A New Adaptive Fault-Tolerant Routing Methodology for Direct Networks*

M.E. Gómez[1], J. Duato[1], J. Flich[1], P. López[1], A. Robles[1], N.A. Nordbotten[2], T. Skeie[2], and O. Lysne[2]

[1] Dept. of Computer Engineering, Universidad Politécnica de Valencia, Camino de Vera, 14, 46071–Valencia, Spain
megomez@disca.upv.es
[2] Simula Research Laboratory, P.O. Box 134, N-1325 Lysaker, Norway

Abstract. Interconnection networks play a key role in the fault tolerance of massively parallel computers, since faults may isolate a large fraction of the machine containing many healthy nodes. In this paper, we present a methodology to design fully adaptive fault-tolerant routing algorithms for direct interconnection networks that can be applied to different regular topologies. The methodology is mainly based on the selection of an intermediate node (if needed) for each source-destination pair. Packets are adaptively routed to the intermediate node and, from this node, they are adaptively forwarded to their destination. This methodology requires only one additional virtual channel, even for tori. Evaluation results show that the methodology is 7-fault tolerant, and for up to 14 faults, more than 99% of the combinations are tolerated, also without significantly degrading performance in the presence of faults.

1 Introduction

There exist many compute-intensive applications that require continued research and technology development to deliver computers with steadily increasing computing power. The required levels of computing power can only be achieved with massively parallel computers, such as the Earth Simulator [8] and the BlueGene/L [1]. The long execution times of these applications requires keeping such systems running even in the presence of failures. However, the huge number of processors and associated devices (memories, switches, links, etc.) significantly increases the probability of failure. In particular, failures in the interconnection network may isolate a large fraction of the machine, wasting many healthy processors that otherwise could have been used. Although network components are robust, they are usually working close to their technological limits and are therefore prone to failures. Increasing clock frequencies leads to a higher power dissipation, which again could lead to premature failures. Hence, fault-tolerant mechanisms for interconnection networks are becoming a critical design issue for large massively parallel computers.

* This work was supported by the Spanish Ministry of Science and Technology under Grant TIC2003-08154-C06-01.

L. Bougé and V.K. Prasanna (Eds.): HiPC 2004, LNCS 3296, pp. 462–473, 2004.

Faults can be classified as transient or permanent. Transient faults are usually handled by communication protocols, using CRCs to detect faults and retransmitting packets. In order to deal with permanent faults in a system, two fault models can be used: static or dynamic. In a static fault model, all the faults are known in advance when the machine is (re)booted. This fault model relies on checkpoints in order to be effective. In a dynamic fault model, once a new fault is found, actions are taken in order to appropriately handle the faulty component.

There exist several approaches to tolerate faults in the interconnection network. Most of them are based on fault-tolerant routing algorithms. However, these strategies require a significant amount of extra hardware resources (e.g., virtual channels), to route packets around faulty components [4], [12]. Alternatively, there exist some fault-tolerant routing strategies that use none or very few extra resources at the expense of providing a lower fault-tolerance degree [4], [9], disabling a certain number of healthy nodes (either in blocks (fault regions) [2], [3] or individually [5], [6]), preventing packets from being routed adaptively [10], or drastically increasing the latencies for some packets [14].

What is really needed is a fault-tolerant strategy for the interconnection network that does not degrade performance at all in the absence of faults, does not significantly decrease network performance in the presence of faults, and tolerates a reasonably large number of faults. This should be achieved without disabling any healthy node and without requiring too many extra resources.

In this paper, we take on this challenge and propose a fault-tolerant routing methodology that satisfies the properties mentioned above. The methodology relies on a static fault model with checkpointing. It allows the use of fully adaptive routing in most cases and it does not sacrifice any healthy node. In order to avoid faults, packets are sent adaptively to an intermediate node[1]. From that node, the packet will be sent adaptively to the destination. The methodology requires the use of at least three virtual channels. Note that two virtual channels are already required to provide fully adaptive routing [13].

It is important to highlight the main differences between the proposed methodology and similar approaches in the literature. Unlike [14], in no case the proposed methodology requires ejecting/reinjecting the packet at the intermediate node, thus drastically reducing latency. Moreover, unlike [10], it does not need to deactivate any *lamb* node to achieve high fault-tolerance. Furthermore, the proposed methodology allows packets to be routed using adaptive routing, instead of deterministic routing, thus increasing the overall network throughput.

The rest of the paper is organized as follows. In Section 2, the proposed methodology is presented. In Section 3, the fault-tolerant routing algorithm resulting from applying the methodology is evaluated. Finally, in Section 4, some conclusions and future work are drawn.

2 Description of the Methodology

The methodology provides fault-tolerance both in n-dimensional mesh and torus networks. For the sake of clarity, the description will mainly be based on a 3D

[1] Intermediate nodes were also used by Valiant [15] for traffic balance.

torus network, with some particular cases explained on a 2D torus network. The methodology tolerates both link and node failures. A node failure can be modeled by the failure of all the links connected to it. Therefore, we will focus only on link failures. When a link fails, we assume that it fails in both directions. The methodology will assume a static fault model, thus, it will know in advance where the failures are located. The proposed methodology will be focused only in the computation of the new routing info for every source-destination pair[2].

The methodology assumes that the initial (i.e., without faults) routing algorithm routes packets by using fully adaptive routing with at least two virtual channels (at least one adaptive and one escape) per physical channel. The adaptive channel(s) enables routing through any minimal path whereas the escape channel guarantees deadlock freedom based on the bubble flow control [13]. A fault-free path is computed by the methodology for each source-destination pair. In the presence of faults, those paths that may use some faulty components are not valid. The methodology avoids those faults by using intermediate nodes for routing. Packets are first forwarded to an intermediate node, and later, from this node to the destination node. Minimal adaptive routing is used in both subpaths. Notice that packets are not ejected from the network at the intermediate node.

Next, we will describe how the intermediate nodes are selected.

2.1 Intermediate Nodes

We denote the source node of a path as S and the destination node as D. A link connects two nodes whose coordinates only differ in one of the dimensions. Faulty links (F_f) are represented by using the identifier of the node (the faulty node F) with the lowest coordinate in the faulty dimension, or the node with the highest coordinate if the link is the wraparound, and the faulty dimension (f). The three coordinates of a given node N are denoted as (X_N, Y_N, Z_N). Intermediate nodes are denoted as I.

If a faulty link F_f can be reached by packets sent from S to D, then an intermediate node I is selected (if possible) in order to avoid F_f. Thus, an intermediate node is used only if there is at least one failure along any possible minimal path between S and D, that is, in a 3-D torus, if the failure is within the cube defined by S and D. On the contrary, if the failure F_f can not be reached for a particular $S - D$ pair, normal routing is used.

The intermediate node I is selected inside the minimal adaptive cube defined by S and D. Thus, both subcubes defined by S and I, and by I and D, are inside this cube, but they are smaller and avoid the failure. Figure 1 shows these subcubes for a given S, D, and I. If the packet is first sent to I and then to D, the possible paths are reduced to the shaded areas, thus, avoiding the failure. Packets are adaptively routed inside each stretch (S-I and I-D).

At the intermediate node, some action must be performed in order to avoid deadlocks. We simply propose the use of two different escape channels. One of them will be used as escape channel for the S-I stretch and the other one for the I-D stretch. Therefore, we define two virtual networks. Each one relies on

[2] Detection of faults, checkpointing, and distribution of routing info is out of the scope of the methodology.

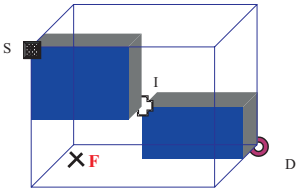

Fig. 1. A minimal adaptive path for a S-D pair using an intermediate node

a different escape channel, but both use the same adaptive channel(s). That is, if a packet is on its way to I, then it uses (if required) the first escape channel. From I, the packet uses (if required) the second escape channel.

2.2 Selecting Intermediate Nodes

There may be several possible I nodes that can be used for each $S - D$ pair. The methodology computes the set of possible I nodes and then selects one of them. Given a faulty link F_f, in order to compute the set of I nodes, some properties can be deduced from the relative positions of S, D, and F in a 3D torus network[3]:

- In one of the three dimensions, the I node must be placed between F and D. That is, I must overcome or leave behind the failure in one of the dimensions. This allows overriding the failure in the path between I and D.
- In another of the dimensions, the coordinate of the I node must lie between S and F. That is, in one of the dimensions I must not overcome the failure. This allows overriding the failure in the cube defined between S and I.
- Finally, in the remaining dimension, the coordinate of the I node can vary between the coordinates of S and D.

Following the previous rules, I will avoid the failure, also providing a minimal path. Figure 2 shows all the possible I nodes in a 3D torus when X is overcome and Y is not overcome and vice versa. Notice that the two first aforementioned rules can also be applied to the Z-dimension, leading to a possibly larger set of possible intermediate nodes.

There are some cases where the previous rules must be done more precise:

- If the S coordinate is equal to the F coordinate in one dimension, then this dimension must be overcome by the I nodes. This case is shown in Figure 3.(a) for a 2D torus, where Y is overcome and X is not, since $Y_S = Y_F$.
- If the D coordinate is equal to the F coordinate in one dimension, then the coordinate of the I node in this dimension must lie between the S and the F coordinates. That is, another dimension should be overcome. In Figure 3.(b), $Y_D = Y_F$, so the I nodes do not overcome Y.

[3] For the sake of simplicity, the dimension f where the faulty link is located has not been taken into account.

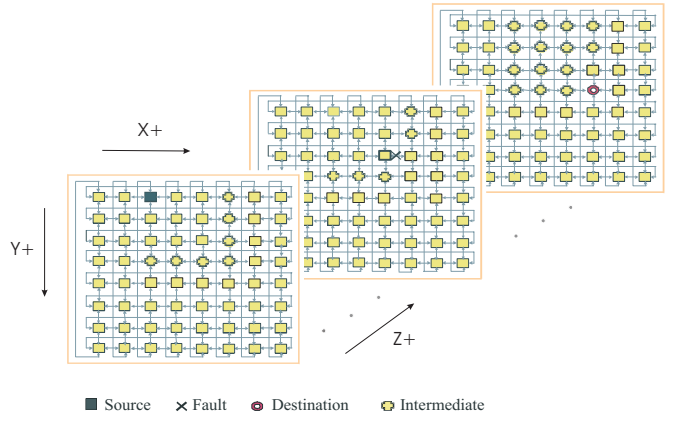

Fig. 2. All the possible intermediate nodes for a particular $S - D$ pair

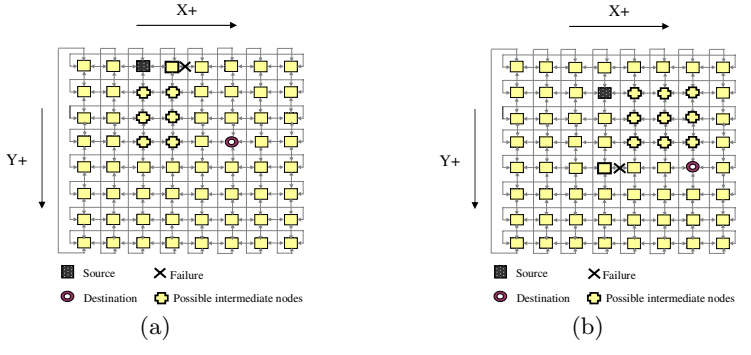

Fig. 3. Possible I nodes for two particular cases: (a) $Y_S = Y_F$. (b) $Y_D = Y_F$

- Another special case arises when S, F, and D are in the same plane. That is, the S, D, and F coordinates are the same for one dimension (for instance, $Z_F = Z_S = Z_D$). In this situation, if we want to follow a minimal path, then the coordinate of I in this dimension must be also the same (i.e., I must also be located in the same plane). Regarding the remaining coordinates, one of them must overcome the fault and the the other one must not overcome the fault. Such situations can be seen in Figures 3.(a) and 3.(b).
- If S, D, and F are in the same ring and the failure is between S and D through the minimal path, it is impossible to find an I node in the minimal path from S to D.

  A possible solution is to travel in the opposite ring direction in order to avoid the failure, selecting an intermediate node along this path. This case can be seen in Figure 4.(a). A possible I node is the one that is halfway between S and D through the non-minimal path. Notice that both paths $S - I$ and $I - D$ are minimal, but the resulting $S - D$ path is non-minimal.

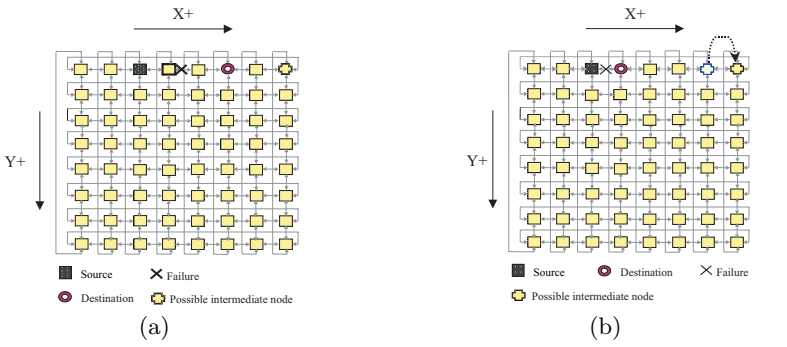

Fig. 4. Situations when S, F, and D are aligned in the same ring

– If S and D share the faulty link (see Figure 4.(b)) and the number of nodes in the dimension is even, it is impossible to find a valid I node in the non-minimal path from S to D as in the previous case.

  The problem is that distance from S to the computed I is the same both using the positive (faulty) and the negative directions of the dimension. Hence, the routing algorithm may select the wrong path. However, the solution is simple. All we have to do is to move I nearer to S. Notice that selecting I in this way also solves the previous case.

As the methodology uses at most one I node for every path, a selection of the final I node is required. Although this selection may affect system performance, at the current stage of this research, the selection is performed randomly.

2.3 Extension to More than One Failure

With the previous rules, all the 1-fault combinations are tolerated. In order to support more than one failure, we define a forbidden zone where all the failures are confined and use I nodes to avoid this zone. Moreover, for a given $S-D$ pair, only those failures located along any minimal path between them will be considered. The I node must be computed following the rules presented in Section 2.2. Figure 5.(a) shows the forbidden zone and the possible intermediate nodes for a 2D Torus network.

However, in a scenario with more than one failure it may be necessary to use additional mechanisms. E.g., with two faults located in the same ring, it is impossible for some $S - D$ pairs to find an adequate I. Figure 5.(b) shows a case, where a combination of seven faults can not be tolerated with one I node and adaptive routing in both stretches.

The figure shows a possible path that could be used, using the node located at $X_S - 1, Y_S + 1$ as I node. If minimal adaptive routing were used from S to I, then some faults could be encountered. In order to properly reach I from S, misrouting and switching off adaptive routing will be used. Misrouting will force routing packets several hops along different directions. Once misrouting is completed, then normal routing (or deterministic routing if adaptive routing

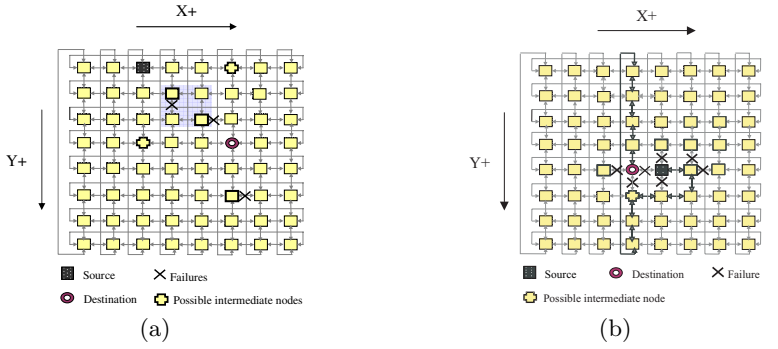

Fig. 5. 2D tori with several failures. (a). A faulty region is defined and I nodes are computed. (b). Misrouting and disabling adaptive routing is required

is switched off) will be applied to the packet. In order to be deadlock-free, the directions to misroute a packet must be used according to the order established by the deterministic routing. In particular, the methodology will use the $X+Y+Z+X-Y-Z-$ direction-order routing, which is deadlock-free and adds routing flexibility (it allows routing packets in both directions of the same dimension).

In the example shown in Figure 5.(b), by using direction-order routing, misrouting, and switching off adaptive routing, the packets will be misrouted one hop in the $X+$ direction, and then forwarded deterministically (by using direction-order routing) to the I node along the $Y+$ and $X-$ direction. In order to reach D from I, misrouting must also be used in order to avoid faults. The packet will be misrouted seven hops in the $Y+$ direction to reach the D node. In each subpath, we assume that packets are misrouted in at most three directions.

To sum up, in a scenario with more than one failure, the methodology will try to override the faults by using all the strategies we have shown so far. Notice that by applying the three mechanisms, it will be possible to compute different fault-free paths for a given $S-D$ pair, thus, being necessary to select among them. In particular, for every $S-D$ pair the methodology will first try to get a minimal path. If no minimal path is found then it will try non-minimal paths by switching off adaptivity and using misrouting in both stretches. The methodology will always pick the path that provides the shortest path.

2.4 Required Resources and Complexity

A packet routed through intermediate nodes requires two subheaders. The first one is used for routing the packet towards the intermediate node, and the second one for routing the packet towards the final destination. At the intermediate node, the first header is removed. However, nodes must select the proper escape channel (if required). If the packet has two headers, then it must select the first escape channel. If it has only one header, the second escape channel must be selected. Packet subheaders also include control fields about misrouting (direction and hops) and switching off adaptive routing (one bit).

The methodology also requires routing info to be stored at each source node. For every destination, this info includes the possible intermediate node (if required) and info about misrouting and switching off adaptive routing. Notice that the amount of required memory is low. In a large system with 65,536 nodes and 5 bytes for routing purposes, the memory size required will be 320KB.

The computational cost of the proposed methodology is low, especially if we take into account that routing info is computed off-line. Most of the affected paths only use an intermediate node to avoid faults, and the cost of computing each intermediate node is $O(1)$. Thus, for all the paths, the computational cost is $O(n^2)$, where n represents the number of nodes. However, in a few cases, misrouting has to be used, thus increasing the computational complexity. The algorithm has to explore all the possible hops along each dimension (up to the network radix, k) until a fault-free path is found. For a 3D Torus (with $n = k^3$ nodes), in the worst case, the computational cost is $O(k^3) = O(n)$. Hence, the computational cost for all the paths by using the proposed methodology is $O(n^3)$.

3 Evaluation of the Methodology

In this section, we evaluate the proposed methodology. First, we are interested in analyzing its fault-tolerance. The methodology is $n-$fault tolerant, if it is able to tolerate any combination of n failures. A given combination of failures is tolerated if every $S - D$ pair in the network can communicate avoiding the failures. Some fault combinations may physically disconnect some nodes from the network. This is not considered as a not tolerated combination.

We are also interested in analyzing how the methodology influences network performance. For this purpose, we compare the performance degradation experienced by our methodology against the one experienced by a mechanism similar to the one used in the BlueGene/L supercomputer. This system is chosen because it represents a state-of-the-art system, and uses adaptive routing and a direct network (3-D torus). Thus, the methodology could be applied to this system.

3.1 BlueGene/L Supercomputer

BlueGene/L [1] is configured as a $64 \times 32 \times 32$ torus of computing nodes constructed with point-to-point serial links between the routers. It uses virtual cut-through [11] and provides both adaptive and deterministic minimal-path routing. Physical channels are multiplexed into up to four virtual channels. Virtual channels are divided into two groups. Two of the virtual channels are used for adaptive minimal routing [7], and two for deterministic minimal routing. One of the deterministic channels is used as escape channel. The bubble flow control [13] is used in order to guarantee deadlock-freedom. The last deterministic virtual channel is reserved for high-priority packets.

The BlueGene/L supercomputer uses a static fault model with checkpointing. Fault-tolerance is achieved by marking healthy nodes as faulty in order to preserve topology and routing, which is extremely convenient when it is hardwired in each router. All the nodes included in four planes (4,096 or 8,192 nodes) that

contain the faulty node/link are marked as faulty. A special hardware bypasses the four planes.

3.2 Simulation Model

A detailed event-driven simulator has been used which models a direct interconnection network with point-to-point bidirectional serial links. Each router has a non-multiplexed crossbar with queues only at the input ports. Each physical input port uses four virtual channels, each providing buffering resources in order to store up to two packets. A round-robin policy has been chosen to select among packets contending for the same output port.

Packets are adaptively routed with minimal paths by using the two adaptive virtual channels. In the two escape channels, packets are deterministically routed following the $X+Y+Z+X-Y-Z-$ order. The two escape channels are used according to the bubble flow control mechanism. When a packet arrives at an input port, the escape queue will be used only if the adaptive queues are full. The output port selected for each routed packet will take into account the information located in the packet header, the status of available output ports, and the status of the neighbor nodes queues.

In all the presented results, the network topology is a 3D torus. Each node has four internal ports connected to the processing node. We present results for $3 \times 3 \times 3$ (27 nodes) and $8 \times 8 \times 8$ (512 nodes) tori. Although actual systems are built with larger topologies (e.g., a $32 \times 32 \times 64$ torus for BlueGene/L), smaller networks can be evaluated exhaustively from a fault-tolerant point of view and the results can easily be extended to larger networks.

For each simulation run, the traffic has the following features. Packet generation rate is constant and the same for all the nodes. The destination of a message is chosen randomly with the same probability for all the nodes. The packet length is set to 128 bytes.

3.3 Evaluation Results

First, we have analyzed all the fault combinations for up to 5 faults in a $3 \times 3 \times 3$ torus. With 5 faults, 25,621,596 fault combinations have been analyzed, and all of them are tolerated by the methodology.

As the number of faults increases, the number of possible fault combinations exponentially increases. Therefore, from a particular number of faults (six faults for the $3 \times 3 \times 3$ torus), it is impossible to explore all the fault combinations in a reasonable amount of time. We will tackle this problem with two approaches. First, we will focus on faults confined in a limited region of the network. Notice that, the worst combinations of faults to be solved by the methodology are those where they are closely located. As the number of fault combinations within such a region is much lower than for the entire network, all the fault combinations can be evaluated. This gives us an approximation of the effectiveness of the methodology in the worst case. Secondly, a statistical analysis is performed, analyzing a subset of the fault combinations, where the faults are randomly located over the entire network. From the obtained results, statistical conclusions are extracted about the fault-tolerance degree of the proposed methodology.

For the first study, the faults are located over a region which will be formed by all the links in the positive direction (in each dimension) of the nodes that are one hop away from a node (the center node) randomly selected[4]. Such a region will be referred to as a *distance 1* region, and will be formed by 21 links (3 links · 6 neighbours + 3 links of the center node) in a $3 \times 3 \times 3$ Torus. Notice that for a high number of faults, the center node will be hardly accessible, as very few links will be not faulty. We have defined the *distance 1* region in such a way that only the center node can become disconnected.

All the fault combinations for up to 14 faults have been analyzed in the *distance 1* region. The methodology is 7-fault tolerant for the *distance 1* region, since the 116,280 fault combinations are tolerated. From 8 faults upwards, some cases are not handled by the methodology. However, the methodology is able to tolerate up to 11 faults in more than 99% of the analyzed combinations. For a greater number of faults, the percentage of supported combinations progressively decreases. Nevertheless, it is hard to imagine a system working for a long time with such a high number of faults, without being repaired.

Next, we present a more realistic scenario, where the faults are randomly located over the entire network. We generate random combinations of n faults and analyze them in order to know if they can be tolerated by our mechanism. We have analyzed 28,400,000 fault combinations in a $3 \times 3 \times 3$ torus with different numbers of link failures (up to 14). For all the cases analyzed, all the fault combinations were solved by the methodology with an error always lower than 0.00074. This error represents the maximum probability that a fault combination is not tolerated by the methodology. Therefore, the mechanism handles faults very efficiently, even for more than 7 link faults.

It must be noticed that the fault tolerance analysis has been performed in a $3 \times 3 \times 3$ Torus. However, because the faults will be at the same or greater distance in a larger network, it is reasonable to expect an equal or even better fault-tolerance degree in larger networks.

Following, we focus on the performance analysis. In order to make the results independent of the relative positions of the failures, we have run 50 simulations (for each number of failures), each of them corresponding to a different randomly-selected failure combination. Figure 6.(a) shows the mean overall network throughput achieved for different numbers of failures in a $8 \times 8 \times 8$ Torus. The confidence intervals are always lower than ± 1.6. As can be seen, the network performance is not seriously affected by the presence of failures (throughput decreases by 5.5% for 6 faults and by 10% for 14 faults).

Finally, we compare the performance degradation when using our methodology against the performance degradation that would be obtained by a fault-tolerant mechanism similar to the one used in the BlueGene/L supercomputer. The BlueGene/L system disables four planes of nodes in order to deal with a fault. As we are using a smaller torus network, we model the mechanism of the BlueGene/L system by only disabling one plane. Figure 6.(b) shows the network

[4] The selection of the center node will not affect results due to the symmetry property of the torus network.

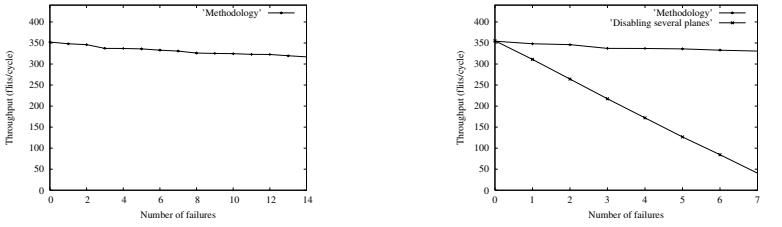

Fig. 6. (a). Mean overall network throughput (flits/cycle) degradation in a $8 \times 8 \times 8$ torus network. (b). Mean overall network throughput degradation for the proposed methodology and for the BlueGene/L like mechanism, in an $8 \times 8 \times 8$ Torus network

throughput obtained with the two methodologies when there are up to 7 faults in a $8 \times 8 \times 8$ Torus. Error bars are not shown as they are too small.

Notice that this is a worst case for the fault-tolerant mechanism similar to the BlueGene/L mechanism, as we assume that the seven faults are located in seven different planes. If the seven faults were located in the same plane, only one plane would be disconnected. As shown in Figure 6.(b), when seven faults are present, our proposed mechanism achieves better performance results than the BlueGene/L-like mechanism does with only one fault present. Thus, even if all the seven faults were in the same plane, our proposed mechanism would still outperform the BlueGene/L-like mechanism. Network throughput degrades only by up to a 6.4% with 7 random faults when using our mechanism, whereas when using the BlueGene/L mechanism, the network performance drops by 88% when disabling seven planes. These results must be put in context. That is, they are obtained in a $8 \times 8 \times 8$ Torus. For larger networks, in particular for the $32 \times 32 \times 64$ Torus used in the BlueGene/L supercomputer, a fault would disconnect four planes of at least 32×32 nodes. That is, 4,096 out of 65,536 nodes. So, in the presence of seven faults, performance would decrease at least 6.25% (if the seven faults are in the same four planes). For our mechanism, in a larger network, performance degradation in the presence of seven faults should be significantly lower than the 6.4% obtained in the $8 \times 8 \times 8$ Torus. This is because the traffic unbalance introduced by the faulty links will be lower in a larger network.

4 Conclusions

In this paper, we have proposed a fully adaptive fault-tolerant methodology valid for n-dimensional mesh and torus networks with a static fault model. The methodology relies mainly on the use of intermediate nodes in order to avoid faults. However, to deal with particular fault configurations, some source-destination pairs communicate through non minimal paths. Also, for some pairs of nodes, adaptive routing is disabled in the subpaths in order to tolerate more faults. Unlike other fault-tolerant approaches, the proposed methodology does

not need to disable any healthy node, only requires one additional virtual channel, and does not degrade performance in the absence of failures.

Evaluation results on a 27-node tori show that the proposed methodology is 7-fault tolerant. Additionally, the percentage of tolerated fault combinations is greater than 99.9% when up to 14 failures are considered. Also, network throughput degrades less than 10% when injecting 14 random failures in a 512-node Torus. In contrast, a fault-tolerant mechanism similar to the one used in the BlueGene/L may degrade network throughput by 88%.

References

1. IBM BG/L Team, An Overview of BlueGene/L Supercomputer, *ACM Supercomputing Conference*, 2002.
2. A.A. Chien and J.H. Kim, Planar-adaptive routing: Low-cost adaptive networks for multiprocessors, *Proc. of the 19th Int. Symp. on Computer Architecture*, pp. 268-277, May 1992.
3. S.Chalasani and R.V. Boppana, Communication in multicomputers with nonconvex faults, *IEEE Trans. on Computers*, vol. 46, no. 5, pp. 616-622, May 1997.
4. W.J. Dally and H. Aoki, Deadlock-free adaptive routing in multicomputer networks using virtual channels. *IEEE Trans. on Parallel and Distributed Systems*, vol. 4, no 4. pp. 466-475, April 1993.
5. W. J. Dally et al., The Reliable Router: A Reliable and High-Performance Communication Substrate for Parallel Computers, *Proc. Parallel Computer Routing and Communication Workshop*, 1994.
6. J. Duato, A theory of fault-tolerant routing in wormhole networks, *Proc. of the Int. Conf. on Parallel and Distributed Systems*, pp. 600-607, Dec. 1994.
7. J. Duato, A Necessary and Sufficient Condition for Deadlock-Free Outgoing in Cut-Through and Store-and-Forward Networks, *Proc. of IEEE Trans. on Parallel and Distributed Systems*, vol. 7, no. 8, pp. 841-854, August 1996.
8. Earth Simulator Center, http://www.es.jamstec.go.jp/esc/eng/index.html.
9. G.J. Glass, and L.M. Ni, Fault-Tolerant Wormhole Routing in Meshes without Virtual Channels, *IEEE Trans. on Parallel and Distributed Systems*, vol. 7, no. 6, pp. 620-636, 1996.
10. C.T. Ho and L. Stockmeyer, A New Approach to Fault-Tolerant Wormhole Routing for Mesh-Connected Parallel Computers, *Proc. of 16th Int. Parallel and Distributed Processing Symp.*, April 2002.
11. P. Kermani and L. Kleinrock, Virtual cut-through: A new computer communication switching technique, *Computer Networks*, vol. 3, pp. 267-286, 1979.
12. D.H. Linder and J.C. Harden, An Adaptive and fault tolerant wormhole routing strategy for k-ary n-cubes, *IEEE Trans. Computers*, vol. C-40 no. 1, pp. 2-12, 1991.
13. V. Puente et al., Adaptive Bubble Router: A Design to Balance Latency and Throughput in Networks for Parallel Computers, *Proc. of the 22nd Int. Conf. on Parallel Processing*, September 1999.
14. Y.J. Suh, B.V. Dao, J. Duato, and S.Yalamanchili, Software-based rerouting for fault-tolerant pipelined communication, *IEEE Trans. on Parallel and Distributed Systems*, vol. 11, no. 3, pp. 193-211, 2000.
15. L.G. Valiant, A Scheme for Fast Parallel Communication, *SIAM Journal on Computing.* vol. 11, pp. 350-361, 1982.