Online Adaptive Fault-Tolerant Routing in 2D Torus

Yamin Li[1], Shietung Peng[1], and Wanming Chu[2]

[1] Hosei University, Tokyo 184-8584 Japan,
yamin@k.hosei.ac.jp; speng@k.hosei.ac.jp
[2] University of Aizu, Aizu-Wakamatsu 965-8580 Japan,
w-chu@u-aizu.ac.jp

Abstract. In this paper, we propose efficient routing algorithms for 2D torus with possible large number of faulty nodes. There is no presumption on the number and the distribution of faulty nodes. The proposed algorithms find a fault-free path between any two nonfaulty nodes with high probability in linear time by using only the local routing information of the network. The results of our empirical analysis through simulations show that the algorithms can find a fault-free path between any two nonfaulty nodes with high probability. For example, in a torus of size up to 128×128, where, the number of faulty nodes up to 15%, the heuristuc-square routing algorithm finds a fault-free path with a probability of 90% or higher. The experimental results are impressive for 2D torus with only four links per node.

1 Introduction

The two dimensional mesh/torus has constant node degree, recursive structure, simple communication algorithms, and good scalability. Due to these attractive properties, the mesh/torus has been the common interconnection network for several commercially available parallel computers, such as MPP (Goodyear Aerospace), Paragon (Intel), Victor (IBM), AP3000 (Fujitsu), and Toroidal Net (IRECE), Alpha 21364 [9].

A 2D mesh can be laid out on a VLSI chip in an area that increases linearly with the number of processors. Since the implementation of 2D mesh uses short, local links only, it is possible to perform communication at very high speed. A 2D torus has wraparound links. However, the method of folding can be used to lay out a 2D torus in such a way that it uses only short, local links too.

In this paper, we focus our designs on 2D torus. However, the ideas used in the proposed algorithms should be applicable to higher dimensional torus. The 2D torus has been and will continuously be a popular interconnection network for high-performance parallel computers due to its high bandwidth nearest neighbor connectivity for efficient computation and fast communication in many scientific applications.

Fault-tolerant routing is a dominant issue facing the design of interconnection networks for large-scale parallel computers [10]. There are many fault models

used for designing fault-tolerant routing algorithms [1][2][3][6][7][8][11][12][13]. Some of these algorithms set conditions on the number of faulty nodes or the shape of faulty components. Others use global fault information (off-line) or partially global information. Chen et. al [4][5] introduced the concept of local-subcube connectivity for hypercubes. In this paper, we develop fault-tolerant routing algorithms on 2D torus using local information only (on-line), and allow arbitrary number of faulty nodes with no restriction on the shape of the faulty nodes (blocks). Our algorithms find a fault-free path between any two non-faulty nodes with high probability in linear time.

The rest of this paper is organized as follows: In the next section, we give necessary definitions used throughout the paper. We also show a theorem that is a theoretical ground of the proposed routing algorithms. In Sections 3, 4, and 5, respectively, three fault-tolerant routing algorithms are proposed on 2D torus with possible large number of faulty nodes. In Section 6, simulations are performed and the results are analyzed and discussed. Finally, in the last section, we conclude this paper with some remarks.

2 Locally-Safe Torus

A kD n-torus T_n^k has k dimensions, n nodes per dimension, and $N = n^k$ nodes. Each node is uniquely indexed by a radix-n k-tuple. Each node is connected via communication links to two other nodes in each dimension. The neighbors of node $s = (s_0, \ldots, s_{k-1})$ in dimension i are $(s_0, \ldots, s_{i-1}, s_i \pm 1, s_{i+1}, \ldots, s_{k-1})$, where addition and subtraction are performed modulo n. For simplicity, throughout this paper, all arithmetics on the indices of nodes in a given torus should be modulo n implicitly. The distance between two nodes s and t in T_n^k is $d(s,t) = \sum_{i=0}^{k-1} \min(|s_i - t_i|, n - |s_i - t_i|)$. In this paper, we work on 2D torus only. For simplicity, we use term T, instead of T_n^2, to denote a 2D n-torus if no confusion arises.

For a given node $s = (s_0, s_1)$ in T, we denote its two neighbors in dimension i by s^{i+} and s^{i-}, respectively. For example, $s^{0+} = (s_0 + 1, s_1)$ and $s^{0-} = (s_0 - 1, s_1)$. An m-square M_m^2, or simply M, in a 2D torus T is a subgraph of T, and M is a 2D mesh of width m (m nodes in each dimension). T is locally-m-safe if the following conditions are satisfied:

1. for every m-square M in T, the subgraph formed by all nonfaulty nodes in M is connected, and
2. every boundary line segment of an m-square in T contains at least one nonfaulty node.

If there exists an integer m, $2 \leq m \leq n$, such that T is locally-m-safe then we say that T is locally-safe. The following theorem shows that local-safety implies connectedness of T.

Theorem 1. *If a 2D torus T is locally-safe then T is a connected graph.*

Proof Let $s = (s_0, s_1)$ and $t = (t_0, t_1)$ are two nonfaulty nodes in T. Without loss of generality, we assume that $s_i < t_i$ for $i = 0,\ 1$. We also assume that s^{i+} and s^{i-} be the neighbors of s on dimension i such that s^{i+} is the neighbor of s closer to t, that is, $d(s^{i+}, t) = d(s, t) - 1$. Since T is locally-safe there exists an integer $m,\ 2 \leq m \leq n$, such that T is locally-m-safe. Let $P = (s \rightarrow t)$ be the shortest path from s to t constructed by the dimension-order routing (routing along dimension 0, and then dimension 1). We construct an L-shape chain of width m that contains the path P as shown as in Figure 1.

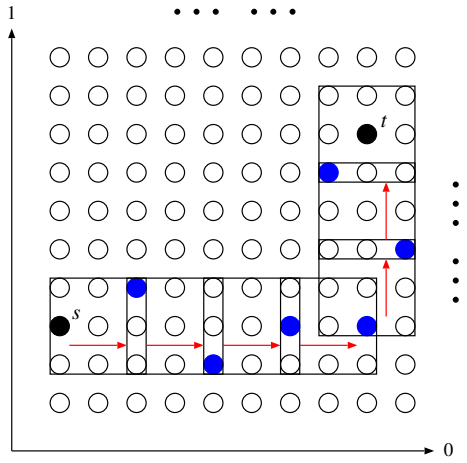

Fig. 1. Routing in chains

Let $P = P_0 \cup P_1$, where P_0 and P_1 are the line segments $s \rightarrow u$ and $u \rightarrow t$ along 0 and 1 dimensions, respectively, where $u = (u_0, u_1) = (t_0, s_1)$. The chain contains ch_0, and ch_1 defined as follows:

1. $ch_0 = \{v \in T \mid s_0 \leq v_0 \leq t_0$, and $s_1 - \lfloor m/2 \rfloor \leq v_1 \leq s_1 + \lceil m/2 \rceil - 1$; and
2. $ch_1 = \{v \in T \mid u_1 - \lfloor m/2 \rfloor \leq v_1 \leq t_1$ and $u_0 - \lfloor m/2 \rfloor \leq v_0 \leq u_0 + \lceil m/2 \rceil - 1$.

Consider ch_0 as a sequence of m-squares $M_i, 0 \leq i \leq p$, where $p = \lceil (t_0 - s_0)/(m-1) \rceil$, such that

1. $L_i = M_{i-1} \cap M_i,\ 1 \leq i \leq p$, are line segments of length $m-1$ and $ch_0 \subset \cup_{i=0}^{p} M_i$;
2. $s \in M_0$ and $u \in M_p$.

Let $L_u = \{v \in ch_0 | v_0 = u_0\}$. Obviously, we have $L_u \subset M_p$. Since T is locally-m-safe, there exist nonfaulty nodes $v^i \in L_i,\ 1 \leq i \leq p$, $u' \in L_u$, and fault-free paths: $(s \rightarrow v^1) \subset M_0, (v^1 \rightarrow v^2) \subset M_1, \ldots, (v^{p-1} \rightarrow v^p) \subset M_{p-1}, (v^p \rightarrow u') \subset M_p$. Then, the path $(s \rightarrow v^1 \rightarrow v^2 \ldots \rightarrow v^p \rightarrow u')$ is the fault-free path from s

to u'. From the definition of ch_1, we have $u' \in ch_1$. By the similar argument, we can find a fault-free path in ch_1 from u' to t. Therefore, s and t can be connected through the fault-free path $s \rightarrow u' \rightarrow t$. We conclude that T is connected. $\diamond$

The proof of the theorem is a constructive one. It provides the necessary background for our first fault-tolerant routing algorithm to be presented at next section.

3 Chain Routing Algorithm

For practice, we do not presume that T is locally-safe. The number of faulty nodes or its distribution is arbitrary. Our routing algorithms are local-information-based: no global information about the situation of the network is needed. If T is locally-m-safe then from theorem 1, the algorithm will generate a fault-free path. Otherwise, it will either generate a fault-free path or report a failure.

The algorithm follows the constructive proof of theorem 1. A chain of meshes with width m that contains the shortest path P is used for the fault-tolerant routing. While routing from source s to destination t, the path is allowed to move inside the chain through a sequence of squares as specified in the proof of theorem 1. However, for routing with higher successful routing rate, we construct the ch_0 such that, for any $v \in ch_0$, $d(v_1, t_1) \leq d(s_1, t_1) + 1$. The ch_1 is constructed similarly. The details are specified in Algorithm 1. We call this algorithm Chain_Routing (see Algorithm 1). The rate of successful routing of the algorithm will be analyzed empirically and the simulation results will be used to compare with that of the other routing algorithms proposed in this paper.

In Algorithm 1, we route source node $s = (s_0, s_1)$ to destination node $t = (t_0, t_1)$ through an L-shape chain. The chain is divided into a sequence of m-squares. A square is uniquely determined with two nodes: b and B, as shown as in Figure 2.

If routing in the first part of the chain succeeds, a node $r = (t_0, r_1)$ will be reached where r_1 is in the line segment bounded by b_0 and B_0. The path $(s \rightarrow r)$ may goes through many squares. To route in an m-square, we can use any search algorithm, depth-first search (DFS) or breadth-first search (BFS) algorithm for instance. If two parts of the chain are routed successfully, a fault-free path $(s \rightarrow t)$ is found. Whenever the routing in a square fails, the algorithm reports a failure and terminates. Assuming that the local routing inside a 2D m-mesh takes constant time, the algorithm runs in $O(n)$ time. We summarize these results into the following theorem.

Theorem 2. *The Chain_Routing algorithm will terminate in $O(n)$ time. When the algorithm terminates, it either generates a fault-free path from s to t or reports that the path cannot be found.*

Algorithm 1 (Channel_Routing(T_n, m, s, t))
Input: 2D n-torus T_n, width of local mesh m, source node
$s = (s_0, s_1)$, and destination node $t = (t_0, t_1)$
Output: a fault-free path $P = (s \rightarrow t)$ or report failure
begin
$P = \phi$;
$r = s$;
$dir_0 = dir_1 = 1$; /* determine routing direction */
if $(0 \le r_0 - t_0 \le n/2)$ OR $(0 \le t_0 - r_0 > n/2)$ $dir_0 = -1$;
if $(0 \le r_1 - t_1 \le n/2)$ OR $(0 \le t_1 - r_1 > n/2)$ $dir_1 = -1$;
for $i = 0, 1$ **do** /* for each dimension i */
/* determine *mesh* boundaries of i and j dimensions */
$j = (i + 1) \bmod 2$; /* dimension j */
$b_j = (r_j - dir_j + n) \bmod n$; /* $[b_j, B_j]$ in dimension j */
$B_j = (r_j + (m - 2) \times dir_j + n) \bmod n$;
while $r_i \neq t_i$ **do**
/* determine *mesh* boundaries of i dimension */
$b_i = r_i$; /* $[b_i, B_i]$ in dimension i */
$B_i = (r_i + (m - 1) \times dir_i + n) \bmod n$;
if t is in *mesh*
if there is a fault-free path $P' = (r \rightarrow t)$ in *mesh*
$P = P \cup P'$;
return P; /* path constructed */
else return failure; /* failed */
else
if there is a fault-free path $P' = (r \rightarrow r')$ in *mesh*
such that $(r'_i = B_i)$ OR $(r'_i = t_i)$
$P = P \cup P'$;
$r = r'$; /* continue */
else return failure; /* failed */
endwhile
endfor
end

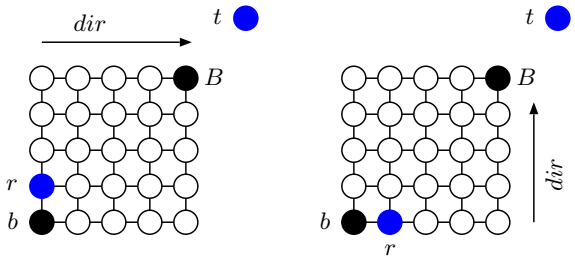

Fig. 2. Boundary of square ($m = 5$)

4 Adaptive-Square Routing Algorithm

In this section, we describe another local-information-based, fault-tolerant routing algorithm, called Adaptive_Square Routing. The idea is as follows. Instead of

arranging the sequence of m-squares as an L-shape chain, a sequence of squares is found recursively such that the routing direction of each square should be along the dimension i such that the distance $d(r_i, t_i)$ is a maximum, where r is the new source node after a local routing. Intuitively, the sequence of squares is arranged to contain a ladder-shaped shortest path that might have many turns, Each square will adapt itself such that the new source node r and the path segment along dimension i are inside the square (not at the boundary of the square).

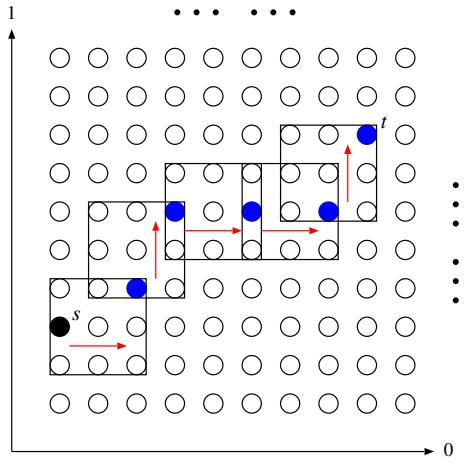

Fig. 3. Routing with Squares

To describe the algorithm, we need a notation to show the position of an m-mesh inside T. Referring to Figure 3, we associate each node r in T with a unique m-square M_r to be used by the algorithm. The M_r is determined by two nodes b and B, representing the lower-leftmost node b and an upper-rightmost node B of the two boundary lines along the routing direction.

More precisely, let i be the current routing direction, and dir_j be the unit direction ($+1$ or -1) of the shortest path along dimensions j. Then, we have $b_i = r_i$ and $b_j = r_j - dir_j$, $B_i = r_i - (m-1) \times dir_i$ and $B_j = r_j - (m-2) \times dir_j$ (all arithmetics are modulo n).

The proposed recursive algorithm is similar to that of the chain approach. The difference is that the square is adaptable in all dimensions instead of just in the dimension of the chain. Let $r = s$. The algorithm first determines the routing dimension i and the local m-square M_r, and then performs local routing in M_r that routes node r to a nonfaulty node r', a node located at the opposite boundary line of M_r from r along the ith dimension. If the local routing $r \rightarrow r'$ successes then we consider r' as a new source r and route from r recursively.

If the local routing fails, the algorithm terminates unsuccessfully and reports a failure. The algorithm is formally specified as in Algorithm 2.

Algorithm 2 (Adaptive_Square_Routing(T_n, m, s, t))
Input: 2D n-torus T_n, width of local mesh $m \geq 3$, source node
$s = (s_0, s_1)$, and destination node $t = (t_0, t_1)$
Output: a fault-free path $P = (s \rightarrow t)$ or report failure
begin
$P = \phi$;
$r = s$;
while $r \neq t$ **do**
$dir_0 = dir_1 = 1$; /* determine routing direction */
if $(0 \leq r_0 - t_0 \leq n/2)$ OR $(0 \leq t_0 - r_0 > n/2)$ $dir_0 = -1$;
if $(0 \leq r_1 - t_1 \leq n/2)$ OR $(0 \leq t_1 - r_1 > n/2)$ $dir_1 = -1$;
Find the dimension i so that the distance between r_i and t_i
$d(r_i, t_i) = \max(d(r_0, t_0), d(r_1, t_1))$;
/* determine *mesh* boundary $[b, B]$, referring to Figure 3 */
$j = (i + 1) \bmod 2$; /* dimension j */
$b_i = r_i$; /* $[b_i, B_i]$ in dimension i */
$b_j = (r_j - dir_j + n) \bmod n$; /* $[b_j, B_j]$ in dimension j */
$B_i = (r_i + (m-1) \times dir_i + n) \bmod n$;
$B_j = (r_j + (m-2) \times dir_j + n) \bmod n$;
if t is in *mesh* bounded by $[b, B]$
if there is a fault-free path $P' = (r \rightarrow t)$ in *mesh*
$P = P \cup P'$;
return P; /* path constructed */
else return failure; /* failed */
else
if there is a fault-free path $P' = (r \rightarrow r')$
in *mesh* such that $(r'_i = B_i)$ OR $(r'_i = t_i)$
$P = P \cup P'$;
$r = r'$; /* continue */
else return failure; /* failed */
endwhile
end

Next, we show that the algorithm will terminate in $O(n)$ time, and either finds a fault-free path from s to t or reports a failure.

Theorem 3. *Adaptive_Square_Routing algorithm terminates in $O(n)$ time, and either outputs a fault-free path from source s to destination t or reports a failure.*

Proof We first show that, for $m \geq 3$, the local routing always makes progress toward destination t. Since the box M_r for node r is constructed in the way that the farthest upper node B is toward to t. That is, $B = (r_i + (m-1) \times dir_i, r_j + (m-2) \times dir_j)$. The worst case is that r is routed to $r' = (r_i + (m-1) \times dir_i, r_j - dir_j)$. Since $d(r, t) - d(r', t) = (m-1) - 1 = m - 2 > 0$ for $m \geq 3$, the local routing

always makes progress toward t. Therefore, the algorithm will terminate after at most $O(n)$ local routings. For a fixed m, the running time of the local routing is a constant. Therefore, the total running time of the algorithm is $O(n)$. $\diamond$

5 Heuristic-Square Routing Algorithm

In this section, we make an effort to improve the performance of Adaptive_Square_Routing algorithm by allowing the routing to continue when the local routing along the dimension of the longest distance, say i, fails. In the new algorithm, the routing continues by trying the local routings in the squares set along the other dimension when the distance between r and t along that dimension is nonzero. If the distance between r and t along the other dimension, say j, $d(r_j, t_j) < m - 1$ then the local routing will route r to r', where $d(r_j, r'_j) = d(r_j, t_j)$. Once the local routing along the dimension j successes, the square constructed for the new source should be arranged along dimension i again. The algorithm that adds this heuristic strategy to the Adaptive_Square Routing is called Heuristic_Square routing. In the next theorem, we show that Heuristic_Box routing works.

Theorem 4. *Heuristic_Square_Routing algorithm terminates in $O(n)$ time, and either outputs a fault-free path from s to t or reports a failure.*

Proof In the new algorithm, we continue to route along dimension j when the local routing along dimension i fails. The local routing along dimension j might not make progress when $d(r_j, t_j) = 1$ and $d(r', t) = d(r, t)$. However, the next local routing will be along dimension i and make progress as shown in Theorem 3. Therefore, the number of local routing in the new algorithm is at most twice of that of the adaptive-square routing algorithm. For fixed m, the running time for the local routing is a constant. Therefore, the running time of the algorithm is $O(n)$. $\diamond$

6 Simulation Results

We have performed a set of simulations on the performance of the proposed algorithms. For the experiments concerning the sizes of the 2D torus and the local squares used, we divide the values of the parameters n and m into two groups. In the first group, we set $n = 16,\ 32$ and $m = 3,\ 4,\ 5$, while in the second group, $n = 64,\ 128$ and $m = 6,\ 7,\ 8$. That is, the size of the m-square used for local routing is larger in the second group than that in the first group. In any case, the values of m is still small compared with the values of n. For the performance of each routing algorithm, two figures (one per group) are used to show the successful routing rate and/or improvement of the algorithm. For the fault model, we use uniform distribution of node failures. The number of faulty nodes generated range between 5% and 25% with a 5% increment. For each case, we simulate 10, 000 times. The simulation results of the successful routing

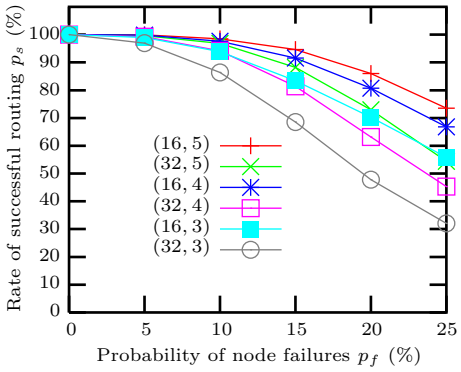

Fig. 4. Chain_Routing: Group #1

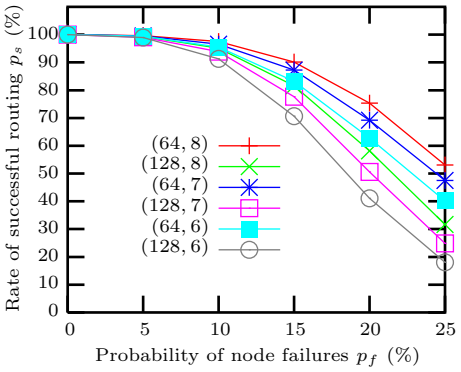

Fig. 5. Chain_Routing: Group #2

rate, improvement, and the length of the routing path for the set of parameters specified above are shown in the figures below.

Figure 4 and Figure 5 plot the successful routing rate of the simplest algorithm, Chain_Routing, where the two numbers in brackets in the figures are n and m, respectively. We can see that for a given n, increasing m improves the successful routing. On the other hand, for a given m, routing in a small torus has higher successful routing rate than that in a large torus.

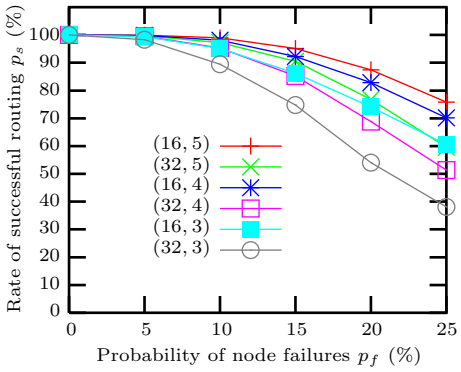

Fig. 6. Square_Routing: Group #1

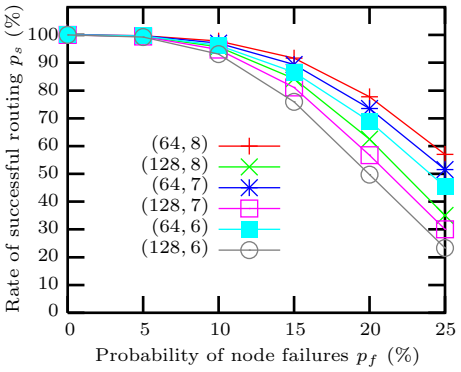

Fig. 7. Square_Routing: Group #2

Figure 6 and Figure 7 show the successful routing rate of the Adaptive_Square_Routing algorithm. Figure 8 and Figure 9 show the performance improvement of the adaptive-square algorithm compared to that of the chain algorithm. From the figures, we conclude that the adaptive-square approach is better than the chain approach in most of the cases.

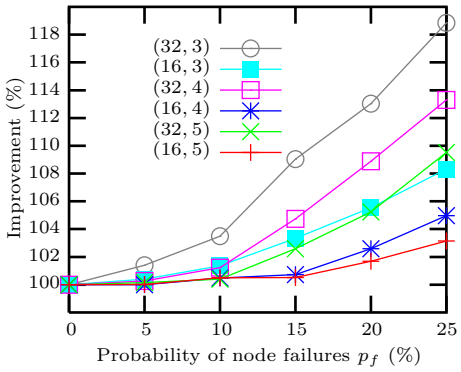

Fig. 8. Square vs Chain: Group #1

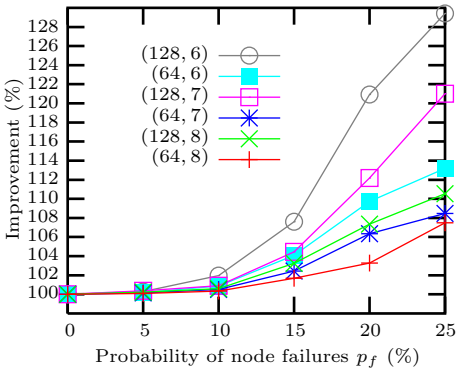

Fig. 9. Square vs Chain: Group #2

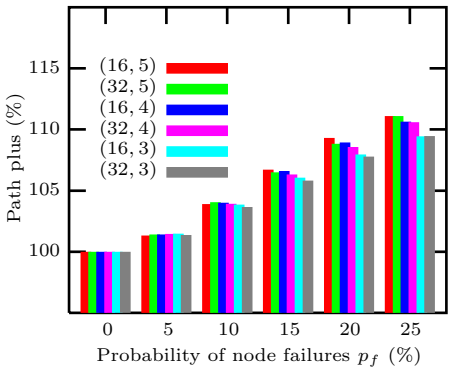

Fig. 10. Square_Routing: Group #1

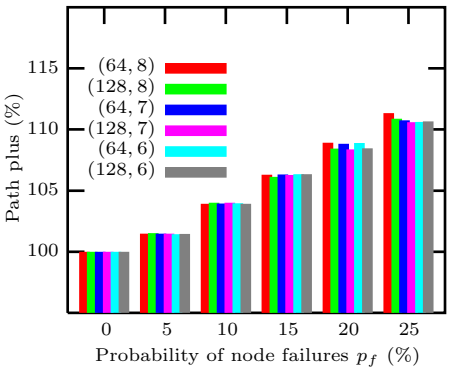

Fig. 11. Square_Routing: Group #2

The *improvement* in Figure 8 and Figure 9 is defined as

$$\frac{\text{Successful routing rate of adaptive-square algorithm}}{\text{Successful routing rate of chain algorithm}}$$

When n is small and m is large, $n = 16$ and $m = 4$ for instance, the adaptive-square algorithm is not much better than the chain algorithm. However, when m is smaller, especially as n increases, the improvement of the adaptive-square algorithm grows faster. As an example, when the number of faulty nodes is 25%, the improvement are about 1.2 and 1.3 for $n = 32$, $m = 3$ and $n = 128$, $m = 6$, respectively, in two groups. For fixed n, using a larger square will have a better performance with a cost of increasing time complexity at a rate proportional to m^2.

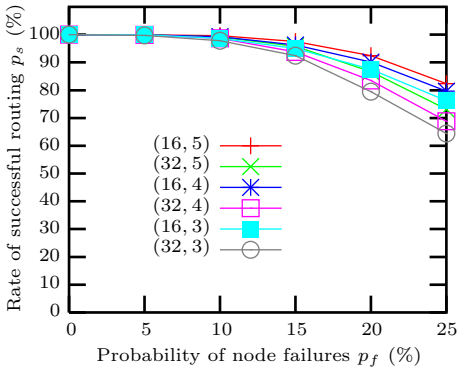

Fig. 12. Heuristic_Routing: Group #1

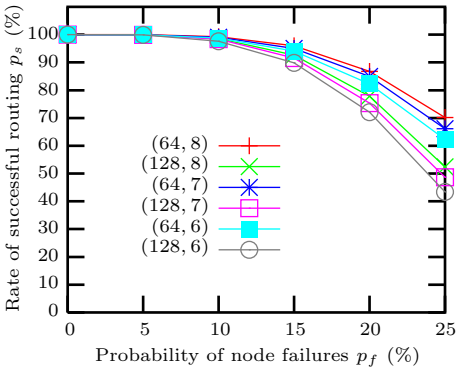

Fig. 13. Heuristic_Routing: Group #2

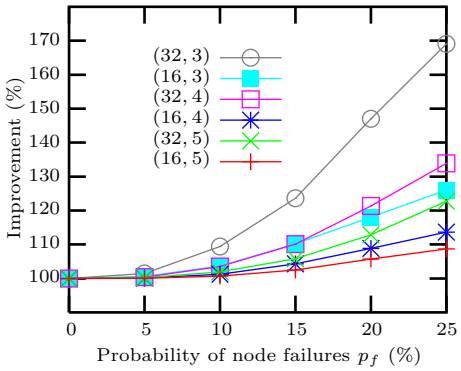

Fig. 14. Heuristic vs Square: Group #1

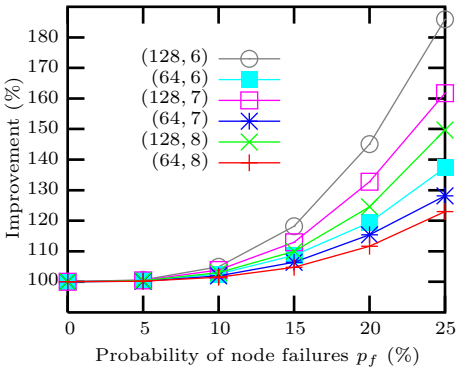

Fig. 15. Heuristic vs Square: Group #2

Figure 10 and Figure 11 display the path plus of the adaptive-square algorithm, which is calculated by

$$\text{Path plus} = \frac{\text{Path length of } P = (s \to t)}{\text{Distance between } s \text{ and } t}$$

Figure 12 and Figure 13 plot the successful routing rate of the heuristic-square routing algorithms. Figure 14 and Figure 15 depict the improvement of successful routing rate gained by using the heuristic-square routing algorithms to that of the adaptive-square algorithm. From the figures, we conclude that the improvement of the heuristic-square routing over the adaptive-square is significant. For example, when the number of faulty nodes is 25%, the improvement are about 1.7 and 1.85 for $n = 32$, $m = 3$ and $n = 128$, $m = 6$, respectively, in two groups. It is worth to adopt the heuristic-square routing algorithm when the probability of faulty nodes is high.

7 Conclusions

In this paper, we first presented a concept of local-safety for a kD n-torus. Then, we proposed two different approaches for fault-tolerant routing in a 2D n-torus with possible large and arbitrarily faulty nodes. The algorithms are online (only local information is used) and efficient ($O(n)$ time assuming that the local routing is $O(1)$). The simulation results show that the rates of successful routing of the algorithms are quite high considering that there are only four links per node in a 2D torus. The possible directions of the further research include 1) Provide theoretical analysis on the performance of the proposed algorithms; and 2) Investigate the practical issues (e.g., deadlock-free) while implement the proposed routing algorithms on certain switching models.

References

1. L. D. Aronson. Homogeneous routing for homogeneous traffic patterns on meshes. *IEEE Transactions on Parallel and Distributed Systems*, 11(8):781–793, August 2000.
2. R. V. Boppana and S. Chalasani. Fault-tolerant wormhole routing algorithms for mesh networks. *IEEE Transactions on Computers*, 44(7):848–864, July 1995.
3. S. Chalasani and R. V. Boppana. Fault-tolerant wormhole routing in tori. *IEE Proc.: Computers and Digital Techniques*, 142(11):386–394, Nov 1995.
4. Jianer Chen, Guojun Wang, and Songqiao Chen. Routing in hypercube networks with a constant fraction of faulty nodes. *Journal of Interconnection Networks*, 2(3):283–294, September 2001.
5. Jianer Chen, Guojun Wang, and Songqiao Chen. Locally subcube-connected hypercube networks: Theoretical analysis and experimental results. *IEEE Transactions on Computers*, 51(5):530–540, May 2002.
6. Q-P Gu and S. Peng. Fault tolerant routing in toroidal networks. *IEICE Transactions on Information and Systems*, E-79D:1153–1159, August 1996.
7. Q-P Gu and S. Peng. Unicast in hypercubes with large number of faulty nodes. *IEEE Transactions on Parallel and Distributed Systems*, 10:964–975, October 1999.
8. Z. Jiang and J. Wu. Fault-tolerant broadcasting in 2D wormhole-routed meshes. *The Journal of Supercomputing*, 25(3):255–275, July 2003.
9. Shubhendu S. Mukherjee, Peter Bannon, Steven Lang, Aaron Spink, David Webb The Alpha 21364 Network Architecture *IEEE Micro* 22(1):26-35, January/February 2002.
10. V. Puente, J. A. Gregorio, F. Vallejo, and R. Beivide Immunet: a cheap and robust fault-tolerant packet routing mechanism *Proceedings of The 31st Annual International Symposium on Computer Architecture*, pages 198-209, June 2004.
11. M.-J. Tsai and S.-D. Wang. Adaptive and deadlock-free routing for irregular faulty patterns in mesh multicomputers. *IEEE Transactions on Parallel and Distributed Systems*, 11(1):50–63, January 2000.
12. J. Wu. Fault-tolerant adaptive and minimal routing in mesh-connected multicomputers using extended safety level. *IEEE Transactions on Parallel and Distributed Systems*, 11(2):149–159, February 2000.
13. D. Xiang. Fault-tolerant routing in hypercube multicomputers using local safety information. *IEEE Transactions on Parallel and Distributed Systems*, 12(9):942–951, September 2001.