2012 International Conference on Computer Networks and Communication Systems (CNCS 2012)
IPCSIT vol.35(2012) © (2012) IACSIT Press, Singapore

A Parallel Routing Algorithm for Torus NoCs

Khaled Day, Nasser Alzeidi, Bassel Arafeh, Abderezak Touzene

Department of Computer Science, Sultan Qaboos University, Muscat, Oman

Abstract. This paper proposes a parallel routing algorithm for routing multiple data streams over disjoint paths in the torus Network-on-Chip (NoC) architecture. We show how to construct a maximal set of disjoint paths between any two nodes of a torus network topology and then make use of the constructed paths for the simultaneous routing of multiple data streams between these nodes. Analytical performance evaluation results are obtained showing the effectiveness of the proposed parallel routing algorithm in reducing communication delays and increasing throughput when transferring large amounts of data in an NoC-based multi-core system.

Keywords: network-on-chip (NoC), 2D mesh, torus, multipath routing, disjoint paths

1. Introduction

With advances in technology, chips with hundreds of cores are expected to become a reality in the near future. Traditionally, communication between processing elements was based on buses. When the number of processing elements is large, the bus becomes a bottleneck from performance, scalability and power dissipation points of view [1]. A network-on-chip (NoC) is instead used to interconnect the processing elements. The topology of the network-on-chip has a major impact on the overall multi-core system performance [2]. Several topologies have been proposed and studied for NoCs including mesh-based and tree-based topologies [2]. Mesh-based topologies (especially the 2D mesh and the torus) have been the most popular of these topologies. Their popularity is due to their modularity (they can be easily expandable by adding new nodes and links without modifying the existing structure), their ability to be partitioned into smaller meshes (a desirable feature for parallel applications), their simple XY routing strategy, and their facilitated implementation. They also have a regular structure and short inter-switch wires. They have been used in many systems such as the RAW processor [3], the TRIPS processor [4], the 80-node Intel's Teraflops research chip [5], and the 64-node chip multiprocessor from Tilera [6].

In this paper we contribute to the study of the torus topology by showing how to construct a maximal set of disjoint paths between any two nodes of a torus and how to use these paths for parallel routing. The proposed parallel routing algorithm allows the transfer of multiple data streams between any two nodes in the torus over disjoint paths resulting in faster transfer of large amounts of data and higher throughput. PRA can also be used for fault-tolerance purposes by sending multiple copies of critical data on disjoint paths. The critical data can still be delivered even if only one of the disjoint paths is fault-free and the others are faulty. Sources of communication faults in NoCs include crosstalk, faulty links and congested network areas.

2. Notations and Preliminaries

A 2D mesh NoC consists of $k \times k$ switches interconnecting IP nodes. One disadvantage of the 2D mesh topology is its long diameter which has a negative effect on the communication latency. A torus NoC, illustrated in figure 1a, is basically the same as a 2D mesh NoC with the exception that the switches on the edges are connected with wrap-around links. Every switch in a torus has five active ports: one connected to the local IP node, and the other four connected to the four neighboring switches (left, right, up and down). The torus topology reduces the latency of the 2D mesh while keeping its simplicity. In order to reduce the length of the wrap-around links, the torus can be folded as shown in figure 1b.

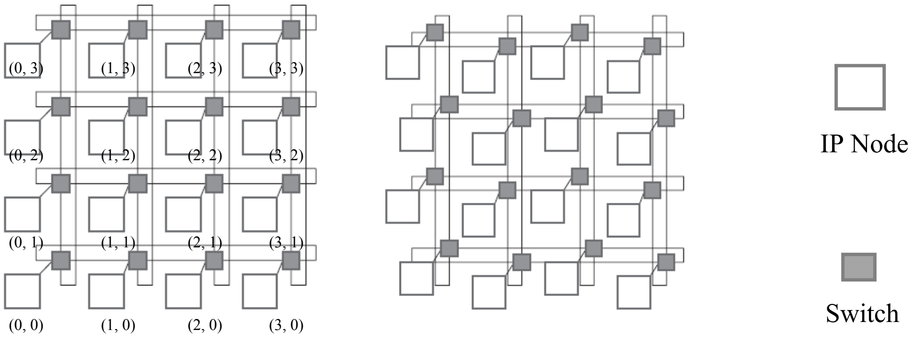

Fig. 1: (a) A Torus NoC Topology (k = 4) (b) A Folded Torus NoC Toplogy (k = 4)

We refer to a node in the torus topology by its pair of X-Y coordinates as illustrated in figure 1a. We show in the next section how to construct node-disjoint paths from a source node S to a destination node D. A path from S to D is a sequence of nodes starting at S and ending at D such that any two consecutive nodes in the sequence are neighbor nodes. We say two paths are node-disjoint if they do not have any common nodes other than the source node and the destination node. A path from S to D can be specified by the sequence of node to node moves that lead from S to D. There are four possible moves from a node to a neighbor node (right, left, up, and down). We denote these moves as $+X$, $-X$, $+Y$ and $-Y$ respectively. When a node is on the rightmost border of the torus, a move to the right uses the wrap-around link that leads to a node on the leftmost border of the torus. Similarly for nodes on the leftmost, top and bottom borders. In other word all + and – operations on the X-Y coordinates are modulo k. With respect to a given source node S and a given destination node D, all $+X$ moves are called forward X moves (denoted F_X) if and only if an initial $+X$ move from S decreases the distance to D along the X dimension. Otherwise all $+X$ moves are called backward X moves and are denoted B_X. These F_X and B_X notations are only defined when S and D differ in the X dimension. Figure 2 provides more precise definitions for F_X and B_X in the form of two functions $F_X(S, D)$ and $B_X(S, D)$ which return the moves that correspond to F_X and B_X for a given source $S = (x_S , y_S)$ and a given destination $D = (x_D , y_D)$. The forward Y moves and backward Y moves (along the Y dimension) and their corresponding F_Y and B_Y notations are similarly defined. We can obtain $F_Y(S, D)$ and $B_Y(S, D)$ functions simply by replacing the x's by y's in the $F_X(S, D)$ and $B_X(S, D)$ functions of Figure 2 respectively.

```
Fx (S, D)
{  // Forward X move for a given source node S = (xS,yS) and a given
   // destination node D = (xD,yD). Only defined when xS ≠ xD
   if (xD > xS and xD - xS ≤ k/2) return +X
   else if (xS > xD and xS – xD > k/2) return +X
   else return -X
}
Bx (S, D)
{  // Backward X move for a given source node S = (xS,yS) and a given
   // destination node D = (xD,yD). Only defined when xS ≠ xD
   if (xD > xS and xD - xS ≤ k/2) return -X
   else if (xS > xD and xS – xD > k/2) return -X
   else return +X
```

Fig. 2: Forward X Moves (F_X) and Backward X Moves (B_X)

3. Node-Disjoint Paths in A Torus

Let $S = (x_S, y_S)$ and $D = (x_D, y_D)$ be any source and destination nodes in the torus. There are at most four node-disjoint paths from S to D corresponding to the four possible starting moves $+X$, $-X$, $+Y$ and $-Y$ from S. We now show how to construct a maximal set of four disjoint paths from S to D. Each of the constructed paths is defined by a sequence of moves that lead from S to D. In a path description we use a superscript notation to indicate the number of consecutive times a move is repeated. For example $+X^2$ denotes a sequence of two consecutive $+X$ moves. Let δ_x be the distance from S to D along the x dimension (i.e. $\delta_x = \min(|x_D - x_S|, k-|x_D - x_S|)$ and let δ_y be the distance from S to D along the y dimension (i.e. $\delta_y = \min(|y_D - y_S|, k-|y_D - y_S|)$. We distinguish the following three cases in the construction of disjoint paths from S to D:

Case 1: If $x_S \neq x_D$ and $y_S \neq y_D$ (S and D on different rows and different columns): Table 1 shows sequences of routing moves of four node disjoint paths from S to D for Case 1 and Figure 3 illustrates these four paths for a 5×5 Torus. The wrap-around links are not shown for clarity of the figure.

Table 1: Four Disjoint Paths from S to D for Case 1

Path	Sequence of Routing Moves
π_{11}	$F_x^{\delta x}, F_y^{\delta y}$
π_{12}	$F_y^{\delta y}, F_x^{\delta x}$

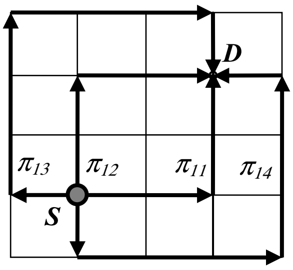

Fig. 3: Disjoint Paths for Case 1 ($k = 5$)

Case 2: If $x_S = x_D$ and $y_S \neq y_D$ (S and D on the same column but different rows): Table 2 shows sequences of routing moves of four node disjoint paths from S to D for Case 2 and Figure 4 illustrates these four paths defined for a 5×5 Torus. The wrap-around links are not shown for clarity of the figure.

Table 2: Four Disjoint Paths from S to D for Case 2

Path	Sequence of Routing Moves
π_{21}	$F_y^{\delta y}$
π_{22}	$+X, F_y^{\delta y}, -X$
π_{23}	$-X, F_y^{\delta y}, +X$
π_{24}	$B_y, +X^2, F_y^{\delta y+2}, -X^2, B_y$

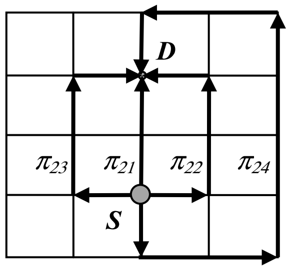

Fig. 4: Disjoint Paths for Case 2 ($k = 5$)

Case 3: If $x_S \neq x_D$ and $y_S = y_D$ (S and D on the same row but different columns): Symmetric to Case 2.

4. The Parallel Routing Algorithm (Pra)

We now propose a parallel routing algorithm (PRA) that allows any source node S in the torus to send to any destination node D, a set of m packets in parallel over disjoint paths. Figure 5 outlines the operation of the parallel routing algorithm at a source node S that wants to send m packets $p_1, p_2, \ldots, p_m$ in parallel to a destination node D. The source node scatters the m packets over the disjoint paths in a round-robin fashion. Upon receiving a forward request packet, an intermediate node (x,y) executes the algorithm of figure 6.

```
Parallel_Route (S, D, p1, p2, …, pm)
{   // Source node S wants to send to destination node D the m packets: p1, p2, …, pm
    i = case(S, D) //identify which of: Case 1, Case 2 or Case 3 applies
    j = 0; k = 0
    while (k < m)                    //while more packets to send
         k = k + 1                   // pk is the next packet to send
         j = j%4 + 1                 // πij is the next path to use
         move = First_Move(πij)      //first move of routing path πij
         next = Neighbor(S, move)    //neighbor to reach after 1st move on πij
         π = Delete_First_Move(πij)  //π = remaining routing path
         send Forward_Request packet <pk, S, D, π > to next
}
```

Fig. 5: Routing at a Source Node

```
Forward (p, S, D, π)
 { // Local node has received a Forward_Request packet <p, S, D, π>
    if (D = local)            // local node is the destination
    {   deliver p to local IP; exit }
    move = First_Move(π)              //first move of routing path π
    next = Neighbor(local, move) //neighbor to reach after 1st move of π
    π = Delete_First_Move(π)   //update remaining routing path
    send Forward_Request packet <p, S, D, π > to next
}
```

Fig. 6: Routing at an Intermediate Node

5. Performance Evaluation

In this section we derive performance characteristics of PRA. We first obtain the lengths of the constructed parallel paths. These lengths are readily obtained from Table 1 and Table 2. The lengths, d_{ij}, of the constructed four π_{ij} paths, $1 \le i \le 3$, $1 \le j \le 4$, are shown in Table 3.

Table 3: Lengths of the Constructed Parallel Paths

Path	Case 1	Case 2	Case 3
1	$\delta_x + \delta_y$	δ_y	δ_x
2	$\delta_x + \delta_y$	$\delta_y + 2$	$\delta_x + 2$
3	$\delta_x + \delta_y + 4$	$\delta_y + 2$	$\delta_x + 2$
4	$\delta_x + \delta_y + 4$	$\delta_y + 8$	$\delta_x + 8$

The proposed routing algorithm splits a message of size M flits over four disjoint paths resulting in approximately $M/4$ flits sent on each path. The message latency for a message can then be calculated as the maximum latency of transferring $M/4$ flits on the four disjoint paths. There are in total $k^2(k^2\text{-}1)$ source-destination pairs (where the source and the destination are different) of which $k^2(k\text{-}1)^2$ pairs correspond to Case 1, $k^2(k\text{-}1)$ pairs correspond to Case 2, and another $k^2(k\text{-}1)$ pairs correspond to Case 3. Therefore, the probability p_i of generating a message that belongs to Case i is given by the following formula:

$$p_i \begin{cases} (k-1)/(k+1) i = 1 \\ 1/(k+1) i = 2 \\ 1/(k+1) i = 3 \end{cases} \tag{1}$$

Averaging over the three possible cases, the average message latency can be calculated as:

$$mean\ message\ latency = \Sigma_{i=1}^{3} p_i \max(T_{i1}, T_{i2}, T_{i3}, T_{i4}) \tag{2}$$

In what follows we calculate T_{ij}. Under uniform traffic pattern, the channel arrival rate can be found by dividing the total channel arrival rates over the number of channels in the network. If each IP generates an average of λ_g messages per network cycle then a total of $N\lambda_g$ messages will be generated in the network. Since each message on path π_{ij} traverses d_{ij} hops and there are 4 output channels in each node, the rate of message received by each channel in the path π_{ij} can be calculated as:

$$\lambda_{ij} = \frac{N\lambda_g d_{ij}}{4N} = \frac{\lambda_g d_{ij}}{4} \tag{3}$$

Since the traffic is uniform and the network is symmetric, all channels in the network have similar statistical characteristics. Therefore the message latency along path π_{ij} is composed of the time to transmit the flits, $M/4$, the routing time, d_{ij}, and the blocking delay encountered at each hop along the path. We assume here that each flit takes one network cycle to be transmitted from one node to the next and the routing decision takes also one cycle. Hence the message latency along path π_{ij} can be written as:

$$T_{ij} = \left[\frac{M}{4} + d_{ij} + d_{ij}\left(w_{ij} PB_{ij}\right)\right] V_{ij} \tag{4}$$

where V_{ij} is the multiplexing factor and $W_{ij}PB_{ij}$ is the blocking delay calculated by multiplying the mean waiting time to acquire a channel W_{ij} by the blocking probability PB_{ij}. The mean waiting time to acquire a channel can be approximated as the mean waiting time of an M/G/1 queue [7]:

$$W_{ij} = \frac{\lambda_{ij}\left(T_{ij}\right)^2\left[1 + \left(T_{ij} - M/4\right)^2 / \left(T_{ij}\right)^2\right]}{2\left(1 - \lambda_{ij} T_{ij}\right)} \tag{5}$$

To calculate the blocking probability for path π_{ij}, we assume that $V \geq 1$ virtual channels can be used per physical channel where any available virtual channel can be selected to route the message to the next hop. Therefore a message will be blocked only when all virtual channels are busy. The probability P_{ij}^{v} that v virtual channels are busy at a physical channel can be determined using a Markovian model [8] as follows:

$$P_{ij}^{v} = \begin{cases} (1-\lambda_{ij}T_{ij})(\lambda_{ij}T_{ij})^{v} & 1 \leq v < V \\ (\lambda_{ij}T_{ij})^{v} & v = V \end{cases} \tag{6}$$

When multiple virtual channels are used per physical channel they share the bandwidth in a time-multiplexed manner. Therefore the message latency has to be scaled by the average degree of multiplexing, V_{ij} (see equation (3) above) which takes place at a physical channel. This can be calculated as follows [8]:

$$V_{ij} = \Sigma_{v=1}^{v} v^2 p_{ij}^{v} / \Sigma_{v=1}^{v} v p_{ij}^{v} \tag{7}$$

An iterative technique with error bound of 0.0001 has been used to evaluate the different variables of the above model. Figure 7 shows plots of the obtained message latency (in network cycles) against the traffic load (message generation rate) for different scenarios. It can be seen from this figure that the message latency is significantly reduced by the use of parallel routing of the message flits over the constructed disjoint paths. It should be mentioned however, that an extra overhead will be needed to assemble the flits of a message at destination nodes. This also requires that flits utilize sequence numbering to maintain the correct order of the message flits. The plots of figure 7 also reveal that using path 1 or path 2 only gives lower latency. This is an expected behavior as these two paths correspond to the optimal routing paths (i.e. shortest paths between any source and destination nodes). It is also clear from the plots that when PRA is used for routing in the torus NoC, the network saturates at high traffic loads which results is higher throughput rates.

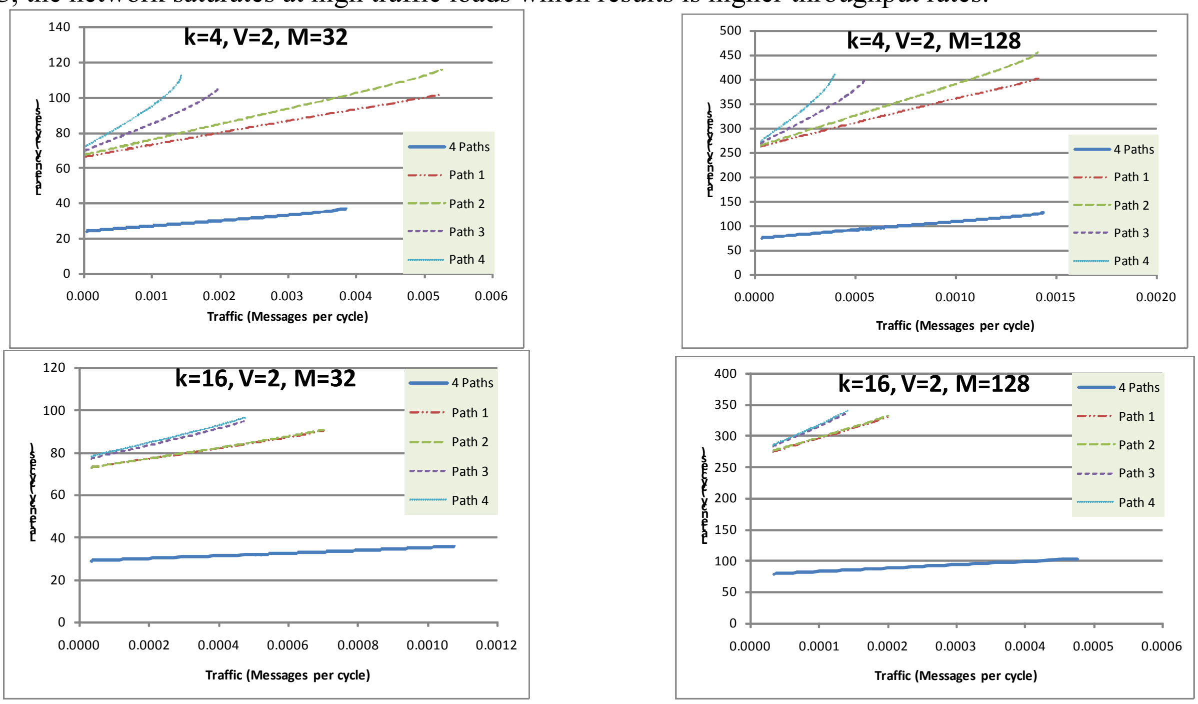

Fig. 7: Message Latency vs Traffic Load

6. Conclusion

We have proposed a parallel routing algorithm (PRA) for transferring multiple data streams over disjoint paths in a torus NoC architecture. The algorithm is based on a construction of disjoint paths between network nodes. Analytical performance evaluation results have been obtained showing the effectiveness of the proposed parallel routing algorithm in reducing communication delays and increasing throughput. The algorithm can be adapted to support fault-tolerant routing of multiple copies of critical data in a Torus NoC over the multiple disjoint paths.

7. References

[1] L. Benini and G. D. Micheli, Networks on Chips: A New SoC Paradigm, *Computer*, vol. 35, no. 1, Jan 2002, pp. 70-78.

[2] L. Benini and G. D. Micheli, *Networks on Chips: Technology and Tools*, Morgan Kaufmann, 2006.

[3] M. B. Taylor, W. Lee, S. Amarasinghe, and A, Agarwal. Scalar Operand Networks: On-Chip Interconnect for ILP in Partitioned Architectures, *International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 341–353, Anaheim, California, 2003.

[4] P. Gratz, C. Kim, R. McDonald, S. Keckler, and D. Burger, Implementation and Evaluation of On-Chip Network Architectures, *International Conference on Computer Design (ICCD)*, 2006.

[5] S. Vangal et al. An 80-Tile 1.28TFLOPS Network-on-Chip in 65nm CMOS, *IEEE Int'l Solid-State Circuits Conference, Digest of Technical Papers (ISSCC)*, 2007.

[6] A. Agarwal, L. Bao, J. Brown, B. Edwards, M. Mattina, C. - C. Miao, C. Ramey, and D. Wentzlaff, Tile Processor: Embedded Multicore for Networking and Multimedia, *Hot Chips 19*, Stanford, CA, Aug. 2007.

[7] L. Kleinrock, Queuing Systems: Theory, vol. 1, New York: John Wiley, 1975.

[8] W.J. Dally, Virtual channel flow control, *IEEE Transactions on Parallel and Distributed Systems*, 3(2), 1992.