Application-Specific Deadlock Free Wormhole Routing on Multicomputers

Xiaoxiong Zhong and Virginia Lo

CIS-TR-92-03
January 1992

Department of Computer and Information Science
University of Oregon

Application-Specific Deadlock Free Wormhole Routing on Multicomputers *

Xiaoxiong Zhong and Virginia M. Lo†
Computer Science Department
University of Oregon
Eugene, Oregon 97403-1202
lastname@cs.uoregon.edu

Abstract

We show that given *a priori* knowledge of the messages to be routed for a specific parallel application, efficient algorithms can be designed to generate low message traffic congestion and deadlock free routings at compile-time. The problem can be formulated as a graph theoretic problem. Since the problem of finding a deadlock free routing with minimal maximum message traffic congestion is shown to be NP-hard, an efficient heuristic is proposed. Performance of the heuristic for both random message distribution and for several specific applications on hypercube and torus topologies is evaluated by simulations. We show that, compared with the E-cube and XY fixed routings, the heuristic has significant improvement with respect to maximum congestion for the specific applications as well as for the nonuniform message distribution case which models communication locality. For the uniform message distribution case, the heuristic has moderate improvement.

1 Introduction

Wormhole routing has been widely used in many advanced multicomputers such as Symult 2010, nCUBE-2, iWarp, and Intel's Touchstone project. This routing scheme has been shown to be more efficient than routing schemes such as store-forward and virtual cut-through. However, wormhole routing introduces a new problem: deadlock can occur because blocked messages remain in the communication channels [DS87]. One way to prevent deadlock is to provide a fixed (oblivious) routing scheme which guarantees freedom from deadlock. For example, in a hypercube system, the E-cube routing scheme always routes a message in the order of decreasing dimensions. Another way is to partition the whole physical network into several virtual

*Supported by Oregon Advanced Computing Institute (OACIS)
†Partially supported by NSF-grant CCR-8808532

networks [DS87, LH91, LN91, JMY89] such that routings on the individual virtual networks are guaranteed to be free from deadlock. This approach provides more connectivity than the fixed routing scheme and hence is more adaptive. The disadvantage of this approach is that it is too expensive to implement for many network topologies. For example, it has been suggested in [LH91] that to attain all possible shortest paths for a pair of nodes in a k-ary n-dimensional hypercube, 2^{n-1} virtual networks should be provided. In addition, the overhead of multiplexing may degrade performance.

The above approaches are designed specifically to optimize the overall network performance such as network latency [Dal90]. While this approach has been successful for general purpose multicomputers, it may have some shortcomings for high performance applications. In particular, since the above routing schemes are oblivious to the message passing requirements of specific applications, such routings may cause serious traffic congestion and thus affect total execution time and system throughput. This is especially undesirable for real time applications [SA91]. Fortunately, for many practical applications, the task structure can be known a priori at compile time and therefore, **application specific routings** can be generated after tasks are assigned to processors [LRG+90, BS87a]. The objective of our research is to design algorithms to generate routings with low maximum message congestion based on knowledge of the message passing structure at compile time. In addition to minimizing maximum message congestion, the challenging problem here is to guarantee that the generated routes are deadlock free.

In this paper, the problem of finding low maximum message congestion and deadlock free paths for a given set of messages is first formulated as a graph theoretic optimization problem on a graph which we call the generic physical channel dependency graph (GPCDG). The GPCDG captures all *possible* channel dependencies for the physical network. With the GPCDG, this problem can be formulated as that of finding paths for messages in the GPCDG such that the subgraph formed by these paths is acyclic and the maximum congestion is minimal. Based on a result given in [KS90], it is shown that the problem of finding a deadlock free, minimal maximum message congestion routing is NP-hard for an arbitrary network. We propose an efficient heuristic which iteratively decreases the maximum congestion while still preserving freedom from deadlock. Performance of the heuristic is studied for both hypercube and torus topologies. Three kinds of simulations are performed. In the first set of simulations, messages are uniformly distributed on the network. In the second set, messages are nonuniformly distributed on the network to model the communication locality. The third set of simulations consists of two specific parallel applications. The performance of the heuristic is compared with that of the E-cube and XY fixed routing schemes respectively.

The rest of the paper is organized as follows. In Section 2, we review previous work for other routing schemes such as store-forward and virtual cut-through. We show that routing algorithms developed for these routing schemes are not applicable to wormhole routing. In Section 3, we introduce the metrics used for message congestion and formulate the problem as

an optimization problem on the GPCDG. In Section 4, we describe the heuristic in detail. In Section 5, we discuss the performance of the heuristic. In Section 6, we summarize the results and discuss future work along this direction.

2 Related Work

The problem of choosing paths for a given set of messages has been studied for various routing schemes. In [BS87b], Bianchini and Shen propose a scheduling algorithm for message traffic in a multiprocessor network. The problem is formulated as a network flow problem based on the assumption that a message can be split into several small flows at a node. This assumption is not applicable to current tightly coupled, high speed multiprocessor networks since the overhead to manage message splitting and combining may well exceed that of the whole message transmission through the network.

In [SA91], a scheduled routing framework is proposed by Shukla and Agrawal for real-time periodical pipelining applications. In their scheme, the router of a processor sets up channel connections based on switch setting commands received from the application running in that processor. The switch setting commands are generated statically at compile time based on a priori knowledge of the task structure in a way that guarantees there will be an unobstructed path for each message during the whole message transmission period. Since messages are routed through the network without collision, the generated routing is always deadlock free. The scheme can only be used for an architecture which has specialized communication modules. Furthermore, it is not always possible to find paths such that messages do not collide. In this case, it is not clear how the deadlock is avoided.

The previous work which is most closely related to ours is the traffic routing scheme proposed by Kandlur and Shin in [KS90]. They study the problem of choosing paths for an arbitrary set of messages to be routed in a network with virtual cut-through routing capability. They show that the problem of choosing pairwise edge-disjoint paths for a given set of messages is NP-complete and an efficient heuristic algorithm is presented. The heuristic first randomly chooses a path for every message and then, tries to reroute one message at a time to decrease the *total cost*, which is defined as the summation over all links of the squares of the total message volume passing through each link. If the total cost can not be improved for all messages, the algorithm stops. It has been shown that the algorithm performs very well by simulation.

Wormhole routing, however, differs from virtual cut-through in that once the head of a message packet is blocked due to channel contention, the whole packet remains in the network, occupying all the channels it is traversing. It is possible that a wormhole routing can be deadlocked due to circular waiting for communication channels [DS87]. In virtual cut-through routing, blocked packets are buffered at a processor, thus releasing occupied channels; therefore, deadlock is not a concern. In Fig. 1, the final routing of Kandlur and Shin's algorithm is optimal

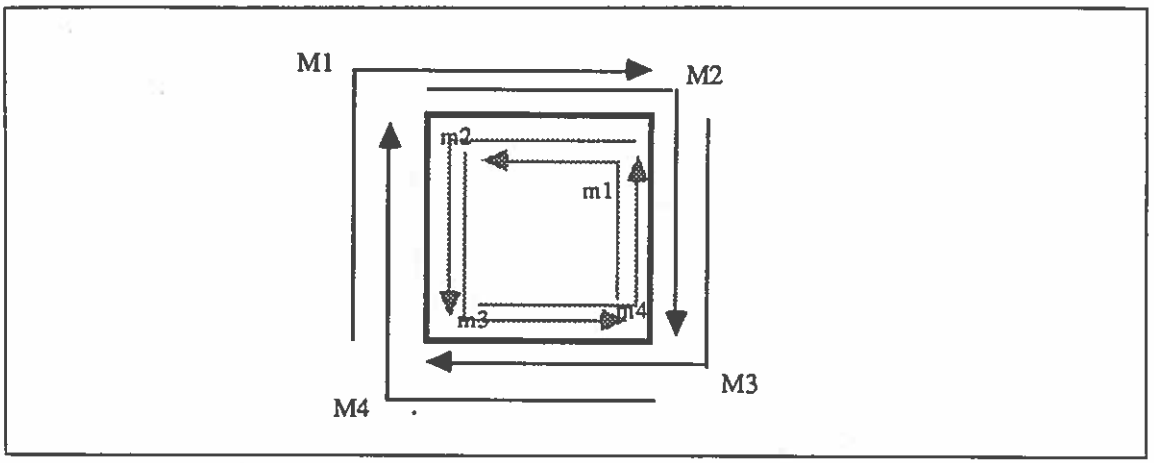


Figure 1: An example where Kandlur and Shin's algorithm generates a deadlocked wormhole routing

with respect to the cost function they define. However, the routing is deadlocked since in the example, both sets of paths for m_1, m_2, m_3, m_4 and M_1, M_2, M_3, M_4 create channel dependency cycles.

3 Problem formulation

By application-specific routing, we mean a routing designed specifically for the messages used in an application. This contrasts with the fixed (oblivious) schemes such as E-cube and XY routings which are independent of the messages to be routed. More precisely, we assume that tasks of the application have been assigned to processors and we also assume that the weights of interprocess communications are known. Such weights can be the total volume of the messages sent between processors [LRM+] or the frequencies of message passing between two processors obtained by methods such as program profiling [Sar89]. In graph-theoretic terms, we formally define the problem as follows.

Definition 1 Let $A = (P, E)$ be a *directed* graph where nodes correspond to processors and edges to communication channels in the physical network. Let a set of messages be represented as $S = (M, W)$ where M is a set of pairs of nodes in P where $(s, d) \in M$ represents the source processor and the destination processor for a single message respectively, and W is the weight function which maps pairs in M to nonnegative integers.

A **routing** is a function $\mathcal{R}$ which maps every pair $(s, d) \in M$ to a simple path $(p_0, p_1), (p_1, p_2), \ldots, (p_{k-1}, p_k)$ in A such that $p_0 = s, p_k = d$.

In the wormhole routing scheme, message collision has more impact on routing performance than the distance (hops) a message has to travel. In wormhole routing, a message is divided into smaller units called flits which are continuously transmitted along the path. Thus, a pipelining effect occurs so that latency without considering message collision is

$$L_f D/B + L/B$$

where L_f is the length of a flit, D is the distance of the path, L is the length of the message and B is the bandwidth of a channel. Since in most cases, $L_f << L$, the contribution of distance D to the latency can be considered to be negligible. If two messages collide in a channel, one message has to wait until the other message passes through the channel completely. Thus message congestion in the wormhole routing is the predominant factor for performance.

Below, we define two metrics which reflect the degree of message collision occurring in a routing.

Definition 2 For routing $\mathcal{R}$, the congestion of a channel is defined as the number of the messages passing through it. The maximum congestion of $\mathcal{R}$, $C(\mathcal{R})$, is the maximum congestion over all channels. In particular, if the number of messages passing through a link h is $C(\mathcal{R})$, we call h a hot spot. If $C(\mathcal{R}) = 1$, we say $\mathcal{R}$ is congestion-free.

Intuitively, each channel whose congestion equals $C(\mathcal{R})$ is one of the hottest traffic spots in the network. Hot spots are especially undesirable in a real-time application since the delay of messages may slow down the whole application and the deadlines may not be met. Our objective, in this paper, is to find a routing which has minimal maximum congestion.

Definition 3 The T-Cost of $\mathcal{R}$, $TC(\mathcal{R})$, is defined as

$$TC(\mathcal{R}) = \sum_{e \in E} (\sum_{m \in M, e \in \mathcal{R}(m)} W(m))^2$$

TC was first defined and used as a cost metric in [KS90]. It reflects both the congestion and dilation of the routing. In the heuristic we propose here, we use this cost function as a secondary metric to control the algorithm so that it does not detour messages too far to reduce the maximum congestion.

In addition to the minimal maximum congestion objective, the routing generated must be deadlock free in a wormhole routing. Deadlocks occur in a wormhole routing scheme when a circular channel waiting state is reached. A well known condition for a deadlock free wormhole routing is given by Dally in [DS87] based on the concept of the channel dependency graph. The channel dependency graph $CDG_{\mathcal{R}}$ for a routing $\mathcal{R}$ is a directed graph whose nodes are channels used in the routing. If there is a path in $\mathcal{R}$ such that c_1, c_2 are two *successive* channels in this

path, then there is an edge in $CDG_{\mathcal{R}}$ from node c_1 to c_2 (also called c_1 depends on c_2). It is clear that a channel dependency graph depends on a specific routing.

Property 1 *[DS87] Routing $\mathcal{R}$ is deadlock free for wormhole routing mechanism iff the channel dependency graph $CDG_{\mathcal{R}}$ is acyclic.*

Our optimization problem is to find a deadlock free routing which has the minimal maximum congestion. Based on a result given in [KS90], we have

Theorem 1 The problem of deciding whether there exists a deadlock free congestion-free routing for an arbitrary set of messages and an arbitrary network is NP-Complete.

Proof: In [KS90], Kandlur and Shin prove that the decision problem of whether there exists a congestion-free routing for an arbitrary set of messages and an arbitrary network is NP-Complete. Since in a congestion-free routing $\mathcal{R}$ all messages are routed through *edge-disjoint* paths, its channel dependency graph $CDG_{\mathcal{R}}$ is acyclic. Thus, based on Property 1, $\mathcal{R}$ is deadlock free. Therefore, testing the existence of a deadlock free congestion-free routing is equivalent to testing the existence of a congestion-free routing. ∎

Hence, it is unlikely that a polynomial time algorithm for the optimization problem can be designed and efficient heuristics must be developed.

Since the deadlock free condition requires to check links in the original network A, it is useful to have a structure to represent link relationships explicitly. We define a graph called the Generic Physical Channel Dependency Graph(GPCDG) for this purpose.

Definition 4 For a given network (directed graph) $A = (P, E)$, its GPCDG is a directed graph $D_A = (V, D)$ where its node set V is $E \cup P$ and its edge set D is defined as follows, for all $e_1 = (a, b), e_2 = (b, c) \in E$, $(e_1, e_2) \in D$. Furthermore, for every edge $e = (p_1, p_2) \in E$, $(p_1, e), (e, p_2) \in D$ also. We call a node $l \in E$ a link node and a node $p \in P$ a processor node.

Intuitively, a GPCDG D_A captures all possible dependencies among the physical channels of A and also retains the original network topology. A key difference between a channel dependency graph and a GPCDG is that the former corresponds to a specific routing and the latter is independent of any routing. Figure 2 shows an example for a 2×2 mesh. Note the GPCDG is analogous to the *total graph* in graph theory [Har72] which is defined for an undirected graph.

We now reformulate the optimization problem for a deadlock free routing based on the GPCDG.

Problem Definition: Given a GPCDG D_A for a network $A = (P, E)$ and a set of messages $S = (M, W)$, a routing $\mathcal{R}$ is a function which maps each pair $(s, d) \in M$ to a simple path in

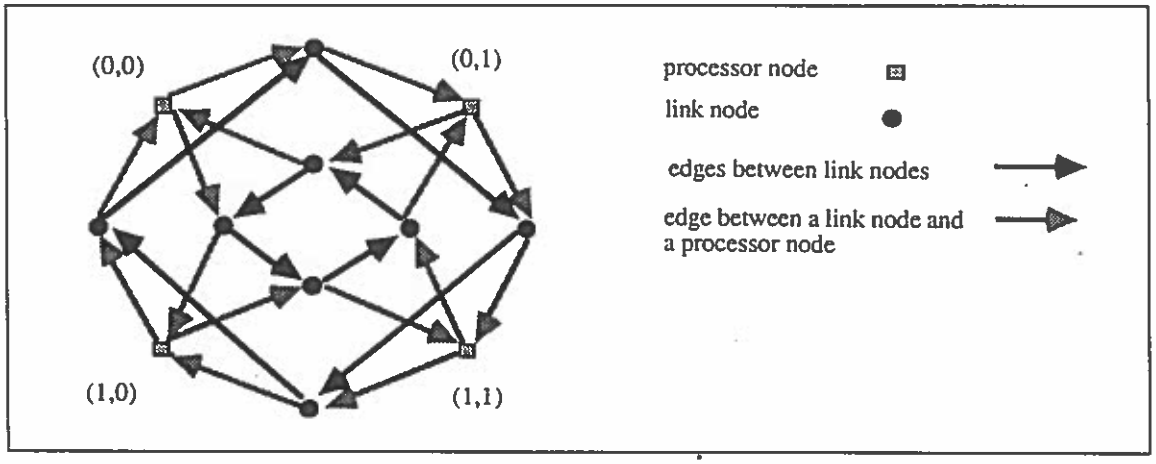


Figure 2: The GPCDG of 2 × 2 mesh

D_A whose endpoints are s and d respectively and whose interior points are link nodes in D_A. Furthermore, a routing $\mathcal{R}$ is deadlock free iff the subgraph formed by message paths in GPCDG D_A is acyclic. The maximum congestion becomes the maximum number of messages passing through a link node of the GPCDG. The objective of our optimization problem is to find a deadlock free routing which minimizes the maximum congestion over all link nodes in D_A.

4 Deadlock Free Low Maximum Congestion Routing

Our deadlock free heuristic (DFH) for the wormhole routing scheme is straightforward. It starts from an initially preselected *deadlock free* routing, iteratively tries to reduce the congestion of hot spots (there might be several such spots) by rerouting messages through other channels with less congestion such that the new routing is also *deadlock free* and its T-Cost $TC(\mathcal{R})$ decreases. The algorithm stops if no further improvement can be achieved. Figure 3 shows the outline of DFH.

DFH has two major procedures, `initialize` and `reroute`. `initialize` selects an initial deadlock free routing. `reroute` tries to reroute a message through other channels other than the hot spot so that the T-Cost is decreased and does not create other hot spots through the rerouting. To efficiently find hottest spots, a priority queue [Tar83] is used to manage link nodes with priorities being their congestions. The following two subsections describe the two routines in more detail.

```
/*DFH */

R = initialize(D_A);
R' = R;
repeat {
        for (every hottest spot h in R){
            if (there exists a message m passing through h which
              can be rerouted through channels other than h and
              the T-Cost decreases) {
              R=reroute(m); }}}
until {R' == R};
```

Figure 3: Outline of the deadlock free heuristic DFH

4.1 Initialization

Finding a deadlock free routing as the initial routing for DFH is straightforward for many regular networks such as hypercubes,meshes and tori. For a hypercube, we can use E-cube routing which always routes a message in the order of decreasing dimensions. This routing has been proved to be deadlock free for wormhole routing. For a 2D mesh or torus, we can use XY routing which always routes a message first along the X-direction and then along the Y-direction. This is also known to be deadlock free. Some applications, however, may use irregular interconnections and there may be no known deadlock free fixed routing for such networks. For example, the iWarp system allows the user to configure an irregular network by setting up so called pathways [BCC+90, Gro89]. We propose a simple method to find an initial deadlock free routing for an irregular network topology. The method uses a spanning tree of the network to find such a routing.

Property 2 *Let $A = (P, E)$ be an arbitrary bidirectional network which, when treated as an undirected graph, is connected and $S = (M, W)$ be a set of messages to be routed, it is always possible to find a deadlock free wormhole routing $\mathcal{R}$ for S in A.*

Proof: Let ST be a spanning tree of A. ST can be found efficiently (for example, by using a depth first search). Let DST be the directed graph generated from ST by replacing a single edge (p_1, p_2) in ST with two directed edges (p_1, p_2) and (p_2, p_1). We define a routing $\mathcal{R}$ as follows. Since for any two nodes $(s, d) \in M$, there exists a unique simple path $P = (p_1, p_2, \ldots, p_n)$ from s to d $(s = p_1, d = p_n)$ in DST, we designate the directed path of P (which exists in A since A is bidirectional) as the routing path for (s, d).

To prove $\mathcal{R}$ is deadlock free, assume the opposite, then by Property 1, the channel dependency graph $CDG_{\mathcal{R}}$ has a simple cycle $c_1, c_2, \ldots, c_n$ where c_i corresponds to edge (s_i, d_i) in DSF for $i = 1, \ldots, n$. Thus $d_i = s_{i+1}$ for $i = 1, \ldots, n-1$ and $d_n = s_1$ and $s_1, \ldots, s_n$

form a cycle C in DST. But the only cycles in ST are those which are formed by cycles (p_1, p_2) and (p_2, p_1). Therefore, there exists an $1 \leq i < n$ such that $e = (s_i, s_{i+1})$ and $e' = (s_{i+1}, s_i)$ are two successive edges in C. But this implies that e depends on e' and e' depends on e, which is not correct since $\mathcal{R}$ always routes messages through a simple path in ST. ∎

The importance of the above property is that it indicates there is *always* a deadlock free routing for any bidirectional network. Furthermore, the spanning tree approach efficiently finds an initial deadlock free routing (linear time complexity in the number of edges of A). The drawback of the algorithm is that the message traffic congestion of the resultant routing is high because of the tree structure. Thus, when a network is regular, we use the fixed fixed routing as its initial routing.

4.2 Reroute a Message away from a Hot Spot

`reroute` takes a routing $\mathcal{R}$, a hot spot h in GPCDG D_A and a message m passing through link h as its inputs and returns a new routing. Assume the nodes immediately before h and immediately after h in the path for message m are a and b respectively (note, a, b may be processor nodes). It then tries to reroute m from a to b by trying to avoid passing through h.

It should be noted that we can not simply reroute m through any path from a to b even though such rerouting can decreases maximum congestion and the T-Cost. This is because such a path may create a deadlock. To simplify the discussion, we need the following definitions.

Definition 5

- For a routing $\mathcal{R}$, $\mathcal{R}$ is also used to denote the channel dependency relation induced by $\mathcal{R}$ on link nodes in D_A.
- A path is said to be **deadlock free with respect to a routing** $\mathcal{R}$ if adding the dependencies in the path to $\mathcal{R}$ still preserves the acyclic condition in D_A.

Let $\mathcal{R}_1$ correspond to the channel dependency relation after the removal of message m from path a, h, b. Note that $\mathcal{R}_1$ may still retain dependency $a \rightarrow h$ or $h \rightarrow b$ since there might be other messages routed through these links. Also note if $\mathcal{R}$ is deadlock free, $\mathcal{R}_1$ is deadlock free too, since removal of dependencies can not introduce deadlock. The goal of `reroute` is to find a deadlock free path from a to b to replace a, h, b such that the T-Cost is decreased.

The algorithm uses depth first search to find such a path. It starts with an initial partial path containing only a and examines the unsearched link nodes adjacent to the most recently added node of P. If adding an unsearched node does not cause deadlock and the new partial T-Cost does not exceed the old T-Cost, then the node is marked as searched and is added to P.

```
/*reroute(R, m, a, h, b) */

P = {a}; mark(a);
Ocost =cost(R);
Pcost=cost(remove-m-from-h(R, m, h));
while (P not empty and last(P) ≠ b) {
        if (there is a deadlock free unmarked node u adjacent to last(P)
          and update(Pcost, u) < Ocost){
          P = P⋃{u};
          mark(u);
          Pcost=update(Pcost, u) }
          else
                {P = P−{last(P)};
                unmark(last(P);
                recover(Pcost);}}
```

Figure 4: reroute function

The procedure continues until it either reaches b or no new node can be found which satisfies the deadlock free condition and decreases T-Cost. In the former case, a new route from a to b has been found. In the latter case, the algorithm unmarks the current node and backtracks to the previous node in P. If no new deadlock free path is found (i.e., if a is examined again), then m is still routed through a, h, b.

Figure 4 shows the algorithm. *Ocost* is the T-Cost of the initial routing $\mathcal{R}$ and *Pcost* is the T-Cost of the current partial routing.

reroute requires testing whether adding a node causes deadlock in $P \bigcup \mathcal{R}$. To efficiently accomplish this, the *transitive* closure of the channel dependency relation induced by $\mathcal{R}$ is maintained. Let $\mathcal{R}^*$ be the transitive closure of $\mathcal{R}$ and $\mathcal{R}_1^*$ be the corresponding transitive closure of $\mathcal{R}_1$. Since computing the transitive closure [AHU74] is relatively expensive ($O(N^3)$ where N is the number of link nodes), we want to avoid recomputing the entire new transitive closure every time we find a new path to replace path a, h, b. This can be achieved by maintaining the transitive closure as a matrix whose ij entry represents the number of directed paths from a link node i to another link node j instead of simply a binary entry to indicate whether there is a path from i to j. Each time a new path is found to replace a, h, b, only some entries of the transitive closure are changed. The time it takes to compute the number of paths for all pairs of link nodes after the initial routing has been chosen is the same as to compute the standard binary transitive closure. Figure 5 shows the algorithm to compute the initial transitive closure and the algorithm to update the transitive closure once a new path has been found. The time complexity to update the transitive closure is reduced to $N^2|P|$, where $|P|$ is the length of P.

```
/*transitive-closure(R), N
returns a matrix R* representing the number of paths for all pairs */
R* = R;
for {i = 2...N } {
    for {l, m = 1...N }
        R*(l, m) = R*(l, m) + R(l, i) × R(i, m);
return R*

/* update-transitive-closure(R*, P, a, h, b)
returns a update R1* by replacing a, h, b with P in R* */
R1* = R*;
/* removal of a → h (if any) */
if {removing m from path a, h, b causes a ↛ h}
  for {all l such that l → a or l = a }
      for {all k such that k → a or k = h }
          R1*(l, k) = R1*(l, k) − 1;
/* similar code for the removal of h → b, omitted*/
/* adding P = (a1, ..., ak) (a1 = a, ak = b) */
for {i = 2, ..., k}
    if {there is no a_{i-1} → a_i in R }
      for {all l such that l → a_{i-1} or l = a_{i-1} }
          for {all k such that k → a_i or k = a_i }
          R1*(l, k) = R1*(l, k) + 1;
```

Figure 5: Computing and Updating Functions for a Transitive Closure

%**deadlock-free-test**$(u, P, \mathcal{R}^*, b)$

if {there is no $p \in P$ such that $u\mathcal{R}^*p$ *and* not$(b\mathcal{R}^*u)$ }
return true;
else return false

Figure 6: `deadlock-free-test` function

The algorithm to test whether a node is deadlock free or not when P is added to $\mathcal{R}_1$ is shown in Figure 6. Since it is possible that no new path other than a, h, b can be found, we would like to retain $\mathcal{R}$ and not update it until a new path is found. The algorithm in Figure 6 uses $\mathcal{R}$ instead of $\mathcal{R}_1$ to accomplish the deadlock testing. This is justified in Property 3.

Property 3 *Path P, as chosen by algorithm* `reroute` *is deadlock free with respect to $\mathcal{R}_1$.*

Proof: We first prove that P is deadlock free with respect to $\mathcal{R}$.

First notice P defines a total order $<$. If adding P to $\mathcal{R}$ causes a simple cycle C, C must contain some subpaths in P. Otherwise, C is a cycle in $\mathcal{R}$, which is a contradiction. Figure 7 shows the structure of C where P_i $(i = 1, \ldots, m)$ are subpaths of P in C. Let h_i and t_i are head and tail of P_i respectively.

If there exists an i such that $t_1 < h_2, t_2 < h_3, \ldots, t_{i-1} < h_i$ and $t_i \not< h_{i+1}$, we have $h_{i+1} < t_i$. From the structure of C, we notice that there exists a path L in $\mathcal{R}$ (refer to Figure 7) from t_i to h_{i+1}. But this is a contradiction since the algorithm `deadlock-free-test` guarantees that there is no such a path in $\mathcal{R}$.

If there is no such i, then $h_1 < t_1 < t_m$ and there exists a path in $\mathcal{R}$ from t_m to h_1. Based on the similar argument, this is impossible.

Since $\mathcal{R}_1$ is contained in $\mathcal{R}$. P must be deadlock free with respect to $\mathcal{R}_1$. ∎

Since `deadlock-free-test` is based on $\mathcal{R}$ rather than $\mathcal{R}_1$, a natural question is whether it eliminates some paths which are deadlock free with respect to $\mathcal{R}_1$ but not with respect to $\mathcal{R}$ (it is possible since $\mathcal{R}$ contains $\mathcal{R}_1$). The following property answers this question.

Property 4 *A deadlock free path $P = a_0, \ldots, a_k$ where $a = a_0$ with respect to $\mathcal{R}_1$ is also deadlock free with respect to $\mathcal{R}$.*

Proof: If adding P to $\mathcal{R}$ causes a simple cycle C, then $\mathcal{R}$ must have more dependencies than $\mathcal{R}_1$. It is safe to assume that $\mathcal{R}$ has both $a \rightarrow h, h \rightarrow b$. Based on the similar argument in the proof of Property 3, there exists a situation where there are $a_1, a_2 \in P$ such that $a_1 <_P a_2$ and there is a path L in $\mathcal{R}$ from a_2 to a_1.

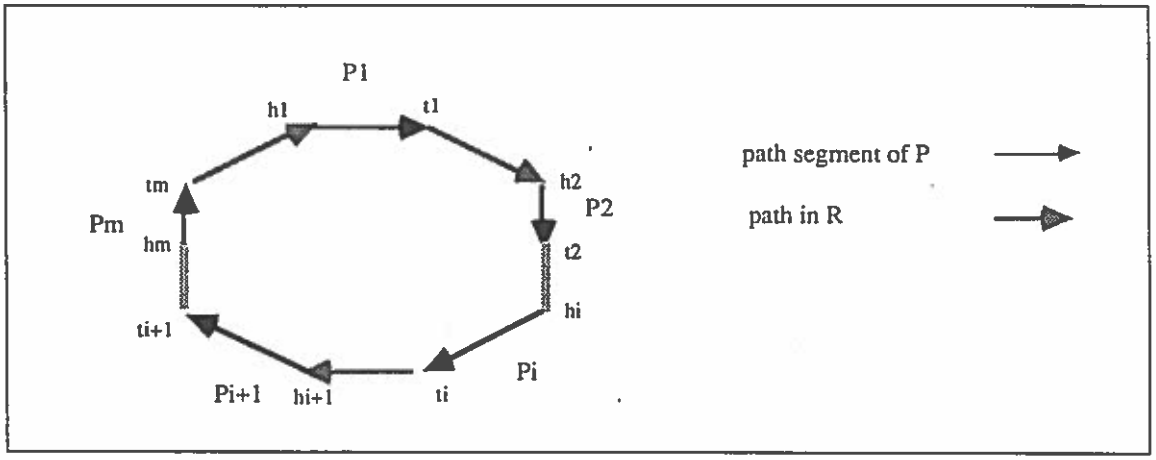


Figure 7: Illustration of the cycle C

If there is no $a \rightarrow h$ and $h \rightarrow b$ in L, then, C is a cycle in the channel dependency graph $\mathcal{R}_1 \bigcup P$, which is a contradiction.

Therefore, there exists $a \rightarrow h$ or $h \rightarrow b$ in L. If there exists $a \rightarrow h$, then the path segment from a_2 to a does not have any $a \rightarrow h$ and $h \rightarrow b$ since C is simple. Based on the previous argument, we know this is impossible.

The only remaining case is when there is only an $h \rightarrow b$ in L, in addition to other dependencies in $\mathcal{R}_1$. But since for a_1 to be in P, `deadlock-free-test` must have guaranteed that there is no path from b to a_1 in $\mathcal{R}$, which includes $h \rightarrow b$. This is clearly a contradiction. Hence, we prove the property. ∎

5 Performance

Two test suites are used to evaluate DFH. Each is evaluated on the torus and binary cube topologies. In both tests, performance is evaluated using both maximum congestion and T-Cost. The performance of DFH is compared with those of the E-cube and XY fixed routing schemes respectively. These fixed routing schemes are also used as the initial deadlock free routings.

The first test suite includes two types of randomly generated message distribution. In the first case, messages are uniformly distributed. Three independent uniform random number generators are used to generate source nodes, destination nodes and message weights respectively. Message weights range from 1 to 50. The number of messages ranges from 10 to 200. The topologies are a 6×6 torus and a 5-dimensional binary cube.

In the second case, messages are nonuniformly distributed on the network in a way to capture communication locality which is usually exhibited after the assignment of tasks to processors

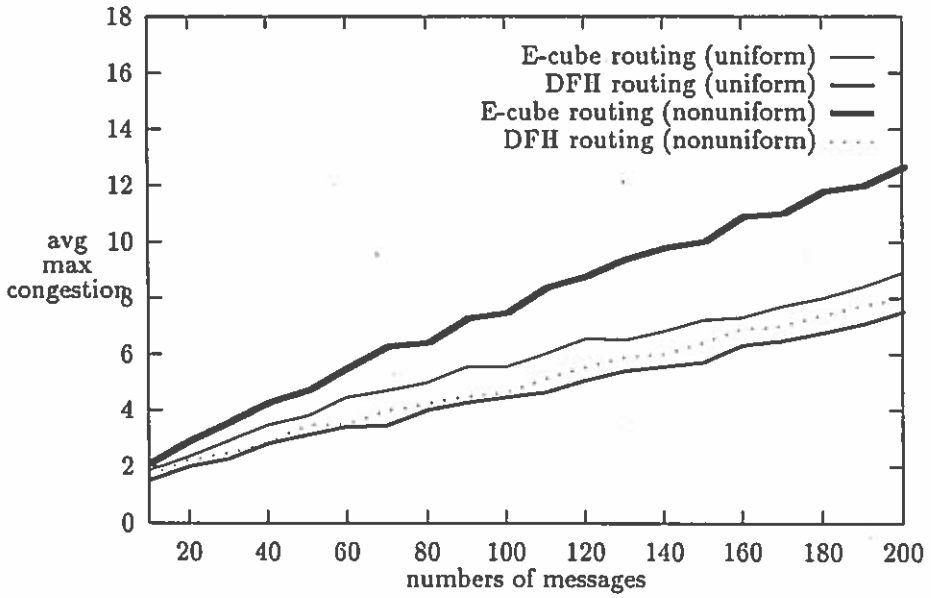


Figure 8: Average Maximum Congestion for a 5-dimensional Cube for Both Uniform and Nonuniform Message Distribution

for an application. This is modeled by dividing the network into four equal parts and messages are distributed uniformly in two quadrants and no messages exist in the other two quadrants. For the topologies considered in this study, a 5-dimensional cube is divided into four subcubes labeled as $00xxx, 01xxx, 10xxx, 11xxx$ where x represents a don't care bit and xxx represents a subcube of dimension 3. Messages are uniformly distributed in subcubes $00xxx$ and $11xxx$. No messages exist in subcubes $01xxx$ and $10xxx$. A 6×6 torus is divided into four 3×3 meshes with some additional wrap around links. In the upper left and the lower right quadrants, messages are uniformly distributed and no messages exist in the upper right and lower left quadrants. The same message weight range and number of messages, as in the first case, are used.

For both cases, performance metrics are averaged over 25 runs of DFH for a given number of messages. It is observed that the standard deviation of the mean maximum congestion for all data is less than 6

Fig. 8 compares the maximum congestion of the routing generated by DFH with that of E-cube routing. Fig. 9 shows the percentage improvement of DFH over E-cube for both maximum congestion and the T-Cost. For nonuniform case, DFH consistently reduces maximum congestion by 30% to 40% and for the uniform case, by 15% to 25% compared with E-cube routing. The T-Cost has much less percentage improvement than maximum congestion. This is because DFH takes the maximum congestion as its primary optimization goal.

Fig. 10 and Fig. 11 show the maximum congestions for DFH routing and XY routing and

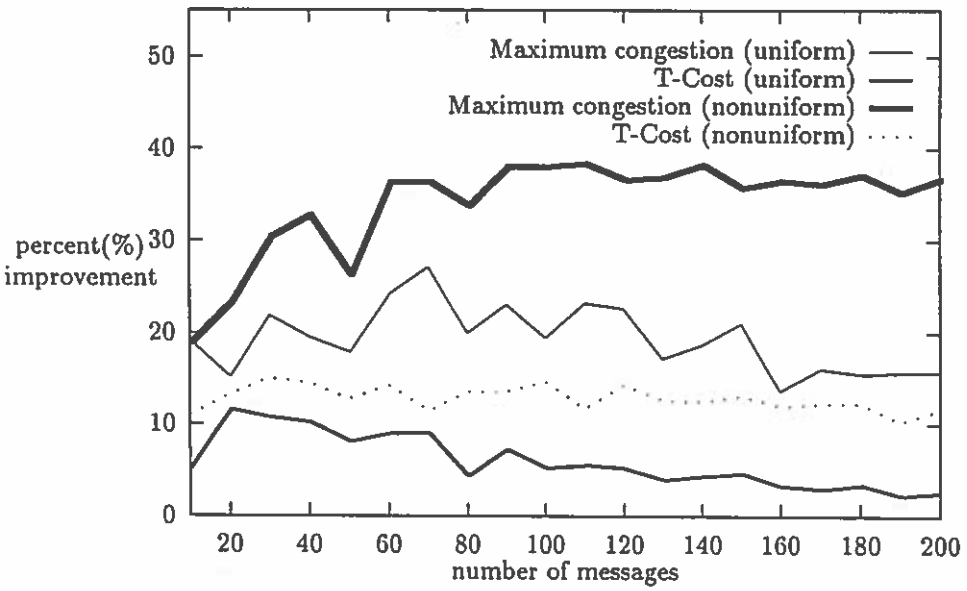


Figure 9: Percentage Improvement of Maximum Congestion and T-Cost for a 5-dimensional Cube for Both Uniform and Nonuniform Message Distribution

the percentage improvement for both maximum congestion and the T-Cost on a 6×6 torus. Again, for the percentage improvement, we see a fairly good gain for maximum congestion and a moderate gain for T-cost. It can be seen also that the improvement is less than that in the binary cube case for the same number of messages. This is because the binary cube has more connectivity than the torus.

The second test suite includes two specific applications. For both applications, a simple greedy embedding algorithm is used to assign tasks to target topologies. The first application is the n-body algorithm which is designed for the Caltech Cosmic Cube [AS88]. Fig 12 gives a description of the problem. In the simulation for 15-body problem both 5×3 torus and 4-dimensional cube topologies are used.

The second application is a program called AVHTST which is used to determine cloud properties from satellite imagery data [RL90, JR90]. Fig 13 gives a description of the problem and its task structure. In the simulation, an 18 nodes AVHTST program is routed on a 4×4 torus and a 4-dimensional cube respectively.

Table 1 shows the simulation data for the two applications. In all cases, there is significant improvement with respect to max congestion. In addition, all percentage improvements for T-Cost are above 10%.

We observe that in the simulations, the number of hot spots is usually much greater than 1 and as the number of contending messages decreases, the number of links which have that number

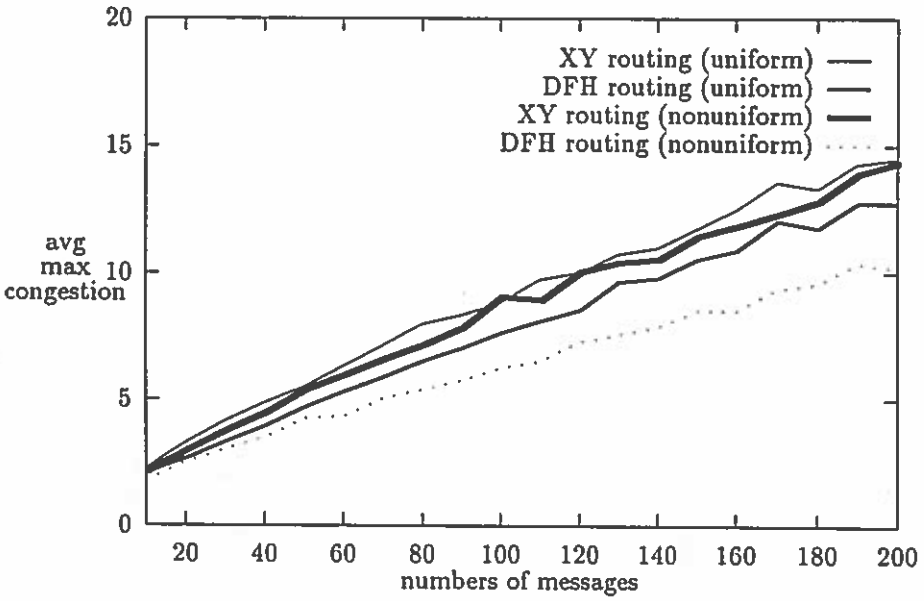


Figure 10: Average Maximum Congestion for a 6×6 Torus for Both Uniform and Nonuniform Message Distribution

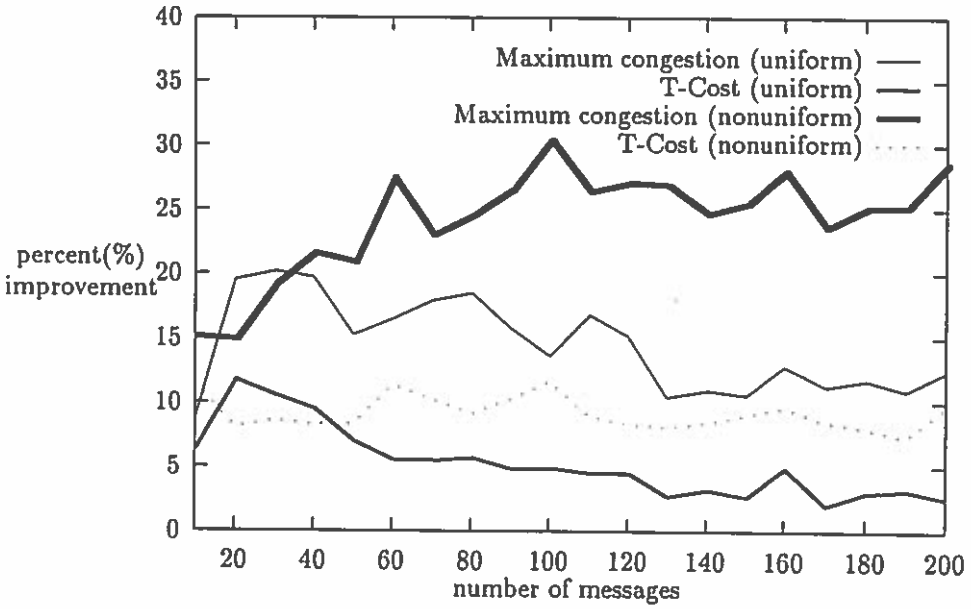
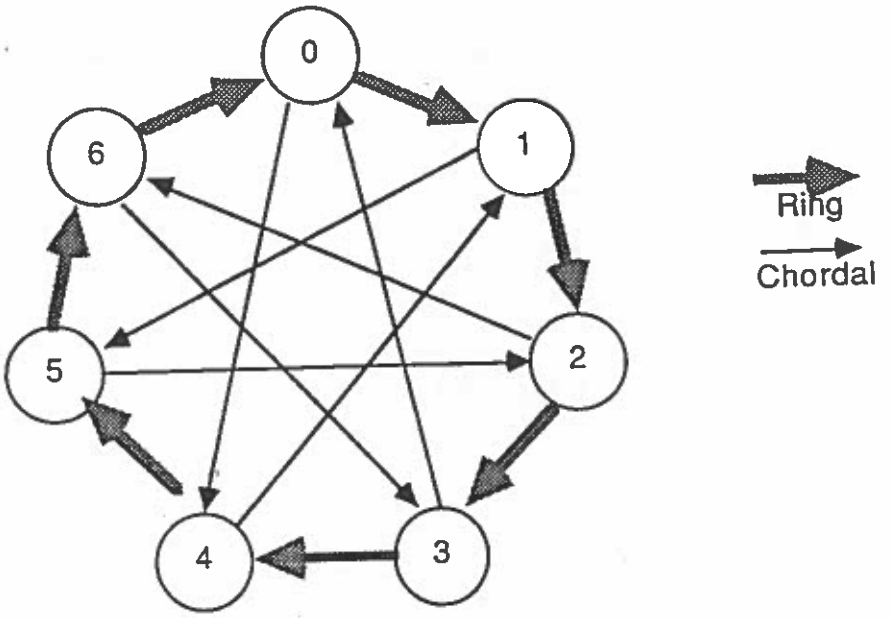
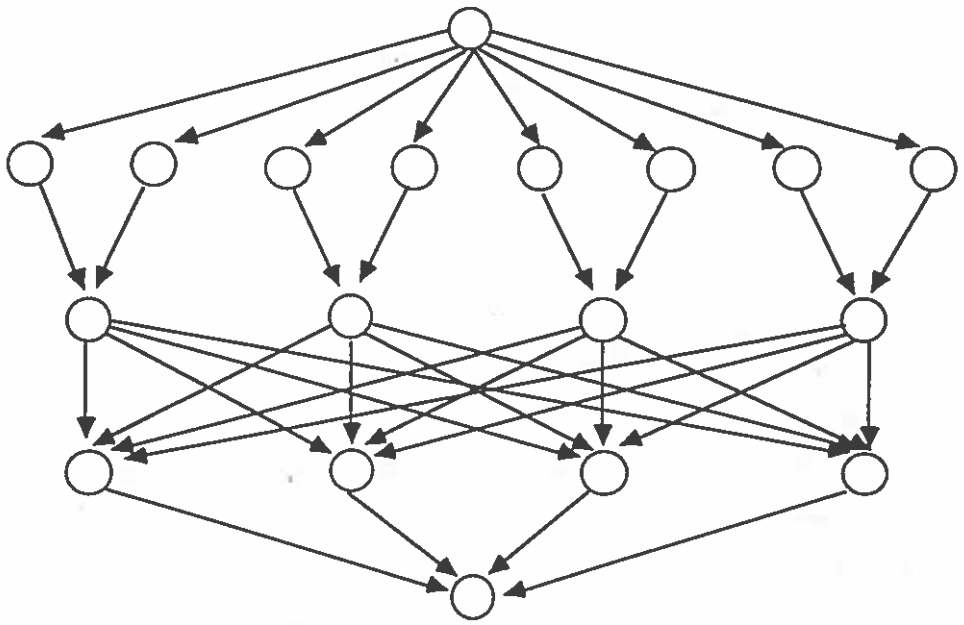


Figure 11: Percentage Improvement of Maximum Congestion and T-Cost for a 6×6 Torus for Both Uniform and Nonuniform Message Distribution



The *n-body problem* requires determining the equilibrium of n bodies in space (where n is odd) under the action of a (gravitational, electrostatic, etc.) field. This is done iteratively by computing the net force exerted on each body by the others (given their "current" position), updating its location based on this force, and repeating this until the forces are as close to zero as desired. The parallel algorithm presented by Seitz uses Newton's third law of motion to avoid duplication of effort in the force computation. It consists of n identical tasks, each one responsible for one body. The tasks are arranged in a ring and pass information about their accumulated forces to its neighbor around the ring. After $(n-1)/2$ steps, each task will have received information from half of its predecessors around the ring. Each task then acquires information about the remaining bodies by receiving a message from its chordal neighbor halfway around the ring. This is repeated to the desired degree of accuracy. In the above is the task graph of 7-body problem..

Figure 12: The Description of n-body Problem and Its Task Graph



AVHTST is part of a program for the study of cloud properties based on satellite imagery data. It is responsible for data analysis and reduction. Input to *AVHTST* consists of c channels of data, , where each channel is represented by a pixel array corresponding to a scene of data at a distinct wavelength. Furthermore, each array is divided into $f \times f$ frames. Each frame is first processed independently and then the results for the corresponding frames in all c channels are combined. Finally, the combined results for $f \times f$ physical frames are broadcast to all other frames for an adjustment of local results. In the last step, the results are aggregated. Above is the task graph for *AVHTST* with 2 channels of data each with 2 by 2 frames per scene. Nodes are labeled with a triple where the first component represents the channel number and the remaining components are the frame coordinates. All edges have unit weight.

Figure 13: The Description of AVHTST Benchmark and Its Task Graph

Applications	Max. Congestion			T-Cost		
	Fixed	*DFH*	Percentage	Fixed	*DFH*	Percentage
15-body on 5×3 torus	5	3	40%	27800	20800	25.2%
15-body on 4-dim cube	3	2	33.3%	23400	19000	18.8%
AVHTST on 4×4 torus	7	5	28.6%	410	367	10.5%
AVHTST on 4-dim cube	5	3	40%	198	166	16.2%

Table 1: Performance for the Applications

of contending messages increases. This explains the fact that when the maximum congestion is small, it is extremely hard to further reduce it.

The performance results conform with our intuition. When messages are uniformly distributed on the network, commonly used fixed routings may evenly vload network links. As a result, it is difficult for DFH to improve the T-Cost. For the cases where messages are nonuniform distributed or for specific applications, since message distribution often exhibits spatial locality, the fixed routing schemes are more likely to yield higher congestion. Thus, DFH can have more improvement in both maximum congestion and T-Cost.

6 Conclusions

The deadlock avoidance problem, which arises from wormhole routing, poses some challenging problems in the design of application specific routings. More sophisticated techniques need to be developed. Since channel dependencies cause deadlock, we formulate the problem of finding a low maximum congestion deadlock free routing as a graph theoretic problem on the Generic Physical Channel Dependency Graph of the original network. A simple heuristic is proposed. To avoid expensive updating of the channel dependencies, a modified transitive closure algorithm is used. The performance of the heuristic is studied. The algorithm converges very fast in all the testing sets. The heuristic has significant improvement for nonuniform message distribution as well as two specific applications and has moderate improvement for uniformly distributed messages over well known fixed routing schemes.

The application-specific wormhole routing generated by our heuristic can be used in advanced multicomputers. An example is Intel's iWarp multiprocessor system [BCC+90, Gro89]. In an iWarp system, the physical interconnection network is a torus. However, the interconnection can be logically reconfigured by setting up *pathways*. A default routing called *street sign routing*, which is essentially the XY routing scheme, is supported by the system. However, applications can change routes by generating necessary routing information in the header of a message.

References

[AHU74] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The design and analysis of computer algorithms,*. Addison-Wesley Publishing Company,, 1974.

[AS88] W.C. Athas and C.L. Seitz. Multicomputers:Message-passing Concurrent Computers. *IEEE Computer*, pages 9–23, August 1988.

[BCC+90] S. Borkar, R. Cohn, G. Cox, T. Gross, H.T. Kung, M. Lam, M. Levine, B. Moore, W. Moore, C. Peterson, J. Susman, J. Sutton, J. Urbanski, and J. Webb. Supporting systolic and memory communication in iWarp. In *Proceedings of the 17th Annual International Symposium on Computer Architecutre,*, pages 70–81, May 1990.

[BS87a] F. Berman and L. Snyder. On mapping parallel algorithms into parallel architectures. *Journal of Parallel and Distributed Computing*, 4(5):439–458, Oct. 1987.

[BS87b] B.P. Bianchini and J.P. Shen. Interprocessor traffic scheduling algorithm for multiprocessor networks. *IEEE Trans. Comput.*, C-36(4):396–409, Apr. 1987.

[Dal90] W.J. Dally. Performance analysis of k-ary n-cube interconnection networks. *IEEE Trans. Comput.*, C-39(6):775–785, June 1990.

[DS87] W.J. Dally and C.L. Seitz. Deadlock-free message routing in multiprocessor interconnection networks,. *IEEE Trans. Comput.*, C-36(5):547–553, May 1987.

[Gro89] T. Gross. Communication in iWarp systems. In *Proceedings of Supercomputing'89,*, pages 436–445, November 1989.

[Har72] F. Harary. *Graph theory,*. Addison-Wesley Publishing Company,, 1972.

[JMY89] C.R. Jesshope, P.R. Miller, and J.T. Yantchev. High performance communications in processor networks,. In *Proceedings of the 16th Annual International Symposium on Computer Architecutre,*, pages 150–157, June 1989.

[JR90] D.V. Judge and W.G. Rudd. A test case for the parallel programming support environment: parallelizing the analysis of satellite imagery data,. Technical Report, Dept. of CS, Oregon State University,, 1990.

[KS90] D.D. Kandlur and K.G. Shin. Traffic routing for multi-computer networks with virtual cut-through capability,. In *Preceedings of the 10th International Conference on Distributed Computer Systems,*, pages 398–405, May 1990.

[LH91] D.H. Linder and J.C. Harden. An adaptive and fault tolerant wormhole routing strategy for k-ary n-cubes. *IEEE Trans. Comput.*, C-40(1):2–12, January 1991.

[LN91] X. Lin and L.M. Ni. Deadlock free multicast wormhole routings in multicomputer networks,. In *Proceedings of the 18th Annual International Symposium on Computer Architecutre,*, pages 116–125, May 1991.

[LRG+90] V.M. Lo, S. Rajopadhye, S. Gupta, D. Kelsen, M.A. Mohamed, and J. Telle. OREGAMI: software tools for mapping parallel computations to parallel architectures. In *Proceedings of International Conference on Parallel Processing*, pages II:88–92, August 1990.

[LRM+] V.M. Lo, S. Rajopadhye, M.A. Mohamed, S. Gupta, B. Nitzberg, J. Telle, and X. Zhong. LaRCS: a Language for Regular Ccommunication Strucutres. Technical Report CIS-TR-90-16, Dept. of CS, University of Oregon,1990.

[RL90] W. Rudd and T.G. Lewis. Architecture of the parallel programming support environment,. In *Proceedings of CompCon'90,*, pages 589–594, Feb. 1990.

[SA91] S.B. Shukla and D.P. Agrawal. Scheduling pipelined communication in distributed memory multiprocessors for real-time applications,. In *Proceedings of the 18th Annual International Symposium on Computer Architecutre,*, pages 222–231, May 1991.

[Sar89] V. Sarkar. *Partitioning and scheduling parallel programs for multiprocessors.* The MIT Press, 1989.

[Tar83] R. Tarjan. *Data structures and network algorithms.* Society for Industrial and Applied Mathematics., 1983.