be considered. When considering transmission scheduling the dynamic approach has the disadvantage of not providing the designer with a guarantee that the time constraints can be satisfied. The problem with most existing static scheduling techniques is that they do not consider the intercommunication time or even the network congestion which may prevent the designer from finding a realistic solution. This paper presents a new static scheduling technique which focuses on the communication overhead introduced by assigning tasks to different processors. By introducing a new model known as a Collision Graph (CG), a new scheduling framework is developed to deal with this problem.

The most time consuming parts of recursive DSP applications can be represented by data flow graphs (DFGs) where the nodes correspond to tasks and the edges indicate data dependencies. A data dependency occurs when data must be carried to future iterations. When scheduling these cyclic DFGs it is possible to pipeline loops [1] in such a way as to increase system throughput. Communication costs are not usually considered in classical scheduling methods [2] nor are most of the techniques developed for parallel compilers [3, 4]. Tongsima, Passos and Sha presented new static scheduling techniques [5] which do not include a discussion of network traffic congestion. The cyclo-compaction scheduling algorithm, CCSA, presented by Tongsima [6] considers communication costs and loop-carried dependencies while performing loop pipelining to optimize a schedule. In [7] the cyclo-compaction scheduling algorithm was extended to handle network congestion. However, this static scheduling technique utilizes a simple first-come first-served (FCFS) approach when messages are competing for the same communication link. This work significantly improves such an algorithm by applying the new framework based on the CG model to arbitrate who gets access to the communication links.

To see the importance of communication scheduling, consider the task graph of Figure 1(a). Figure 1(b) gives one possible assignment of this graph to a network of four processors connected in a two-dimensional mesh arrangement. Tasks assigned to the same processor incur no com-
communication delays since no network communication is required. However, node 1 must send messages to nodes 2, 3, and 4, and node 4 must send a message to node 3. Since there is only a single bidirectional link between each node, network collisions occur. In Figure 1(c) two possible orderings of the message traffic are given. In these orderings, messages appearing on the same line can be transmitted simultaneously. Assuming each message takes the same amount of time, \( x \), to traverse the network then scenarios 1 and 2 give orderings which complete at time \( 3x \) and \( 2x \) respectively. Thus, it is possible to reduce the communication overhead by 33% based on the message schedule.

## 2. PROBLEM DESCRIPTION AND GRAPH MODEL

The scheduling process begins with what Tongsima calls a communication sensitive data-flow graph (CSDFG). Figure 1(a) is an example CSDFG where nodes correspond to processing tasks and edges denote messages being sent between these tasks. Communication costs associated with each edge is one unit unless marked otherwise. These costs are incurred for each link traversed in the network. Obviously, then, if two corresponding tasks are assigned to the same node, the communication time is zero. The bar lines on the arcs between nodes \( I \) and \( A, D \) and \( A, D \) and \( X \) and between \( E \) and \( X \) denote the distance between iterations or an inter-iteration dependency. As a starting point for the cyclo-compaction scheduling algorithm, well-known techniques similar to list-scheduling [8] are used to produce the start-up schedule of Figure 2(a). The CCSA then begins by trying to implicitly retime the graph by shifting the current iteration boundary down by one control step. This is analogous to moving the first row of the schedule table down to the bottom of the current iteration. The purpose of doing this rescheduling is to find a better position for those nodes so that the length of the schedule table will be decreased [7]. In other words, the goal is to find spots in the table where these row members can be placed without violating constraints imposed by direct dependencies and also by the network congestion constraints. By moving the row, what actually is being done is inserting the next iteration of the row members into the schedule table. Thus, the first iteration of those elements must go into a preamble or prologue.

![Figure 1](image1.png)

![Figure 2](image2.png)
tion for X then the entire row is skipped. Now, if a valid arrangement for these nodes but with a different location for A exists then the row can be shifted and the schedule shortened. Finding the arrangement of valid locations for the two nodes involves using the new CG model. The definition of a CG is as follows:

**Definition 1** Given a set of messages, a Collision Graph is defined as \( G = (V, E) \) where \( V \) is the set of nodes \( v_1, v_2, \ldots, v_N \) representing messages \( M_1, M_2, \ldots, M_N \); and \( E = \{(v_i, v_j)\} \) the paths of \( M_i \) and \( M_j \) intersect.

Note that the CG has a one-to-one correspondence between the number of nodes and the number of messages being transmitted in the network. Also note that this model is not restricted to any topology or routing scheme. The particular network architecture and routing strategy are only used when the edge determination is made.

In Figure 3 three CGs are given for the first iteration in the cyclo-compaction scheduling algorithm. Figure 2(d) shows the graph after it has been retimed. These collision graphs show the four messages that need to be transmitted. The notation \( I_1-A_3 \) denotes that a message from node \( I \) on processor 1 is sending a message to node \( A \) on processor 3. These CGs are for the case where \( A \) has already been assigned to PE3 at control step 5. In part (a), the attempt is being made to assign \( A \) to PE4. However, from the CG it is found that it has a collision with the message \( H_3-G_3 \). This is a collision because in the two dimensional network arrangement with XY routing it is not possible for these two messages to proceed simultaneously. If \( H_3-G_3 \) proceeds first then the link is not available for the \( D_3-X_4 \) transmission. Part (b) shows that \( X \) cannot be assigned to PE2 because there will be a collision on link 2-4. Part (c) shows that assigning \( X \) to PE1 will not work either due to multiple collisions. Consequently, these CGs show that assigning \( A \) first results in there not being a valid location for \( X \).

What the original cyclo-compaction scheduling algorithm would do at this point would be to skip this row and continue with the next iteration. The first pass of this algorithm is shown in Figure 2(b). Note that the schedule length is the same. The only difference is that a preamble exists and that the next iteration of \( A \) and \( X \) has been inserted into the schedule table. Now, by embedding the usage of the CG the algorithm can arrive at valid locations for these two nodes. To do this, more CGs would be constructed with node \( A \) in different locations. Figure 2(c) shows the results of using this new procedure, called the modified cyclo-compaction scheduling algorithm, MCCSA, to find a valid arrangement of the two nodes. It is the case that this arrangement is the best location which can be obtained. The schedule is shortened by one control step and the communication costs of the schedule is minimized.

The process just outlined continues for every step in the modified cyclo-compaction scheduling algorithm. Doing this finds optimal locations for the nodes in each step of the algorithm. Finding these locations from using the CG is a difficult problem. For each step, many collision graphs must be constructed and analyzed. In fact, a CG cannot be constructed for every possible valid location combination for the nodes. We call the problem of determining the best node placement from the CG the **Best Node Placement Problem (BNPP)**.

### 3. Algorithm and Description

The modified cyclo-compaction scheduling algorithm, MCCSA is now presented. This algorithm uses the cyclo-compaction scheduling algorithm of Tongsima [6] but with modifications to incorporate usage of the collision graph. The main difference between these algorithms is the em-
bedded procedure Check-collision which is used in the MCCSA.

Algorithm 1

```
1 input : G = (V, E, T, d, e); # forwarding buffers, and # of pipelines
2 Output : shortest schedule table S
3 begin
4 S := Start-Up-Schedule(G); Q := S;
5 for ; i := 1 to |V| step 1 do
6 (G, S) := Rotate(G);
7 if length(S) < length(Q) then Q := S; fi
8 od
9 proc Cyclo-compact(G, S) =
10 /* extract nodes from the table */
11 N := Deallocate(S);
12 /* operate retiming technique on the nodes */
13 R := Retime(G, N);
14 S := Re-schedule(G, S, N);
15 return(G, S)
16 proc Re-schedule(G, S, N) =
17 /* Tor each v ∈ N do
18 /* * get a minimum required control step */
19 cs_max :=
20 max([parents(v).cs + c[v], cs_max := length(S));
21 /* * get an upper bound schedule length */
22 cs := cs_max;
23 /* * find a minimum available legal control step to schedule v */
24 while (cs < cs_max) ∧ ((legal(cs, v, G) = false)
25 ∨ (|pad := entry(S, cs)| = available)) do
26 cs := cs + 1;
27 od
28 Check-collision(G, S, N);
30 od
31 return(S)
32 proc Check-collision(G, S, N) =
33 /* Construct CG’s V nodes ∈ N
34 Compare node position information and communication
35 costs for valid CG’s
36 Select best node locations
37 if old node locations are better than new locations
38 then do not change node locations
39 else assign to new locations
40 fi
41 end
```

The algorithm proceeds in the manner demonstrated by the example of Figure 2. Determination of the start-up schedule begins the process. Next the cyclo-compact procedure is called which acts as the engine of the algorithm. The nodes to be retimed are selected and the rescheduling process is started. Usage of the collision graph is found in procedure check-collision. In this procedure the various CGs are constructed and the best node locations are selected. It should be noted that a result of the procedure could be that no new locations can be found which improves the schedule table. Consequently, the schedule table is left unchanged. As can be seen from the example, using the CG to select node positions reduced the schedule length by one step after the first iteration. From experiments performed, it was found that employing this technique always yields a schedule at least as short as the one found from the FCFS approach with significantly smaller number of cyclo-compaction iterations. This demonstrates that embedding the use of the CG model into the cyclo-compaction scheduling algorithm results in a major improvement in the determination of the final schedule.

4. CONCLUSION

The communication overhead inherent in parallel systems significantly impacts the overall performance. When considering the static scheduling of messages in a network, it was shown that taking into account this overhead can lead to the determination of better scheduling algorithms. The development and use of the Collision Graph was shown to be useful in this process. The developed techniques can either be used as stand-alone routines or as embedded procedures in existing scheduling algorithms. An example of the latter process was shown when the cyclo-compaction scheduling algorithm was significantly improved by embedding the usage of the CG into it. The development of the CG and its importance to communication scheduling, as well as its applicability to the newly presented cyclo-compaction scheduling algorithm, constitutes a significant contribution to the scheduling of DSP applications.

REFERENCES


