Run-Time Deadlock Detection in Networks-on-Chip Using Coupled Transitive Closure Networks

Ra'ed Al-Dujaily†, Terrence Mak†, Fei Xia†, Alex Yakovlev† and Maurizio Palesi‡
†School of Electrical, Electronic & Computer Engineering, Newcastle University, UK
{raaed.aldujaily, terrence.mak, fei.xia, alex.yakovlev}@newcastle.ac.uk
‡Kore University, Italy, maurizio.palesi@unikore.it

***Abstract*—Interconnection networks with adaptive routing are susceptible to deadlock, which could lead to performance degradation or system failure. Detecting deadlocks at run-time is challenging because of their highly distributed characteristics. In this paper, we present a deadlock detection method that utilizes run-time Transitive Closure (TC) computation to discover the existence of deadlock-equivalence sets, which imply loops of requests in networks-on-chip (NoC). This detection scheme guarantees the discovery of all true deadlocks without false alarms unlike state-of-the-art approximation and heuristic approaches. A distributed TC-network architecture which couples with the NoC architecture is also presented to realize the detection mechanism efficiently. Our results based on a cycle-accurate simulator demonstrate the effectiveness of the TC-network method. It drastically outperforms timing-based deadlock detection mechanisms by eliminating false detections and thus reducing energy dissipation in various traffic scenarios. For example, timing based methods may produce two orders of magnitude more deadlock alarms than the TC-network method. Moreover, the implementations presented in this paper demonstrate that the hardware overhead of TC-networks is insignificant.**

I. Introduction

Deadlocks may appear in interconnection networks such as Networks-on-Chip (NoCs) [1] and may lead to performance degradation or even system failure. It is crucial to remove deadlocks when implementing a routing algorithm. In the literature, there are two strategies to deal with deadlocks: deadlock avoidance and deadlock recovery [2].

In deadlock avoidance, resources are granted to packets in a way that the overall network is deadlock free. This can be based on a turn model which prohibits the routing algorithm from making certain turns in the network [3], [4] or based on the strict ordering of virtual channels [5]. In general, avoidance techniques require restricted routing functions or additional resources to prevent deadlocks [2].

Deadlock recovery, which has yet to be implemented in real NoCs, implies that resources are granted to packets without any routing restrictions. Hence, deadlocks may occur and efficient detection and recovery mechanisms are required to intervene. However detecting deadlock from a network is challenging, simply because of the distributed nature of deadlocks. Heuristic approaches, such as time-out mechanisms, are often employed to monitor the activities at each channel for deadlock speculations. These techniques may produce substantial false detections, especially with the network close to saturation where blocked packets could be flagged as deadlock. This makes it difficult to determine the best threshold value in these mechanisms which could depend on packets length, traffic load, traffic type and network size [6], [7], [8], [9], [10].

978-3-9810801-7-9/DATE11/©2011 EDAA

Several techniques have been proposed for reducing the number of false detections in general computer networks. In [8] a packet is suggested as deadlocked if all requested channels by the packet are inactive for a given time out. The work in [10] proposes a technique which is less susceptible to false deadlocks at the expense of extra hardware. In [9] the author proposed a technique that employs special control packets to cross along inactive channels for more accurate deadlock detections. Although these reduced false deadlock alarms compared to the crude time-out mechanism, they are still based on the time-out idea and finding the best threshold values for different network settings is not trivial.

There are two deadlock recovery schemes, regressive and progressive. A regressive recovery is based on an abort and retry mechanism [11] which kills the suspected packet and reinjects it after a time out. A progressive recovery however utilizes additional hardware to bypass the suspected packets to their destination sequentially [6] or concurrently [5].

In this paper, we present a deadlock detection method, which guarantees true deadlock detection for NoCs. A run-time transitive closure (TC) computation scheme is employed to discover the existence of deadlock-equivalence sets, which imply loops of requests. Also, the proposed detection scheme can be realized using a distributed architecture, which closely couples with the NoC infrastructure, to speed up the necessary computation and to avoid introducing traffic overhead in the communication network. The contributions of this paper are:

- Introducing a deadlock-equivalence set criterion for detecting loop of packet requests.
- Presenting a new deadlock detection scheme to discover the existence of deadlock-equivalence sets based on TC computation. In addition a distributed architecture, TC-network, is proposed to implement the detection.
- Evaluating the proposed deadlock detection scheme through experimental studies and comparisons with the state-of-the-art timeout detection scheme using various traffic scenarios and evaluating the hardware area and power overhead of the TC-network.

II. Methodology

A. Assumptions

In line with existing work on deadlock detection/recovery [6], [7], [8], [9], [10], we make the following assumptions:

- The network is a collection of routers connected by channels. Each router is connected to a single core that can inject/consume packets in the network via the router.
- Fully adaptive routing algorithm with minimal paths is used.
- A finite time is required to consume a packet at the destination.
- A buffer cannot contain flits belonging to different packets, $i.e.$ atomic channel allocation [7].

B. Equivalence Set Criterion for Deadlock

Network resources and dependencies at any particular time can be expressed as a Channel Wait-for Graph (CWG) [2], [5]. It is readily translated to an $n \times n$ adjacency Boolean matrix $[G_{ij}]_{n\times n}$ as follows: 1) $G_{ij} = 1$ when there is a head flit in channel "i" requesting channel "j" ($e.g.$ in Fig. 1 a head flit occupy ch_1 and requesting ch_2) or if a non head flit in channel "i" and its head flit in channel "j" ($e.g.$ in Fig. 1 a data or tail flit occupy ch_2 and its head flit in ch_3); 2) $G_{ij} = 0$ otherwise (including when $i = j$). Let the vertices (channels) from network N be $V = \{v, v_2, \ldots, v_n\}$, and consider a subset $M = \{v, v_2, \ldots, v_m\}$ of vertices for some $m < n$. M is a Deadlock Equivalent Set (DES) if and only if in all its vertices there are packets waiting for one another in a cyclic manner to progress to their respective destinations. The evaluation of equivalence relationships can be based on the Transitive-Closure (TC) matrix $[T_{ij}]_{n\times n}$ of the original matrix $[G_{ij}]_{n\times n}$.

The transitive-closure relationship can then be used to determine whether there is a set of channels in the network forming a DES. Suppose we have channel "a" and channel "b", $a, b \in N$. If these two channels form a deadlock-equivalence set $\Omega(a, b)$, their corresponding TC will be $T_{ab} = T_{ba} = 1$ and $T_{aa} = T_{bb} = 1$. This can be extended to m channels, such that all pairs of elements in M meet the DES condition:

$$\Omega(a,b) = \begin{cases} 1, & \text{if } T_{ab} = 1 \wedge T_{ba} = 1 \wedge T_{aa} = 1 \wedge T_{bb} = 1 \\ 0, & \text{otherwise} \end{cases}$$

$$\forall a,\, b \in M \qquad (1)$$

In interconnection networks, a channel can appear in a DES only once and cannot appear in multiple DES's at the same time. Members of the set of simultaneous DES's, $S = \{S_i\}$, are therefore pairwise disjoint, that is S_i, $S_j \in S$ and $i \neq j$ implying $S_i \cap S_j = \phi$.

C. Equivalence Set Computational Complexity

The DES provides a simple criterion for deadlock detection. The technique is to derive the transitive-closure of the CWG and identify the channels that satisfy the criterion. Figure 2 exemplifies this idea. The derived TC graph (Fig. 2-b) clearly shows four vertices (channels) with self-reflexive paths and all pairs of these satisfy the condition of the DES (Eq.1). The computational procedure is outlined in Algorithm 1. It has three nested loops containing a $\Theta(1)$ core. In lines 3-7, it converts the directed graph (CWG) to a Boolean adjacency matrix to reflect the existing paths in the graph. In lines 8-12, the TC is computed. The code state $T^{(k)}$ should show a path from vertex i to vertex j if 1) $T^{(k-1)}$ already shows a path from i to j, passing through one of the vertices in $1..k-1$, or 2) $T^{(k-1)}$ shows a path from i to k and a path from k to j; hence there will be a path from i to j through k.

The computation of the transitive-closure is expensive. The algorithm requires a computational complexity of $O(n^3)$. In the next section, we present a distributed architecture to realize the TC computation and also exploit the reflexive property to further simplify DES detections in NoCs.

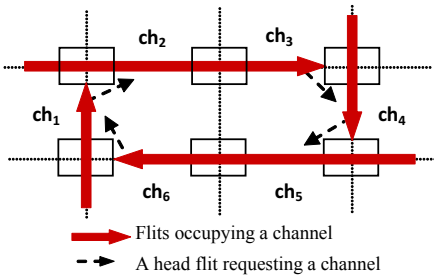


Figure 1: Four packets in a 2D network are waiting for one another and forming a deadlock configuration.

Algorithm 1 A DES discovery using TC computation

1: Definitions: CWG is a directed graph generated from the NoC at certain time; V_{CWG} is a set contains all the vertices in CWG; E_{CWG} is a set contains all the edges in CWG.
2: $n \leftarrow | V_{CWG} |$;
3: for $i \leftarrow 1$ *to* n
4: for $j \leftarrow 1$ *to* n
5: if $(i \neq j \wedge E(i,j) \in E_{CWG})$ then ${T_{ij}}^{(0)} \leftarrow 1$;
6 else ${T_{ij}}^{(0)} \leftarrow 0$;
7: end for j, i
8: for $k \leftarrow 1$ *to* n
9: for $i \leftarrow 1$ *to* n
10: for $j \leftarrow 1$ *to* n
11: ${T_{ij}}^{(k)} = {T_{ij}}^{(k-1)} \vee \left({T_{ik}}^{(k-1)} \wedge {T_{kj}}^{(k-1)}\right)$;
12: end for j, i, k
13: Compute DES $\forall a, b \in n$ that satisfy:
$$T^n_{ab} = 1 \wedge T^n_{ba} = 1 \wedge T^n_{aa} = 1 \wedge T^n_{bb} = 1$$
14: return DES

III. Transitive closure Network Architecture

A. Transitive Closure Computation with TC-Networks

Dynamic programming (DP) can yield the solution for the transitive closure [12]. DP provides an opportunity for solving the computation using a parallel architecture with improved computation performance. Mapping TC computation to a parallel computational platform can be achieved with the introduction of a TC-network. The network has a parallel architecture, and can be used to compute the TC solution through the simultaneous propagation of successive inferences.

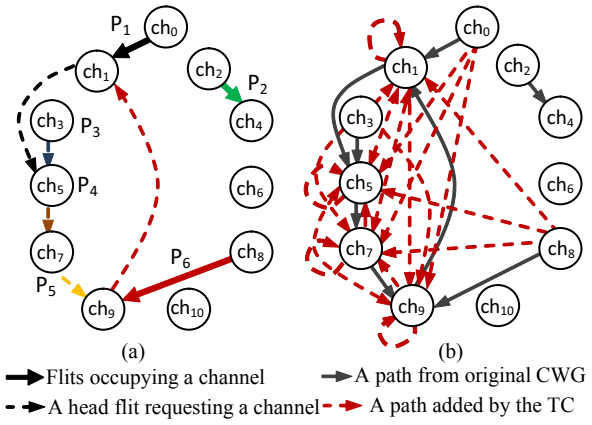


Figure 2: a) The CWG of a particular network at a certain time, b) CWG^* represents the TC graph, the set of channels $\{ch_1, ch_5, ch_7, ch_9\}$ satisfy the DES definition (Eq.1).

Lam and Tong [13] introduced DP-networks to solve a set of graph optimization problems with an asynchronous and continuous-time computational framework. This new class of inference networks is inherently stable in all cases and has been shown to be robust with an arbitrarily fast convergence rate [13]. A parallel computational network for solving the shortest path problem is also proposed in [14].

A TC-network is constructed by the interconnection of autonomous computational units. Figure 3 shows the structure of a unit and the connections in a general inference network. Each unit represents a binary relation (i, j) between two objects and there are N sites to perform the inference action as defined in the site function. The value of the corresponding relation between i and j is then determined by resolving the conflict among all of the site outputs. Basically, if $S_k(i, j)$ represents the site output at the k-th site and $g(i, j)$ stands for the unit output of unit (i, j), then the TC computation can be stated in terms of network structure

$$S_k(i, j) = g(i, k) \wedge g(k, j) \tag{2}$$

$$g(i, j) = \vee_{\forall k} S_k(i, j) \tag{3}$$

where $\wedge$ is the inference for the site function and the conflict-resolution operator for the unit function. The operator $\vee$ denotes the unit which resolves the binary relation (i, j).

The computational units will be interconnected in the same way as the TC computation structure. Each unit represents a node and an interconnection represents an edge. Such a TC network converges to the solution and can be readily implemented using a distributed network. Also, this network architecture, with its simplicity and parallelism, is ideally suitable for on-chip DES detection.

B. Coupling TC-networks to NoCs

An on-chip communication network itself defines the graph vertices of its TC-network. This provides an opportunity for TC computation embedding a TC-unit at each node. Unlike general computer networks, where internode information can

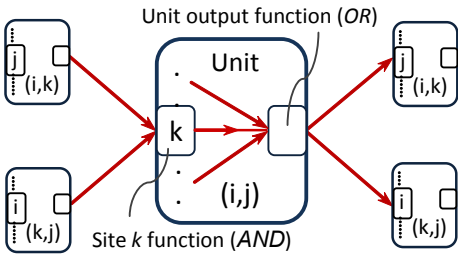


Figure 3: Unit interconnection in a general TC-network where $1 \leq i, j, k \leq n; k \neq i, j$.

only be exchanged through packets, on-chip networks can take advantage of additional dedicated wires to transmit data between routers. The TC-network shown in Fig. 4 consists of distributed computational units and links between them. The topology of the network resembles the defined graph topology, which is the communication structure of a NoC. At each node, there is a computational unit, which implements Eqs. 2 and 3. The output of the unit will be propagated to neighbour units via interconnects. The TC-network tightly couples to the NoC and each computational unit locally exchanges control and system parameters with the router. The run-time information, such as local channels' occupation and request status, will be input to the computational unit simultaneously. The simplicity of the computational unit provides for a run-time response and does not consume any data-flow network bandwidth.

Back to Fig. 2, the CWG vertices do not include self-reflexive paths. Any self-reflexive path in the TC graph corresponds to a DES. This property can be used to further simplify DES detection. Deadlocks have been reported to be infrequent events [7], [10], [15] in networks. As a result, it is possible to employ a light-weight network based on mutual exclusion units to implement TC-networks to discover the existence of the self-reflexive path for each channel. The mutual exclusion circuit ensures that only one channel can use the TC-network at any time and channels are checked sequentially using a simple token-ring protocol. One possible token-ring path for mesh and torus networks is the one presented in [6], a Hamiltonian cycle. The token-ring could be implemented as an asynchronous circuit [6] or could be clocked using router clocks.

Each TC-unit seizing the token will initiate checking of the corresponding router channels. The rest of the units will implement a transitive property that passes the test signal to neighbour units if and only if there is a chain of channel dependencies between its input and output. This makes the TC-network self-pruning and will keep switching power to minimum. The function performed in the TC-unit that seizes the token starts by asserting a logic high test signal on the output TC-link corresponding to the channel to be tested. The corresponding input, TC-link, is then checked (see Algorithm 2). In case of a match between an input and output, a self-reflexive path exists and the channel is part of a DES. The TC-unit then sets a detection flag that may be used by the node router to trigger a recovery. Otherwise the detection flag

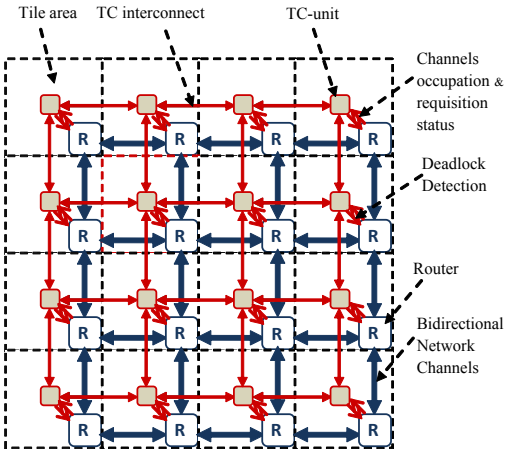


Figure 4: A TC-network coupled to a mesh network.

Algorithm 2 Pseudo code of the TC-unit computation

1: Inputs:
$tc\_rx[m]$: TC input signals from neighbor routers,
$ch\_oc[m]$: channels occupation input from the local router,
$ch\_req[m][n]$: channels request input from the local router,
$token\_in$: token input signal
2: Outputs:
$tc\_tx[n]$: TC output signals to neighbor routers,
$dd[n]$: deadlock detection flags, output to the local router
3: Definitions:
"Par": denotes parallel operations
n : number of router output channels
m: number of router input channels
tp: a temporary buffer;
k: channel number to check if has a self-reflexive loop
4: Par: (for $i \leftarrow 1$ *to* n):
5: if $(token\_in = 1 \wedge k = i)$ then $tc\_tx[k] = 1$;
6: $tp = 0$;
7: Par: (for $j \leftarrow 1$ *to* m)
8: $tp = tp \vee tc\_rx[j] \wedge ch\_oc[j] \wedge (ch\_req[j] = i)$;
9 End Par
10: if $(i = k \wedge token\_in = 1 \wedge tp = 1$) then $dd[i] = 1$;
11: else $dd[i] = 0$; $tc\_tx[i] = tp$;
12:End Par

is reset and the next channel will be tested. Once the TC-unit finishes checking all the channels of the corresponding router, it passes the token to the next neighbour unit. The delay time of the TC-network to converge and provide useful information will mainly depend on the NoC size. However, the TC-network can produce a legitimate output even if does not converge in a single clock cycle since a deadlock is a steady and persistent event [2].

IV. Result and Discussions

A. Evaluation Methodology

We measured and compared the percentage of detected deadlocks using the TC-network method and the heuristic time-out mechanism [6] for different network traffic and different Packet Injection Rates (PIR). The percentage of detected deadlocks is calculated as the ratio between the number of packets detected as deadlock over the total number of packets (received and detected). Moreover we calculated the consumed energy caused by dropping detected packets. In general the energy dissipated by a network is divided into the following groups: 1) Routing energy which depends on the routing type, 2) Selection energy which refers to the type of selection if the routing algorithm returns more than one option, 3) Forwarding energy which is used in sending a flit, 4) Receiving energy which is used in receiving a flit and 5) Standby energy. The packet dropping energy is defined as follows:

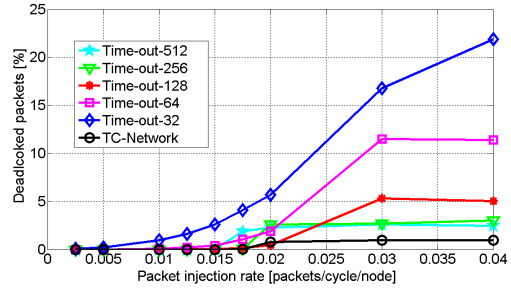
$$E_{wasted} = hop_{count} \times (E_{forward} + E_{receiving}) + Flit_{age} \times [E_{standby} + Flit_{head} \times (E_{routing} + E_{selection})] \quad (4)$$

where E denotes energy, hop_{count} is the number of hops the flit passed before being aborted, $Flit_{age}$ is the number of clock cycles the flit lived in the network (either moving or waiting resources to be freed) and $Flit_{head}$ is a Boolean flag that add the last term to the equation if the flit type is head, $i.e.$ if the blocked flit is head it will continue consume energy by trying to reserve an output channel in each clock cycle.

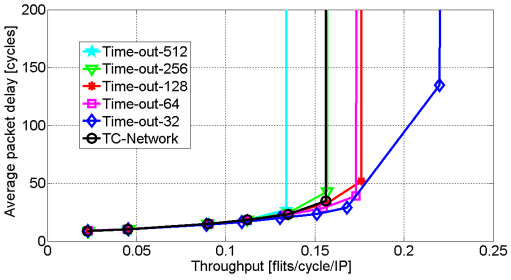
The performance evaluation was carried out using a modified version of Noxim [16]. In particular, we modified Noxim by introducing the TC-network and the crude time-out deadlock detection techniques. The simulated NoC is a mesh with four port architecture, a fully adaptive routing with random selection function, no virtual channels, and a crossbar switch. Each input channel consists of four flit buffers and one clock cycle is assumed for routing and transmission time across the crossbar and a channel. The result are captured after a warm up period of 10,000 clock cycles. The simulation time is set to 300,000 clock cycles. To ensure the accuracy of results captured with a higher confidence, the simulation at each PIR is repeated many times with different seeds and their mean values taken.

B. Evaluation Result

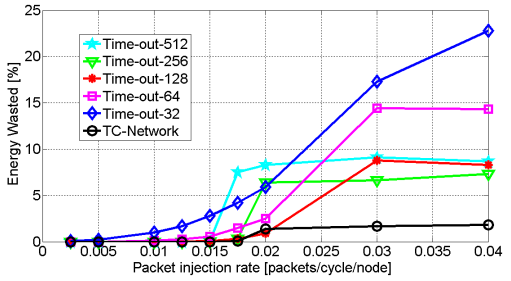
Figure 5 shows the performance results for a 4×4 2D mesh NoC, and a uniform distribution of packet destinations with different PIR. The packet lengths are randomly generated between 2 and 16 flits. The majority of detected deadlocks using the time-out mechanism are false alarms. For instance, 22% of the packets injected in the network are detected as part of deadlocks with the threshold value set to 32 (Fig. 5-a). The TC-network instead detected that less than 1% of packets are in true deadlocks, consistent with literature [15] which stated that deadlock is an infrequent event. Figure 5-b shows the network average delay versus the throughput for full load range. It shows that smaller threshold values used in the time-out mechanism improve these two important network metrics because the time-out mechanism works as a congestion monitoring rather than deadlock detection. Examining Fig. 5 (a and b) one could select the threshold value of 256 as the best value for such a network setting as it produces a minimum detection percentage of 2.7% with good throughput and latency. The selection of the best threshold value for different network settings (packets length, buffer size and traffic type) was the goal of several studies [10], [8], [7], [9].



(a)

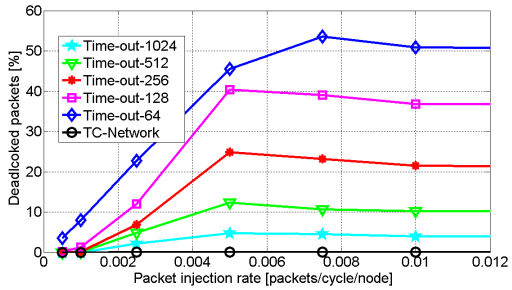


(b)

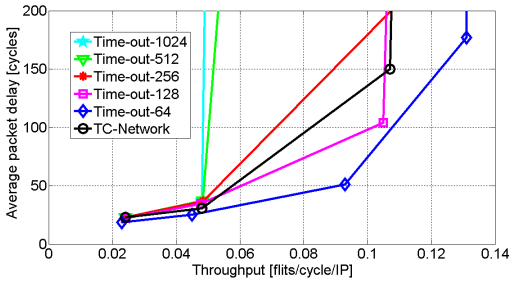


(c)

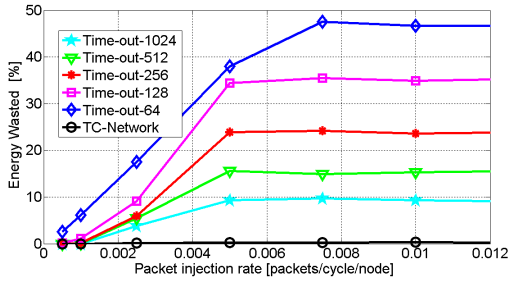
Figure 5: TC-Network and Time-out mechanism performance under a uniform traffic scenario.



(a)



(b)



(c)

Figure 6: TC-Network and Time-out mechanism performance under the bitreversal traffic scenario.

Figure 5-c shows the percentage of wasted energy due to packet dropping to the total consumed energy. The figure is similar to but not linearly proportional to the detection percentage figure (Fig. 5-a). This is because significant energy consumption is caused by the routing function [16] which repeatedly try to route the head flit till the time out value has elapsed in the case of the time out method (see Eq.4). For instance, the time-out-128 detects 5.3% at saturation and it waste 8.8% of the total consumed energy by aborting these packets while the Time-out-512 detecting 2.6% at saturation and wastes 9.1% energy. There are two reasons behind that: the first one because the network with bigger threshold value is delivering less flits at the given simulation time (Fig. 5-b). The second reason is the $Flit_{age}$ in (Eq.4) is directly proportional to the threshold value, in case when a packet is detected as deadlocked.

To investigate different traffic scenarios and network sizes, Fig. 6(a, b and c) shows the performance results with the bitreversal traffic. The network size is 8×8 mesh and the packet lengths are randomly chosen between 32 and 64 flits. The results in general show a similar trend to the previous example. Here we could select the threshold value of 1024 as the best threshold. The TC-network method detects around 0.07% of packets as deadlocked and dropping them consumed energy of less than 3.6% compared to 4% detected using Time-out-1024 and wasting energy of around 10%. For other traffic scenarios the results are summarized in Table I for 8×8 mesh with packets size randomly chosen between 32 and 128. The table also summarizes the TC-network improvements.

C. Area, Power and Delay Estimation

It is crucial in designing NoCs that routers should not consume a large percentage of silicon area compared to the core blocks. We have designed in Verliog two fully adaptive routers based on the Time-out and the TC-network methods. These are then synthesised using Synopsys Design Compiler and mapped onto the UMC 90nm technology library. We found that the TC circuit uses 75% less area than the Time-out circuit and will add only 0.76% area overhead to the total router area

Table I: TC-Network improvement compared to Time-out for different threshold values and traffic scenarios.

		Time-out 64			Time-out 256			Time-out 1024			TC-Network		
Traffic Type	PIR (Sat.)	$DD^{\star}\%$	$EW^{\dagger}\%$	$Thro^{\ddagger}$	DD%	EW%	Thro	DD%	EW%	Thro	DD%	EW%	Thro
Shuffle	0.0035	49.5	39.8	0.145	32.8	29.6	0.156	13.3	14.6	0.142	0.10	0.29	0.140
Transpose	0.015	35.2	27.6	0.269	12.6	11.1	0.269	0.90	1.00	0.266	0	0	0.267
Butterfly	0.005	28.9	33.8	0.280	11.7	16.7	0.290	2.30	4.20	0.290	0	0	0.295
Random[1]	0.001	18.1	16.3	0.039	2.80	2.70	0.047	0.20	0.22	0.048	0.07	0.11	0.047
Random[2]	0.001	20.2	15.5	0.039	4.90	4.60	0.045	1.28	2.30	0.048	0.69	1.2	0.044
TC Improvement		176x	83x	0.97x	75x	41x	1x	21x	14x	1x			

*Detected Deadlocks, †Energy Wasted, ‡Throughput

[1]Random traffic with 4 hot spots located at the corners, [2]Random traffic with 4 hot spots located at the center.

Table II: TC-Network delay estimation for different mesh network sizes

Network Size	Min.Delay* (ns)	Min. Clock Cycles	Max.Delay† (ns)	Max. Clock Cycles
3×3	0.74	< 1	1.67	< 2
4×4	0.74	< 1	2.97	< 3
6×6	0.74	< 1	6.69	< 7
8×8	0.74	< 1	11.90	< 12
$8 \times 10^{\ddagger}$	0.74	< 1	14.87	< 15

*Shortest deadlock cycle will consist of 4 channels for 2D mesh

†Longest deadlock cycle will consist of $N \times N$ channels for 2D mesh

‡Tile number of Intel TeraFLOPS chip [17]

compared to 2.9% for the Time-out implementation. A TC circuit consumes 27% less power than the Time-out circuit. The power overhead added by TC circuit to the total router power is 0.08% compared to 0.11% added by the Time-out circuit, with 10 bit threshold counter.

It is also important to estimate the delay of the TC circuit and TC interconnects. First the TC unit critical path gates delay is calculated using the *sdf* file generated after synthesising the circuit using worst case library. Second the interconnect delay calculation assumes the tiles are arranged in a regular manner on the floorplan with $2mm \times 1.5mm$ tile size, similar to Intel TeraFLOPS chip [17]. The maximum interconnect length between routers will be 2 mm. The load wire capacitance and resistance are estimated using the Predictive Technology Model (PTM) [18] to be $0.146fF$ and 1.099 Ohm per micron respectively. The wire delay between TC units can be readily calculated based on the distributed RC model [19]. Table II shows the minimum and maximum delay required to discover different DES's for different network sizes. The table also shows the equivalent maximum number of clock cycles, assuming a router operating clock frequency of 1GHz.

V. Conclusion and Future Work

This work studies deadlock detection and recovery in NoCs as opposed to deadlock prevention. We proposed a new deadlock detection method based on computing deadlock-equivalence sets. Also, a transitive-closure network architecture is proposed to realize the detection computation in parallel. The method eliminates the need of any kind of time out mechanism, and delivers true deadlock detections independent of the network load and message lengths. The proposal is rigorously evaluated using a cycle-accurate simulator to demonstrate the effectiveness of the TC-network based detection method compared to the time-out mechanism. In the future, we will investigate the methodology of using TC-network on real-time deadlock detection and recovery in a large-scale NoCs.

References

[1] L. Benini and G. De Micheli, "Networks on chips: a new SoC paradigm," *IEEE Computer*, vol. 35, no. 1, pp. 70 –78, 2002.

[2] W. J. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers, 2004.

[3] J. G. Christopher and M. N. Lionel, "The turn model for adaptive routing," *Parallel Processing Symposium, Proceedings 9th International*, vol. 41, no. 5, pp. 874–902, 1994.

[4] G.-M. Chiu, "The odd-even turn model for adaptive routing," *IEEE Transac. on Parallel Distrib. Syst.*, vol. 11, no. 7, pp. 729–738, 2000.

[5] J. Duato, S. Yalamanchili, and L. M. Ni, *Interconnection Networks: An Engineering Approach*. Morgan Kaufmann Publishers, 2004.

[6] K. Anjan and T. Pinkston, "DISHA: a deadlock recovery scheme for fully adaptive routing," Parallel Processing Symposium, Proceedings 9th International, pp. 537 –543, 1995.

[7] J. M. Martinez-Rubio, P. Lopez, and J. Duato, "FC3D: Flow control-based distributed deadlock detection mechanism for true fully adaptive routing in wormhole networks," *IEEE Trans. on Parallel Distributed Syststem*, vol. 14, no. 8, pp. 765–779, 2003.

[8] J. M. Martínez-Rubio, P. López, and J. Duato, "A cost-effective approach to deadlock handling in wormhole networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 12, no. 7, pp. 716–729, 2001.

[9] L. Soojung, "A deadlock detection mechanism for true fully adaptive routing in regular wormhole networks," *Computer Communications*, vol. 30, no. 8, pp. 1826–1840, 2007.

[10] P. Lopez, J. M. Martinez-Rubio, and J. Duato, "A very efficient distributed deadlock detection mechanism for wormhole networks," in *HPCA '98*. IEEE Computer Society, 1998.

[11] J. H. Kim, L. Ziqiang, and A. A. Chien, "Compressionless routing: a framework for adaptive and fault-tolerant routing," *IEEE Trans. on Parallel and Distrib. Syst.*, vol. 8, no. 3, pp. 229–244, 1997.

[12] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. MIT Press and McGraw-Hill, 2001.

[13] K. Lam and C. Tong, "Closed semiring connectionist network for the bellman-ford computation," *IEE Proceedings on Computers and Digital Techniques*, vol. 143, no. 3, pp. 189 –195, 1996.

[14] T. Mak, P. Y. Cheung, W. Luk, and K. P. Lam, "A DP-network for optimal dynamic routing in network-on-chip," in *CODES+ISSS '09*. ACM, pp. 119–128, 2009.

[15] S. Warnakulasuriya and T. Pinkston, "Characterization of deadlocks in interconnection networks," in *IPPS '97*. IEEE Computer Society, 1997, pp. 80–86.

[16] F. Fazzino, M. Palesi, and D. Patti, "Noxim: Network-on-chip simulator," http://noxim.sourceforge.net/.

[17] S. R. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, A. Singh, T. Jacob, S. Jain, V. Erraguntla, C. Roberts, Y. Hoskote, N. Borkar, and S. Borkar, "An 80-tile sub-100-w teraflops processor in 65-nm cmos," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 1, pp. 29–41, 2008.

[18] "PTM: Predictive technology model," http://ptm.asu.edu/.

[19] J. M. Rabaey, A. Chandrakasan, and B. Nikolic, *Digital Integrated Circuits: A Design Perspective*. Upper Saddle River, N.J.; London: Prentice Hall, 2002.