

# Mapping to Reduce Contention in Multiprocessor Architectures\*

Loren Schwiebert and D. N. Jayasimha  
Department of Computer and Information Science  
The Ohio State University  
Columbus, Ohio 43210-1277  
Email: {loren,jayasim}@cis.ohio-state.edu

## Abstract

*Reducing communication overhead has been widely recognized as a requirement for achieving efficient mappings which substantially reduce the execution time of parallel algorithms. This paper presents an iterative heuristic for static mapping of parallel algorithms to architectures. Special attention is given to measuring and reducing channel contention. Experimental results are used to show the effects of channel contention for packet-switched networks and the improvement realized by our heuristic. We also present preliminary results for wormhole-routed networks.*

**Keywords:** multiprocessor architectures, channel contention, mapping problem, directed acyclic task graph, critical edges.

## 1 Introduction

It is well known that communication overhead can significantly increase the execution time of parallel algorithms. The mapping problem explores the difficulties of scheduling the computation and communication components of the parallel algorithm on the processors so the execution time is minimized. In practice, a parallel architecture is not completely connected, so the overhead associated with communicating between processors is not uniform. Additionally, channel contention can increase the transmission time of messages. These factors can be used by the mapping heuristic to reduce the execution time.

Finding an optimal mapping is NP-complete in general [3], so researchers have focused on designing tractable heuristics that generate efficient mappings. The heuristic requires an appropriate model of the computation and the communication components of the parallel algorithm and the communication properties of the parallel architecture. The heuristic also needs an appropriate goal, since an inappropriate goal may lead to an inferior mapping.

Approaches to the mapping problem can be divided between compile-time (static) and run-time (dynamic) mapping methods. The disadvantages of run-time mapping are the reduced efficiency that results from making decisions without global information and the overhead of simultaneously mapping and executing the algorithm. These disadvantages are avoided by using compile-time mapping. The drawback to compile-time mapping is the difficulty of determining the execution time of branches and loops. A compile-time approach is adopted.

## 2 Previous Work

Early research results [1, 6] assume each task is assigned to a separate processor. Also, the heuristics focus on minimizing the total communication overhead. This goal does not necessarily result in a good execution time, because the arrival time of some messages may have to be delayed so that more critical messages can be expedited. Unreasonable restrictions are also placed on the communication model of the architecture or algorithms. For example, [1] assumes that all communications transmit the same amount of data and ignores channel contention. In [6], channel contention is explicitly measured, but only when all the communication occurs in phases.

In contrast, recent research results [2, 5, 8, 9] focus on minimizing the execution time which is also the goal of this paper. Additionally, no restrictions are placed on the number of tasks. However, various inadequate assumptions are made about the communication properties of the parallel architecture. Infinite bandwidth between the processors is assumed in [9]. In [5], the mapping heuristic uses the proximity of the clusters to reduce contention. However, the contention is not measured, so the heuristic is unable to vary the mapping to avoid contention. Channel contention is measured in [2, 8], but the communication model is inaccurate and so the contention is not accurately determined. This leads to incorrect decisions. For example, in [2] it is shown that over a third of their mappings were worse when contention was considered.

\*Published in the IEEE 7<sup>th</sup> International Parallel Processing Symposium - April, 1993

### 3 Problem Statement

The parallel architecture is modeled as a set of  $p$  homogeneous processors connected by a network with local memory on each processor. A hypercube topology is assumed, although the mapping heuristic can be used for any topology. The network consists of a fixed number of bi-directional channels that directly connect neighboring processors. This paper uses packet-switched networks and provides preliminary results on wormhole-routed networks.

The parallel algorithm is represented by a directed acyclic graph (DAG),  $G = (V, E)$  where  $V$  is the set of nodes (or tasks) and  $E$  is the set of directed edges. The number of nodes in the graph is  $v = |V|$  and the number of directed edges is  $e = |E|$ . Directed edge  $e_{ij}$  represents a message from node  $n_i$  to node  $n_j$  with a communication cost of  $c_{ij}$ . The communication cost is given in number of packets for packet switching and number of flits for wormhole routing. Node  $n_i$  has an execution time of  $x_i$ . No restrictions are placed on either the communication costs of the edges or the execution times of the tasks.

A general model requires an arbitrary number of tasks. An effective means of handling this requirement is via a two-step approach. First, a clustering algorithm is used to cluster the tasks on an unbounded number of processors and these clusters are merged until there is exactly one cluster for each processor in the target architecture. After clustering, the mapping problem is simplified to choosing the appropriate processor for each cluster. This two-step approach yields a substantial improvement over the one-step unclustered approach [5]. Although the tasks are in clusters, each task maintains its own communication and computation properties. It is assumed that the clustering step has already been performed. Heuristic algorithms for clustering are given in [4, 5, 10].

The clustering algorithm imposes a total ordering on the tasks in a cluster. This total ordering allows for more efficient execution of our algorithm. The tasks are numbered based on their level in the DAG.

Channel contention can have a significant effect on the execution time of the parallel algorithm. If infinite channel bandwidth is assumed when evaluating a mapping, not only is an unrealistic estimation of the execution time made, but also an inefficient mapping may be chosen over a more efficient one.

In Figure 1, the clusters have been mapped to a four processor hypercube. All tasks in a column are in the same cluster. The horizontal axis gives the processor of the corresponding cluster and the vertical axis gives the time. Each rectangle represents a task with the associated task number inside the rectangle. The execution time of a task is the length of its rectangle. Each directed edge is a commu-

nication message. The communication cost of each edge,  $c_{ij}$ , indicates the time to transmit the message along a single channel without contention.

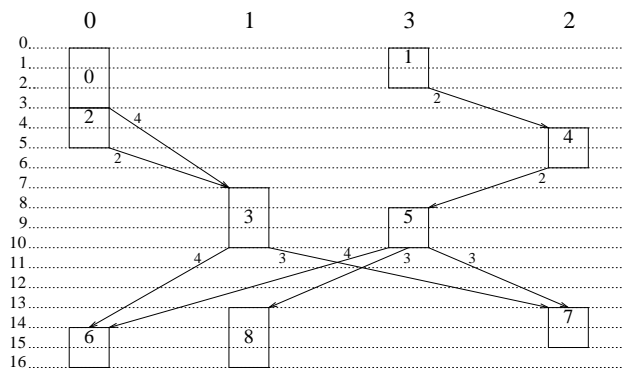


Figure 1: **Examples of Channel Contention**

Channel contention occurs for three reasons:

- **Finite Bandwidth on any Channel.** Consider the message from Task 0 to Task 3 and from Task 2 to Task 3. Both messages require the same channel, however, Task 0 is still using this channel when Task 2 is ready to begin transmission. The message from Task 2 to Task 3 must be delayed.
- **A Limited Number of Channels at each Processor.** There are three messages being transmitted by Task 5. Since this processor has only two channels, one of the messages must be delayed.
- **Limited Paths through the Interconnection Network.** Task 3 and Task 5 are sending messages at the same time. The message from Task 3 to Task 6 can begin transmission immediately. However, the message from Task 3 to Task 7 must either wait until the message from Task 3 to Task 6 has completed transmission, or be routed through the processor that executed Task 5. Since Task 5 is currently using all its channels, the message must be delayed in either case. This situation could also arise because of restrictions imposed by the routing algorithm.

Since the mapping algorithm given in [9] and our mapping algorithm both use the same computation model and assume clustering as a prior step, simulation results using the two algorithms are compared.

### 4 The Algorithm

Like many other researchers, our heuristic uses the critical path while attempting to improve the current mapping. *A critical edge is an edge that increases the execution time*

of the parallel algorithm if the transmission of the edge is delayed. The source and destination tasks of a critical edge are critical tasks. A cluster with at least one critical task is a critical cluster.

The heuristic, described in Sections 4.1 – 4.7, uses an iterative approach to find a mapping. The channel contention is measured to accurately determine the execution time of a mapping. Then the set of critical edges is updated using contention information and a new mapping is generated. The algorithm uses a function,  $distance()$ , which returns the length of the shortest path between any two processors. For complexity analysis, it is assumed that this function can be computed in constant time.

#### 4.1 Calculate the Communication Matrix

Since the communication among clusters does not change during the mapping, the results of this step are reused for each iteration. Each directed edge has a sending task and a receiving task that are assigned to different clusters. The amount of communication between different clusters is stored in a  $p \times p$  matrix. After all the edges have been processed, *each element in the matrix corresponds to the total message traffic between the two specified clusters.* The amount of communication for each cluster is also totaled. The matrix is used to map the clusters that are not critical. Since all edges are examined once, and the matrix must be initialized, the time complexity is  $O(e + p^2)$ .

#### 4.2 Map the Clusters to the Processors

A mapping is first generated on a completely connected architecture. This is done using the identity mapping, that is, cluster  $i$  is mapped to processor  $i$ . The time complexity of this step is  $O(p)$ . Information from the mapping on the completely connected architecture is used as input to the first iteration of our heuristic. The remaining iterations use the output from the previous iteration.

The critical edges have a direct impact on the execution time. The heuristic tries to find the mapping which has the least total distance between critical task pairs. This minimizes the execution time if there is no channel contention and reduces the chance of contention because fewer channels are required.

The total distance between critical task pairs is calculated for the current mapping. This is done in  $O(p^2)$  time. The algorithm then swaps each critical cluster with every other cluster and recalculates the total distance. The two clusters are swapped back if the total distance has increased. This sequence of exchanges does not guarantee that the total distance is reduced, however, this approach works well in practice. In the worst case, all clusters are critical clusters, so  $O(p^2)$  different mappings are tried.

Only two clusters are swapped each time, so the total distance for each new mapping can be computed in  $O(p)$  time. The time complexity of this step is  $O(p^3)$ .

The remaining clusters must be mapped to prevent delays in the non-critical edges from increasing the execution time of the parallel algorithm. It is not clear how this can be done efficiently; the method used is to try to minimize the channel contention by using fewer channels for communication. This is of secondary importance to the critical edge assignment, so these clusters are assigned only to available processors.

The mapping of the non-critical clusters is done by repeatedly choosing the unassigned cluster with the most total communication. It is placed on the available processor which minimizes the total distance of the messages between this cluster and the clusters that are already assigned. At most  $O(p)$  clusters must be assigned using this method, trying at most  $O(p)$  different processors. Checking the total distance requires  $O(p)$  time, so the time complexity of this step is  $O(p^3)$ .

#### 4.3 Compute the Execution Windows

The execution window is the time range in which a task must begin execution to avoid increasing the execution time of the parallel algorithm. The size of the execution window is the difference between the earliest and latest starting times. The execution window is calculated iteratively from the immediate predecessors and successors of each task. The earliest starting time is calculated from the start node(s) of the DAG as follows:

Let  $n_i, n_j, n_k$  denote tasks.  $n_i$  is any task that sends a message to task  $n_j$ ;  $n_j$  is the task whose earliest starting time is being computed; and  $n_k$  is the previous task in the same cluster. Using the definitions from Section 3, the transmission time of a message, ignoring channel contention, is:

$$t_{ij} = c_{ij} + distance(n_i, n_j) - 1$$

The earliest starting time of task  $n_j$  is:

$$earliest(n_j) = \max(earliest(n_k) + x_k, \max_i(earliest(n_i) + x_i + t_{ij}))$$

The latest starting time is similarly calculated from the exit node(s). The time complexity of this step is  $O(v + e)$ .

#### 4.4 Measure the Channel Contention

Figure 1 illustrates the reasons there could be channel contention. Measuring the contention allows a more realistic execution time for the mapping to be found. Since

the execution time of the mapping is more accurately determined, inferior mappings can be correctly rejected. In order to simulate message transmission, it is necessary to make several assumptions about how messages are routed through the interconnection network. These are typical assumptions made by other researchers, a complete list can be found in [7]. Fully-adaptive minimal routing is used for packet switching and *e-cube* routing is used for wormhole routing.

In order to evaluate the impact of channel contention on the mapping, it is necessary to consider the traffic on the interconnection network during the entire execution of the algorithm. This execution is modeled with an event-driven simulator. An event occurs whenever a task completes execution (and places messages on the channel queues), a packet enters the network (and utilizes channels), or a packet leaves the network (and frees channels). The execution windows are also updated with channel contention information.

For complexity analysis, a  $p$ -processor hypercube which has a diameter of  $\log p$  is assumed. The time complexity of this step is  $O(v + e \log^2 p)$ , assuming that the maximum length of any message is fixed. Our measurement of the channel contention is much more accurate than the method proposed in [2], while still achieving a lower time complexity.

#### 4.5 Update the Critical Edge Set

All critical edges are transmitted between tasks with an execution window of size zero. The proof can be found in [7]. This simplifies the problem of determining the critical edges. The set of critical edges depends on the mapping. Both the distance between the source and destination processors and channel contention can increase the transmission time of an edge. It is necessary to recompute the critical edges for each mapping. The time complexity of this step is  $O(e + p^2)$ .

#### 4.6 Calculate the Execution Time

The execution time is calculated while measuring the channel contention. The goal of the mapping heuristic is to minimize the execution time of the parallel algorithm. Within our model, it is impossible to produce a mapping with a execution time less than that of the identity mapping on the completely connected architecture, so this time is a lower bound on the execution time.

**Theorem 1** *For a given clustering, no mapping on the target architecture can have a execution time that is less than the execution time of the identity mapping on the completely connected architecture.*

**Proof:** The proof can be found in [7].

This lower bound is loose, in the sense that it is unlikely that *any* mapping on the target architecture can achieve this time bound. However, this lower bound is used for comparison because no other basis of comparison was available.

#### 4.7 Termination

The mapping heuristic terminates for any of three reasons:

- A previous mapping of the clusters to the processors has been repeated. The mapping process is deterministic, so a cycle of mappings has been found.
- A mapping has an execution time equivalent to that of the completely connected architecture.
- It is necessary to balance the complexity of the algorithm with the need to generate an efficient mapping. In order to guarantee an upper bound on the time complexity of the mapping algorithm, the mapping algorithm is terminated after  $p$  iterations. Based on measurements taken while computing the results presented in Section 5, this gives a reasonable trade-off between complexity and performance.

Since there are at most  $p$  iterations of the algorithm, the worst-case time complexity is  $O(p^4 + vp + ep \log^2 p)$ .

### 5 Results

All tests assume a hypercube architecture. All task graphs were randomly generated using task computation times that randomly vary from 1 – 10 and message lengths that randomly vary from 1 – 10 packets. Each task generates from 1 – 5 messages. The tasks are uniformly distributed between the clusters. Additional results are presented in [7].

The average of 100 randomly generated graphs was used for the results. The results are weighted relative to the execution time on the completely connected architecture. The three algorithms used are:

- **MsCntn(Measure Contention)** The heuristic in this paper is used to choose a mapping.
- **IgCntn(Ignore Contention)** The heuristic in [9] is used to choose a mapping. The differences between the two heuristics are that their heuristic ignores channel contention and does not allow the reassignment of the critical clusters after the initial mapping. The mapping chosen by their heuristic is recomputed with channel contention and this corrected measure of the execution time is used in the results.

Table 1: Execution Times (Lower Bound = 100)

Problem Size	MsCntn	IgCntn	Random
$p = 4, v = 200$	112.1	113.0	113.2
$p = 8, v = 200$	112.3	117.4	117.5
$p = 8, v = 400$	113.8	118.1	117.6
$p = 16, v = 400$	114.1	121.1	123.3
$p = 16, v = 800$	115.2	122.2	124.4
$p = 32, v = 800$	115.6	122.3	123.8
$p = 32, v = 1600$	121.5	131.3	134.5
$p = 64, v = 1600$	123.5	138.5	142.6
$p = 64, v = 3200$	131.6	145.8	150.1

- **Random** A random mapping is generated.

Table 1 provides results for various configurations of the number of nodes in the DAG and the hypercube architecture. Figure 2 shows the results of using a 16-processor hypercube and varying the size of the DAGs. Figure 3 shows the results of using DAGs with 800 tasks on different sized hypercubes. Table 1 and Figures 2 – 3 clearly show the improvement when channel contention is considered. Table 1 shows the *increasing* difference in execution times between the heuristics as the problem size increases. As seen in Figure 2, the relative improvement is almost constant when the number of processors is fixed and the number of tasks is increased. Although the relative improvement is nearly constant, the absolute improvement increases. Figure 3 shows how the difference between the heuristics increases with the number of processors when the number of tasks is fixed. We expect these trends to continue for larger numbers of processors, because of the continued increase in distance and channel contention.

Except for small problem sizes, the random mapping algorithm has the worst performance. The variation between the results in this paper is smaller than the variation given in [9]. The reason for this reduced variation is that packet switching is used instead of message switching. Message switching is more sensitive than packet switching to the distance between processors. This has the most effect for random mappings, which are significantly worse.

The heuristic given in [9] does not perform significantly better than the random mapping, while our heuristic shows noticeable improvement. *Our heuristic shows the least variance*, although both heuristics have less variance than the random mappings.

The heuristics are extended to wormhole-routed hypercubes, but are otherwise unchanged. The random graphs are modified so that communication costs are given in num-

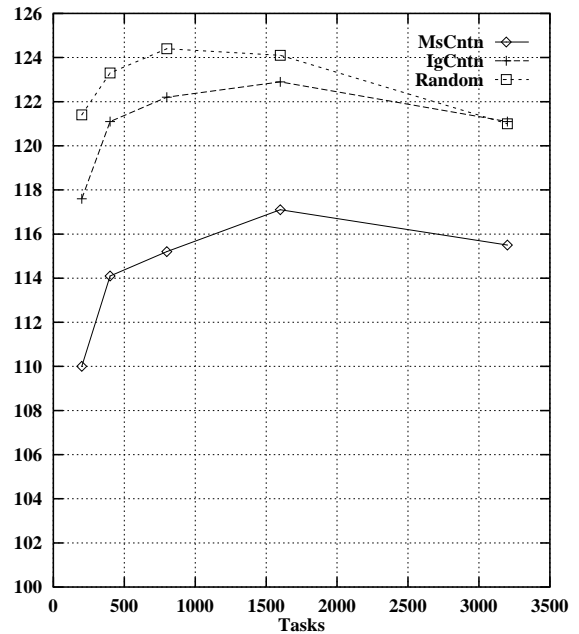


Figure 2: Execution Times with 16 Processors<sup>1</sup>

ber of flits rather than number of packets. The execution times of the tasks are modified to maintain the same computation to communication ratio. The results are shown in Table 2.

Table 2: Execution Times (Lower Bound = 100)

Problem Size	MsCntn	IgCntn	Random
$p = 4, v = 200$	111.6	112.9	113.3
$p = 8, v = 200$	111.7	116.4	117.1
$p = 8, v = 400$	112.5	117.5	118.8
$p = 16, v = 400$	112.5	119.6	121.6
$p = 16, v = 800$	115.4	121.7	123.9
$p = 32, v = 800$	114.4	122.6	125.5
$p = 32, v = 1600$	119.9	128.3	130.7
$p = 64, v = 1600$	117.1	126.1	128.7
$p = 64, v = 3200$	124.3	134.2	135.8

The results in Table 2 show important differences from the results in Table 1 and Figures 2 – 3. Both heuristics generate better mappings than the random mapping, however, the improvement is not as great. Accurate estimates of the execution time of random mappings are comparable to inaccurate estimates of more sophisticated mapping heuristics. This can be clearly seen since the random mapping is nearly as good as the heuristic that ignores channel contention. Our heuristic shows the best performance for all problem sizes, but the improvement over the random map-

<sup>1</sup>The y-axis starts at 100, the lower bound for an optimal mapping.

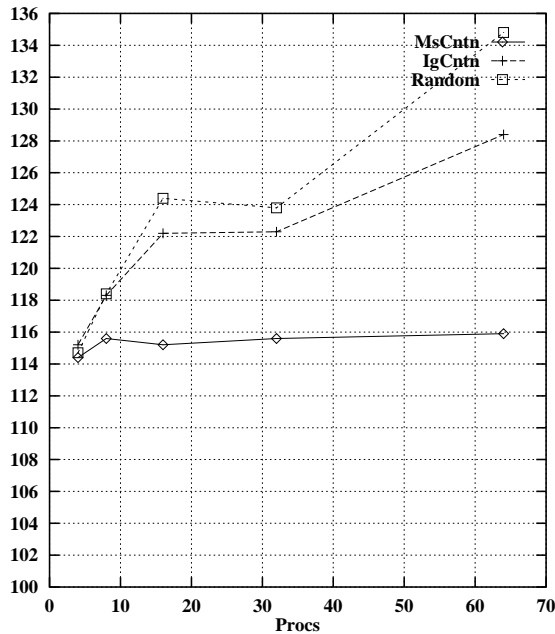


Figure 3: Execution Times with 800 Tasks<sup>1</sup>

pings is relatively small. This suggests that *different heuristics are required for wormhole-routed architectures*. These heuristics should be more sensitive to channel contention than distance.

## 6 Conclusion and Future Work

The mapping strategy presented in this paper demonstrates the need for a realistic model of both the communication requirements of a parallel algorithm and the communication bandwidth of the interconnection network to produce efficient mappings. Ignoring channel contention in the interconnection network can result in the selection of an inferior mapping.

Experimental evidence has been presented to justify the use of this mapping heuristic. One aspect that deserves further exploration is deriving a tighter lower bound on the optimal mapping. This examination is useful both for quantifying the worst-case and expected results of the mapping algorithm, and for identifying potential refinements to the heuristic. The heuristics have been compared using random graphs. We plan to use DAGs generated from a variety of applications.

Our approach uses clustered task graphs which are then mapped onto the processors. Considering channel contention during the clustering step as well as the mapping step may result in a better solution to the general mapping problem. We are currently developing heuristics that reduce channel contention for both the clustering and map-

ping steps on wormhole-routed architectures and also perform significantly better than random mappings.

## References

- [1] S. H. Bokhari. On the Mapping Problem. *IEEE Transactions on Computers*, C-30(3):207–214, March 1981.
- [2] H. El-Rewini and T. G. Lewis. Scheduling Parallel Program Tasks onto Arbitrary Target Machines. *Journal of Parallel and Distributed Computing*, 9:138–153, June 1990.
- [3] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W. H. Freeman and Company, New York, NY, 1979.
- [4] A. Gerasoulis, S. Venugopal, and T. Yang. Clustering Task Graphs for Message Passing Architectures. In *ACM International Conference on Supercomputing*, pages 447–450, 1990.
- [5] S. J. Kim and J. C. Browne. A General Approach to Mapping of Parallel Computations upon Multiprocessor Architectures. In *International Conference on Parallel Processing*, volume III, pages 1–8, 1988.
- [6] S. Lee and J. K. Aggarwal. A Mapping Strategy for Parallel Processing. *IEEE Transactions on Computers*, C-36(4):433–442, April 1987.
- [7] L. Schwiebert and D. N. Jayasimha. Mapping Parallel Computations to Multiprocessor Architectures Considering the Effects of Communication. Technical Report OSU-CISRC-5/92-TR-14, The Ohio State University, 1992.
- [8] G. C. Sih and E. A. Lee. Scheduling to Account for Interprocessor Communication within Interconnection-Constrained Processor Networks. In *International Conference on Parallel Processing*, volume I, pages 9–16, 1990.
- [9] J. Yang, L. Bic, and A. Nicolau. A Mapping Strategy for MIMD Computers. In *International Conference on Parallel Processing*, volume I, pages 102–109, 1991.
- [10] T. Yang and A. Gerasoulis. A Fast Static Scheduling Algorithm for DAGs on an Unbounded Number of Processors. In *ACM International Conference on Supercomputing*, pages 633–642, 1991.