RC 20644 (10/18/96)
Computer Science/Mathematics

Research Report

A Deadlock Avoidance Method for Computer Networks

Bulent Abali

IBM Research Division
T.J. Watson Research Center
P. O. Box 218
Yorktown Heights, New York

IBM Research Division
Almaden · T.J. Watson · Tokyo · Zurich

page intentionally left blank for double sided printing.

to appear in *Workshop on Comm. and Arch. Support for Network Based Parallel Comp. (CANPC'97)*

A Deadlock Avoidance Method for Computer Networks

Bulent Abali

IBM Thomas J. Watson Research Center,
P.O.Box 218, Yorktown Heights, NY 10598
abali@watson.ibm.com

Abstract. A deadlock avoidance method that minimizes the number of turn restrictions in wormhole routed networks is described. Fewer restrictions generally result in lower message latency and higher network bandwidth. The method is based on recursive partitioning of a graph model of the network and then removing interpartition edges to eliminate cycles. The method is applicable to a variety of network topologies including bidirectional multistage networks (BMIN), hypercubes, meshes, and irregular networks. In comparison to other algorithms, the method produces fewer or same number of turn restrictions on a set of benchmark networks.

1 Introduction

The main contribution of this paper is a deadlock avoidance method that minimizes the number of turn restrictions in wormhole routed networks. Fewer restrictions result in lower message latency and higher network bandwidth in general. Wormhole routing [1], a form of cut-through switching, is increasingly being used for clustering of workstations and in local area networks. Examples of wormhole routed networks are Tandem ServerNet [2] for clustering PCs, IBM SP [3, 4] for clustering RISC workstations, Myricom Myrinet, Fiber Channel Standard (FCS) switches, FDDI switches, and switched Ethernet [5].

An important issue in wormhole routed networks is deadlock avoidance. Deadlocks may occur whenever there is a cyclical dependency between message packets on the network links interconnecting the switches. Multiple packets block each other's path in the network and wait indefinitely for paths to be cleared. There are two main techniques for avoiding deadlocks in wormhole routed networks: virtual channels [1] and turn restrictions. Virtual channels eliminate cyclical dependencies by implementing multiple "virtual" channels passing through the same physical link. Virtual channels require replication of buffers in the switches and therefore they may increase switch complexity and delay (see [6] for example). Turn restrictions prohibit message packets from using particular input-output port combinations in particular switches in order to eliminate cyclical dependencies. The Turn Model [7], ServerNet [2], SP networks [3, 4, 8], the *e–cube* algorithm, and many other routing algorithms implement turn restrictions in some form. Turn restrictions serve a similar purpose to the "No Left Turn" traffic signs posted at street intersections. The signs prohibit some of the alternative routes, but at least the traffic will flow freely.

In this paper, we describe a method for minimizing the number of turn restrictions in wormhole routed networks. The method is based on a well known graph partitioning

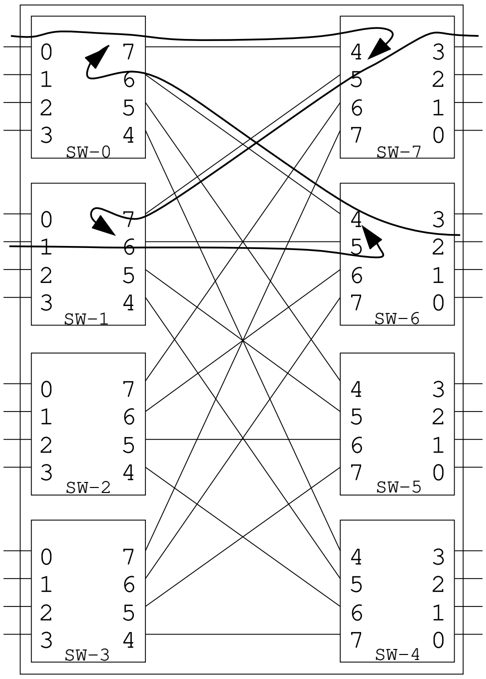

Fig. 1. A BMIN with 8 switches. An example cycle of four messages is also shown.

algorithm called Kernighan-Lin partition heuristic [9]. The method is applicable to many network topologies including bidirectional multistage networks (BMIN), hypercubes, meshes, and irregular networks. In comparison to other algorithms, the method produces fewer or same number of turn restrictions on a set of benchmarks, as summarized in Table 1. For example, it produces as few as 26 turn restrictions for the BMIN topology shown in Fig. 1, which compares well to the previously known 48 restrictions described in [8], and subsequently described 32 in [10].

2 Deadlocks

We will illustrate our method on a bidirectional multistage network (BMIN) shown in Fig. 1. We chose this BMIN, because it is one of the challenging topologies for which no fewer than 32 turn restrictions were published. However, the method does not depend on this particular topology and it can be used on other topologies as well. The BMIN topology shown in the figure is identical to the SP switch board [3, 4], which interconnects processors and/or other switch boards. The BMIN consists of eight 4×4 bidirectional cross–point switches. Each link interconnecting a pair switches is a full duplex bidirectional link, allowing simultaneous message transmission in both directions. All of the networks discussed in this paper have full duplex bidirectional links.

BMINs allow for *turn–around* routing where a message entering a switch from one side may turn around and leave the switch from the same side, as shown in Fig. 1. Deadlocks may occur since the head and tail of a message may span several switches. For example, four messages each represented by an arrow may enter the BMIN simultaneously as shown in Fig. 1. Each message wants to turn–around from particular switch, however finding its intended destination blocked by the tail of another message. Since no message can retreat, it will wait for the others to clear its path. Result is a deadlock in which the four messages wait indefinitely.

Typically, BMINs such as fat–trees avoid this deadlock problem by allowing turns in only one direction—turning only from "up" to the "down" direction [11], since processors are connected only to one end (i.e. side) of the tree. For example, Fig. 1 is a typical fat–tree for a 16 processor system with nodes connected along one side. However, it is possible to connect 32 processors along both sides of the BMIN to reduce the network cost, thus motivating our examination of the deadlock scenario of Fig. 1.

The deadlock in Fig. 1, in graph theoretic terms is a *cycle* of directed edges (arcs). A cycle is a contiguous sequence of input and output ports in the network, where the first and the last port is the same. The existence of a cycle in a graph may be determined by the well known *depth–first search* algorithm [12] that we will not describe here.

To avoid deadlocks we must ensure that set of routes used in the network are *acyclic*. One way to accomplish this is by prohibiting messages from turning around from a particular set of input/output ports in particular switches. For example, the cycle in Fig. 1 can be avoided by putting a "no turn" restriction at one of the four corners from which messages turn. Turn restrictions for the SP networks are first described in [8]. Horst describes turn restrictions for ServerNet in [2]. Glass and Ni describe turn restrictions for meshes in [7]. Qiao and Ni describe turn restrictions for irregular topologies in [5]. Note that Duato [13] describes sufficient conditions for deadlock-free routing that allows cyclic dependencies in adaptive networks, if there exists a connected acyclic channel subset in the network. However, Duato's proposal is not applicable to non-adaptive networks.

Note that there 12 different ways to make an internal turn in each switch, excluding entering and exiting from the same bidirectional port. Here, an internal turn refers to the ports 4–7 of the BMIN in Fig. 1. We make the assumption that every processor eventually receives its messages. Hence, final destinations –external ports 0, 1, 2, and 3– cannot participate in any cycles, and we will not consider them in further discussions. There are 8 switches in the BMIN, and therefore the total number of internal turns is 96.

Exhaustively searching for a minimal set of turn restrictions is a difficult task, since there are 2^{96} possible ways to restrict the turns. In the method of [8] one column of switches, either the left or the right column of switches allow for no turns at all (either the switches 0,1,2,3 or 4,5,6,7 in Fig. 1). This method restricts a total of 48 turns out of a possible 96 turns. The number of restrictions are subsequently reduced to 32 [10].

3 Deadlock Avoidance by Graph Partitioning

This paper's method of finding the turn restrictions is based on recursive partitioning of a graph model of the network and then removing inter-partition edges to eliminate cyclic

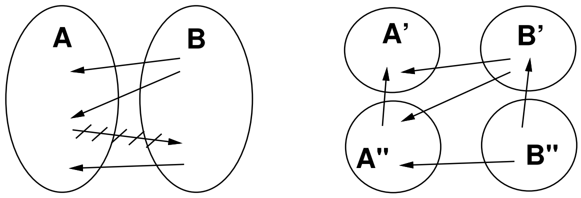

Fig. 2. Finding restrictions by graph partitioning and eliminating cycles.

```
FIND_RESTRICTIONS(G(V, E))
input: graph representation of the network
output: a set of turn restrictions
1     CYCLE_DETECT(G)
      if no cycles in G then
            return empty set.
      endif
2     Partition V in to set of vertices A and B such that
            A and B have nearly equal number of vertices
            AND the number of arcs from A to B is minimum
            AND partition constraints are satisfied.
3     Remove the set of arcs TR from A to B
4     TR ← TR + FIND_RESTRICTIONS(G_A)
5     TR ← TR + FIND_RESTRICTIONS(G_B)
6     return TR
```

Fig. 3. The pseudo-code for finding turn restrictions

dependencies, as illustrated in Fig. 2. We used the Kernighan-Lin heuristic procedure for graph partitioning [9], although other graph partitioning algorithms may be used as well. The objective of graph partitioning algorithms is to minimize the cut size, which is the number of edges between the partitions. This is the essence of our method. Smaller cut size leads to fewer turn restrictions.

A directed graph representation $G(V, E)$ of the network is used where graph vertices V represent the *network links*, and each directed edge (arc) in E represents an *input–output connection* in a switch. For example, 4 bidirectional internal ports of the BMIN is represented by 4 vertices representing switch input links, and 4 vertices representing switch output links, and a total of 12 arcs from 4 input vertices to 4 output vertices representing every possible connection. We assume that messages will not enter and exit from the same bidirectional port, since it serves no useful purpose.

The method is described by the pseudo–code in Fig. 3. In Step 2, vertices of the graph are partitioned in to 2 equal size vertex sets A and B (Fig. 2). Here, the most important issue is minimizing the cut size: We must split vertices V so that the number of arcs from set A to set B ($A \rightarrow B$) is as small as possible. The cut size of $B \rightarrow A$ is not important. We describe in Section 3.1 how cut size can be minimized.

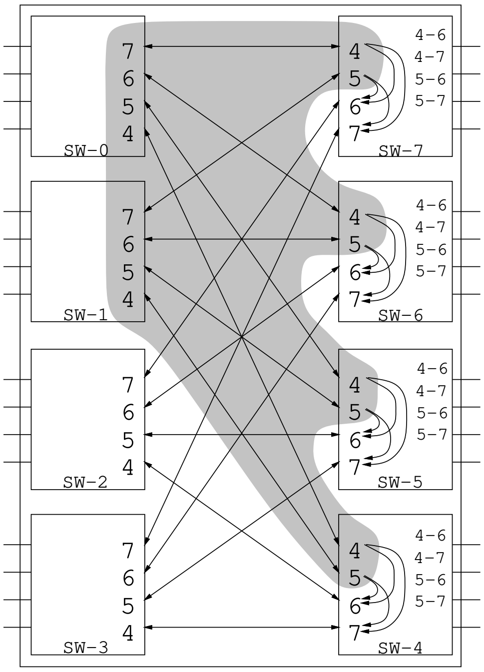

Fig. 4. Arcs removed between the two subgraphs to avoid cycles.

In Step 3, the arcs $A \rightarrow B$ are removed from the graph (indicated by dashed arrows Fig. 2). The removed arcs correspond to turn restrictions, since each arc represents an input–output connection in a switch. Since the remaining arcs between A and B are only in one direction ($B \rightarrow A$), no cycle spanning both A and B can exist. This observation is a key to our method. The two subgraphs A and B are then recursively partitioned (Steps 4,5) and more arcs are removed until no subsubgraph with a cycle (Step 1) remains in the network.

We now illustrate the method on the BMIN shown in Fig. 1. The BMIN is partitioned in to two set of vertices (e.g. links) shown in Fig. 4. Links in the shaded area represent one partition and links in the unshaded area represent the other partition. The arcs from one subgraph (shaded) to the other subgraph (unshaded) are removed. We represent the removed arcs with arrows in Fig. 4. This is the first partition, and it results in 16 turn restrictions: namely (input $\rightarrow$ output) $4 \rightarrow 6$, $4 \rightarrow 7$, $5 \rightarrow 6$, $5 \rightarrow 7$ in switches 4, 5, 6, and 7. Messages may not enter and exit from the restricted input–outputs. The first subgraph is further partitioned in to two subsubgraphs, and the arcs $4 \rightarrow 5$ in switches 4, 5, 6, and 7 are removed. Likewise, the second subgraph is partitioned and the arcs $6 \rightarrow 7$ are removed. The final result has a total of 24 turn restrictions $4 \rightarrow 5$, $4 \rightarrow 6$, $4 \rightarrow 7$, $5 \rightarrow 6$, $5 \rightarrow 7$, $6 \rightarrow 7$ in switches 4, 5, 6, 7 as shown in Fig. 5. This is a better result than the 48 restrictions described in [8] and 32 restrictions described in [10].

There are few noteworthy observations in Fig. 5, (1) although the turn restrictions on all 4 switches are identical in this particular example, this may not happen in general,

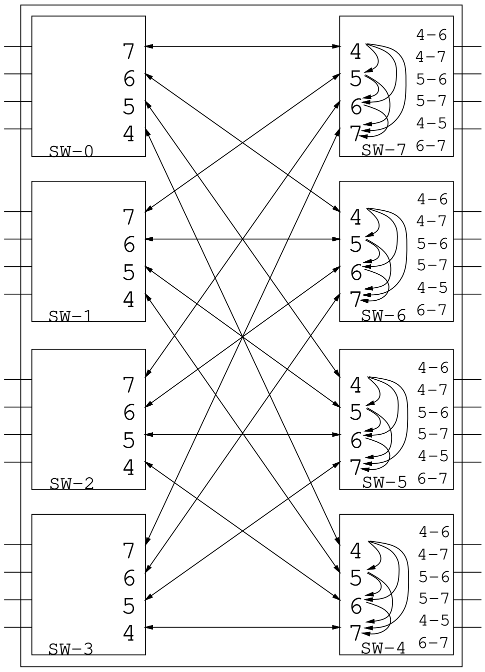

Fig. 5. The entire set of turn restrictions found by this paper's method. Restrictions are represented by arrows

(2) the 24 turn restrictions make some switches unreachable. For example, switch 3 is unreachable from switches 0, 1, 2. To deal with this problem we may use *partitioning constraints* which we will discuss later in Section 4.4.

3.1 Partitioning Step

The graph partitioning is an NP–complete problem. However, many good heuristics exist for graph partitioning. We used the well known Kernighan–Lin (KL) [9] heuristic in the partitioning step (Step 2). KL has been extensively used in many applications including VLSI circuit design, linear programming, and sparse matrix computations. We modified the KL algorithm to handle directed graphs and to reduce the cut size only in one direction. We will only present a sketch of the algorithm here. Details can be found in [9].

The modified KL algorithm starts with a randomly selected initial partition of two vertex sets A and B. The objective of the algorithm is to find two subsets A' and B' of same size (where $A' \subset A$, $B' \subset B$), such that exchanging the two subsets between A and B reduces the cut size. Here, cut size is defined as the number of arcs incident from set A to set B. After two subsets A', B' are exchanged between A and B, the procedure is repeated iteratively until no further reduction in cut size can be obtained. We assume that arcs do not have any associated weights. For characterizing heterogenous networks with differing switch and link capacities, one may use weighted arcs as well. Then, the cut size may be defined as the sum of the weights of arcs incident from set A to set B.

Given two randomly selected initial sets A, B, and $n = min(|A|, |B|)$, we determine the two vertices $a_1 \in A$ and $b_1 \in B$ such that exchanging them would correspond to the maximum possible reduction in cut size of $A \rightarrow B$. Call this reduction *the partial gain* g_1. In [9] it is described how maximum partial gain can be determined fast. We temporarily exchange vertices a_1 and b_1 between sets A and B, and "lock" them meaning that they will not be moved again in this iteration.

We now determine the two vertices $a_2 \in A - \{b_1\}$ and $b_2 \in B - \{a_1\}$ such that exchanging them would correspond to the maximum possible gain g_2 in cut size of $A \rightarrow B$. Continuing in this manner, we identify all the vertex pairs $(a_3, b_3), (a_4, b_4), \ldots (a_n, b_n)$ and their respective gains $g_3, g_4, \ldots, g_n$.

We now determine the k that maximizes the sum $G = \sum_{i=1}^{k} g_i$. If total gain $G > 0$, we then commit the vertex exchanges $(a_1, b_1), \ldots (a_k, b_k)$, and we undo the exchanges $(a_{k+1}, b_{k+1}), \ldots (a_n, b_n)$, i.e. we move back the vertex pairs for steps $k + 1$ through n to their original set. The procedure is then repeated with the newly formed sets A and B. However, if total gain $G \leq 0$ it means that we arrived at a local minimum in the previous iteration and we cannot further reduce the cut size. We undo all the exchanges for $i = 1, \ldots, n$, and terminate the procedure.

Few observations: (1) A partial gain g_i may be negative due to lack of a better choice. However, this may lead to positive gains g_j later $(j > i)$, resulting in a positive total gain G. Therefore, the algorithm can climb out of some local minimums. (2) Cut size decreases monotonically. (3) Due to the randomized initial selection of vertex sets, we must execute the algorithm several times, and use the best result.

Algorithm – Topology	BMIN	3-cube	Irregular	4x4 Mesh
This paper's method	26	12	14	
SP32 (Sethu et al [8])	32			
SP48 (Sethu [10])	48			
3-cube ex. (Horst [2])		12		
ET (Qiao and Ni [5])			37	
ET* (see text)	36		36	
Turn Model (Glass and Ni [7])				18
e-cube (dimension ordered)		24		36
Lower Bound (see text)	24	12	14	18

Table 1. Number of turn restrictions produced by several algorithms on benchmark networks.

4 Results

We compared our method to several algorithms on a set of network topologies. Table 1 compares the number of turn restrictions determined by various algorithms. All of the networks discussed here have full duplex bidirectional links interconnecting the switches and processors.

Switch	Turn Restrictions
0	7→4, 7→5
1	6→7
2	5→4, 6→4, 6→7, 7→4
3	5→4, 5→6, 5→7, 6→4, 6→5, 6→7
4	4→7, 5→4, 5→6, 5→7
5	5→4, 5→6, 6→4, 6→5
6	5→4, 6→4
7	4→7, 5→7, 6→7

Table 2. The 26 BMIN turn restrictions produced by this paper's method.

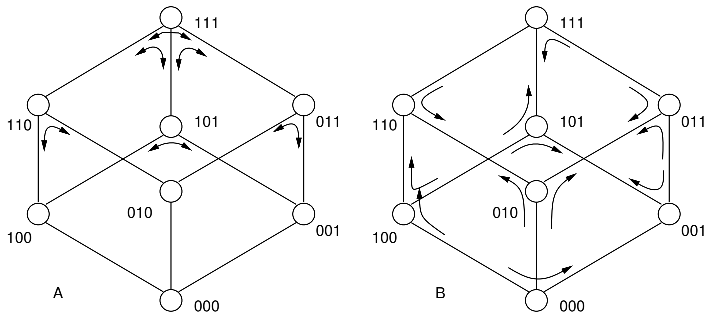

Fig. 6. Two different sets of turn restrictions for a 3-dim hypercube

4.1 BMIN

For the BMIN topology shown in Fig. 1 implementation of our method produced a minimum of 26 turn restrictions shown in Table 2. This is the best result obtained out of a few hundreds runs, each started with a different randomly selected initial partition. Note that this result has every switch reachable from every other switch unlike the 24 restriction example shown in Fig. 5. Note that we derived a lower bound of 24 turn restrictions for the BMIN in Fig. 1 using the method described in Section 4.2.

4.2 Hypercube

Horst [2] gives an example of a 3 dimensional hypercube with 12 turn restrictions when describing Tandem ServerNet, as shown in Fig. 6A. The number of restrictions are distributed as 0,0,0,2,0,2,2,6 in nodes 0–7, respectively. Turn restrictions are represented by arrows in the figure.

Application of our method also gives 12 restrictions, as shown in Fig. 6B. The number of restrictions are distributed as 1,1,2,2,2,2,1,1 in nodes 0–7, respectively. The distribution of turn restrictions is an issue of concern. With evenly distributed restrictions, one would expect the network to perform better in terms of latency and bandwidth. From that point of view, Fig. 6B is a better solution than Fig. 6A.

Note that for a 3–cube, 12 turn restrictions is the necessary minimum for a cycle free network. We can determine a lower bound for any network by counting *turn–disjoint* cycles. Turn–disjoint means that no pair of cycles in the counted set share any turns. For example, in a hypercube no two cycles of length 4 turn from the same corner, hence they are turn-disjoint: The 3–cube has 6 faces (each face is a 2–subcube). Each one has 2 cycles of length 4 along its perimeter. One cycle runs in the clockwise direction and the other runs in the counter–clockwise direction. Each cycle can be eliminated with one turn restriction. Therefore a total of 12 turn restrictions is necessary in a 3–cube. We can generalize this reasoning to n dimensional hypercubes. There are $2^{n-3}n(n-1)$ two subcubes in n-cube, with each 2-subcube having two cycles in opposite directions. Therefore, $2^{n-2}n(n-1)$ turn restrictions are the minimum necessary.

Note that the well known *e-cube* routing algorithm, which is also known as *dimension ordered routing* allows routing of messages from smaller dimension to larger dimension. Therefore, *e-cube* effectively implements $n(n-1)/2$ turn restrictions per node in $n-$cube. That is 24 turn restrictions total in a 3-cube. Table 1 summarizes the results.

4.3 Irregular Topology

Qiao and Ni [5] describe an algorithm for finding turn restrictions in irregular network topologies. Their algorithm is based on finding an Eulerian Trail (ET) in the network. If messages are sent along an ET, then no deadlocks can occur since the ET is cycle free by definition. Then "shortcuts" in the ET are found in order to reduce the number of turn restrictions. Qiao and Ni [5] demonstrate the ET algorithm on the topology shown in Fig. 7. The ET algorithm produces 37 turn restrictions for this topology. We describe here the algorithm ET* inspired by ET. ET* is easier to describe and implement. ET* produces 36 turn restrictions for this particular example. In comparison, our graph partitioning based approach produces 14 turn restrictions. (This is the best result obtained out of a few hundred runs.) Results are summarized in Table 1. The turn restrictions produced by our method are shown in Fig. 7 by arrows. Note that 14 turn restrictions are the minimum necessary for this topology. This lower bound is determined by counting turn–disjoint cycles in the network, as described in Section 4.2.

We now describe the ET* algorithm. A *trail* is a sequence of unidirectional links of the network in which all the links in the sequence are distinct. An *Eulerian trail* is a trail that contains all the links of the network. For example, Fig. 8 shows an Eulerian trail for the network in Fig. 7. The notation ij denotes the link from node i to node j. By definition an Eulerian trail has no cycles, hence messages sent along the trail cannot deadlock. For example, processor 0 can send a message to processor 1 along the trail $05 \rightarrow 54 \rightarrow 43 \rightarrow 38 \rightarrow 71$ (each processor i not shown in the figure is connected to switch i.)

Although the method routing messages along the trail is deadlock-free, it does not take advantage of the fact that switch 0 is directly connected to switch 1, hence there exists a *shortcut* to the intended destination. Shortcuts are created by adding arcs to the trail. A message may take a shortcut if does not cause any cycles. In the ET* algorithm, we allow a shortcut only if it is in the direction of the Eularian trail (forward shortcut). We do not allow backward shortcuts in order to avoid possible cycle(s). We define a shortcut from ij to jk as forward, if ij is before jk in the trail, otherwise it is a backward

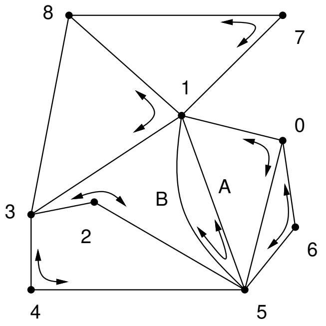

Fig. 7. Irregular topology of Qiao and Ni, and the turn restrictions determined by this paper's method

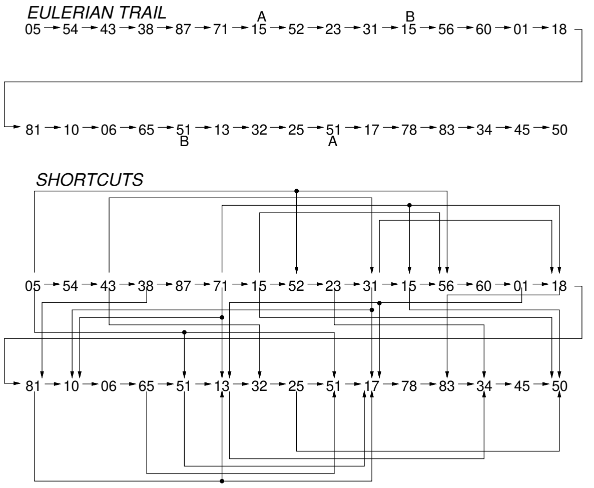

Fig. 8. A Eulerian trail and forward shortcuts added to the trail

shortcut. For example, the shortcut $05 \rightarrow 52$ is allowed. However, the shortcut $45 \rightarrow 52$ is not permitted since it is a backward shortcut that creates a cycle.

The arcs of the final graph represent the permitted turns. For example, $05 \rightarrow 52$ in Fig. 8 means that, switch 5 can forward messages incoming from switch 0 to switch 2. Arcs missing from the graph are the turn restrictions. For example, $45 \rightarrow 52$, missing from the figure, indicates that switch 5 can not forward messages from switch 4 to switch 2. By counting the missing arcs, we arrive at the figure of 36 turn restrictions. Note also that application of ET* algorithm to the BMIN in Fig. 1 produces 36 turns restrictions. Results are summarized in Table 1.

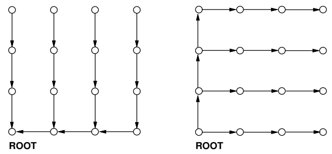

Fig. 9. Up and down trees for identifying turns that should not be removed

4.4 2-D Mesh (Constrained Partitioning)

We applied our graph partitioning based method to 2-D meshes as well, since meshes are one of the well known and studied network topologies. We first describe related work and then we describe the constrained partitioning procedure.

The *e-cube* routing algorithm prevents deadlocks by routing the messages along the x-axis first and then y-axis next. Thus, *e-cube* allows East to South turns (ES), and WS, EN, WN, EW, WE, NS, SN turns. The NE, NW, SE, and SW turns are not permitted. The *turn model* of Glass and Ni [7] implements fewer turn restrictions. In the "West First" turn model, the NW and SW turns are not permitted. In an $n \times m$ processor mesh the turn model requires $2(n-1)(m-1)$ turn restrictions. This is also the minimum number necessary for meshes which we can determine by counting the turn disjoint cycles, as described in Section 4.2

Our method, when applied without constraints, does not work at all for a mesh. The turn restrictions always make one or more nodes unreachable from another node, just as was the case for the BMIN in Fig. 5. In a 4×4 mesh, multiple runs with different initial partitions always resulted in 10–15% of the node pairs disconnected. The best way to avoid a cycle is to disconnect the network elements, which our method does very well in this case. Of course, this defeats the purpose of having a network. Due to its relatively long diameter, a mesh is disconnected easily when partitioned without constraints. We thus devised a constrained partitioning procedure to solve this problem.

Before partitioning we designate some turns as "cannot be restricted", so that there always exists a path between every node pair in the network, i.e. the graph stays connected. Then, during the partitioning step (Section 3.1) we ensure that those arcs (i.e. turns) do not appear in the cut from vertex set A to B.

The graph of constrained arcs (turns) should be acyclic, and the following strategy is devised to identify those arcs. A "down" spanning tree of the network and an "up" spanning tree of the network rooted at one node is created, such as shown in Fig. 9 for a 4×4 mesh. Every node is reachable from every other node by going up in the "up tree" towards the root, and then going down in the "down tree" towards the final destination. The turns used in the two trees are marked as "cannot be restricted", and then the partitioning step is applied. A tree is cycle free by definition, therefore constrained arcs cannot form a cycle.

5 Conclusions

In this paper we described a deadlock avoidance algorithm for minimizing turn restrictions in wormhole routed networks with arbitrary topology. The method is based on recursive partitioning of the channel dependence graph of the network and removing interpartition edges to make the graph acyclic. We used the KL graph partitioning algorithm to minimize the cut size between partitions, i.e. to minimize the number of turn restrictions.

We compared the method to existing algorithms on a small set of benchmark networks, and observed that fewer turn restrictions are usually obtained. We have preliminary (not presented here) results—simulated and measured on real systems—that show that fewer turn restrictions lead to lower latency and higher bandwidth. Further simulations of various networks are needed to validate this relationship and quantify performance improvement.

A strength of our method is that it does not require new hardware, switch design, or topology. The method can be easily applied to existing networks by changing the route tables.

Acknowledgements

We are grateful to anonymous reviewers for their valuable comments and criticism, Caroline Benveniste for preliminary results of simulation, and Mike Rosenfield for his support.

References

1. W. J. Dally and C. L. Seitz, “Deadlock-free message routing in multiprocessor interconnection networks,” *IEEE Trans. on Computers*, vol. C-36, pp. 547–553, May 1987.
2. R. Horst, “ServerNet Deadlock Avoidance and Fractahedral Topologie0s,” in *Proc. 10th Int. Parallel Processing Symp. (IPPS’96)*, pp. 274–280, April 1996.
3. C. B. Stunkel et al, “The SP2 High-Performance Switch,” *IBM Systems Journal*, vol. 34, no. 2, pp. 185–204, 1995.
4. B. Abali and C. Aykanat, “Routing Algorithms for IBM SP1,” *Lecture Notes in Computer Science, Springer–Verlag*, vol. 853, pp. 161–175, 1994.
5. W. Qiao and L. M. Ni, “Adaptive routing in irregular networks using cut–through switches,” in *Proc. 25th Int. Conf. Parallel Processing (ICPP)*, August 1996.
6. A. A. Chien, “A cost and speed model for k-ary n-cube wormhole routers,” in *Proc. Hot Interconnects’93*, August 1993.
7. C. Glass and L. M. Ni, “The turn model for adaptive routing,” in *Proc. 19th Int. Ann. Symp. Computer Architecture*, pp. 278–287, 1992.
8. H. Sethu, R. F. Stucke, and C. B. Stunkel, “Technique for accomplishing deadlock free routing through a multi–stage cross–point packet switch.” U.S. Patent 5,453,978, issued 9/26/1995.
9. B. W. Kernighan and S. Lin, “An efficient heuristic procedure for partitioning graphs,” *Bell System Tech. J.*, vol. 49, pp. 291–307, 1970.
10. H. Sethu, “Routing Restrictions.” unpublished.
11. I. D. Scherson and C.-H. Chien, “Least Common Ancestor Networks,” in *Proc. 7th Int. Parallel Processing Symp.*, pp. 507–513, 1993.
12. T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. NY: McGraw-Hill, 1990.
13. J. Duato, “A new theory of deadlock-free adaptive routing in wormhole networks,” *IEEE Trans. Parallel and Distributed Systems*, vol. 4, no. 2, pp. 1320–1331, 1993.