An Efficient Heuristic for Application-Specific Routing On Mesh Connected Multiprocessors

Xiaoxiong Zhong and Virginia Lo

CIS-TR-92-04
January 1992

Department of Computer and Information Science
University of Oregon

An Efficient Heuristic for Application-Specific Routing On Mesh Connected Multiprocessors *

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

Key Words: routing, multicomputer networks, deadlock avoidance, mesh, mapping.

Abstract

Wormhole routing has been widely used in practical systems such as iWarp and Ncube-2. To avoid potential deadlock, a physical network can be partitioned into several virtual networks to break possible cyclic dependencies. An n-dimensional mesh can be partitioned into 2^{n-1} virtual networks and, if messages are always routed through paths with shortest Manhattan distance, deadlock can be avoided. This paper studies the problem of finding low congestion deadlock free routing on mesh connected multiprocessors which support virtual networks. We show that finding such a minimal maximum congestion routing for an arbitrary set of messages is NP-hard for an n dimensional mesh ($n > 2$). An efficient heuristic is proposed and performance of the heuristic is evaluated.

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 in a multicomputer network [Sei90]. 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 mesh-connected multiprocessor system, the XY-routing scheme

*Supported by Oregon Advanced Computing Institute (OACIS) and NSF-grant CCR-8808532

always routes a message through the X (horizontal) direction and then the Y (vertical) direction. Another way is to provide an adaptive routing algorithm to reduce message traffic congestion dynamically while still preserving deadlock freedom [NS89].

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 for a specific application. This is especially undesirable for real time applications [SA91]. Fortunately, for many practical applications, the message requirements 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].

A way to avoid deadlock is to partition the physical network into virtual networks [DS87, LH91, LN91, JMY89] such that routings on the individual virtual networks are guaranteed to be free from deadlock. In particular, an n-dimensional mesh can be partitioned into 2^{n-1} virtual networks and, a message, depending on its source and destination addresses, is injected to a virtual network and is routed through a path of shortest Manhattan distance. However, there is still considerable freedom to choose which shortest path to route a message. In this paper, we show that, given the message passing structure at compile time, message routes can be chosen to considerably reduce message traffic congestion .

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 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 Virtual Network Partitioning on a Mesh

A way to determine whether a wormhole routing has a potential deadlock or not is to test whether its channel dependency graph is acyclic or not (see [DS87] for more detail). If the channel dependency graph has a cycle, a deadlock potentially exists. There have been many suggested methods for deadlock-free routing design [DS87, LH91, LN91, JMY89]. One effective method is to partition the physical network into several virtual networks such that a channel dependency cycle is split across several virtual networks and each virtual network is deadlock free. Since the virtual networks are multiplexed over time, the whole routing is also deadlock free. This provides more connectivity for a router than a fixed routing scheme such as XY-routing. The disadvantage of this approach is that it increases the complexity and cost of a router.

In [JMY89], a simple way is suggested to partition an n-dimensional mesh into 2^n virtual

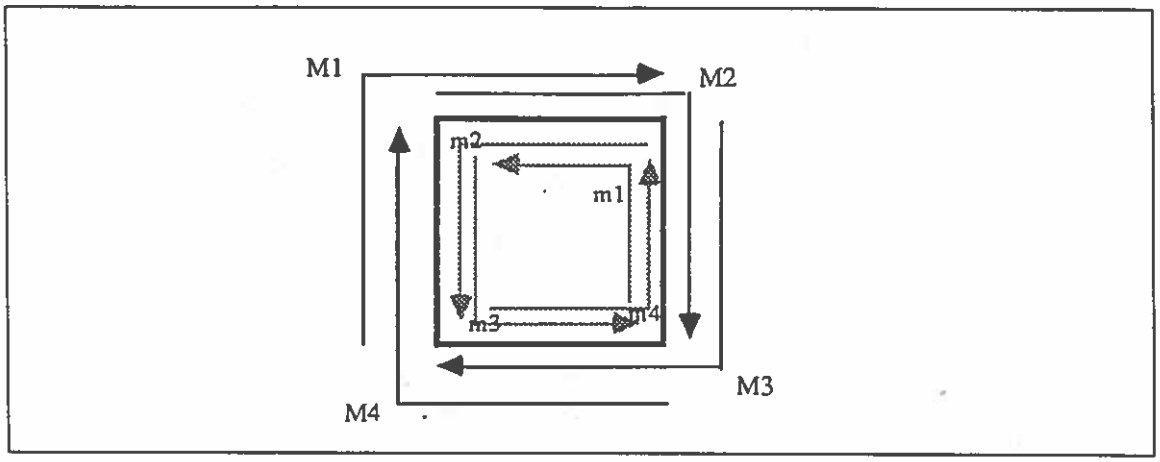


Figure 1: An example where Kandlur and Shin's algorithm generates a deadlocked wormhole routing

networks such that all Manhattan shortest paths * are captured. More precisely, suppose that G is a n dimensional mesh with nodes labeled with standard Cartesian coordinates. Clearly, a given channel lies in a single dimension. The direction of each channel in G has two possibilities, specified as 1 or -1. G is partitioned into 2^n virtual networks where each virtual network N is specified uniquely by a vector $(d_1, \ldots, d_n)$ where $d_i \in \{1, -1\}$. $N(d_1, \ldots, d_n)$ consists of all channels whose orientation is d_i in dimension i for each $i = 1, \ldots, n$.

Fact 1 It is easy to shown that the above partition covers all possible Manhattan shortest paths in an n dimensional mesh.

Fact 2 If a routing routes message through a Manhattan shortest path in one of the virtual networks, then it is deadlock-free. This is because the virtual networks *themselves* are acyclic and hence its corresponding channel dependency graph is also acyclic.

Furthermore, since channels in virtual network $N_1(d_1, \ldots, d_n)$ and virtual network $N_2(-d_1, \ldots, -d_n)$ are disjoint, combining N_1 and N_2 does not create any channel dependency cycles. Thus, we can further combine N_1 and N_2 into one single virtual network. This results in a total of 2^{n-1} virtual networks. For example, The four virtual networks for a 2-D mesh are shown in Fig. 2. (a) and (b), (c) and (d) can be further combined to form two virtual networks.

*A Manhattan shortest path has a shortest rectilinear distance between the source and the destination.

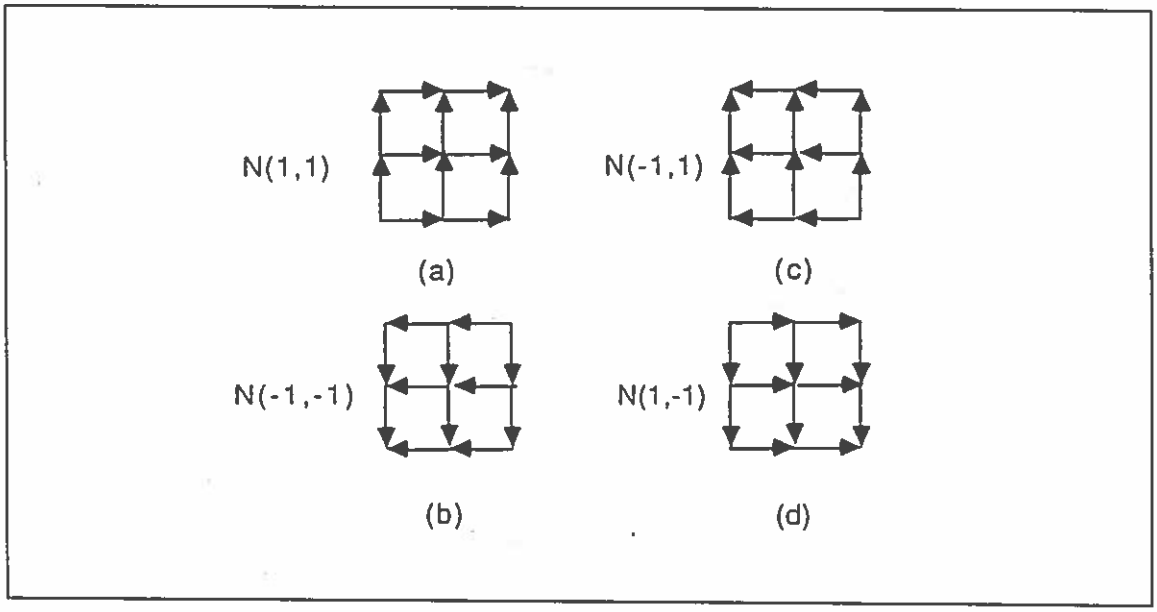


Figure 2: Partitioning of a 2-D Mesh into Four Virtual Networks

The drawback of the above partition scheme is that, as the number of dimensions grows, the number of virtual networks grows exponentially. The scheme is best for a low dimensional mesh. For example, for a two dimensional mesh, the number of virtual networks is only two, which can be implemented practically. In fact, in the newly released Intel iWarp system, two virtual networks are supported for a two dimensional torus, which is the physical configuration of the system [BCC+90, Gro89].

In next section, we formulate the main problem.

4 Application-specific Routing on a Partitioned Mesh

In many practical applications, task structures can be known a priori. This gives us an opportunity to fine tune the performance. For message routing, instead of using a system-supported default routing scheme such as XY-routing, we can design a routing based on knowledge of the message passing requirements off line to reduce message traffic congestion. For example, consider a matrix transpose application where processor (i, j) needs to exchange data with processor (j, i). If we use XY-routing, it will cause some unnecessary congestion. On the other hand, an off line routing can reduce the congestion dramatically. Fig. 3 shows

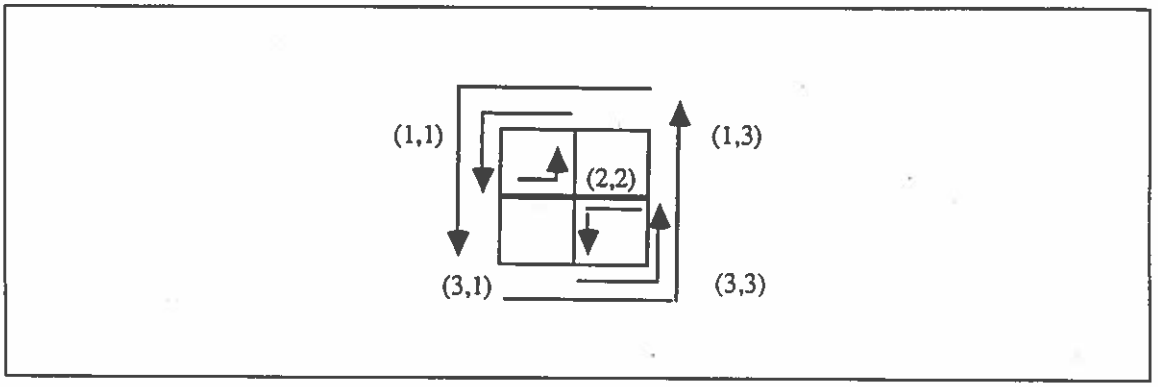


Figure 3: XY-routing for 3×3 matrix transpose

the XY-routing of a 3×3 matrix transpose. In fact, we can route the message from $(1,3)$ to $(3,1)$ through $(1,3),(1,2),(2,2),(2,1),(3,1)$ and the message from $(3,1)$ to $(1,3)$ through $(3,1),(3,2),(2,2),(1,2),(1,3)$. This gives a congestion-free routing.

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. For the purpose of this paper, we consider our target architecture to be an n dimensional mesh which is partitioned as above. Formally, we define a routing on a *partitioned* mesh as follows.

Definition 1 Let $A = (P, E)$ be a directed graph representing an n dimensional mesh where nodes correspond to processors and edges to communication channels in the physical network and it is partitioned into 2^{n-1} virtual networks as discussed in Section 3 . Let a set of messages be represented as M which is a *multiset* of pairs of nodes in P where $(s,d) \in M$ represents the source processor and the destination processor respectively for a single message . A **routing** on A is a function $\mathcal{R}$ which maps every pair $(s,d) \in M$ ($s = (s_1,\ldots,s_n)$ and $d = (d_1,\ldots,d_n)$) to a simple path $P = (p_0, p_1, p_2, \ldots, p_k)$ in A such that $p_0 = s, p_k = d$ and the length of P is $k = \sum_{i=1}^{n}(|s_i - d_i|)$. Such a routing is called a *minimal* routing in [BH85] and is abbreviated as **m-routing.**

It is possible that M is not a permutation. Furthermore, two messages with the same source and same destination addresses may be present in M (i.e., M is a multiset).

For a minimal routing, message collision is the predominant factor for performance. This is because all messages are routed through one of *shortest* paths and the effect of distance for the routing performance is minimized.

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 Optimization Problem: Given an n-dimensional mesh A, of size $m_1 \times m_2 \times \ldots \times m_n$, and a set of messages M to be routed, find a m-routing $\mathcal{R}$ such that $C(\mathcal{R})$ is minimal.

The Congestion-Free Decision Problem (CFD) is to decide whether there exists a m-routing $\mathcal{R}$ such that $C(\mathcal{R}) = 1$ for a given set of messages M.

We show, in the following, that this problem is NP-hard for an n-dimensional mesh when $n > 2$. In particular, we show that the decision problem can be polynomially reduced to the 3-Sat problem [GJ79]. The proof is based on the method used in [KS90] where the problem of finding a congestion-free routing (not necessarily minimal) on an *arbitrary* network (graph) is proved to be NP-complete.

Theorem 1 The congestion-free decision (**CFD**) problem is NP-complete for $n > 2$.

Proof: We show that the problem is NP-complete for three-dimensional meshes. The conclusion for higher dimension can be deduced easily.

The problem is clearly in NP since given a guess of a possible routing, we can easily verify whether the routing is congestion-free or not in polynomial time. We show that 3-Sat(isfiability) problem [GJ79] is polynomially reducible to **CFD**.

Let Θ be a proposition which is a conjunction of k disjunctive clauses with a total of m variables $x_1, \ldots, x_m$ in a 3-Sat problem. We construct a routing problem instance R_P on a three dimensional mesh A such that R_P has a congestion-free m-routing iff Θ is satisfiable.

We construct, in A, a subgraph A_1 which is crucial for the construction of R_P.

Figure 4 shows the construction of A_1. We denote a node in A by its coordinate (x, y, z) where the first node is $(1, 1, 1)$. Corresponding to the m variables, we construct m rectangles in plane $z = k + 1$. Each rectangle represents the occurrences of one variable

*In this paper, we assume the mesh has two directional channels for each edge and there is no traffic conflict between them. This assumption can be modified properly.

in Θ. The size of the i-th rectangles is $2L_i \times 2$ where L_i is the number of occurrences of both literal x_i and $\bar{x}_i$. Two corner nodes in the i-th rectangle are denoted as C_1^i, C_2^i respectively where, in the rectangle, C_1^i, C_2^i are the closest and farthest points to origin $(1,1,1)$ respectively. We call nodes with smaller y coordinates lower trail nodes and nodes with larger y coordinate upper trail nodes in a rectangle. We further use U_j^i to denote the node whose x-coordinate is the j-th smallest among all possible x coordinates in the upper trail of the i-th rectangle. In the same way, we label the j-th lower node as V_j^i. Notice that $C_1^i = V_1^i, C_2^i = U_2^{2*L_i}$. We intend to relate the jth-occurrence of x_i in Θ to edge $(U^i_{2*(j-1)+1}, U^i_{2*(j-1)+2})$ or edge $(V^i_{2*(j-1)+1}, V^i_{2*(j-1)+2})$, depending on whether x_i or $\bar{x}_i$ occurs. These rectangles are connected by a horizontal edge which connects C_2^i and C_1^{i+1}. Furthermore, we designate two special nodes S_F and D_F which will be used as the source and destination nodes for a message. We connect S_F to C_1^1 and, C_2^m to node D_F horizontally. Figure 4 shows the coordinates of all of nodes we have defined in A_1.

For the i-th clause ($i = 1, \ldots, k$), we designate two nodes S_i, D_i which are intended to be source and destination nodes for a message. S_i is placed in plane $z = i$ which is under plane $z = k+1$. D_i is placed in plane $z = k+i+1$, which is over plane $z = k+1$. Their coordinates are shown in Figure 4. Furthermore, we connect S_i and D_i by paths which pass through rectangles as follows. If the j-th occurrence of literal x_p or $\bar{x_p}$ is in the i-th clause C, then we construct a path from S_i to D_i by joining the follow path segments: S_i to $I_p = (2 + \sum_{u=1}^{i-1} L_u + 2*(j-1), 1, i)$ in plane $y = 1$, notice that the x-coordinate of I_p is the same as the x-coordinate of $U^i_{2*(j-1)+1}$ or $V^i_{2*(j-1)+1}$; I_p to $J_p = I_p + (0, \delta + p, 0)$ where $\delta = 1$ if this occurrence is x_p otherwise $\delta = 2$; J_p to $K_p = J_p + (0, 0, k-i+1)$; K_p to $K'_p = K_p + (1,0,0)$; Here, K_p and K'_p are $U^i_{2*(j-1)+1}, U^i_{2*(j-1)+2}$ or $V^i_{2*(j-1)+1}, V^i_{2*(j-1)+2}$, depending on δ. K'_p to $Q_p = K'_p + (0,0,i)$; Q_p to $R_p = Q_p + (0, m+3-\delta-p, 0)$; R_p to D_i. Here, δ is chosen such that, when the occurrence is x_p, the path passes through an edge in the lower trail of the rectangle and when the occurrence is $\bar{x_p}$, the path passes through an edge in the upper trail.

It can be verified that paths from S_i to D_i are disjoint from paths from S_j to D_j in A_1 for any $i \neq j$.

The largest coordinates in A_1 are from $D_k = (2 + \sum_{i=1}^{m} L_i, m+3, 2k+1)$. We choose the size of A as $(2 + \sum_{i=1}^{m} L_i) \times (m+3) \times (2k+1)$.

The routing problem R_P is constructed as follows: the message set M consists of a message from S_F to D_F, a message from S_i to D_i for $i = 1, \ldots, k$ and messages from a to b for any edge (a, b) in A but *not* in A_1 (we consider A to be a directed graph and every edge in a mesh corresponds to two directed edges in A).

We claim that the 3-Sat problem has a true assignment iff R_P has a congestion-free m-routing in A.

($\Rightarrow$) If the 3-Sat problem has a true assignment, the i-th clause (for any $i = 1, \ldots, k$) has at least one literal, say, x_p or $\bar{x_p}$ for the smallest index p which is true under the

assignment. By the construction of A_1, there is a path in A_1 from S_i to D_i which passes a path segment from K_p to K'_p in rectangle p. We choose this path as the route for the message from S_i to D_i. Since we choose K_p and K'_p consistently: if x_p occurs, we choose lower nodes; if $\bar{x_p}$, we choose upper nodes, either upper or lower nodes in a rectangle are chosen, but not both. Therefore, we can choose the spare upper trail or lower trail as one of path segments for message from S_F to D_F. For other messages whose source and destination nodes form an edge not in A_1, we simply choose this edge as the route for the message. This routing is a congestion-free m-routing.

($\Leftarrow$) Supposed that R_P has a congestion-free m-routing. First of all, notice that messages from S_i to D_i and message from S_F to D_F must be routed through edges in A_1 since for every edge (a, b) not in A_1, there is a message to be routed from a to b, which makes (a, b) the only possible route for a m-routing. This constrains the routes for the remaining messages to edges in A_1. For message from S_F to D_F to be unobstructed, only one trail (either lower part or upper part but not both) in any rectangle in plane $z = k + 1$ can be routed for messages from S_i to D_i. We construct a truth assignment as follows: for any $p = 1, \ldots, m$, if message from S_F to D_F is routed through upper trail in rectangle p, x_p is assigned **True**, otherwise, x_p is assigned **False**. Since there exists at least one unobstructed path P_i for message from S_i to D_i and P_i contains at least one edge in a rectangle, say, p, this means that clause i has an occurrence of literal x_p or $\bar{x_p}$. Furthermore, if P_i contains a lower edge in rectangle p, we know clause i has an occurrence of x_p, which has been assigned **True**. If P_i contains an upper edge in the rectangle, we know clause i has an occurrence of $\bar{x_p}$. However, in this case, x_p is assigned **False**. In both cases, the truth value of clause i is **True**. This concludes that the 3-Sat problem is satisfiable.

Since the above construction can be finished in polynomial time, we prove the theorem. ∎

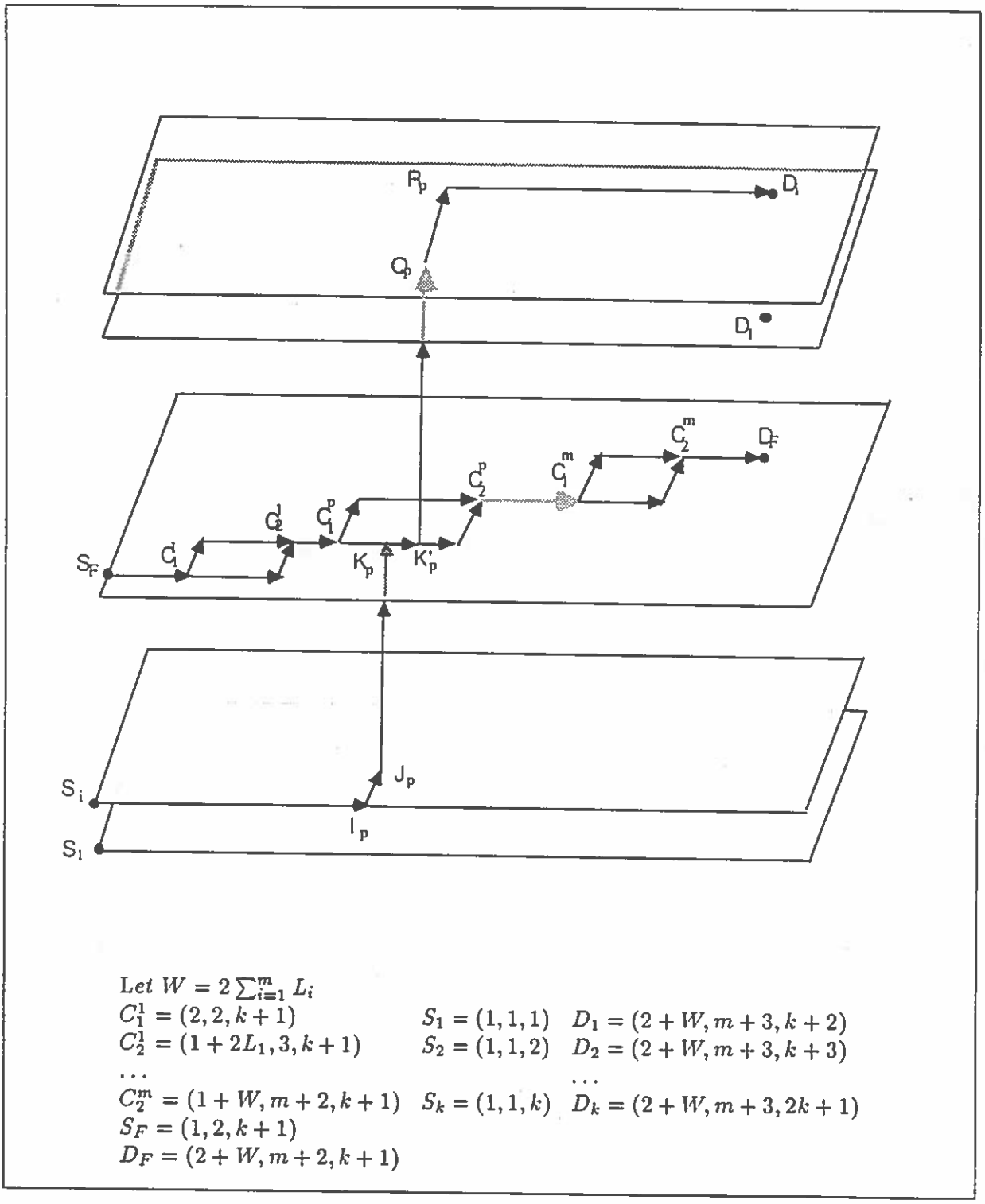
The case for a two dimensional mesh is unknown. We conjecture that it is still NP-complete for the congestion-free decision problem.

Therefore, the Optimization Problem is NP-hard.

5 A Heuristic Algorithm

It is therefore justified to design an efficient heuristic algorithm for the optimization problem. We present a simple and efficient heuristic algorithm *BLOCK*.

The idea behind heuristic *BLOCK* is that, in an n dimensional mesh, given a message to be routed from source node $s = (s_1, \ldots, s_n)$ to destination node $d = (d_1, \ldots, d_n)$, we know that it is going to be routed in a shortest Manhattan path from s to d. This means that it can only pass through the *directed* edges in the rectangle $\{(x_1, \ldots, x_n) | s_i \leq x_i \leq d_i\}$. We call the rectangle "an affected rectangle". Here, the direction of an edge in the rectangle is of the same



Let $W = 2\sum_{i=1}^{m} L_i$

$C_1^1 = (2, 2, k+1)$
$C_2^1 = (1+2L_1, 3, k+1)$
...
$C_2^m = (1+W, m+2, k+1)$
$S_F = (1, 2, k+1)$
$D_F = (2+W, m+2, k+1)$

$S_1 = (1, 1, 1)$
$S_2 = (1, 1, 2)$
...
$S_k = (1, 1, k)$

$D_1 = (2+W, m+3, k+2)$
$D_2 = (2+W, m+3, k+3)$
...
$D_k = (2+W, m+3, 2k+1)$

Figure 4: Illustration of The Proof of Theorem 1

direction from s to d. For example, if $s_i < d_i$, then only edges whose i-th component is increased will be considered. We can therefore associate a weight label for a directed edge in the mesh. The labeling scheme initially labels all edges as zero and then examines each message. When a directed edge is in the affected rectangle of the message, its label is increased by one. These labels represent the potential congestion resulting from routing of the messages. Fig. 6 shows the labeling of a 6×5 mesh. We also observe that, for a message to be routed from s to d, the number of shortest Manhattan paths can be calculated easily. This number is called **Freedom** for the message and is determined by the absolute differences of all components of s, d. Let $x_i = |s_i - d_i|, i = 1, \ldots, n$, we denote the freedom function as $F(x_1, \ldots, x_n)$. F can be calculated recursively as shown in Fig. 5.

$$
\begin{aligned}
F(x_1, \ldots, x_n) &= F(x_1 - 1, \ldots, x_n) + \ldots + F(x_1, \ldots, x_n - 1) \\
F(0, x_2, \ldots, x_n) &= F(0, x_2 - 1, \ldots, x_n) + \ldots + F(0, x_2, \ldots, x_n - 1) \\
&\ldots \\
F(x_1, 0, \ldots, 0) &= 1 \\
&\ldots \\
F(0, 0, \ldots, 0, x_n) &= 1
\end{aligned}
$$

Figure 5: Calculating freedom function

In particular, when $n = 2$, $F(x_1, x_2) = \begin{pmatrix} x_1 + x_2 \\ x_1 \end{pmatrix}$.

For a message, the smaller its **Freedom** value, the fewer paths are eligible for its routing. To route messages on a labeled mesh, *BLOCK* first sorts messages based on their **Freedom** values in an increasing order and then routes messages with lowest **Freedom** first. Routing an individual message is done by seeking a shortest Manhattan path such that maximum weight among all edges in the path is minimized. Such a path is called a shortest *minimum* path. We can use standard techniques for shortest paths such as Dijkstra's algorithm to find such a path (here, the distance of a path is defined as the maximum weight (label) of edges in the path) among all paths with shortest Manhattan distance. In fact, a more careful examination reveals that the subgraph formed by all shortest Manhattan paths for a message is an acyclic graph and hence, a more efficient shortest path algorithm for acyclic graphs can be used to find such a **minimum** path.

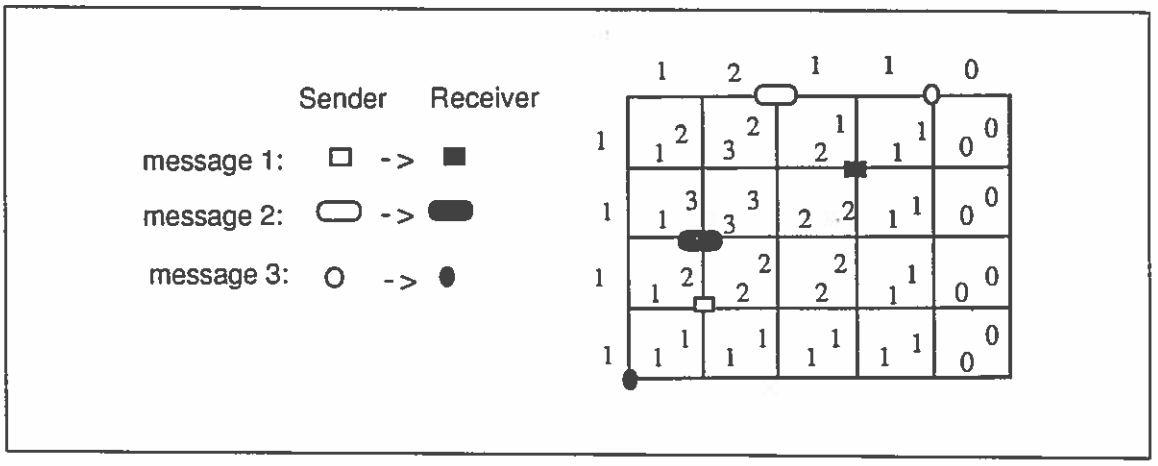


Figure 6: Illustration of the Labeling Scheme in $BLOCK$

After finding such a path P, labels of edges which are in the affected rectangle but not in P are decreased by one. The new labels serve for finding shortest *minimum* paths for the next message.

The outline of the algorithm $BLOCK$ is shown in Fig. 7.

The time complexity of $BLOCK$ can be analyzed as follows. Let A be an n dimensional mesh with N nodes. The number of edges it has is denoted as K. The number of messages is $|M|$. **Step 1** takes K time. The worst time for **Step 2** is $\mathcal{O}(|M|K)$. The time to calculate freedom for each message is also no more than $\mathcal{O}(|M|)N$ since we can compute $F(x_1,\ldots,x_n)$ in $x_1x_2\ldots x_n$ time steps based on the recursive relation in Fig. 5. For a fixed-size $N_1 \times \ldots \times N_n$ mesh, we can even calculate the freedom function off-line for all values $F(x_1,\ldots,x_n), x_i \leq N_i$ by the recursive relation in Fig. 5. The sorting step takes $|M|\log(|M|)$ steps. Finally, **Step 5** takes at most $|M|K$ time to complete since we can use the shortest path algorithm for acyclic graphs (see, page 203 in [Man89]) whose complexity is only $|E|$ where $|E|$ is the number of edges of the graph). This gives us the total complexity $\mathcal{O}(|M|(K+\log(|M|)))$. But since $K \leq nN$, the time complexity is $\mathcal{O}(|M|(nN+\log(|M|)))$.

6 Performance

Two test suites are used to evaluate $BLOCK$. In both tests, performance is evaluated with respect to the maximum congestion. The performance of $BLOCK$ is compared with that of XY fixed routing scheme, which has been used as a default routing scheme in many systems.

```
/* Input: Set of Messages M, n dimensional mesh A */
/* Output: A m-routing */

/* initialization */
1: for   edge e ∈ A {label(e)=0;}
/* labeling */
2: for   message m ∈ M {
         for edge e in the affected rectangle of m {label(e)++;} }
/* Freedom calculation */
3: for   message m ∈ M { F(m)=calculate-freedom(m);}
/* sorting based on freedom F */
4: M=sort(M);
/* routing a message */
5: for   message m ∈ M {
         find a shortest minimum path P for m in A;
         for edge e in the affected rectangle of m but not in P {label(e) − −;} }
```

Figure 7: Outline of Heuristic BLOCK

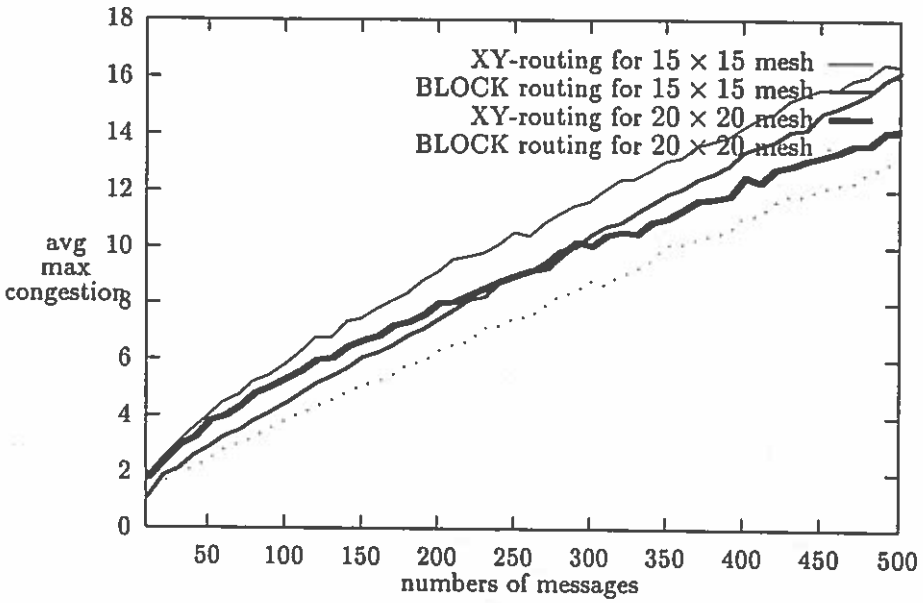


Figure 8: Average Maximum Congestion for 2-D 15 × 15 and 20 × 20 Meshes under Uniform Message Distribution

In the first test suite, messages are randomly distributed using a uniform distribution. Two independent uniform random number generators are used to generate source nodes and destination nodes respectively. The number of messages ranges from 10 to 500. The topologies used are 2-D 15 × 15 and 2-D 20 × 20 meshes. For a given number of messages, maximum congestion is averaged over 100 runs of *BLOCK*. It is observed that the standard deviation of the mean maximum congestion for all data point is less than 3%.

Fig. 8 shows the maximum congestions of the XY-routing and the routing generated by *BLOCK*. Fig. 9 shows the percentage improvement over XY-routing for maximum congestion in the two meshes. It can be seen that as the number of messages grows, the percentage improvement drops. This is because when more and more messages are injected to the network, the network is more and more saturated and it is harder for *BLOCK* to reduce the congestion.

The second test suite includes five benchmark applications representing a variety of task structures. The first is 16 node perfect broadcasting on a 2-D 4 × 4 mesh where processes (tasks) exchange information (such as identities) to achieve a consensus [Fin87]. The second is Gaussian elimination of a 32 × 32 matrix on a 2-D 8 × 4 mesh where a process is responsible for an entry of the matrix . The third one is the 15-body problem on a 2-D 5×3 mesh which was designed for the computation of planetary gravitational forces for the Caltech Cosmic Cube [AS88]. The fourth is an 18 node program called AVHTST on a 2-D 4 × 4 mesh which is used to determine cloud

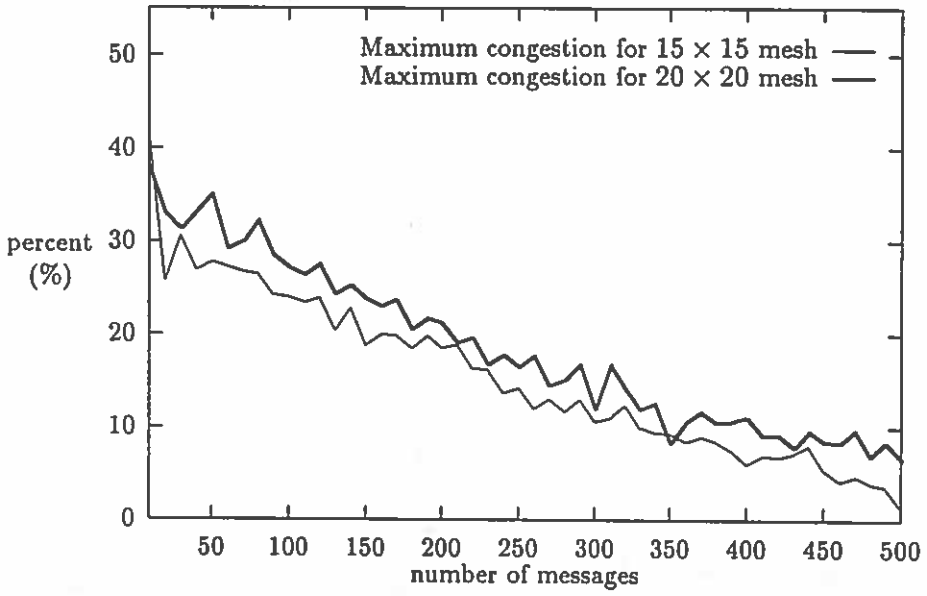


Figure 9: Percentage Improvement of Maximum Congestion for 2-D 15×15 and 20×20 Meshes Under Uniform Message Distribution

properties from satellite imagery data [RL90, JR90] and finally, matrix transpose for various sizes. The task structures of the above problems but the last are first assigned to the processors of meshes by a simple, greedy heuristic. For the matrix transpose problem, a 2-D mesh which has the same size of the matrix is used and a natural one-to-one processor assignment is used (i.e., the (i, j)-entry of the matrix is assigned to processor (i, j)).

Table 1 shows the simulation data for the first four applications. In all cases, there is significant improvement with respect to maximum congestion. Fig. 10 shows the percentage improvement of the maximum congestion of the routing produced by $BLOCK$ over that of XY-routing for matrix transpose. The sizes of matrices range from 10×10 to 19×19. It can be seen that an improvement of approximately 40% is achieved in all data points. It is also interesting to note that, in applications like Gaussian elimination and matrix transpose, message congestion is very heavy. In fact, in the 32 Gaussian elimination benchmark, the maximum congestion of XY-routing is 60 and, in 19×19 matrix transpose, the maximum congestion of XY-routing is 29. However, $BLOCK$ is still able to reduce the maximum congestion considerably (40 for Gaussian elimination and 20 for the matrix transpose). This is because in such applications, unlike in the case where messages are distributed *uniformly*, there exists communication locality for further improvement.

Applications	Maximum Congestion		
	XY-routing	*BLOCK*	Percentage
16 Perfect Broadcasting on 4×4 mesh	8	7	12.5%
32 Gaussian Elimination on 8×4 mesh	60	40	33.3%
15-body on 5×3 mesh	5	3	40%
18 node AVHTST on 4×4 mesh	7	6	14.3%

Table 1: Performance for the Applications

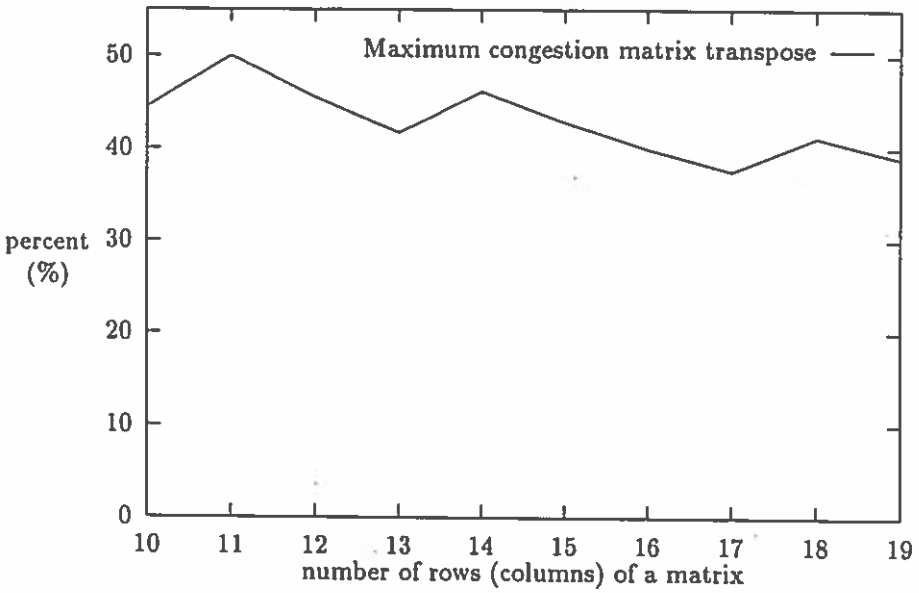


Figure 10: Percentage Improvement of Maximum Congestion for Matrix Transpose

7 Applications

The results presented here have several potential applications. First of all, the heuristic can be applied to the hypercube, torus and other interconnection structures. For example, it has been also pointed out in [LH91] that, to avoid deadlock, an n-dimensional binary cube can be partitioned into 2^n virtual networks (using the similar technique presented in Section 3). All possible shortest paths are captured in these virtual networks. A similar "affected area" concept can be used to label communication channels in the cube. Furthermore, although our original motivation is for a partitioned mesh using wormhole routing, the heuristic can be applied to other routing schemes such as virtual cut-through or store-forward, provided that we are seeking a minimal routing.

The above heuristic was developed within a project called **OREGAMI** which aims at providing a software environment and algorithms for mapping parallel computations to distributed memory machines. After tasks (processes) have been assigned to processors, for a target architecture whose routings can be controlled by the user, the heuristic can be used to find routes for messages. Architectures supporting user-controlled routing schemes include iWarp systems and Transputers. For example, in an iWarp system, pathways can be set up based on the routes generated [Gro89] and whenever a message need to be transmitted, it is injected to the proper virtual network and is routed through a pathway set up previously.

8 Conclusions

We have presented an efficient heuristic for application-specific wormhole routing in a partitioned mesh-connected multiprocessor system. Such a heuristic can also be used for other routing schemes such as virtual cut-through and store forward and for the hypercube and other well known topologies. The performance of the heuristic is studied for various benchmarks and uniformly distributed messages on the network. For all performance suites, the heuristic achieves very good performance. Such an application-specific routing technique can be used in many architectures and is especially useful for real-time applications which require high performance.

Future research includes applying the technique to practical applications in practical machines, improving the heuristic and analyzing its theoretical behavior. Finally, we are trying to prove that even for 2-D meshes, the congestion-free decision problem is still NP-complete. This seems to require a very different approach than the one we presented for n-dimensional meshes for $n > 2$.

References

[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.

[BH85] A. Borodin and J. Hopcroft. Routing,Merging, and Sorting on Parallel Models of Computation. *Journal of Computer and System Sciences*, 30:130–145, 1985.

[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.

[Fin87] R.A. Finkel. Large-grain parallelism - Three case studies. In Leah Jamieson, Dennis B. Gannon, and Robert J. Douglass, editors, *The Characteristics of Parallel Algorithms*, pages 21–64, Cambridge, Massachusetts, 1987. The MIT Press.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractability.* W.H.Freeman and Company, 1979.

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

[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.

[Man89] Udi Manber. *Introduction to Algorithms, A Creative Approach.* Addison-Wesley Publishing Company, 1989.

[NS89] J.N. Ngai and C.L. Seitz. A Framework for Adaptive Routing in Multicomputer Networks. In *Proc. of the 1989 ACM Symposium of Parallel Algorithms and Architectures*, pages 1–9, 1989.

[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.

[Sei90] C.L. Seitz. Concurrent Architectures. In Robert Suaya and Graham Birtwistle, editors, *VLSI and Parallel Computation,Chapter 1*, pages 1–84, San Mateo, California, 1990. Morgan Kaufmann Publishers, Inc.