A Framework for Designing Congestion-Aware Deterministic Routing

Abbas Eslami Kiasari, Axel Jantsch, and Zhonghai Lu
Royal Institute of Technology (KTH), Sweden
{kiasari, axel, zhonghai}@kth.se

ABSTRACT

In this paper, we present a system-level Congestion-Aware Routing (CAR) framework for designing minimal deterministic routing algorithms. CAR exploits the peculiarities of the application workload to spread the load evenly across the network. To this end, we first formulate an optimization problem of minimizing the level of congestion in the network and then use the simulated annealing heuristic to solve this problem. The proposed framework assures deadlock-free routing, even in the networks without virtual channels. Experiments with both synthetic and realistic workloads show the effectiveness of the CAR framework. Results show that maximum sustainable throughput of the network is improved by up to 205% for different applications and architectures.

Categories and Subject Descriptors

C.2.1 [Network Architecture and Design]: Network communications

General Terms

Algorithms, Design, Performance

1. INTRODUCTION

Thanks to high performance and low power budget of ASICs (application specific integrated circuits), they have been common components in the design of embedded systems-on-chip. Advances of semiconductor technology facilitate the integration of reconfigurable logic with ASIC modules in embedded systems-on-chip. Reconfigurable architectures are used as new alternatives for implementing a wide range of computationally intensive applications, such as DSP, multimedia and computer vision applications [1]. In the beginning of the current millennium, network-on-chip (NoC) emerged as a standard solution in the on-chip architectures [7][8]. In network-based systems, the performance of the communication infrastructure is critical, as it can represent the overall system performance bottleneck. The performance of networks depends heavily on the routing algorithm effectiveness, since it impacts all network metrics such as latency, throughput, and power dissipation.

Routing algorithms are generally categorized into deterministic and adaptive. A deterministic routing algorithm is oblivious of the dynamic network conditions and always provides the same path between a given source and destination pair. In contrast, in adaptive routing algorithms, besides source and destination addresses, network traffic variation plays an important role for selecting channels to forward packets. However, adaptive routing may cause packets to arrive out-of-order since they may be routed along different paths. The re-order buffers needed at the destination for ordering the packets impose large area and power on system [13]. Deterministic routers not only are more compact and faster than adaptive routers [4], but also guarantee in-order packet delivery. Therefore, it is not surprising that designers would like to use deterministic routing algorithms in the NoCs which suffer from limited silicon resources. However, in deterministic routing a packet cannot use alternative paths to avoid congested channels along its route; this leads to degraded performance of the communication architecture at high levels of network throughput.

A well-designed routing algorithm utilizes the network resources uniformly as much as possible and avoids the congested channels, even in the presence of non-uniform traffic patterns, which are usual in the embedded systems. In this paper, we propose a system-level Congestion-Aware Routing (CAR) framework for designing minimal deterministic routing algorithms for network-based platforms. Especially, CAR is appropriate for reconfigurable embedded systems-on-chip which host several applications with high computational requirements and static workloads. Before the execution of a new application, the routing tables are configured with pre-computed routes, as well as other components in the system. After selecting the route and adding it to the packet, no further time is needed on routing at the intermediate nodes along the path. Due to advantages of table-based routing, it is one of the most widely used routing methods for implementing deterministic routing algorithm, e.g., IBM SP1 and SP2 [4].

To calculate the expected load on various channels in the network, CAR uses off-line analysis based on the global knowledge of application traffic. The results obtained from simulation experiments confirm that the proposed routing framework can find efficient routes for various networks and workloads.

The rest of the paper is organized as follows. We start by reviewing previous studies in Section 2. The CAR framework is proposed in Section 3. Experimental results in Section 4 show that our proposed approach can improve the system performance. Finally, concluding remarks are given in Section 5.

2. RELATED WORK

Turn model for designing partially adaptive routing algorithms for mesh and hypercube networks was proposed in [6]. Prohibiting minimum number of turns breaks all of the cycles and produces a deadlock-free routing algorithm. Turn model was used to develop the Odd-Even adaptive routing algorithm for meshes [3]. This model restricts the locations where some turns can be taken so that deadlock is avoided. In comparison with turn model, the degree of routing adaptivity provided by the Odd-Even routing is more even for different source-destination pairs.

DyAD routing scheme, which combines deterministic and adaptive routing, is proposed in [9] for NoCs, where the router works in deterministic mode when the network is not congested, and switches to adaptive mode when the network becomes congested. In [17] the authors extend routers of a network to measure their load and to send appropriate load information to their direct neighbours. The load information is used to decide in which direction a packet should be routed to avoid hot-spots. Recently, the authors in [14] present APSRA, a methodology to develop adaptive routing algorithms for NoCs that are specialized for an application or a set of concurrent applications. APSRA exploits the application-specific information regarding pairs of cores that communicate and other pairs that never communicate in the NoC platform to maximize communication adaptivity and performance.

Since all of these approaches are based on adaptive routing, they suffer from out-of-order packet delivery. Our proposed routing framework overcomes this problem while it spreads the load more evenly across the network.

Also, an application-aware oblivious routing is proposed in [11] that statically determines deadlock-free routes. The authors presented a mixed integer-linear programming approach and a heuristic approach for producing routes that minimize maximum channel load. However, in case of realistic workload, they did not study the effect of task mapping on their approach.

3. CAR FRAMEWORK

The CAR framework consists of 5 steps as its flowchart is shown in Figure 1. At first, we represent the architecture and application using *topology graph* (TG) and *communication graph* (CG), respectively. Then we construct the *channel dependency graph* (CDG) based on TG and CG. In the third step, an acyclic CDG is extracted by deleting some edges from CDG to guarantee the deadlock freedom. After that, we find all possible shortest paths for each flow to create the routing space. Finally, we formulate an optimization problem over the routing space and solve it. In the following subsections, each step is described in detail.

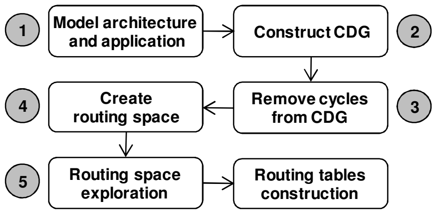
1
Model architecture and application
Construct CDG
2
4
Create routing space
Remove cycles from CDG
3
5
Routing space exploration
Routing tables construction

Figure 1. The flowchart of CAR framework

3.1 Model Architecture and Application

In order to characterize the network performance, a network model is essential. As shown in Figure 2, a directed graph, which is called topology graph (TG), can represent the topology of network architecture. Vertices and edges of TG show nodes and links of the network, respectively. Every node in TG contains a core and a router. Such a core is a local computing or a storage region.

An application can be modelled by a graph called communication graph (CG). CG is a directed graph, where each vertex represents one selected task, and each directed arc represents the communication volume from source task to destination task.

3.2 Construct Channel Dependency Graph

Dally and Seitz simplified designing deadlock-free routing algorithms with a proof that an acyclic channel dependency graph

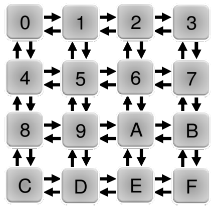

Figure 2. TG of a 4x4 mesh network

(CDG) guarantees deadlock freedom [5]. Each vertex of the CDG is a channel in TG. For instance, vertex 01 in Figure 3 corresponds to the channel from node 0 to node 1 in Figure 2. There is a directed edge from one vertex in CDG to another if a packet is permitted to use the second channel in TG immediately after the first one. To find the edges of a CDG, we use the *Dijkstra's algorithm* to find all shortest paths between source and destination of any flows in corresponding TG. CDG of a 4x4 mesh network (Figure 2) under minimal fully adaptive routing is shown in Figure 3.a, when any two nodes have the need to communicate such as in the uniform traffic pattern.

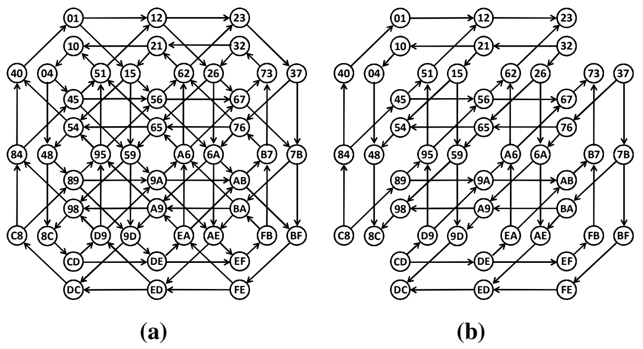

Figure 3. CDG of 4x4 mesh network for minimal fully adaptive routing under (a) uniform and (b) transpose traffic patterns

3.3 Remove Cycles from CDG

Traditional routing algorithms, such as *dimension-order routing* (DOR) and turn model, extract an acyclic CDG by systematically removing some edges from CDG regardless of the traffic pattern. This may result in poor performance of routing algorithm due to prohibition of unnecessary turns. For instance, as shown in Figure 3.b, there is no cycle in CDG of 4x4 mesh network under transpose traffic pattern, which the node in row *i* and column *j* sends packets to the node in row *j* and column *i*. However, traditional routing algorithms conservatively remove some edges from CDG.

We modify the *depth-first-search* (*dfs*) algorithm to find cycles in a given CDG. Since we want to remove minimum number of edges, we delete an edge from CDG which is shared among more cycles. Note that, this edge is removed if the reachability of all flows is guaranteed. For example, in a CDG of 4x4 mesh network, shown in Figure 3.a, there are 6,982,870 cycles and the edge from vertex 40 to vertex 01 is shared among 5,041,173 cycles. Thus by removing this edge from CDG, the number of cycles is considerably reduced to 1,941,697. These steps are repeated again while there is a cycle in CDG. Table 1 shows the numbers of cycles found by CAR in CDG of different mesh networks. As it can be vividly seen, number of cycles is exponentially grown with the size of TG and it takes a long time to find all cycles in the CDG. Hence, we find cycles in CDG till certain number of cycles, and then remove an edge from CDG which is shared among more cycles.

Table1. Number of cycles in CDG of mesh networks

TG	Number of cycles in corresponding CDG
Mesh (2x2)	2
Mesh (2x3)	8
Mesh (3x3)	292
Mesh (3x4)	14,232
Mesh (4x4)	6,982,870
Mesh (4x5)	3,656,892,444

3.4 Create Routing Space (RS)

In this step, we apply Dijkstra's algorithm to the acyclic CDG to find all shortest paths between source and destination of flows in corresponding TG and create a set of f flows $RS = \{F_1, F_2, \dots, F_f\}$ where f is the number of all flows in the system. $F_i = (\lambda_i, n_i, P_i)$, where λ_i and n_i are the packet generation rate and the number of available shortest paths for flow i, respectively. Also, P_i is itself a set and includes all n_i routes for flow i.

Usually more than one shortest path is available between two nodes $(n_i > 1)$ in the routing space RS, so it is reasonable to choose a path such that the load is evenly spread across the network. In the next subsection, we formulate an optimization problem over RS to find a suitable route for each flow and then use the simulated annealing heuristic to solve this problem.

3.5 Routing Space Exploration

In this subsection, we define an optimization problem to explore the routing space of RS. It is essential to define *decision variables* and *objective functions* in formulating the optimization problem. As previously mentioned, our goal is to select a path for flow i $(1 \le i \le f)$ among n_i available paths. Therefore, we define $X = \{x_1, x_2, \dots, x_f\}$ as decision variables in the space of RS where x_i refers to a path number for flow i $(1 \le x_i \le n_i)$. A routing algorithm prevents congestion in the network by balancing the load over network channels, so the standard deviation of channels throughput can be used as a criterion for load balance in the network. The more balanced the channel load, the smaller the standard deviation of channels throughput. Hence, we consider standard deviation of channels throughput as an objective function.

Assuming the network is not overloaded, the throughput of channel c in TG, T_c, can be calculated using the general equation

$$T_c = \sum_{i=1}^{f} \lambda_i \times R(i, c) \qquad (1)$$

where the $R(i, c)$ is a Boolean function and equals 1 if the flow i passes through channel c and equals 0 otherwise. Note that we assume a deterministic routing algorithm, thus the function of $R(i, c)$ can be predetermined, regardless of topology and routing algorithm. After computing T_c for all channels in the TG, the standard deviation of channels throughput can be calculated using the following equation

$$\sigma = \sqrt{\frac{1}{L} \sum_{i=1}^{L} (T_i - \bar{T})^2} \qquad (2)$$

where L is the number of channels in the network and $\bar{T}$ is the average throughput of all channels. If the load is completely balanced over the channels, then $\sigma = 0$. CAR framework uses the simulated annealing heuristic to minimize the objective function (σ) as described briefly in the following.

The name and inspiration of simulated annealing algorithm come from physical annealing technique in metallurgy. To simulate the physical annealing process, simulated annealing algorithm will randomly choose a neighbour solution to replace the current solution. As we mentioned before, $X = \{x_1, x_2, \dots, x_f\}$ is the set of decision variables where x_i is the path number for flow i $(1 \le x_i \le n_i)$. To choose a neighbour of X, we generate a random number r where $1 \le r \le f$ to choose a flow, and then generate another random number x_r^{new} where $1 \le x_r^{new} \le n_r$ and $x_r^{new} \ne x_r$ to choose another path for flow r. The new solution is accepted based on an equation that depends on the difference of the objective function values between the two states. We follow the Metropolis algorithm [2] as the acceptance criterion which accepts all downhill moves (from higher value to lower value) and probabilistically accepts uphill moves (from lower value to higher value). The acceptance of uphill moves allows saving the method from becoming stuck at a local minimum. Detailed information about simulated annealing approach can be found in [12].

4. Experimental Results

To evaluate the capability of CAR framework, we developed a discrete-event simulator that mimics the behaviour of routing algorithm in the networks at the flit level. Due to the popularity of the mesh network in NoC domain, our analysis focuses on this topology but CAR framework can be equally applied for other topologies without any change. We compare the performance of CAR with DOR which becomes XY routing algorithm in 2D mesh networks.

To achieve a high accuracy in the simulation results, we use the batch means method [15] for simulation output analysis. There are 10 batches and each batch includes 1000 up to 1,000,000 packets depending on the workload type, packet injection rate, and network size. Statistics gathering was inhibited for the first batch to avoid distortions due to the startup transient. The standard deviation of latency measurements is less than 1.8% of the mean value. As a result, the confidence level and confidence interval of simulation results are 0.99 and 0.02, respectively.

For the sake of comprehensive study, numerous validation experiments have been performed for several combinations of workload types and network size. In what follows, the capability of CAR will be assessed for both synthetic and realistic traffic patterns. Since their applications differ starkly in purpose, these classes of NoC have substantially different traffic patterns.

4.1 Synthetic Traffic

Synthetic traffic patterns used in this research include *uniform, transpose, shuffle, bit-complement,* and *bit-reversal* [4]. After developing models describing spatial traffic distributions, we should use an appropriate model to model the temporal traffic distribution. In the case of synthetic traffics, we use the Poisson process for modelling the temporal variation of traffic. It means that the time between two successive packet generations in a core is distributed exponentially. The Poisson model widely used in many performance analysis studies, and there are a large number of papers in many application domains that are based on this stochastic assumption.

The average packet latencies in the 4x4 and 8x8 mesh networks are plotted against offered load in the network in Figure 4 and Figure 5, respectively. We observe that under uniform and bit-complement traffic patterns CAR converges to DOR, because in such traffic patterns the standard deviation of channels throughput is minimum for DOR. This result is consistent with other results reported in [3][6][9][14]. The main reason is that the DOR distributes packets evenly in the long term [6]. Previous works,

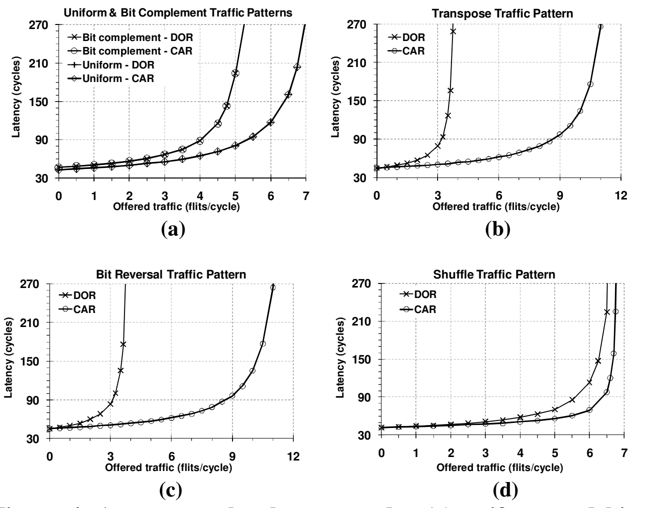

Figure 4. Average packet latency under (a) uniform and bit-complement, (b) transpose, (c) bit-reversal, and (d) shuffle traffic patterns in 4x4 mesh network

Odd-Even [3], turn model [6], DyAD [9], and APSRA [14] indicate that in the case of uniform traffic, their proposed approaches underperform DOR. However, as can be seen in Figure 4.a and 5.a, our proposed framework has the same performance as DOR for different traffic loads.

Figure 4.b and 4.c compare the latency of DOR and CAR in 4x4 mesh network under transpose and bit-reversal workloads, respectively. It can be vividly seen that CAR considerably outperforms DOR. In these cases, CAR can find routes for flows such that the standard deviation of channels throughput equals zero. This means that the load is completely balanced across the network channels. Also, in the case of 8x8 mesh network, CAR has better performance than DOR as shown in Figure 5.b and 5.c.

Figure 4.d and 5.d reveal that under shuffle traffic pattern CAR slightly outperforms DOR.

Table 2 shows the maximum sustainable throughput of the network for each workload and for each routing algorithm in 4x4 and 8x8 mesh networks. It also shows the percentage improvement of CAR over DOR and reveals that on average CAR outperforms DOR. The maximum load that the network is capable of handling using CAR is improved by up to 205%.

Also, the performance of CAR framework is compared against DyAD routing scheme [9] which combines deterministic and adaptive routing algorithms. We simulate the uniform and transpose workloads on the similar architecture (6x6 mesh network) and compare their improvement over DOR. Table 3 shows the percentage improvement of DyAD and CAR over DOR. In case of uniform workload, DyAD underperforms DOR while CAR has the same performance as DOR. In case of transpose

Table 2. Improvement in maximum sustainable throughput of CAR as compared to DOR for different synthetic workloads

Workload	4x4 mesh network			8x8 mesh network		
	DOR	CAR	Impr.	DOR	CAR	Impr.
Uniform	7.4	7.4	0	15.9	15.9	0
Transpose	3.8	11.6	205%	7.7	10.3	34%
Bit-comp.	5.6	5.6	0	8.8	8.8	0
Bit-rev.	3.8	11.6	205%	7.6	9.0	18%
Shuffle	6.6	6.9	5%	12.2	13.1	7%

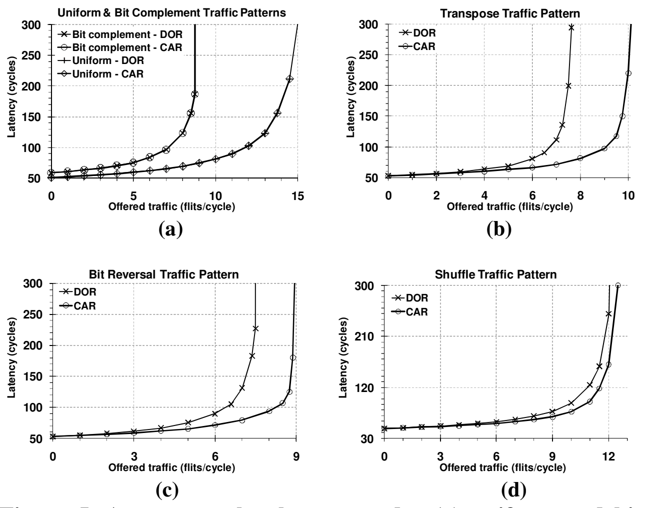

Figure 5. Average packet latency under (a) uniform and bit-complement, (b) transpose, (c) bit-reversal, and (d) shuffle traffic patterns in 8x8 mesh network

traffic pattern, DyAD and CAR give about 62% and 56% improvement over DOR, respectively. This means that our deterministic routing policy can compete with adaptive routing policies (DyAD switches to adaptive mode under high traffic load) and meanwhile guarantees in-order packet delivery.

Table 3. Improvement in maximum sustainable throughput of DyAD and CAR over DOR

Workload	Improvement over DOR	
	DyAD	CAR
Uniform	-26%	0
Transpose	62%	56%

4.2 Realistic Traffic

In case of realistic traffic, we consider two virtual channels for links to show the consistency of proposed framework with multiple virtual channel routing. As realistic communication scenarios, we consider a generic multimedia system (MMS) and the video object plane decoder (VOPD) application. MMS includes an H.263 video encoder, an H.263 video decoder, an mp3 audio encoder, and an mp3 audio decoder [10]. The communication volume requirements of this application are summarized in [10]. VOPD is an application used for MPEG-4 video decoding and its communication graph is available in [18]. Several studies reported the existence of bursty packet injection in the on-chip interconnection networks for multimedia traffic [16][19]. Poisson process is not the appropriate model in case of bursty traffic; consequently, we used two-state Markov modulated process as stochastic traffic generators to model the bursty nature of the application traffic [4]. The two states represent an "on" and "off" mode for injection process with average communication bandwidth matching the applications' average communication bandwidth.

Since in such systems, there are various types of cores with different bandwidth requirements, placement of tasks on a chip has strong effect on the system performance. To find a suitable mapping of these applications, we formulate another optimization problem to prune the large design space in a short time and then again use the simulated annealing heuristic to find a suitable mapping vector.

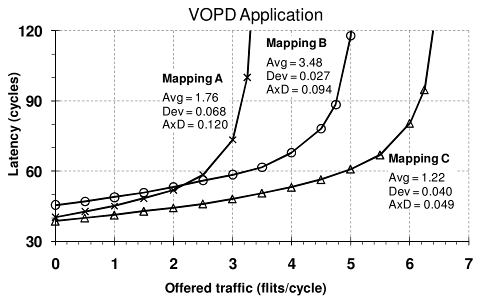

Figure 6. Average packet latency of VOPD application for three different mapping configurations vs. offered load

An efficient mapping tries to balance load over channels (minimizes the standard deviation of channels throughput) and also keeps the average hop count as small as possible. However, minimizing the standard deviation and minimizing the average hop are not always in the same direction. Therefore, to improve load balance, we have to increase the average path length [4]. To show how these parameters, average hop and standard deviation of channels throughput, affect the communication latency, we consider three different task mappings of VOPD application to the tiles of a 4x4 mesh on-chip network. Then, the system is simulated for these mapping configurations and the corresponding average packet latency values are plotted against offered load in Figure 6. Although the average hop in mapping A (1.76) is about half of the average hop in mapping B (3.48), mapping A underperforms mapping B. This is due to more balanced load in mapping B (the standard deviation of channels throughput in mapping B (0.027) is less than the standard deviation in mapping A (0.068)). On the other hand, the load in mapping B is more balanced in comparison to mapping C. However, due to smaller average hop in mapping C, it outperforms mapping B for all levels of network throughput. Thus, we define a new criterion named AxD (Average hop x standard Deviation) and use it as the objective function in the optimization problem of congestion-aware mapping (similar to congestion-aware routing).

Initially, we map task *i* to node *i* and then try to minimize the AxD through the simulated annealing approach. Figure 7.a shows that in the case of MMS application and DOR, for the initial mapping M1, AxD equals 0.57 and after a certain number of tries, the mapping vector converges to the mapping M4 with AxD = 0.04. Furthermore, AxD values for mappings M2 and M3, which are two local minimum points in simulated annealing process, are shown in the figure.

After the mapping phase, we apply the CAR framework to these four mapping vectors. Figure 7.a reveals that in case of mapping M1, CAR can significantly reduce the AxD from 0.57 to 0.24. This great difference is due to the unbalanced load of DOR. However, for more efficient mapping vectors (M2, M3, and M4), we achieve less improvement. Specially, in the case of best mapping (M4), AxD is reduced insignificantly from 0.0397 to 0.0395. It is reasonable that DOR is congestion-aware for the best mapping, because during the mapping problem solving process, we fix the routing policy to DOR and strive to minimize AxD for this routing policy. Figure 7.b shows that the simulation results confirm this conclusion. In the case of mapping M1, CAR significantly outperforms DOR, but in the case of M4, the latency is the same for both DOR and CAR. Likewise, as shown in Figure 7.c and 7.d, for the VOPD application, the analysis result is the same as MMS application.

Figure 7.b and 7.d reveals that in case of application-specific traffic patterns, the improvement in the performance of routing

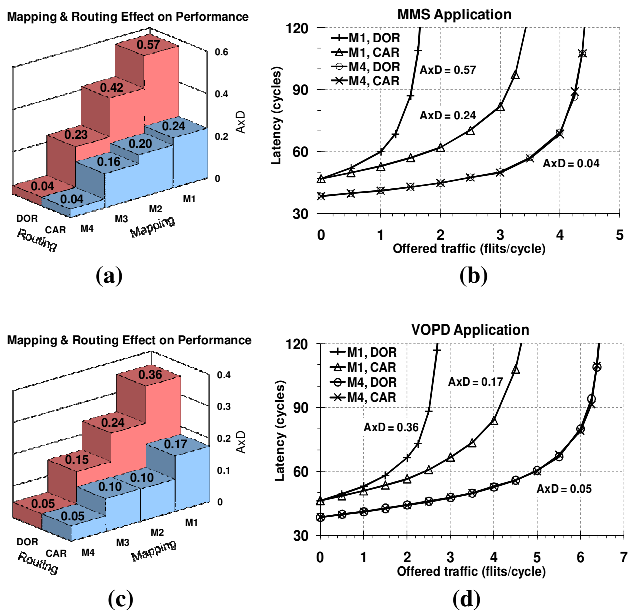

Figure 7. (a) The effect of mapping and routing on the performance of MMS application, (b) average packet latency for different mapping and routing schemes in the case of MMS workload, (c) the effect of mapping and routing on the performance of VOPD application, (d) average packet latency for different mapping and routing schemes in the case of VOPD workload

schemes highly depends on how the application tasks are mapped to the topology. This fact was not considered in the related works such as [11]. Also, Table 4 reports the maximum acceptable traffic for different mapping vectors under MMS and VOPD workloads. The better mapping vector results in smaller improvement in the saturation point.

Table 4. Improvement in Maximum Sustainable Throughput of CAR as Compared to DOR for Realistic Applications

Mapping	MMS application			VOPD application		
	DOR	CAR	Impr.	DOR	CAR	Impr.
M1	1.8	3.6	100%	2.9	4.9	69%
M2	2.1	3.9	86%	3.5	5.2	49%
M3	3.2	4.3	34%	5.1	6.4	25%
M4	4.7	4.7	0	6.7	6.7	0

Nowadays, in embedded systems-on-chip there are several different types of cores including DSPs, embedded DRAMs, ASICs, and generic processors which their places are fixed on the chip. On the other hand, such a system hosts several applications with completely different workload. Furthermore, modern embedded devices allow users to install applications at run-time, so a complete analysis of such systems is not feasible during design phase. As a result, it is not feasible to map all applications such that the load is balanced for all of them with specific routing algorithm and we should balance the load in routing phase.

In this section we used the CAR framework to find low congestion routes in the mesh network. Due to simplicity, regularity, and low cost merits of 2D mesh topology, it is the most popular one in the field of NoC. However, for large and 3D NoCs, which will be popular in the future, the communication in mesh architecture takes a long time. In the next subsection we use CAR to find deadlock-free paths in an arbitrary topology.

4.3 Find Routes in an Arbitrary Topology

To show the capability of CAR framework to find deadlock-free routes in an arbitrary topology, we consider the topology shown in Figure 8.a. CAR reports that under uniform traffic pattern there are 2 cycles in the corresponding CDG and by prohibiting turns 52 to 21 and 87 to 73 (shown in Figure 8.b) the deadlock-freedom is guaranteed.

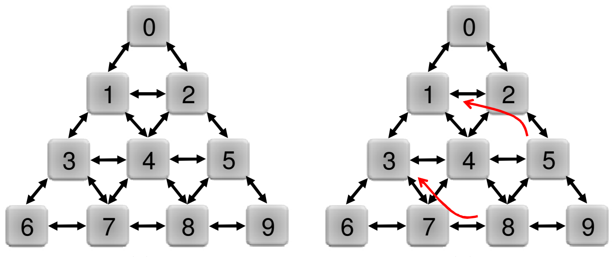

Figure 8. (a) A custom topology and (b) prohibited turns

Table 5 shows the routing table for node 0 of the topology in Figure 8.a. Each route in the table specifies a path from node 0 to a given destination as channels name. SE, SW, and EJ specify South East, South West, and ejection channels, respectively. To route a packet, the routing table is indexed by destination address to look up the pre-computed route by CAR. This route is then added to the packet. Since there are 7 channels in this network (E, S, NE, NW, SE, SW, and EJ), they can be encoded as 3-bit binary numbers. Also, there are techniques to reduce the size of routing tables [4][14].

Table 5. Routing table for node 0 of topology in Figure 8.a.

dst.	route	dst.	route
0	No packet	5	SE, SE, EJ
1	SW, EJ	6	SW, SW, SW, EJ
2	SE, EJ	7	SE, SW, SW, EJ
3	SW, SW, EJ	8	SW, SE, SE, EJ
4	SW, SE, EJ	9	SE, SE, SE, EJ

5. Conclusion

On-chip packet routing is extremely crucial because it heavily affects performance and power. This calls for a great need of routing optimization. However, due to the diverse connectivity enabled by a network and the interferences in sharing network buffers and links, determining good routing paths, which are minimal and deadlock free for traffic flows, is nontrivial. In this paper, we have addressed the congestion-aware routing problem. With the analysis technique, we first estimate the congestion level in the network, and then embed this analysis technique into the loop of optimizing routing paths so as to quickly find deterministic routing paths for all traffic flows while minimizing the congestion level. Our experiments with both synthetic and realistic workloads show that we can extract high quality solutions with small computational time.

The proposed framework is appropriate for reconfigurable embedded systems-on-chip which run several applications with regular and repetitive computations on large set of data, e.g., multimedia and computer vision applications. CAR can not only design minimal and deterministic routing, but also can implement non-minimal and deadlock-free fully adaptive routing without virtual channels in arbitrary topology.

6. Reference

[1] K. Bondalapati and V.K. Prasanna, “Reconfigurable Computing Systems,” *Proceedings of the IEEE*, 90(7):1201-1217, 2002.

[2] O. Catoni, “Metropolis, Simulated Annealing, and Iterated Energy Transformation Algorithms, Theory and Experiments,” *Journal of Complexity* 12(4):595-623, 1996.

[3] G.-M. Chiu, “The Odd-Even Turn Model for Adaptive Routing,” *IEEE Transactions on Parallel and Distributed Systems*, 11(7):729-738, 2000.

[4] W. Dally and B. Towles, Principles and Practices of Interconnection Networks, Morgan Kaufmann Publishers Inc., First edition, 2004.

[5] W.J. Dally and C.L. Seitz, “Deadlock-Free Message Routing in Multiprocessor Interconnection Networks,” *IEEE Transactions on Computers*, 36(5):547-553, 1987.

[6] C.J. Glass and L.M. Ni, “The Turn Model for Adaptive Routing,” *Journal of the Association for Computing Machinery*, 41(5):874-902, 1994.

[7] P. Guerrier and A. Greiner, “A Generic Architecture for on-chip Packet-Switched Interconnections,” *Proceedings of the Design, Automation, and Test in Europe*, pp. 250-256, 2000.

[8] A. Hemani, *et. al.*, “Network on a Chip: An Architecture for Billion Transistor Era,” *Proceedings of the IEEE NorChip*, pp. 166-173, 2000.

[9] J. Hu and R. Marculescu, “DyAD - Smart Routing for Networks-on-Chip,” *Proceedings of the Design Automation Conference,* pp. 260-263, 2004.

[10] J. Hu and R. Marculescu, “Energy- and Performance-Aware Mapping for Regular NoC Architectures,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(4):551-562, 2005.

[11] M. A. Kinsy, *et. al.*, “Application-Aware Deadlock-free Oblivious Routing,” *Proceedings of the ISCA*, pp. 208-219, 2009.

[12] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. “Optimization by Simulated Annealing,” *Science*, 220(4598):671–680, 1983.

[13] S. Murali, *et. al.,* “Analysis of Error Recovery Schemes for Networks on Chips”, *IEEE Design and Test of Computers*, 22(5): 434-442, 2005.

[14] M. Palesi, *et. al.,* “Application Specific Routing Algorithms for Networks on Chip,” *IEEE Transactions on Parallel and Distributed Systems*, 20(3):316-330, 2009.

[15] K. Pawlikowski, “Steady-State Simulation of Queueing Processes: A Survey of Problems and Solutions,” *ACM Computing Surveys*, 22(2):123-170, 1990.

[16] V. Soteriou, H. Wang, L.-S. Peh, “A Statistical Traffic Model for On-Chip Interconnection Networks,” *Proceedings of the MASCOTS*, pp. 104-116, 2006.

[17] W. Trumler, *et. al.*, “Self-optimized Routing in a Network-on-a-Chip,” *IFIP World Computer Congress*, pp. 199-212, 2008.

[18] E.B. van der Tol and E.G. Jaspers, “Mapping of MPEG-4 Decoding on a Flexible Architecture Platform,” *SPIE*, vol. 4674, pp. 1-13, 2002.

[19] G. Varatkar and R. Marculescu, “Traffic Analysis for On-chip Networks Design of Multimedia Applications,” *Proceedings of the Design Automation Conference,* pp. 795-800, 2002.