A CASE STUDY OF FAULT TOLERANT ROUTING MECHANISM FOR TORUS EMBEDDED HYPERCUBE INTERCONNECTION NETWORK FOR PARALLEL ARCHITECTURE

N. Gopalakrishna Kini* , M. Sathish Kumar** & Mruthyunjaya H. S.***

This paper describes a fault-tolerant routing mechanism to facilitate data routing in Torus embedded hypercube interconnection network subject to node failures in parallel computing. It is shown that by only using feasible paths routing can be substantially simplified. Though there are algorithms for fault tolerant communication in torus and hypercube networks, there exits no efficient algorithm for the embedded architecture. We present an algorithm to provide an efficient fault tolerant routing mechanism for a (2, 2, 8)-Torus embedded hypercube interconnection network.

Keywords: Hypercube Network, Torus Network, Embedded Network, Torus Embedded Hypercube Network, Fault Tolerance, Fault List, Routing.

1. INTRODUCTION

It is quite evident that fault tolerance in highly parallel computers is important for achieving reliable and high performance computing [1]-[3]. Fault tolerance is the ability of an interconnection network to continue operating in presence of single or multiple faulty nodes [4]. As the network size scales up [11]-[13] the probability of processor failure also increases. It is therefore essential to design fault tolerant routing algorithms that allow to route messages between non-faulty nodes in the presence of faulty processors.

The hypercube network’s node degree grows logarithmically with number of vertices making it difficult to build scalable architectures [7]-[9]. On the other hand, the torus network supports the scalability as all the nodes are having constant node degree. A better network performance is achieved by embedding the torus and hypercube networks that will give rise to a Torus embedded hypercube network. Such a combination results in a system which can be implemented with small node degree and a reduction in hardware cost per node. Also, a constant node degree results in a system that is scalable without having to modify the individual nodes [4]-[6].

Torus embedded hypercube network, when one or more nodes fail, a large number of available links enable the fault free nodes to continue communicating with other nodes. We consider a (2, 2, 8)-torus embedded hypercube interconnection network for demonstrating our algorithm. Our algorithm gives the optimum path even in the presence of single/multiple faulty nodes in the interconnection network.

2. ARCHITECTURAL PROPERTIES OF TORUS EMBEDDED HYPERCUBE NETWORK

In this section, a brief discussion on embedding of the torus and the hypercube networks is done to obtain the torus embedded hypercube [2], [11] as in (1)-(5).

\[
T_1(i, j, k) = (i, (j + 1) \mod m, k) \quad (1)
\]

\[
T_2(i, j, k) = (i, (m + j - 1) \mod m, k) \quad (2)
\]

\[
T_3(i, j, k) = ((i + 1) \mod l, j, k) \quad (3)
\]

\[
T_4(i, j, k) = ((l + i - 1) \mod l, j, k) \quad (4)
\]
The address of individual node is represented by \( n \)-bit binary vector. A link will exist between two nodes where the addresses of these two nodes differ exactly by one bit [7]. For a \((2, 2, 8)\)-torus embedded hypercube network, a node with a five bit address has its left most bit representing row number, the next bit representing column number and the remaining least significant bits representing the address of a node in the hypercube as shown in Figure 1.

The ring connections of row/column of each torus are not shown in Figure 1 for simplicity. A wraparound connection is done along each row/column if they have same label as a completion of \((2, 2, 8)\)-torus embedded hypercube network.

3. Fault Tolerance in Torus Embedded Hypercube Network

The algorithm developed is explained in this section. This algorithm technique enable to formally prove that even with multiple faulty nodes, the torus embedded hypercube interconnection network can still route the data successfully.

Each node in torus embedded hypercube network is labeled with an \( n \)-dimensional binary vector \((x_1, x_2, \ldots, x_{n-1}, x_n)\), where \( x_i \in \{0, 1\} \) for \( 1 \leq i \leq n \). The neighboring nodes of a node \( Z \) is different from \( Z \) in one bit with reference to their binary representations. In the binary vector the initial bits \( x_1, \ldots, x_i \) and \( x_{i+1}, \ldots, x_n \) are representing the row number and column number respectively where \( i \leq j < n \). Note that, for the \((2, 2, 8)\) network \( i = 1, j = 2 \) and \( n = 5 \) are considered.

In the initial phase of the algorithm, \( x_{j+1}, \ldots, x_n \) bits are taken into account by considering \( x_1, \ldots, x_{i+1}, \ldots, x_i \) as don’t care combinations. The source node presently referred to as the current node which would like to communicate with the destination will compare each bits of its address \( x_{j+1}, \ldots, x_i \) starting from left most bit with the corresponding bits of its destination node. If the bits are found to be same we leave current node address as it is and move on to the very next consecutive bit which is at the right of it. If the bits are different then that bit in the current node is changed as that of corresponding bit of destination node to derive a new node.

Now this new node will become the current node for further comparison. This is done till all the bits \( x_{j+1}, \ldots, x_n \) are processed. We assume that each processor has a fault list for recording the faulty/nonfaulty status of its neighboring nodes. The new node derived, if found in the fault list, will be ignored; otherwise will be considered as current node and the process of comparison will be continued from thereon. As new nodes are encountered which is not in the fault list, they will get added to the list of routing path. With this a data routing path is discovered within the cube by ignoring all the faulty nodes with respect to current node [8]-[10].

In the second phase, \( x_1, \ldots, x_i \) are taken into account by considering \( x_{j+1}, \ldots, x_i \) as don’t care combinations. The current node which is obtained from the previous phase will now continue this process of comparison of bits of its address starting from left with the corresponding bits of its destination node. If the bits are found to be same we leave current node address as it is and go ahead with the very next consecutive bit.

If the bits are different then that bit in the current node is changed as that of corresponding bit of destination node to derive a new node. Now this new node will become the current node for further comparisons. This is done till all the bits in \( x_1, \ldots, x_{i+1}, \ldots, x_i \) are processed. Again the new node derived if found in the fault list will then be ignored; otherwise considered as current node and the process of comparison will continue. As new nodes are encountered which are not in the fault list, they will be added to the list of routing path. With this a data routing path is discovered within the concurrent torus by ignoring all the faulty nodes with respect to current node.

4. Results and Discussion

Table 1 shows the result of the algorithm that has provided the data routing paths in \((2, 2, 8)\)-torus embedded hypercube network. A routing between seven cases of random source and destination nodes are considered for demonstration. An efficient optimal routing path is established between them with reference to the available links as in Figure 1 and
5. Conclusion and Future Work

We have developed a data routing algorithm for a (2, 2, 8)–torus embedded hypercube network. The analysis and functioning of the algorithm has been provided with the situation wherein the network is fault free. Also the case is considered to support our algorithm with source node of address 00000 and a destination node of address 01111 communicating each other by considering all combinations of faulty nodes. A node in torus embedded hypercube network can tolerate not more than \((n-1)\) faulty nodes, where \(n\) is the node degree and also a non-faulty node will get disconnected when all its \(n\) neighbors become faulty. Though the algorithm is for a specific (2, 2, 8) network, we feel that the algorithm can be generalized with some limited modifications.

REFERENCES


