Available online at www.sciencedirect.com
ScienceDirect
Journal of Systems Architecture 53 (2007) 619–628
JOURNAL OF SYSTEMS ARCHITECTURE
www.elsevier.com/locate/sysarc

A heuristic fault-tolerant routing algorithm in mesh using rectilinear-monotone polygonal fault blocks

Dajin Wang *

Department of Computer Science, Montclair State University, Upper Montclair, NJ 07043, USA

Received 5 October 2005; received in revised form 18 August 2006; accepted 22 December 2006
Available online 8 January 2007

Abstract

A new, rectilinear-monotone polygonally shaped fault block model, called Minimal-Connected-Component (MCC), was proposed in [D. Wang, A rectilinear-monotone polygonal fault block model for fault-tolerant minimal routing in mesh, IEEE Trans. Comput. 52 (3) (2003) 310–320] for minimal adaptive routing in mesh-connected multiprocessor systems. This model refines the widely used rectangular model by including fewer non-faulty nodes in fault blocks. The positions of source/destination nodes relative to faulty nodes are taken into consideration when constructing fault blocks. Adaptive routing algorithm was given in Wang (2003), that constructs a minimal "Manhattan" route avoiding all fault blocks, should such routes exist. However, if there are no minimal routes, we still need to find a route, preferably as short as possible. In this paper, we propose a heuristic algorithm that takes a greedy approach, and can compute a nearly shortest route without much overhead. The significance of this algorithm lies in the fact that routing is a frequently performed task, and messages need to get to their destinations as soon as possible. Therefore one would prefer to have a fast answer about which route to take (and then take it), rather than spend too much time working out an absolutely shortest route.

Keywords: Adaptive routing; Fault block model; Fault tolerance; Interconnection network; Mesh

1. Introduction

The *mesh* structure is one of the most important interconnection network models. As a topology to interconnect multiprocessor computer systems, it has been shown to possess many attractive properties. It has been one of the most favored interconnection structures to manufacturers, and parallel computers using mesh or its variants have been commercially available for quite a long time [6,14,16]. The interest in this simple yet advantageous structure has been sustaining. New topologies based on mesh still appeared in recent literature [9]. Because of its importance to achieving high performance, fault-tolerant computing for mesh structures has been the focus of an extensive literature. A very important aspect of mesh fault tolerance is its ability to route from a source node to a destination, avoiding all faulty nodes. Routing is a process to send *messages*, which can be either data or instructions,

* Tel.: +1 973 655 7615.
E-mail address: wang@pegasus.montclair.edu

1383-7621/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.sysarc.2006.12.005

from a source node to a destination node, passing some intermediate nodes. There are basically two types of routing: *deterministic* routing and *adaptive* routing. In deterministic routing, a fixed path is used to send/receive messages for a particular pair of source/destination. The obvious advantages of this routing are its simplicity and ease of implementation. However, it suffers the shortcoming of weak fault tolerance. When one or more nodes on the dedicated path fail, routing cannot be carried out. In adaptive routing, on the other hand, there is no dedicated path for a pair of source and destination. The path is adaptively constructed in the process of routing. In fully adaptive routing, messages can take any fault-free intermediate nodes to reach destination. As a result, adaptive routing can tolerate faulty nodes by taking detours.

A natural goal in adaptive routing is to find a route as short as possible in the presence of faulty nodes, preferably the minimal route. Fault-tolerant routing has been studied extensively [1–5,7,8,10–12,15,17,18,20,21]. In fault-tolerant routing on mesh, most work uses rectangular fault block model [1,2,4,13,17,21]. In rectangular model, all faulty nodes are grouped in disconnected, rectangular areas, called *fault blocks*. A fault block is constructed in a way that includes as few non-faulty nodes as possible while maintaining rectangular shape. All nodes in these fault blocks, whether they are faulty or non-faulty, are to be bypassed in any routes. Rectangular fault block model is a simple and useful model, based on which many routing algorithms were developed. However, as we will show in Section 2, this model may not be optimal for the purpose of finding routes as short as possible. In constructing rectangular blocks, some non-faulty nodes are unnecessarily included, decreasing the number of selectable nodes for routing, thus decreasing the chance for finding minimal routes. In [19], a fault block model, Minimal-Connected-Component (MCC), was proposed to facilitate minimal fault-tolerant adaptive routing. The proposed model is a refinement of the widely used rectangular model. Using MCC, the chances of finding a minimal route can be greatly increased in presence of faulty nodes. An adaptive routing algorithm, which we will refer as *MIN_ROUTING* in this paper, was given in [19] that constructs an optimal "Manhattan" route avoiding all MCC fault blocks, should such routes exist.

In this paper, we will present a heuristic algorithm which, in the event a minimal Manhattan route does not exist, computes a fault-avoiding route using MCC model. With some preprocessing, the algorithm quickly works out convincingly good routes in terms of average path length, deviating from the real shortest route by only a few Hamming distance. Since finding a shortest route among all possible routes, avoiding all faults, is a very time consuming task in general, it makes sense to seek for a trade-off between route length and the time spent finding it. In a multiprocessor computer system, a route for a specific source/destination may only be used a few times. This makes the exhaustive search for a shortest route particularly uneconomical. Moreover, routing from one node to another is taking place all the time in multiprocessor computers. In many cases, it is preferable that a route be worked out quickly, and used to pass messages, rather than spend too much time computing an absolutely shortest route. In summary, the proposed algorithm provides an important option for the task of routing.

The rest of this paper is organized as follows. Section 2 gives the preliminaries, including the motivation of proposing Minimal-Connected-Component (MCC) fault block model. In Section 3, we give a formal description of MCC fault block model. We also introduce a sufficient and necessary condition for the existence of minimal Manhattan routes in presence of MCC fault blocks. In Section 4, we propose an adaptive routing algorithm that heuristically constructs a nearly shortest route without much overhead, in case that a Manhattan route is known to be nonexistent. In Section 5, experiment results are presented and discussed. We give some concluding remarks in Section 6.

2. Preliminaries

It is convenient to represent a two-dimensional (2D) mesh with a 2D coordinate system, in which we use a pair of ordered integers (x, y) to locate and identify a node. Routing is then to send messages from a source node (x_s, y_s) to a destination (x_d, y_d), passing some intermediate nodes. For the convenience of discussion and without loss of generality, we can always define the coordinate system so that $(x_s, y_s) = (0, 0)$. The destination node (x_d, y_d) will then fall in one of the four quadrants. Without loss of generality, we always assume that (x_d, y_d) is in the first quadrant, i.e., $x_d \geqslant 0$ and $y_d \geqslant 0$. The cases for destinations falling in other quadrants need only symmetric treatments.

In a 2D mesh, a non-boundary node (x,y) has four immediate neighbors, located at $(x+1,y)$, $(x-1,y)$, $(x,y+1)$ and $(x,y-1)$, respectively. Following the convention in literature on meshes, we say that node $(x+1,y)$ is (x,y)'s neighbor in its *east* direction. Similarly, $(x-1,y)$, $(x,y+1)$, and $(x,y-1)$ are called (x,y)'s *west*, *north*, and *south* neighbors. In each step of routing, a node can take one and only one direction to its immediate neighbor. So a routing can be represented as a sequence of directions. We define *Manhattan route* and *shortest route* as follows:

Definition 2.1. A route is called Manhattan route if during the routing, at most two directions are used.

Definition 2.2. A route is called shortest route if among all possible routes, it has the shortest Hamming distance.

The *Hamming distance*, in this paper's context, is simply the number of nodes a route passes (a.k.a. "lengths" or "hops") to reach the destination. Obviously, a Manhattan route is a minimal route in terms of route length, and can always be effected for any pair of (x_s,y_s) and (x_d,y_d) in a mesh with no faulty nodes. A Manhattan route must be a shortest route, but not vice versa.

If there are some faulty nodes in the mesh, in the sense that these nodes are unable to receive/send messages, a Manhattan route may or may not exist. In [20], a sufficient condition was proposed for the existence of Manhattan route using the commonly used *disconnected rectangular block* fault model [17]. While a useful model facilitating many routing algorithms [1,2,4,13,17], it is not an optimal one for the purpose of finding routes as short as possible. In forming the rectangular fault blocks, the positions of source and destination as relative to the faults are not taken into consideration. For example, as per the definition of rectangular fault block, in Fig. 1, the fault block *A* contains fault-free but "disabled" nodes *a*, *g*, *i*, and *j*. Since the whole block is to be bypassed, these disabled nodes will not be used in the process of routing no matter where the source and destination are. However, if source *s* and destination *d* are as given in Fig. 1, using them in routing can generate a Manhattan route ($s \rightarrow g \rightarrow a \rightarrow i \rightarrow j \rightarrow d$).

The above example motivates for a finer model of fault block so that the shortest possible routes can be found in the presence of faulty nodes. In [19], the Minimal-Connected-Component fault model, MCC for abbreviation, was proposed for that purpose. The main idea behind this fault model is that a node will be included in a fault block only if using it in a routing will definitely make the resulting route non-Manhattan. In the next section, we will introduce MCC and the minimal routing result using it.

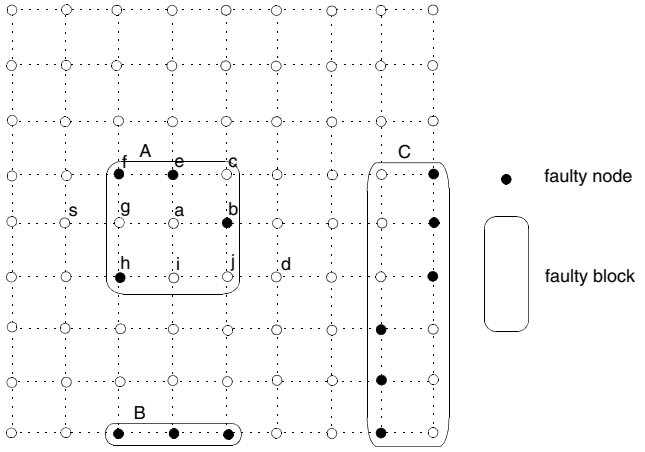

Fig. 1. An example of rectangular fault blocks. Fault-free but disabled nodes in a fault block will not be used in routing.

3. Minimal-connected-component (MCC) Fault block model

3.1. The MCC model

Again, the following discussion assumes that $(x_s,y_s)=(0,0)$ and (x_d,y_d) falls in the first quadrant. Thus, a Manhattan route from $(x_s,y_s)=(0,0)$ to (x_d,y_d) would only take east and/or north directions.

The formation of MCC blocks is a recursive procedure, described as follows:

1. Initially, label all faulty nodes as "faulty";
2. If a node (x,y) is fault-free, but its north neighbor $(x,y+1)$ and east neighbor $(x+1,y)$ are faulty, (x,y) is labeled "useless";
3. If a node (x,y) is fault-free, but its north and east neighbors are either faulty or useless, (x,y) is labeled "useless";
4. If a node (x,y) is fault-free, but its south neighbor $(x,y-1)$ and west neighbor $(x-1,y)$ are faulty, (x,y) is labeled "can't-reach";
5. If a node (x,y) is fault-free, but its south and west neighbors are either faulty or can't-reach, (x,y) is labeled "can't-reach";
6. The nodes are recursively labeled until there are no new useless or can't-reach nodes;

7. Finally, if two adjacent nodes are either faulty or useless or can't-reach, the two nodes are connected with an imaginary edge.

A node labeled "useless" is such a node that once it is entered in a routing, the next move must take either west or south direction, making a Manhattan routing impossible. A node labeled "can't-reach" is such a node that to enter it in a routing, a west or south move must be taken, making a Manhattan routing impossible.

We call the final node set connected this way an MCC fault block. Fig. 2a shows what useless and can't-reach nodes are; Fig. 2b is an example MCC fault block; and the "general shape" of an MCC is shown in Fig. 2c. Note that the construction of MCC is dependent on the relative positions of source and destination. The general shapes of MCC for destinations falling in the second, third, and fourth quadrants, respectively, are shown in Fig. 3. It can be seen that MCCs for destinations falling in quadrants I and III are of the same general shape, while MCCs for destinations falling in quadrants II and IV have the same shape. Therefore, it is not necessary to compute a set of fault blocks for *each* pair of source/destination. Instead, for given faulty nodes, only *two* different sets of fault blocks are computed: The destination d falling in quadrant I or III uses one set for routing, and d falling in quadrant II or IV uses the other set.

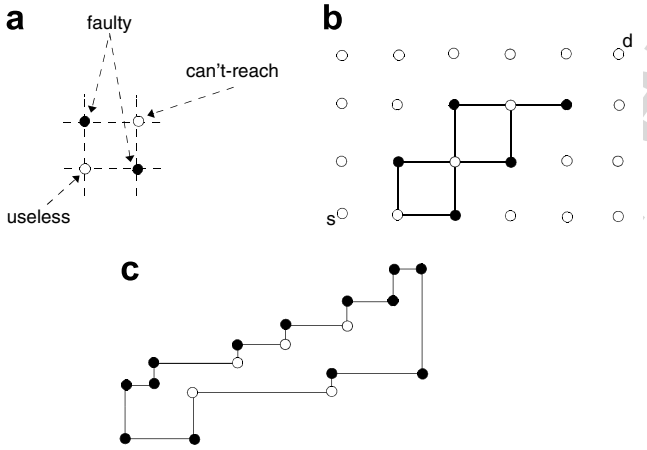

Fig. 2. (a) Definition of useless and can't-reach nodes; (b) an example MCC fault block; (c) a general MCC has the shape of the so-called "rectilinear monotone" polygon. It looks like a staircase.

The following observations of MCC are important:

Proposition 3.1. *All MCCs in a mesh are disjoint to each other.*

Proposition 3.2. *Any two MCCs are separated by at least two Hamming distance, i.e., one can always find a route going "in-between" two MCCs.*

MCC model includes much fewer non-faulty nodes in fault blocks. Many non-faulty nodes that would have been included in rectangular fault blocks now can become candidate routing nodes. Experiments were conducted to compare the two models in terms of (1) the total number of nodes included in fault blocks; and (2) the total number of fault blocks. Comprehensive simulation results [19] showed that the benefit of using MCC model (as opposed to the rectangular model) is quite enormous - the number of nodes (both faulty and fault-free) included in rectangular blocks is much higher than that in MCC blocks. Since all nodes included in faulty blocks are disabled for routing, MCC blocks result in much more non-disabled nodes in the mesh.

In [19], a sufficient and necessary condition was proposed for the existence of Manhattan routes in a mesh using MCC model. Also, [19] presented an algorithm (referred as *MIN_ROUTING* in this paper) to find a Manhattan route provided that such routes exist. The sufficient and necessary condition for the existence of Manhattan routes is stated below.

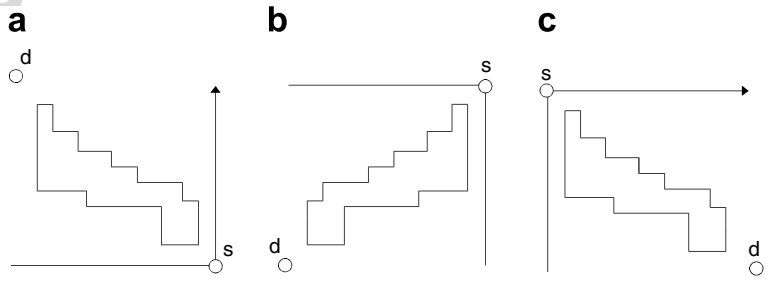

Fig. 3. The shape of MCC for destination falling in (a) second quadrant; (b) third quadrant; (c) fourth quadrant.

Theorem 3.1. *A Manhattan route, from source node $s = (0,0)$ to destination node $d = (x_d, y_d)$, can be found if and only if neither of the following two statements holds.*

1. *There exists a sequence of MCCs $(M_1, M_2, \ldots, M_n)$, such that*
   - *M_1 contains a node $(0, z_1)$ such that $0 < z_1 < y_d$;*
   - *M_n contains a node (x_d, z_n) such that $0 < z_n < y_d$;*
   - *For all M_i, M_{i+1}, $1 \leqslant i \leqslant n-1$,*

$$\min\{a : (a,b) \in M_{i+1}\} - 1 \leqslant \max\{u : (u,v) \in M_i\} \leqslant \max\{x : (x,y) \in M_{i+1}\} - 1$$

   and

$$\max\{v : (u,v) \in M_i\} < \max\{y : (x,y) \in M_{i+1}\}$$

   We call this type of sequence Type-I sequence.
2. *There exists a sequence of MCCs $(M_1, M_2, \ldots, M_n)$ such that*
   - *M_1 contains a node $(z_1, 0)$ such that $0 < z_1 < x_d$;*
   - *M_n contains a node (z_n, y_d) such that $0 < z_n < x_d$;*
   - *For all M_i, M_{i+1}, $1 \leqslant i \leqslant n-1$,*

$$\min\{b : (a,b) \in M_{i+1}\} - 1 \leqslant \max\{v : (u,v) \in M_i\} \leqslant \max\{y : (x,y) \in M_{i+1}\} - 1$$

   and

$$\max\{u : (u,v) \in M_i\} < \max\{x : (x,y) \in M_{i+1}\}.$$

   We call this type of sequence Type-II sequence.

Fig. 4 gives an intuitive illustration of the two types of MCC sequences to help convey the idea. The proof of Theorem 3.1 can be found in [19].

Note that it is possible that the distribution of MCCs makes the conditions in Theorem 3.1 true, thus minimal routing impossible. If that is the case, we still need to find a route, preferably as short as possible. In the rest of this paper, we will propose a heuristic routing algorithm in case there exists no minimal Manhattan routes.

3.2. MCC overlapping graph

Definition 3.1. A node of MCC M_i is called the northeast node of M_i if neither its north nor its east neighbor belongs to any MCC. We use U_i to denote the northeast node of M_i. Similarly, a node of MCC M_i is called the southwest node of M_i if neither its south nor its west neighbor belongs to any MCC. We use L_i to denote the southwest node of M_i.

$x(U)$ is used to denote node U's x-coordinate, $y(U)$ to denote U's y-coordinate.

The algorithm we will propose needs frequent answers to the question "whether a Manhattan route exists between a particular pair of nodes." Based on Theorem 3.1, this information can be conveniently built into and then obtained from the transitive closure on the *MCC overlapping graph*.

An MCC overlapping graph is constructed out of the layout of the MCCs. Given an MCC set $S = (M_1, M_2, \ldots, M_n)$ in the mesh, we construct a *directed graph* $G_S^{\mathrm{I}} = (V, A^{\mathrm{I}})$ with vertex set $V = \{v_1, v_2, \ldots, v_n\}$, where v_i represents MCC M_i, and the "I" superscript in G_S^{I} and A^{I} refers to Type-I. The set of edges A^{I} is defined as follows: If M_i's northeast node, U_i, is "immediately below" M_j and "properly covered" by M_j's west–east span, then there is a directed edge pointing from v_i to v_j. More formally,

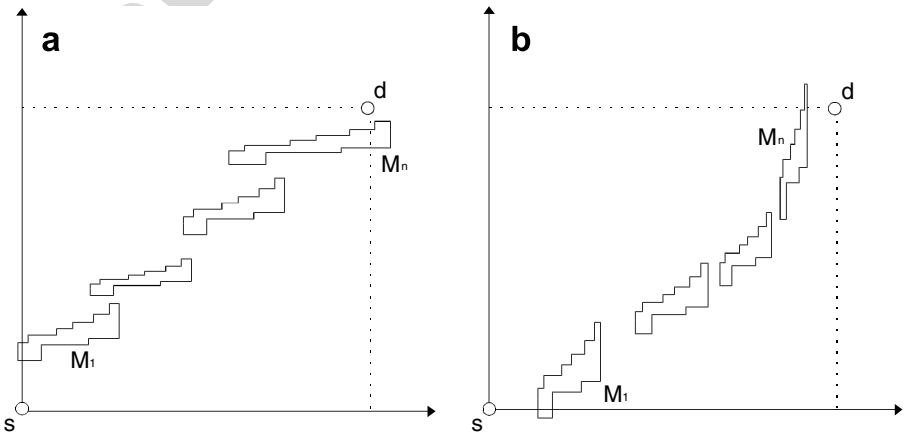

Fig. 4. (a) Type-I MCC sequence; (b) Type-II MCC sequence.

$A^{\mathrm{I}} = \{(v_i, v_j)$
| going north from U_i, M_j is the first encountered MCC such that $y(U_i) < y(U_j) \wedge x(L_j) - 2 < x(U_i) < x(U_j)\}$.

Fig. 5 shows an example of two MCCs M_i and M_j such that there is a directed edge from v_i to v_j in G_S^{I}.

An obvious choice of data structure for G_S^{I} is an $n \times n$ 2D array, denoted g_S^{I}. The value of an element $g_S^{\mathrm{I}}(i,j)$ will be either 0 or 1:

$$g_S^{\mathrm{I}}(i,j) = \begin{cases} 1 & \text{if}(v_i, v_j) \in A^{\mathrm{I}} \\ 0 & \text{if}(v_i, v_j) \notin A^{\mathrm{I}} \end{cases}$$

The *transitive closure* of g_S^{I}, denoted $T(g_S^{\mathrm{I}})$, tells whether there exits a directed path from one vertex to another in G_S^{I}, i.e.,

$$T(g_S^{\mathrm{I}})(i,j) = \begin{cases} 1 & \text{if there exists a directed path from } v_i \text{ to } v_j \\ 0 & \text{otherwise} \end{cases}$$

Symmetrically, we can construct overlapping graph $G_S^{\mathrm{II}} = (V, A^{\mathrm{II}})$ for Type-II MCC sequence, where A^{II} is defined as follows: If M_i's northeast node, U_i, is "immediately left to" M_j and "properly covered" by M_js south-north span, then there is a directed edge pointing from v_i to v_j, i.e.,

$A^{\mathrm{II}} = \{(v_i, v_j)$
| going east from U_i, M_j is the first encountered MCC such that $x(U_i) < x(U_j) \wedge y(L_j) - 2 < y(U_i) < y(U_j)\}$.

Once $T(g_S^{\mathrm{I}})$ and $T(g_S^{\mathrm{II}})$ are computed, based on Theorem 3.1, the existence problem for Manhattan route between a particular pair of nodes, say $(0,0)$ and $(x_{\mathrm{d}}, y_{\mathrm{d}})$, can be solved by an algorithm in $O(n)$ time with $O(n^3)$ preprocessing time, where n is the number of MCC fault blocks [19]. (Compared to mesh size, n is a very small quantity.) We will refer the determination algorithm as *Manhattan* in this paper. *Manhattan* will be run first to determine whether there is a Manhattan route between source and destination nodes. If YES is returned, the minimal routing algorithm *MIN_ROUTING* introduced in [19] will be called. If NO is returned, we will use the routing algorithm described in the next section.

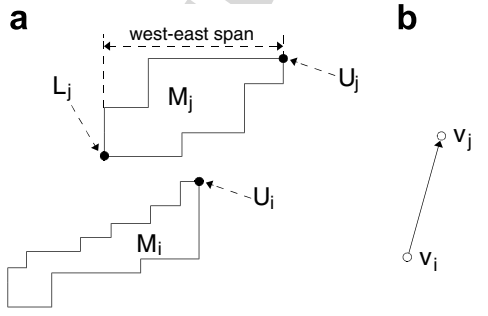

Fig. 5. (a) Two MCC's M_i and M_j; (b) There will be a directed edge from v_i to v_j in G_S^{I}.

4. Heuristic fault-tolerant routing using MCC

4.1. Routing algorithm

g_S^{I}, $T(g_S^{\mathrm{I}})$, g_S^{II}, and $T(g_S^{\mathrm{II}})$, as described in the previous section, are all computed in the preprocessing phase of the routing algorithm.

The greedy routing algorithm we propose will force the route to take some "backward" moves in cases it seems inevitable.

Definition 4.1. Use U_i' to denote the immediate northeast node of U_i, i.e., $x(U_i') = x(U_i) + 1$ and $y(U_i') = y(U_i) + 1$. Use L_j' to denote the immediate southwest node of L_j, i.e., $x(L_j') = x(L_j) - 1$ and $y(L_j') = y(L_j) - 1$.

The simplest way to backwardly get around a reached MCC is shown in Fig. 6. The routing goes straight up from U_i' until M_j is reached. It then goes along M_j's lower boundary ("west-first-south-second") until L_j' is reached.

The heuristic routing algorithm with respect to Type-I MCCs is given below.

1. From node $(x_{\mathrm{s}}, y_{\mathrm{s}})$, go straight up until an MCC M_i is reached; goto Step 2
2. **for** all j such that $T(g_S^{\mathrm{I}})(i,j) = 1$ **do** // M_i is "chainedly" under M_j
{
Call *Manhattan* $(L_j', (x_{\mathrm{d}}, y_{\mathrm{d}}))$
if *Manhattan* $(L_j', (x_{\mathrm{d}}, y_{\mathrm{d}})) = YES$
{
$J \leftarrow j$ // Remember this j

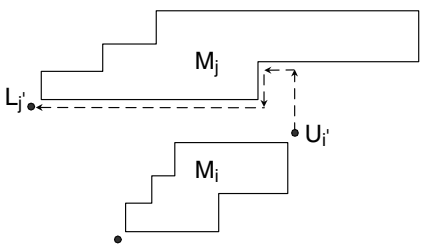

Fig. 6. Backwardly get around M_j.

```
            break for-loop; goto Step 3
        }
    }
    goto Step 4          // Manhattan returns NO for all j
3. // Entering this step, Manhattan has returned YES for some j = J
    3.1. Go along the lower boundary of MCC sequence until M_J is reached
    3.2. Take the backward route (toward southwest) until L'_J is reached
    3.3. From node L'_J, call algorithm MIN_ROUTING to minimally route to node (x_d, y_d)
    Algorithm terminates.
4. // Entering this step, Manhattan did not find any M_j, such that there is minimum route from L'_j to (x_d, y_d)
    4.1. Go along the backward route (toward southwest) of M_i until L'_i is reached
    4.2. Let (x_s, y_s) ← L'_i; goto Step 1
```

Step 1 is straightforward. When the routing enters Step 2, it reaches an MCC block M_i. The purpose of Step 2 is to figure out a place to "get around" the chainedly blocking MCCs. The greedy strategy checks for a lower left corner, from which a Manhattan route to destination can be effected.

Suppose there is a Manhattan route from the lower left corner of an MCC, and M_J is the first such MCC (it could be the case that $J = i$). Then the routing goes along the lower boundary of MCC sequence until M_J is reached. It then takes the backward route (toward southwest) until L'_J is reached. From node L'_J, it calls algorithm *MIN_ROUTING* to minimally route to node (x_d, y_d). This is Step 3, and the idea is illustrated in Fig. 7a.

However, if *Manhattan* did not find any M_j, such that there is minimum route from L'_j to (x_d, y_d), the algorithm will just "get around" from the lower left corner of M_i, and the routing continues at L'_i. This is Step 4, and the idea is illustrated in Fig. 7b.

The phase with respect to Type-II MCC's is just a dual process.

4.2. Complexity analysis

The routing process is initiated only at source node s. The routing program traverses the nodes it chooses until destination d is reached. To assist in making routing decisions, matrices g_S^{I} (g_S^{II}) and $T(g_S^{\mathrm{I}})(T(g_S^{\mathrm{II}}))$ have to be consulted from time to time. Therefore they have to be carried with the routing program.

The majority of computation is done in Step 2, when an MCC is encountered. In the for-loop, algorithm *Manhattan* is run at most $O(n)$ times, with each run taking $O(n)$ time. Since Step 2 will be

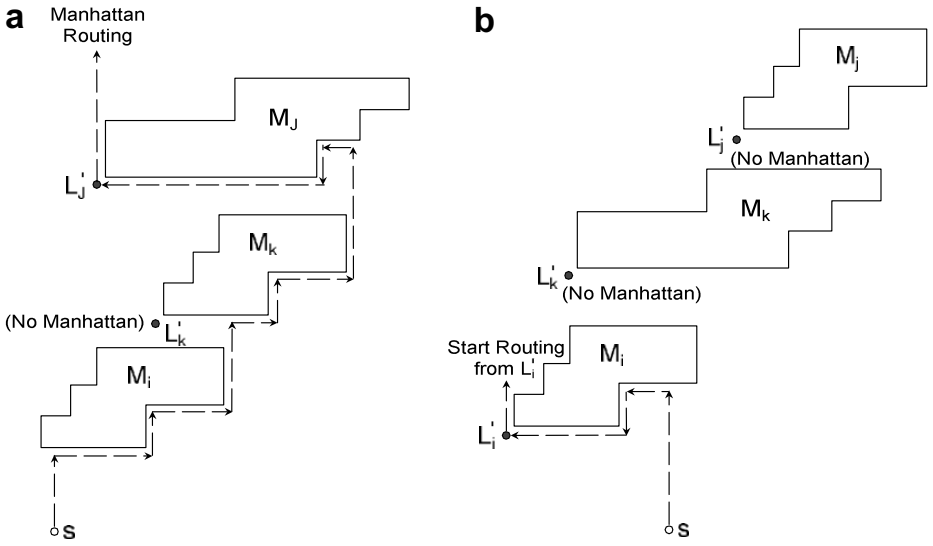

Fig. 7. (a) Step 3 – It has been determined in Step 2 that there is an M_J in the "upward" chain of MCCs, such that from M_J's lower-left corner L'_J there is a Manhattan route to the destination. The routing algorithm then just keeps going "northeast" until reaching M_J, and then goes "backward" to L'_J to take the Manhattan route; (b) Step 4 – It has been determined in Step 2 that there is no such an M_J as in (a). Therefore going "northeast" will not lead to destination. The routing algorithm then goes "backward" right away to L'_i, and start a new probing from there.

Mesh Size	50×50	55×55	60×60	65×65	70×70	75×75	80×80	85×85
Manhattan Distance of Source-Destination	98	108	118	128	138	148	158	168
Average Length of Shortest Route	103.80	114.26	125.74	134.14	144.26	155.42	164.64	174.28
Average Length of Heuristic Route	107.54	117.84	131.84	137.36	148.76	160.24	170.04	183.90

Fig. 8. Heuristic route length vs. shortest route length. The fault rate used for this table/diagram is 0.25.

entered at most $O(n)$ times, the total time spent on decision making is bounded by $O(n^3)$ in the whole routing process, where n is the number of MCC blocks. Note that the experimental results showed that n is a very small number compared to the size of mesh.

Every time Step 2 is entered, the algorithm "greedily" takes the first available lower left corner, from which there is a minimum route to d. The final route worked out by this algorithm is heuristic in nature and is not necessarily shortest. There is no need to distribute any fault block information in non-faulty nodes. The path traversed by the routing program is the same path traversed by messages. No extra nodes are involved in routing process.

5. Experimental results and discussion

We conducted simulation experiments to evaluate the performance of the heuristic route-finding algorithm. For given fault rates, faulty nodes are randomly generated for square meshes of various sizes, and MCC blocks are formed for these faulty nodes. The proposed heuristic routing algorithm is run using these MCC blocks. The length of route worked out by the heuristic routing algorithm is compared with that of the shortest possible route. The table in Fig. 8 shows the result for fault rate 0.25. Simulation results for other fault rates follow exactly the same pattern. The data used in Fig. 8 is the average of 500 runs. The second row in the table gives the Manhattan distance of source/destination under consideration. The third and fourth rows in the table list length of the shortest route and length of heuristic route, respectively, for meshes of different sizes. It can be seen that in all cases, heuristic route length deviates from the shortest route length only by a small distance.

To understand why the heuristic length would deviate very little from the shortest length, another group of experiments were conducted. We randomly generated faulty nodes, and observed the pattern of MCC distributions in terms of allowing feasible routes. Meshes of sizes 50×50, 60×60, 70×70, 80×80, and 90×90 were simulated. The table in Fig. 9 shows the result for meshes of 70×70. Results for other sizes show the same trend. When the fault rate is 0.21 (first column in the table), all 1000 groups of generated faults allow Manhattan routes from source to destination, using MCC fault blocks. As the fault rate increases, the MCC distributions that allow Manhattan routes decrease. When a distribution allows no Manhattan routes, two possibilities exist: either there is no feasible route at all (totally blocked), or there are routes but they are non-Manhattan. For example,

Fault Rate	0.21	0.23	0.25	0.27	0.29	0.31	0.33	0.35	0.37	0.39
Total Distributions Generated	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000
Distributions Containing Manhattan Routes	1000	1000	989	897	574	237	112	43	7	0
Distributions Containing Non-Manhattan Routes	0	0	0	5	1	3	4	0	1	0
Distributions Containing No Routes at All	0	0	11	98	425	760	884	957	992	1000

Fig. 9. Statistics of MCC distributions that allow Manhattan, non-Manhattan routes, and no routes at all, for mesh of size 70×70.

in the fourth column, when fault rate is 0.27, among the 1000 groups of generated faults, 897 groups allow Manhattan routes from source to destination, 98 groups allow no route at all, and five groups allow non-Manhattan routes. The table indicates that the MCC distributions have a tendency of either allowing Manhattan routes, or no routes at all (as fault rate increases). Only by a very small chance, a distribution would allow no Manhattan routes, but non-Manhattan routes. That fact explains the behavior observed in Fig. 8: If a non-Manhattan route exists, the heuristic algorithm will be called. Step 2 is to find the first available "lower left corner," from which there is a Manhattan route to destination. By extremely high probability, such a corner will be found very soon. That is, the "backward detour" will be taken *almost always only once* (in the algorithm, Step 3 will be almost always entered after Step 2). This chosen backward detour may not be the shortest, but the difference between the chosen backward detour and the shortest backward detour (which would take much more time to find) is the only contributor of the deviation, which happens to be a small quantity on average.

The heuristic routing algorithm is much simpler than shortest routing, which would basically resort to exhaustive search. The simulation results suggest that the lengths of routes worked out by the heuristic algorithm are very close to that of shortest routes. Therefore for most routing tasks, heuristic routing should be in order.

6. Conclusion

We proposed a heuristic algorithm that takes greedy approach to compute a nearly shortest route among the Minimal-Connected-Component fault blocks [19], which was proposed for increasing the chance of finding minimal routes among faults. Experiments show that the algorithm works out convincingly good routes in terms of average path length, deviating from the real shortest route by only a few Hamming distance. The total time spent on decision making is bounded by $O(n^3)$ in the whole routing process, where n is the number of MCC blocks and is a very small number compared with the size of mesh.

Since finding a shortest route among all possible routes, avoiding all faults, is a time consuming job, it makes sense to seek for a trade-off between route length and the time spent finding it. In a multiprocessor computer system, a route for a specific source/destination may only be used a few times. This makes the exhaustive search for a shortest route particularly uneconomical. Routing from one node to another is taking place all the time in multiprocessor computers. In many cases, it is preferable that a route be worked out quickly, and used to pass messages, rather than spend too much time computing an absolutely shortest route. The proposed algorithm provides an important option for fault-tolerant routing.

Acknowledgement

The author thanks Peng Du, of the State Key Laboratory for Novel Software Technology at Nanjing University, for his help in implementing the experiments and providing simulation data.

References

[1] R.V. Boppana, S. Chalasani, Fault-tolerant wormhole routing algorithms for mesh networks, IEEE Trans. Comput. 44 (7) (1995) 848–864.
[2] Y.M. Boura, C.R. Das, Fault-tolerant routing in mesh networks, in: Proceedings of the 1995 International Conference on Parallel Processing, 1995, pp. I 106–I 109.
[3] Z. Chen, Z. Liu, Z. Qiu, A deadlock-free wormhole routing scheme in the pan-mesh, Proceedings of the 20th International Conference on Advanced Information, Networking and Applications, 2003, pp. 825–829.
[4] A.A. Chien, J.H. Kim, Planar-adaptive routing: low cost adaptive networks for multiprocessors, in: Proceedings of the 19th International Symposium on Computer Architecture, 1992, pp. 268–277.
[5] G.M. Chiu, S.P. Wu, Fault-tolerant routing strategy in hypercube multicomputers, IEEE Trans. Comput. 45 (2) (1996) 143–155.
[6] W.J. Dally, The J-Machine: system support for actors, in: Hewitt, Agha (Eds.), Actors Knowledge-Based Concurrent Computing, MIT Press, 1989.
[7] P.T. Gaughan, B.V. Dao, S. Yalamanchili, D.E. Schimmet, Distributed, deadlock-free routing in faulty, pipelined, direct interconnection networks, IEEE Trans. Comput. 45 (6) (1996) 651–665.
[8] G.J. Glass, L.M. Ni, Fault-tolerant wormhole routing in meshes without virtual channels, IEEE Trans. Parallel Distributed Syst. 7 (6) (1996) 6201–6360.
[9] H.X. Gu, Z.J. Liu, Building a terabit router with XD networks, Lecture Notes in Computer Science 3740 (2005) 520–528.
[10] T.C. Lee, J.P. Hayes, A fault-tolerant communication scheme for hypercube computers, IEEE Trans. Comput. 41 (10) (1992) 1242–1256.
[11] S. Lee, D. Moon, H. Kim, W. Chang, A genetic routing algorithm for a 2D-meshed fault-tolerant network system, in: Proceedings of the First International on Workshop

Advanced Internet Services and Applications, Lecture Notes in Computer Science, vol. 2402, 2002, pp. 39–46.
[12] A.C. Liang, S. Bhattacharya, W.T. Tsai, Fault-tolerant multicasting on hypercubes, J. Parallel Distributed Comput. 23 (3) (1994) 418–428.
[13] R. Libeskind-Hadas, E. Brandt, Origin-based fault-tolerant routing in the mesh, in: Proceedings of the the First International Symposium on High Performance Computer Architecture, 1995, pp. 102–111.
[14] S.L. Lillevik, The Touchstone 30 Gigaflop DELTA prototype, in: Proceedings of the Sixth Distributed Memory Computing Conference, 1996, pp. 671–677.
[15] R.S. Rajesh, S. Arumugam, An optimistic deadlock free adaptive wormhole routing algorithm for two dimensional meshes, Commun. Comput. (2004) 21–24.
[16] C.L. Seitz, The architecture and programming of the Amete Series 2010 multicomputer, in: Proceedings of the Third Conference on Hypercube Concurrent Computers and Applications,1998, pp. I 33–I 36.
[17] C.C. Su, K.G. Shin, Adaptive fault-tolerant deadlock-free routing in meshes and hypercubes, IEEE Trans. Comput. 45 (6) (1996) 666–683.
[18] Y.-J. Suh, B.V. Dao, J. Duato, S. Yalamanchili, Software based fault-tolerant oblivious routing in pipelined networks, in: Proceedings of the 1995 International Conference on Parallel Processing, 1995, pp. I 101–I 105.
[19] D. Wang, A rectilinear-monotone polygonal fault block model for fault-tolerant minimal routing in mesh, IEEE Trans. Comput. 52 (3) (2003) 310–320.
[20] J. Wu, Reliable unicasting in faulty hypercubes using safety levels, IEEE Trans. Comput. 46 (2) (1997) 241–247.
[21] J. Wu, Fault-tolerant adaptive and minimal routing in mesh-connected multicomputers using extended safety levels, IEEE Trans. Parallel Distributed Syst. 11 (2) (2000) 149–159.