World Applied Sciences Journal 12 (1): 106-113, 2011
ISSN 1818-4952
© IDOSI Publications, 2011

An Efficient Generalized Multi-Fault Tolerant Mapping Algorithm onto a 3-D Torus Interconnection Topology

Mohammed Qatawneh, Azzam Sleit, Moh'd Belal Al- Zoubi,
Ayman Fetyani and Saleh Al-Sharaeh

Department of Computer Information Systems,
King Abdullah II School for Information Technology,
The University of Jordan, Amman 11942, Jordan

Abstract: The Three-Dimensional Torus interconnection topology is widely used in massively parallel machines such as Cray T3D and T3E. The Multi-Fault algorithm presented in this paper is based on base-b reflected gray code approach. The algorithm is efficient for large 3D data domain decompositions, such as large-space simulation problems. The utilization of the algorithm is high, since it utilizes a mechanism to keep a Hamiltonian cycle within the system without the need to have a redundancy in either the nodes or links. The algorithm was implemented on Cray T3D for single fault and it shows a slight degradation in speedup and small overhead as compared to the fault-free case. Since faulty nodes have to be taken out of the running set, we will have two cases to overcome the faulty node(s), either offline or online. The offline case takes an O(n) and the online case takes O(2n). Furthermore, due to the decrease in the size of the targeted hardware, the size of the slab will proportionally increase with the number of faulty nodes. Parallel programming developers should take into account the memory limitations of the nodes. An error ratio must be set before running the algorithm to meet the memory constraints.

Key words: Fault-Tolerant · Parallel Simulation · Hamiltonian Cycle · Gray Codes

INTRODUCTION

High Performance Computing Applications (HPCA) are known to be demanding of computing power. Due to perceived difficulties with programming in a multicomputing environment, interconnection topologies play a vital role [1]. The Massively Parallel Processing (MPP) Cray T3D/T3E architecture employs a 3D torus interconnection topology. Each Processing Element (PE) has a local memory which is physically distributed, but logically shared. Each PE can access any other PE memory without involving the microprocessor in that PE. Cray Research Inc. introduced the first phase which has 256 processors followed by the T3E with 1024 processors and recently Cray XD1 with a shared memory system. In this new generation of MPP based on 3D torus interconnection topologies, each node has 16M bytes that are physically distributed, but logically shared. When a processor needs data at a certain address, which exists in another processor, that processor number combined with the address requested will formulate the effective address and then a very fast search engine will handle the communication without interrupting the computation processor. Latency can be effectively hidden in this manner by transparently overlapping communication with computation, which makes this new generation very effective and scalable [2].

A computation intensive simulation problem demands an efficient parallel processing algorithm [3]. One application is the three-dimensional simulation of the space plasma of Earth's Ionosphere in real time [4-6]. Such a simulation has several advantages, one of which is that it gives realistic results as measured by Satellites probes. Unfortunately, it has a disadvantage when mapped to a system that employs a 3D Torus interconnection topology, which is the probability of having either a faulty node or a link that is relatively high. Moreover, the probability of having a faulty node or even a link proportionally increases as the number of nodes increases.

Corresponding Author: Mohammed Qatawneh, Department of Computer Information Systems, King Abdullah II School for Information Technology, The University of Jordan, Amman 11942, Jordan, E-mail: mba@ju.edu.jo.

One solution to the fault nodes is to have redundant nodes or links by provisioning hardware. Due to chip area limitations [7], it becomes costly to have redundant nodes or links as demonstrated by previous researchers who used the faulty rung concept to find a solution for this problem.

In [8], the researchers proposed the virtual channel concept in which they proposed three virtual channels per physical link. Kim *et al.* [9] proposed four virtual channels per physical link. In the aforementioned work, the solution was restricted to a certain shape region and sometimes it has to disable some nodes around the faulty region, which leads to a degradation in the utilization and even in the traffic flow.

A novel algorithm is presented in [10], in which the researchers divided the nodes into sub-meshes. Alhough the approach showed good results, it has the draw back in terms of utilizing performance metrics such that some nodes adjacent to the faulty node might be skipped. A similar approach done by Huaxi *et al.* [11] considered the faulty node as a hot spot and performed a mechanism to form a balanced ring around the faulty node. Consequently, this resulted in skiping nodes which downgraded the overall system performance solving the even traffic flow problem.

A special characteristic of large-volume data decomposition simulation is that when a fault occurs, it is very costly to restart the simulation all over again. Having redundant nodes as well as redundant links or even increasing the number of virtual channels has a physical limitation and leads to an increase in the system cost. Consequently, a software solution along with a load-balancing technique keeps the hardware cost at minimal as well as increasing other performance metrics such as speedup and utilization.

A recent work on this problem proposed an online software solution with minimal software overhead by utilizing base-b reflected gray codes. In Al-Sharaeh [4], a single fault tolerant algorithm was proposed. The proposed algorithm fully utilized the nodes, but it had the limitation of only a single fault. In this paper, we will extend the work into an efficient multi-fault tolerant scheme. The new approach generalized the single fault problem into a multi fault-tolerant one with the minimum software overhead. To generalize the solution into a multi-fault tolerant one, we proposed a new mathematical model and then analyzed it with an example.

Torus Interconnection Network: Let T be an n-D torus network, such that $N = \prod_{i=1}^{n} k_i$. The nodes in N are arranged in n-dimensions, where k_i is called the radix of the i^{th} dimension, where i: $1 \le i \le n$. In N, the links are either half-duplex or full duplex. The vertices of the graph N are the nodes with address $(a_1,a_2,..,a_n)$ where the vertex α_i represents the node position in the i^{th} dimension. For example, if the node coordinate is (2,3,4), that means it is located two hops in the x-dimension and 3 hops in the y-dimension, 4 hops link count in the z-dimension. Nodes with address $\{a\}$ and $\{b\}$ in a full-duplex n-dimensional torus are connected if there exists i, such that $a_i = (b_i \pm 1) \bmod k_i$ and $a_j = b_j, \forall j \ne i$. Thus, each node is connected to two neighboring nodes in each dimension. In the unidirectional torus, node $(a_1,a_2,..,a_n)$ is linked to node $(b_1,b_2,..,b_n)$ if there exists i such that $1 \le i \le n$. The k-ary n-cube is a restricted case of the torus network in which the radices of all dimensions are equal to k. When k = 2, the k-ary n-cube topology collapses to the well-known hypercube topology [12].

The PIC Code Model: The Particle-In-Cell (PIC) code technique [4-6,13] is an efficient simple method of solving a wide variety of complex scientific problems such as the non-linear evolution of the lower hybrid waves in the Earth's ionosphere and charge interaction with space vehicles, which involves a large number of particles moving under the action of internal and external forces. The defining characteristic of PIC simulations is the method of calculating the force on each particle. By dividing the simulation box into cubic cells in which the particles fall, we are able to use the resulting grid in the solution of a field equation from which the force on each particle can then be determined; a periodic spatial domain is assumed. The calculated forces on each particle determine the new position of the particle. This is much more efficient than summing N^3 coulomb interactions among N particles [4-6].

The three-dimensional geometric space associated with the simulated special domain is subdivided into a number of equal size cubic cells ($\Delta x \times \Delta y \times \Delta z$) appropriately large. Each cell is then initialized with an equal number of particles whose initial velocity components (v_x,v_y,v_z) and position (x,y,z) are chosen randomly from Maxwellian and Uniform distributions,

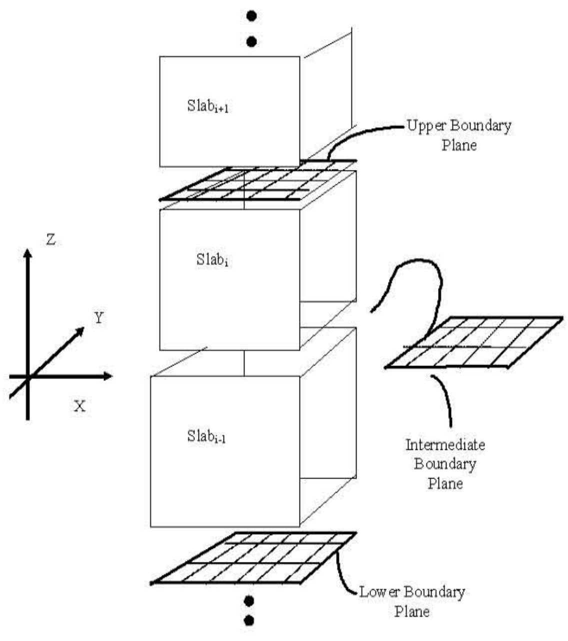

Fig. 1: Communication Boundaries

respectively. As shown in Figure 1, the simulated volume is of size $L_x \times L_y \times L_z$ and the total number of cells is $N_x \times N_y \times N_z$. The particles represent electrons and ions of the plasma.

Each grid point is then assigned a partial charge, based upon the locations of the particles which reside in the eight adjacent cubic cells [5]. The electric potential of each grid point is calculated by solving the Poisson equation:

$$\nabla^2 \phi(x_i, y_j, z_k) = -\rho(x_i, y_j, z_k) \Big/ \varepsilon_0 \qquad (1)$$

Where ϕ is the electric potential, ρ is charge density and ϵ0 is the permittivity of the space. The electric field $\vec{E}$ is determined by solving:

$$\vec{E} = -\nabla \phi(x_i, y_j, z_k) \qquad (2)$$

The equations of motion for the particles are then solved using the leap-frog scheme in a uniform magnetic

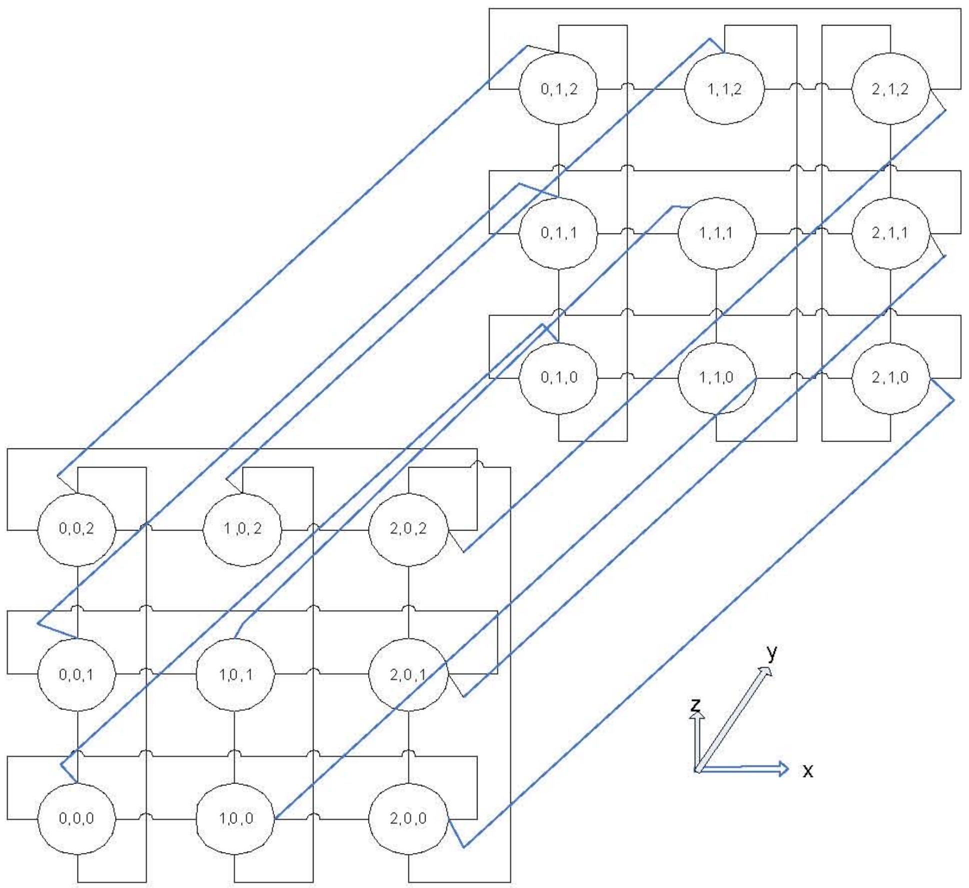

Fig. 2: A (3,2,3) 3D Torus with fault at node (1,0,0)

Table 1: Mapping 4-Cube onto 3-D Torus of Size (2,4,2)

Slab*i*	V_t
0	0,0,0
1	1,0,0
2	1,1,0
3	0,1,0
4	0,2,0
5	1,2,0
6	1,3,0
7	0,3,0
8	0,3,1
9	1,3,1
10	1,2,1
11	0,2,1
12	0,1,1
13	1,1,1
14	1,0,1
15	0,0,1

field, $\vec{B}$, to evaluate the new particle position and velocity in the x, y and z dimension:

$$\frac{d\vec{v}}{dt} = (q/m)(\vec{E} + \frac{\vec{v}}{e} \times \vec{B}) \tag{3}$$

$$\frac{d\vec{r}}{dt} = \vec{v} \tag{4}$$

Where *m* and q are the particle mass and charge and $\vec{v}$ is the velocity (v_x,v_y,v_z).

Decomposition and Mapping Methodology: The 3D box extends along the earth magnetic field that extends from the north pole to the far end of the Earth's south pole. Such a simulation, when implanted on an interconnection topology, demands the nodes to be reconfigured to form a Hamiltonian cycle. Given a graph $G = (V,E)$, if G is a simple graph with $n \geq 3$ vertices and if $\deg(v) + \deg(w) \geq n$, for each pair of non-adjacent vertices v and w, then G is Hamiltonian[13]. A Hamiltonian cycle in G is a path in the graph, starting and ending at the same node, such that every node in V appears on the cycle exactly once. Figure 2, illustrates an example partial of (3,2,3) torus and a Hamiltonian cycle as follows: (0,0,0), (1,0,0), (2,0,0), (2,1,0), (1,1,0), (0,1,0), (0,1,1), (1,1,1), (2,1,1), (2,1,2), (1,1,2), (0,1,2), (0,0,0).

The 3D Box that extends along the Earth's Magnetic field will be partitioned evenly through a pure data decomposition technique. Because of the characteristic of plasma simulation Ions will move freely into adjacent slabs. Hence, adjacent slabs should be mapped to physically adjacent processors. Table 1 shows the mapping of slabs to processors. The targeted processor ID is the V_t using base-b reflected gray code [4].

Generalized Multi-Fault Tolerant Algorithm: A previous research on this problem [4-5] has addressed a single fault solution which is due to either a node or link failure. In this work, we will extend the mentioned previous work to address the multi-fault case with the objective to keep the system fully utilized. This is accomplished by keeping a Hamiltonian cycle and adjusting the slab size to fit the fully operational node. The previous wok mentioned kept the slab size fixed, while in this work, to better utilize the parallel system, the slab sizes are allowed to vary. If we do the restructuring of the 3D simulation space along the z-axis and if the number of nodes are P, then each node will have $(N_x N_y N_z)/P$, where N_x, N_y and N_z are the number of nodes in the x-, y- and z-dimension respectively. If a fault occurs, the new slab size will be $(N_x N_y N_z)/(P-n)$, where n is the number of faulty nodes.

As mentioned earlier, for the 3D Torus of size (2,4,2), the Hamiltonian cycle would be: 000→100→110→010→020→120→130→030→031→131→121→021→011→111→101→001.(000) refers to the address of the first node at the origin and (1,3,1) refers to the node address at 1 hop away from the origin in the x-direction, 3 hops away from the origin in the y-dimension and 1 hop away from the z-dimension in the z-direction. The proposed algorithm for multi-fault is as follows:

Algorithm:

This algorithm takes the argument (CN, FN, DN) and returns a new destination node, where:

Symbol	Description	Symbol	Description
CN	Current node	Cx	Current Node with X axis
FN	Fault node	Cy	Current Node with Y axis
DN	Destination node	Cz	Current Node with Z axis
CN_Set	Set of all neighbors of the Current node	Dx	Destination Node with X axis
DN_Set	Set of all neighbors of the Destination node	Dy	Destination Node with Y axis
CNz	Current node with Z axis	Dz	Destination Node with Z axis
DNz	Destination node with Z axis	Xdim	Rang of X axis used (Dimension)
NDNode	New Destination Node	Ydim	Rang of Y axis used (Dimension)
NCNode	New Current Node	Zdim	Rang of Z axis used (Dimension)

Step 1: Find neighbors for the Current Node (X,Z,Y) Called Cnneighbor_set:

$$\overline{X} = \begin{cases} \begin{cases} X+1, if\,(X+1) \le X_{\dim} \\ OR \\ X-1, if\,(X-1) \ge X_{\dim} \end{cases} \\ Y \\ Z \end{cases}, \quad \overline{Y} = \begin{cases} X \\ \begin{cases} Y+1, if\,(Y+1) \le Y_{\dim} \\ OR \\ Y-1, if\,(Y-1) \ge Y_{\dim} \end{cases} \\ Z \end{cases}, \quad \overline{Z} = \begin{cases} X \\ Y \\ \begin{cases} Z+1, if\,(Z+1) \le Z_{\dim} \\ OR \\ Z-1, if\,(Z-1) \ge Z_{\dim} \end{cases} \end{cases}$$

Step 2: Find neighbors for the Destination Node COORDINATE (X,Z,Y) Called DNneighbor:

$$\overline{X} = \begin{cases} \begin{cases} X+1, if\,(X+1) \le X_{\dim} \\ OR \\ X-1, if\,(X-1) \ge X_{\dim} \end{cases} \\ Y \\ Z \end{cases}, \quad \overline{Y} = \begin{cases} X \\ \begin{cases} Y+1, if\,(Y+1) \le Y_{\dim} \\ OR \\ Y-1, if\,(Y-1) \ge Y_{\dim} \end{cases} \\ Z \end{cases}, \quad \overline{Z} = \begin{cases} X \\ Y \\ \begin{cases} Z+1, if\,(Z+1) \le Z_{\dim} \\ OR \\ Z-1, if\,(Z-1) \ge Z_{\dim} \end{cases} \end{cases}$$

Step 3: Check Destination Node

```
If (Active) → one node fault
{Current Node=CNneighbor → Set n Dnneighbor → Set except the faulty Node
Current Node → Destination Node}
else (Not Active) → Two node fault consecutive
{$if\,(C_x = D_x)$ and $(C_z = D_z)$ Then
{Case Y axis

Current Node = Cnneighbor_ Set∩DN_neighbor_Set exceptionthe faulty Node Return (Current Node)}

else
$if\,(Cn_x = Dn_x)$ and $(Cn_y = Dn_y)$ Then
{Case Z axis
Current Node = Cnneighbor_Set∩DN_neighbor_Set and exception the faulty Node Return (Current Node)}

else
Case X-axis
//Determine New destination node {

$if\,(CN_z \le D_z) And (CN_y \ge DN_y) And (DN_Z < D_{Dim} - 1) Then$
$NDNode = \begin{cases} X \\ Y \\ Z+1 \end{cases}$
else
$if\,(CN_z \ge D_z) And (CN_y \le DN_y) And (DN_Z > 0) Then$
$NDNode = \begin{cases} X \\ Y \\ Z-1 \end{cases}$
else
$if\,(CN_z < DN_z) And (CN_y = DN_y) And (DN_Z = z_{Dim} - 1) Then$
$NDNode = \begin{cases} X \\ Y+1 \\ Z \end{cases}$
```

```
else
$if(CN_z > DN_z) And (CN_y = DN_y) And (DN_Z = 0) Then$
$NDNode = \begin{cases} X \\ Y-1 \\ Z \end{cases}$
//Determine New Current node(NCNode) With the dependence on Current node{

$if(CN_z < DN_z) And (CN_y = DN_y) Then$
$NCNode = \begin{cases} X \\ Y+1 \\ Z \end{cases}$
else
$if(CN_z > DN_z) And (CN_y = DN_y) Then$
$NCNode = \begin{cases} X \\ Y-1 \\ Z \end{cases}$
else
$if(CN_z = DN_z) And (CN_y > DN_y) Then$
$NCNode = \begin{cases} X \\ Y \\ Z+1 \end{cases}$
else
$if(CN_z = DN_z) And (CN_y < DN_y) Then$
$NCNode = \begin{cases} X \\ Y \\ Z-1 \end{cases}$
```

Find all paths for a New Current node

Path X axis

```
$if(C_x < D_x) Then$
$NCNode = \begin{cases} X+1 \\ Y \\ Z \end{cases}$
else
$NCNode = \begin{cases} X-1 \\ Y \\ Z \end{cases}$
```

Path Z axis

```
$if(C_z < D_z) Then$
$While((C_z - D_z) \neq 0)$
{
    $NCNode = \begin{cases} X \\ Y \\ Z+1 \end{cases}$
}
else
$While((C_z - D_z) \neq 0)$
{
    $NCNode = \begin{cases} X \\ Y \\ Z-1 \end{cases}$
}
```

Path Y axis

```
$if(C_y < D_y) Then$
    $NCNode = \begin{cases} X \\ Y+1 \\ Z \end{cases}$
else
    $NCNode = \begin{cases} X \\ Y-1 \\ Z \end{cases}$
```

```
Current Node ← NCNode
Return (Current Node)}
```

Illustration Examples: To illustrate the proposed algorithm, we present the following numerical example. In the previous section, we present the following Hamiltonian cycle for a 3D torus of size (2, 4, 2):

000→100→110→010→020→120→130→030→031→131→121→021→011→111→101→001.

Let us assume we got an error message from the operating system, network layer, that node (x,y,z) is at fault. According to the algorithm, we have to by pass this node to another one since it is not in the network system anymore. Hence we would either backout in all dimensions till we get to a node that is not at fault. For the case, we backed out and we did not find an active node. When backing out, we will be forced to go forward in all dimensions till we reach an active node. To find the sets, (i.e., neighboring sets of all nodes adjacent to the faulty nodes), we follow the steps in the algorithm above. Say, if the node (0, 2, 0) is at fault, then the Current Node is (0, 1, 0) according to Step 1, (i.e., backing out one step in the x-dimension) and Destination Node is (1, 2, 0) because we by pass the faulty node at (0,2,0). In order to keep the routing and slabs allocations at optimal, we have to follow the nodes that constitute the Hamiltonian cycle. We kept the equations as we modeled the algorithm, but we plugged in the numbers for the example case, for the sake of clarity.

Step 1: {(0,0,0) (0,1,1) (1,1,0) (0,2,0)}

Step 2: {(0,3,0) (0,2,1) (1,1,0) (0,2,0)}

In order to avoid selecting a destination node that is not active, step 3 is checked out.

Step 3: Check Destination Node Is not active then Find all paths for a New Current node

Path X axis	Path Y axis	Path Z axis
$if(C_x = 0 < D_z = 1$ *Then* $NCNode = (\underline{1},2,0)$	$if(C_y = 2 < D_y = 2)$ *Then* $NCNode = (1,1,3)$	$if(C_x = 0 < D_z = 1)$ *Then* $NCNode =$
	else	$(0,2,\underline{\underline{0}})$
	$NCNode = (0,3-1,0)$ $NCNode = (0,\underline{\underline{2}},0)$	

Return (Current Node = $NCNode = (1,\underline{\underline{2}},1)$)

From Step 3, the x-axis path results in NCNode = (1,2,0), if the x-coordinate is "1" and if the y-axis NCNode (0,2,0), 2 is the y-coordinate. The same happens if the z-axis = (0,2,0), then"0" is the z-axis. As a result, the algorithm returns the new destination node (1,2,0). The built-in routing algorithm will find an alternative path to the new destination node, which could either be routed through 110 with a cost of 2 hops, or through 011, 021 and then 121, with a cost of 4 hops.

Algorithm Characteristics: This proposed algorithm can be applied online or offline. For the offline case, scanning the operational nodes comes just before constructing the 3D shape mapping. The faulty nodes will be removed from the destination mapping set. The online case is when the node(s) become faulty during the run time. It is evident that the offline method is faster in constructing the shape, but slower in the preprocessing steps. This is because we need to scan all the nodes in the targeted MPP hardware and this will require an *O(n)*. And for the online case, it takes 2*O(n)*, which is an *O(n)* again. Furthermore, with this algorithm, we can apply up to (n-1) faulty nodes. As mentioned earlier, the size of the slab will proportionally increase with the number of faulty nodes. Parallel programming developers should take into account the memory limitations of the nodes. An error ratio must be set before running the algorithm.

CONCLUSION

Large space simulation problems require an efficient mapping technique. For an efficient mapping of 3D simulation problems when the data decomposition technique is used, the mapping should take into consideration the communication cost. A special characteristic of space plasma simulation problems is that the communication only occurs at the adjacent slabs. The targeted MPP should be configured into ring topology (Hamiltonian). One challenge to such a technique is the increasing likelihood of having a single to multi-faulty node. Due to hardware limitation and cost, a software approach is necessary. The proposed algorithm efficiently resolves the multi-fault problem and fully utilizes the operation nodes with an optimal number of steps. It is evident from the example, that for a single fault, it takes from two to four hops at the maximum. It is true that the built-in routing algorithm in the MPP hardware such as Cray 3D might run into flits collision that result in a re-routing, but without the new algorithm, the simulation has to come to halt and the simulation of the application has to be re-submited again.

REFERENCES

1. Duati, J., S. Yalamanchili and L. Ni, 2003. Interconnection Networks, an Engineering Approach, Morgan-Kaufmann Press.
2. Cray Research Inc., 2010. Closing the gap between peak and achievable performance in high performance computing,White Paper WP-0020404. http://www.cray.com/downloads/whitepaper_closi ng_the_gap.pdf.
3. Birdsall, C.K. and A.B. Langdon, 1991. Plasma Physics via Computer Simulation, New York: Adam Hilger.
4. Saleh Al-Sharaeh, H., 2008. Efficient Fault Tolerant Mapping of Large Three-Dimensional Simulation onto 3D Tori Graph, Accepted, pp: 239-248, 21(2). European J. Scientific Research.
5. Saleh Al-Sharaeh, H., 2007. On the Hamiltonian cycle mapping onto 3-D torus interconnection network based on base-b reflected gray codes, Applied Mathematics and Computation, 186(2): 1311-1321.
6. Hosni Al-Sharaeh, S., B. Earl Wells and Nagendra Singh, 1996. A Massively Parallel Particle-In-Cell Technique for Three-Dimensional Simulation of Plasma Phenomena, Proceedings of the 9th International Conference on Parallel and Distributed Computing Systems (PDCS96), Dijon, France, pp: 25-27.
7. Susumu Horiguchi and Takayuki Ooki, 2000. Hierarchical 3D-Torus Interconnection Network International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN '00), pp: 50.
8. Chien, A.A. and J.H. Kim, 1992. Planner-adaptive routing:low-cost adaptive networks for multiprocessor, 19th Annual Int. Symp. On Computer Architecture, pp: 268-277.
9. Seong-Pyo Kim and Taisook Han, 1998. Fault-tolerant adaptive wormwhole routing in2D mesh, IEICE Trans. Information system E81-D., 10: 1064-1072.
10. Yamin Li, Shietung Peng and Wanming Chu, 2005. Adaptive box-based efficient fault-tolerant routing in 3D torus, Parallel and Distributed Systems, Proceedings. 11th International Conference on 1(20-22): 71-77.
11. Huaxi Gu, Jie Zhang, Kun Wang, Zengji Liu and Guochang Kang, 2007. Enhanced fault tolerant routing algorithms using a concept of "balanced ring, J. Systems Architecture: the EUROMICRO J. Archive, 53(12): 902-912.
12. Hashemi-Najafabadi, H and Hamid Sarbazi-Azad, 2007. Combinatorial performance modelling of toroidal cubes, In Press, J. Systems Architecture, xxx: xxx-xxx.
13. Harlow, F.H., 1955. A Machine Calculation Method for Hydrodynamic Problems. Los Alamos Scientific Laboratory Report LAMS-1956.
14. Ore, O., 1962. Theory of Graphs, American Mathematical Society Colloquium Publications, xxxviii.