

ACES: Application-Specific Cycle Elimination and Splitting for Deadlock-Free Routing on Irregular Network-on-Chip

Jason Cong, Chunyue Liu, Glenn Reinman

Computer Science Department, University of California, Los Angeles

Los Angeles, CA 90095, USA

{cong, liucy, reinman}@cs.ucla.edu

ABSTRACT

Application-specific Network-on-Chip (NoC) in MPSoC designs often requires irregular topology to optimize power and performance. However, efficient deadlock-free routing, which avoids restricting critical routes and also does not significantly increase power for irregular NoC, has remained an open problem until now. In this paper an *application-specific cycle elimination and splitting (ACES)* method is presented for this problem. Based on the application-specific communication patterns, we propose a scalable algorithm using global optimization to eliminate as much channel dependency cycles as possible while ensuring shortest paths between heavily communicated nodes, and split only the remaining small set of cycles (if any). Experimental results show that compared to prior work, ACES can either reduce the NoC power by 11%~35% while maintaining approximately the same network performance, or improve the network performance by 10%~36% with slight NoC power overhead (-5%~7%) on a wide range of examples.

Categories and Subject Descriptors

B.4.3 [Interconnections (Subsystems)]: Topology

General Terms

Algorithms, Design

Keywords

Application-specific Network-on-Chip, Deadlock-free routing

1. INTRODUCTION

As we shrink IC feature sizes to nanoscale, interconnect delay and power consumption emerge as the dominant factors in the optimization of Multi-Processor System-on-Chip (MPSoC). As the number of cores on a single die grows, more interconnect bandwidth will be required for on-chip communication. Therefore, the Network-on-Chip (NoC) design is proposed to solve these challenges because of its intrinsic scalability and predictability.

Application-specific NoC offers the opportunity to optimize the NoC for the target problem domain and does not necessarily conform to regular topologies. In general, the selection of the topology can have a dramatic impact on the overall performance, area and power. Recent work in application-specific NoC topology synthesis has shown significant power savings compared

to a regular NoC [1][2]. Alternatively, the performance of regular topologies can be significantly improved with minimal impact on area and energy consumption by adding application-specific long-range links [3] or radio frequency interconnections (RF-I) [4][5]. Clearly, the evolution of application-specific NoCs is leading to irregular topologies. However, efficient deadlock-free routing for NoC with irregular topologies remains an open problem.

There are two main approaches to dealing with deadlock in irregular NoCs. The first class of approaches is based on the theory of [6]. It divides the NoC into two virtual networks (VN): one fully-adaptive (no routing restrictions) and another that is deadlock-free (with routing restrictions). Network packets are routed in the full-adaptive VN at first and will be moved to the deadlock-free VN when there are no available resources in the full-adaptive VN. One recent example is the deadlock detection and recovery method used in [4] and [5]. However, since most application-specific MPSoC in the embedded system domain are power-critical, the power overhead of introducing two virtual channels for each physical channel is significant.

The second class of approaches handles the deadlock-free routing problem in irregular NoCs by imposing routing restriction, such as the turn prohibition algorithm [7] and south-last routing [3]. They restrict routings without considering the application-specific communication patterns; hence they may increase the routing distance between heavily communicated nodes. The recent work in [8] first proposes to remove dependencies based on the application communication requirement. It uses a greedy heuristic and may exhaustively enumerate all possible combinations of channel dependency cycles and thus cannot scale to large designs. In fact, sometimes it is impossible to make the channel dependency graph acyclic without disconnecting the network with unidirectional links. This complication is not considered in [8]. Moreover, [8] only considers whether there is communication between two nodes or not, without consideration of the data size transferred, which may lead to sub-optimal solutions.

Therefore, it is necessary to find an optimal trade-off point between power and performance (i.e., between these two types of approaches) for deadlock-free routing in irregular NoCs. We want something that avoids restricting critical routes in the NoC, but that also does not significantly increase NoC power. In this paper we propose an *application-specific cycle elimination and splitting (ACES)* method for this problem. We first develop a scalable algorithm using global optimization to eliminate as many channel dependency cycles as possible with the guarantee of network reachability, based on the application-specific communication patterns, and then only split the remaining small set of cycles (if any) using virtual channel splitting. Network performance is maintained by ensuring plentiful shortest paths between heavily communicated nodes. Moreover, with the possible existence of split channels, a routing table construction and encoding method is developed to minimize the hardware overhead of ACES.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

DAC'10, June 13–18, 2004, Anaheim, California, USA.

Copyright 2010 ACM 978-1-4503-0002-5 /10/06...\$10.00.

2. DEFINITIONS AND FRAMEWORK

We first present some definitions relevant to the application and the NoC architecture that will be used in the rest of the paper.

Definition 1: An *application characterization graph* $APCG(C, E)$ is a directed graph, where each vertex $c_i \in C$ represents an IP core and each edge $e_{c_i, c_j} \in E$ characterizes the communication from c_i to c_j . Each e_{c_i, c_j} is tagged with v_{c_i, c_j} which characterizes the communication volume (size of transferred data) from c_i to c_j .

Definition 2: A *NoC topology graph* $TG(R, Ch)$ is a directed graph where each vertex $r_i \in R$ represents a router. Each directed edge $ch_{i,j} \in Ch$ represents a physical unidirectional channel that connects an output port of r_i to an input port of r_j .

Here we assume the mapping of the IP cores in a given $APCG(C, E)$ to the routers of a given $TG(R, Ch)$ has already been done. Figure 1(a) shows an example of a TG upon which an $APCG$ is mapped. We use this as a working example in this section. The modeled NoC is a 3×3 baseline mesh overlaid with one shortcut from node A to node L . The different communication volumes between nodes are denoted by dashed lines.

Definition 3: Given a topology graph $TG(R, Ch)$ and a routing function P , a *channel dependency graph* $CDG(Ch, D)$ is a directed graph. Each vertex $ch_i \in Ch$ is a physical channel and each edge $d_{i,j} \in D$ is a channel dependency from ch_i to ch_j , i.e., a network packet is allowed by the routing function P to go from ch_i to ch_j .

According to the theory of [6], a routing function P for a TG is deadlock-free if the CDG is acyclic. Figure 1(b) illustrates the corresponding CDG of the TG in Figure 1(a) without any routing restrictions. A node AB denotes the physical channel from router A to B . The injection queue s_i and the ejection queue t_i of a router r_i are denoted in unfilled circles with their channel dependencies denoted by dashed lines. In the rest of the paper, a shortest path from router r_i to router r_j denotes the shortest path from s_i to t_j .

ACES breaks the cycles in the CDG based on the weight of the channel dependency edges. To appropriately guide the algorithm to remove the less-used edges and ensure a large number of shortest paths between heavily communicated nodes, an edge is weighted by both the number of shortest paths passing through it and the communication volume (size of transferred data). We introduce the *usage probability* to capture this property. Here, we use $\Phi(r_i, r_j)$ to denote the set of shortest paths in CDG from router r_i to r_j and $A(d, r_i, r_j)$ to denote the number of shortest paths in CDG from r_i to r_j that go through edge d .

Definition 4: Given a $CDG(Ch, D)$, the *usage probability* of an edge $d \in D$ for the application $APCG(C, E)$ is:

$$Pr(d) = \sum_{r_i, r_j, \text{ s.t. } d \in \Phi(r_i, r_j)} \sum_{c_i \in IP(r_i), c_j \in IP(r_j)} v_{c_i, c_j} A(d, r_i, r_j) / |\Phi(r_i, r_j)|$$

where, $IP(r_i)$ and $IP(r_j)$ are sets of IPs mapped on router r_i and r_j , respectively; v_{c_i, c_j} is the communication volume from vertex c_i to c_j (defined in Definition 1).

With this weighting function, heavily communicating nodes will make all the edges of their shortest paths have larger weights and thus are less likely to be removed; i.e., the more two nodes communicate, the shorter their distance will be and the more shortest paths they will have. We introduce the application-specific weights (usage probability) for a channel dependency graph $CDG(Ch, D)$ to construct an *application-specific channel dependency graph* ($ASCDG$). Figure 1(c) gives the $ASCDG$ based on the $APCG$ in Figure 1(a) and CDG in Figure 1(b). Figure 1(c) shows that the usage probability of different channel dependency edges varies considerably. This motivates the application-specific

nature of ACES. Without consideration of the application communication pattern, an acyclic CDG generated by the south-last routing algorithm is given in Figure 1(d), where the use of the shortcut is restricted with the forbidden southbound dependency from AL to LH/LF . The shortcut from A to L is augmented to optimize the topology for the application, but by applying south-last routing, the use of the shortcut is severely restricted. Figure 1(e) shows the acyclic CDG generated by ACES, where only the channel dependency edges that are never or rarely used are removed. It should be noted that *reachability* should be guaranteed while breaking cycles in the $ASCDG$; i.e., for each pair of communicating nodes of the application, there is at least one directed path from the source node to the destination node.

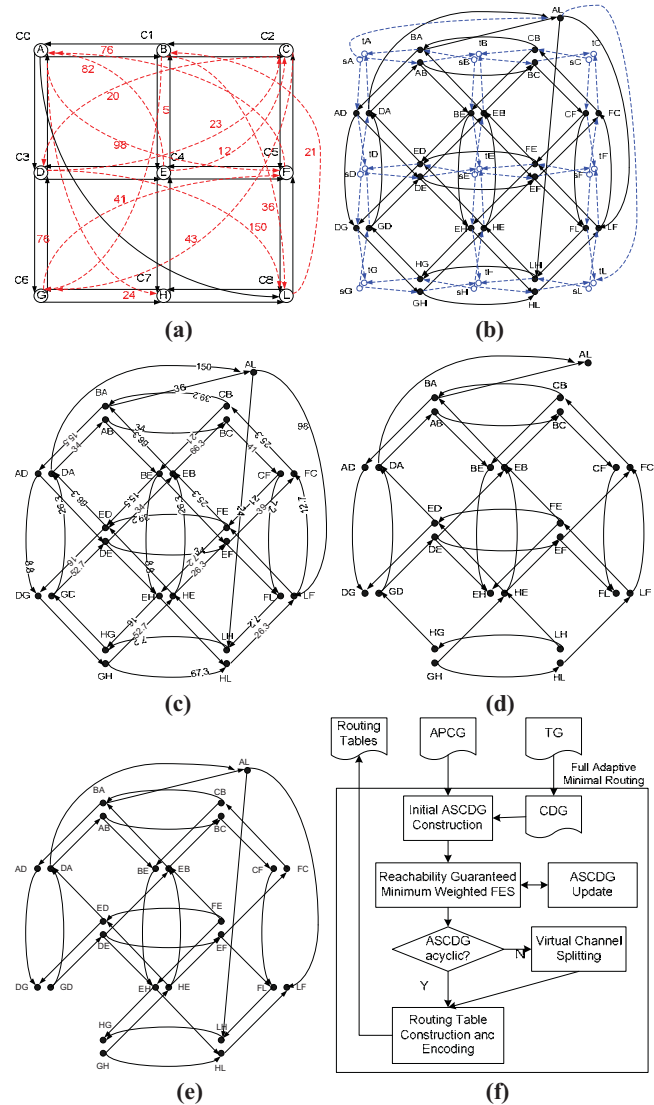


Figure 1. Example of (a) an $APCG$ mapped on a topology graph TG , (b) CDG , (c) $ASCDG$, (d) acyclic CDG by south-last routing, (e) acyclic CDG by the ACES. (f) Overall framework of ACES

An overview of the ACES framework is shown in Figure 1(f). The framework takes the $APCG$ and TG as the inputs. A CDG is constructed based on the given TG without any routing restrictions. From the $APCG$ and CDG , an initial $ASCDG$ is constructed by weighting the channel dependencies with the

application-specific communication patterns. Then the RG-MWFES algorithm (Section 3) is performed to remove channel dependencies that are not part of frequent communication routes. If the algorithm finishes with an acyclic *ASCDG*, virtual channel splitting is bypassed. Otherwise, it must be the case that the channel dependencies in the remaining cycles are kept to maintain the network reachability; thus, virtual channel splitting is used to break these remaining cycles. Furthermore, since application-specific NoC typically use routing tables to guide the routers to route the network packets [10], we construct and encode the routing tables based on the final acyclic *ASCDG*, to minimize the hardware overhead of ACES (Section 4). At the run time, the generated routing table will be loaded into the routers at the beginning of the application. Note that ACES is not restricted to routing-table-based method, its generated *ASCDG* can also be implemented in the logic-based routing method (such as the *routing bits* in LBDR proposed in [9]). To leverage the plentiful shortest paths between heavily communicated nodes in the *ASCDG* preserved by ACES, we use adaptive routing to dynamically balance the network traffic. It should be noted that, if deterministic load-balanced routing algorithms (e.g., [10]) are used, with plentiful shortest-path choices between heavily communicated nodes provided by ACES, these algorithms can also balance the load on each edge more easily; however, this is beyond the scope of this paper.

3. RG-MWFES ALGORITHM

Making the *ASCDG* acyclic is similar to the minimum weight feedback edge set (MWFES) problem on a directed graph, which is known to be NP-Complete [11]. Although good approximation methods are proposed in [12], our problem is more difficult here due to the requirement to guarantee *ASCDG* reachability. Although the existing MWFES algorithms can guarantee that the directed graph is connected after the cycle elimination, they can not guarantee the existence of a directed path between every application communication. Thus some source nodes may not reach their destination nodes in the *ASCDG* after applying existing MWFES algorithms. To overcome this problem, we introduce a new reachability guaranteed minimum weighted feedback edge set (RG-MWFES) problem in this paper as follows.

Given: An *ASCDG*(Ch, D),

Goal: Under the constraint of maintaining the reachability of *ASCDG*(Ch, D), remove edges to minimize the number of vertices that still get involved in cycles in the resulting *ASCDG* and the total weight of the removed edges.

The first goal is to minimize the vertices involved in cycles, since the physical channels represented by them will be split in the subsequent stage, resulting increased power consumption. In the meantime, we want to minimize the total weight of the removed edges in order to maintain network performance, since the weight of an edge characterizes the number of shortest paths and the communication volume that go through that edge.

We choose the MWFES algorithm in [12] as a starting point to develop approximation algorithms with this additional reachability constraint, since [12] represents a recent global optimization approach to MWFES problem on a directed graph. The idea of [12] is to decrease the weight of all the edges in any cycle it finds by the weight of the edge which will be removed from that cycle. Thus the more cycles an edge belongs to, the more likely its weight will be reduced and the more likely it will be removed in subsequent steps. Since more edges may be removed than necessary, a final edge add-back stage is performed to minimize the total weight of the removed edges while avoiding

re-introducing cycles. However, like other MWFES algorithms, [12] does not guarantee the reachability. To overcome this problem, we introduce an edge lock scheme in our RG-MWFES algorithm; i.e., if the removal of an edge will violate the *ASCDG* reachability, it is locked so that it can not be removed in the future. Here, we call this edge a *Critical Edge*. Critical edge checking is performed in the step of updating edge weight: given an edge d , if all the paths in *ASCDG* from a source router r_i to a destination router r_j of an application communication go through edge d , then d is a critical edge. It will be preserved and marked locked. However, if all the edges in a detected cycle are locked, the edge with the least weight in that cycle is *temporarily* removed. If it remains critical after the final edge add-back stage, it will be added back and handled by virtual channel splitting.

It should be noted that the edge weights are changed after each removal of a feedback edge, since the removal of an edge will increase the usage probability of other edges in the *ASCDG*. Moreover, the removal of an edge will make some other edges critical. Therefore, we can not fix the edge weight as the algorithm progresses. Instead, the edge weight needs to be updated at each iteration. However, since the weight of an edge is also reduced when another edge in the same cycle is removed, which is the key idea of the baseline MWFES algorithm [12] to obtain a globally optimized solution, it is not sufficient to use only one weight variable to keep track of these two kinds of weight update. In the RG-MWFES algorithm, for each edge d , we introduce two terms for the edge weight: base weight $w_b(d)$ and dynamic weight adjustment $w_a(d)$. The base weight $w_b(d)$ is initiated by the weight obtained from the initial *ASCDG* (*initial weight*). Each time that an edge is removed from the cycle, the $w_b(d)$ of all the other edges in this cycle will be reduced by the weight of the removed edge, which is the same as [12]. The $w_a(d)$ is initiated as 0 and is handled as follows: each time that we update the edge weights of all the remaining edges in the graph based on the new usage probability when some edges are removed, we calculate the new weight, and record the difference between this new weight and the initial weight as $w_a(d)$. When we want to find the minimum weight edge in a cycle, the sum of these two terms will be used to denote the current weight $w(d)$ of an edge d .

In the edge add-back step, the earlier an edge tries to be added back, the more likely it can be successfully added back without re-introducing cycles. Since the first goal of our algorithm is to minimize the vertices that still get involved in cycles after the algorithm is completed, we first add back all the removed locked edges in decreasing order of the total size of the cycles they once belonged to (at the time of removal) if their additions do not introduce any cycle. Then we add back the previously removed unlocked edges if they do not re-introduce cycles in the decreasing order of the weights. Thus the total weight of the removed edges is minimized, which is the second goal of our algorithm. After adding back the previously removed edges, some of the remaining removed locked edges may no longer be critical. Therefore we check the criticality of the remaining removed locked edges. If they are no longer critical, they will be treated as unlocked edges so that we do not need to add them back. Otherwise, the removed locked edges that remain will finally be added back and be handled by virtual channel splitting.

In sum, the RG-MWFES algorithm has three innovations: (i) network reachability is guaranteed, (ii) edge weights and criticality are properly and efficiently updated, and (iii) the algorithm simultaneously considers both network power

```

Algorithm RG-MWFES(ASCFG(Ch, D))
BEGIN
1  F ← ∅; // Already removed unlocked feedback edge set
2  F_Locked ← ∅; // Already removed locked feedback edge set
3  WHILE(ASCFG is cyclic) DO
4    Let C be a simple cycle in ASCFG;
5    IF all the edges in C are locked THEN
6      Let d be the edge in C with the
        minimum w(d) ← w_b(d) + w_a(d);
7      D = D ∪ d; F_Locked = F_Locked ∪ {d};
8    ELSE Let d be the unlocked edge in C with the
        minimum w(d) ← w_b(d) + w_a(d);
9      FOR each edge d' ∈ C
10     IF w(d') = w(d) and d' is unlocked, THEN
11       F = F ∪ {d'}; D = D ∪ d';
12     ELSE w_b(d') ← w_b(d') - w(d); FI
13     ENDFOR
14   FI
15   Update the w_a(d) of all the edges d of D;
16 ENDWHILE
17 edge_add_back(); // Add back edges and check the criticality
// of the remained critical edges
END

```

Figure 2. Pseudo-code of the RG-MWFES algorithm

consumption (by minimizing the need for channel splitting) and performance (by maximizing application-specific frequently used communication paths). Based on our experimental results (detailed in Section 5), for small NoCs the algorithm can break all cycles, while for large NoCs (more than 500 cycles in the original *ASCFG*) the algorithm can complete with only 4–5 cycles remaining. The pseudo-code of the RG-MWFES algorithm is shown in Figure 2. The complexity of the RG-MWFES algorithm is $O(|Ch|*|D|^3)$, given an *ASCFG*(*Ch*, *D*).

4. ROUTER ARCHITECTURE

The impact of ACES on the router architecture is twofold. On one hand it reduces the number of virtual channels (VC) compared to the virtual-network-based methods, which is the target of ACES. However, on the other hand, since ACES is based on *channel-specific routing* (i.e., each channel of the router may have different routing decisions for the same destination), a straightforward implementation will require a larger memory for the routing table than a routing algorithm where every input channel of the router has the same routing decision for the same destination (referred to as *router-specific routing*, and used in the fully-adaptive routing of the virtual-network-based methods). This may negate the power savings we obtained from reducing the number of VCs. Assuming an *M*-port router and the fact that each channel index can be represented by *L* bits, each routing table entry will contain *kL* bits, where *k* is a parameter related to the NoC adaptiveness. For a channel-specific routing, each input channel will require *NkL* space, and the routing table size will be *MNkL* for one router and MN^2kL for the whole NoC. However, a router-specific routing table only requires N^2kL space.

Without virtual channel splitting, an effective routing table encoding method is as follows. All of the routing entries for the input channels in a router to the same destination are combined into a single entry so that it contains all the possible output channels for these input channels (as in router-specific routing). For each input channel, a small register called the *forbidden register* is used to store the index of its forbidden output channels (corresponding to the removed edges from the *ASCFG* by RG-MWFES algorithm). Then at the routing computation stage, the router first reads the candidate output channels from the routing table and filters them with the forbidden register associated with the input channel. The channel-specific lookup table is avoided.

However, virtual channel splitting complicates this method. It breaks the cycles in *CDG* by splitting each channel along a cycle

into a pair of VCs: *high VC* and *low VC* (shown in Figure 3). There are only channel dependencies in the same level of VCs, except at the *breaking edge*, where there is only channel dependency from high VC to the low VC. The high VC holds all the shortest paths of the original channel while the low VC may not. Different VCs in the same input channels may have different routing decisions for the same destination, which makes the above encoding method impossible, and may result in $2MN^2kL$ space.

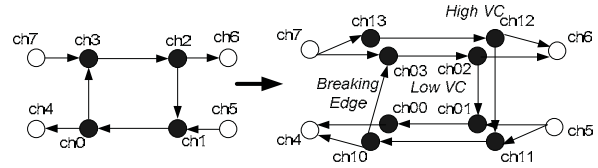


Figure 3. Virtual channel splitting to break a cycle

We overcome this problem by proposing an efficient routing table construction method. To ensure that different input channels use the same VC of an output channel when the output channel is located on the shortest path to one destination, we make the following rule: if both the low and high VCs are located in the shortest path from router r_i to r_j , the low VC will be selected (to leave the high channel available for other routes). This rule also maximizes the use of both low and high VCs.

From the above rule, we can see that for a pair of split VCs, if both have equal distance to the destination, the packet towards that destination must be located in the low VC; otherwise, the packet must be located in the high VC. Then, given a destination, we will know the location of the packet in the input channel; i.e., although the two VCs in an input channel may have different routing decisions to the same destination, the routing decision is fixed for this input channel. Thus, for each pair of split VCs, only one routing table entry is required.

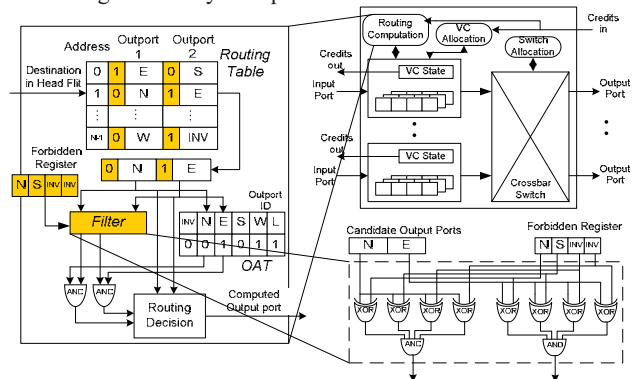


Figure 4. Router architecture for support of ACES

Figure 4 shows a 5-port adaptive router with the shaded modules augmented by ACES. In the output-channel availability table (OAT), a 0 means that the output-channel is not available. In the routing table, a 1-bit prefix is added to each output channel index (0 means low VC and 1 means high VC; the non-split channels are considered low VC). A filter is added and its schematic diagram is presented in Figure 4. If a candidate output channel is the same with one element of the forbidden register, the filter will output a 0 to invalidate the corresponding availability signal from OAT. Results from Cacti 5.3 show that by routing table encoding, the routing table area can be reduced by 66%, power can be reduced by 43% and accessing time can be reduced by 13% compared to that of a channel-specific routing table. [10]

shows that the typical critical path of a router is VC allocation (15-20 FO4), while the access time of the encoded router-specific routing-table is only 0.16ns in 32nm. Thus, the introduced logic (<3 FO4) will not impact the clock cycle. Orion 2.0 [14] reports that the area of the above router is 285,028 μm^2 , and the synthesis results of the augmented logic is only 167 μm^2 , which is less than 0.1% of the total router area.

5. EXPERIMENT RESULTS

5.1 Experiment Platform

The proposed ACES framework is implemented in C++. To accurately capture the NoC performance, we leverage the Garnet [13] network simulator and implement the dynamic routing table look-up method and virtual channel splitting scheme onto it. To accurately capture the NoC power, Orion 2.0 [14] is used to obtain the data of router dynamic/leakage energy and area with various router configurations and links.

We evaluate the ACES method on a platform of a regular mesh overlaid with RF-I shortcuts [5]. Because shortcuts can be allocated arbitrarily on the mesh, most existing irregular topologies can be generalized to it. Different benchmarks have different optimal shortcut allocations (via [5]), yielding different irregular topologies. ACES does not rely on the underlying mesh to achieve deadlock-free routing. For the power estimation of the RF-I, we adopt the method described in [5], which shows a 10×10 mesh (2cm \times 2cm chip) with a single cycle packet traversal delay across all RF-I shortcuts at 2GHz frequency. Thus in this paper, we assume that the delay to travel all the shortcuts is one cycle.

We compare the ACES method with two other commonly used methods for deadlock-free routing in irregular topology: deadlock detection and recovery (DDR) [5] (based on the theory of [6]) and south-last routing (SLR) [3]. SLR is used in the dead-free VN of DDR instead of using XY routing in [5] to achieve the best performance of DDR. As discussed in Section 1, DDR sacrifices power for performance, while SLR sacrifices performance for power. We will show that ACES can make a good trade-off between these two endpoints. We do not compare ACES to [8] as it can not scale to the size of most of our evaluated networks. The number of VCs for each port is 1 for SLR, 2 for DDR, 1 for non-split channel in ACES, 2 for split channel in ACES. Wormhole switching and adaptive routing are used. Each flit is 8 bytes and each packet can have 36, 12 or 4 flits based on the message type.

5.2 Benchmarks

To explore the interconnect demand of future multithreaded applications, we generate probabilistic traces to represent a variety of communication patterns as in [5]. The first pattern is *hotspot*, in which there is one component generating a disproportionate amount of traffic. This can be exhibited by caches holding frequently used synchronization variables or a master/worker paradigm. The second set of patterns, *uniDF* and *biDF*, clusters components into groups which are functionally laid out in a dataflow-like fashion. Components are biased to communicate with components within their group and with components in groups that neighbor them on either one side (*uniDF*) or both sides (*biDF*). These patterns would be seen in data decomposition or a functional decomposition into a pipelined pattern. The probabilistic trace files are generated for a 10×10 mesh overlaid with 16 shortcuts. Both low and high injection rate are tested for each trace (e.g., *hotspot_L/H*).

We also explored some real-life applications: three kernels and four applications from the *SPLASH* benchmark suite [15], an

MPEG-4 decoder [16], two embedded system applications, *auto_industry* and *telecom*, from the *E3S* benchmark suite [17].

For *SPLASH* benchmarks, the NoC is configured as a 10×10 mesh overlaid with eight shortcuts. Memory controllers are placed at chip corners. The largest network packets are between memory controllers and L2 cache banks; thus, we surround the memory controller with eight L2 banks (256KB, 8-way set associative, 128B block size) in each 3×3 corner to reduce the distance traveled by these messages. The other 64 nodes are tiles with one processor and one local L1 I/D cache (8KB, 4-way set associative, 32B block size). We collect network message injection traces of these benchmarks on a 64-core SPARC processor using Simics [18], and then execute these traces on Garnet.

A parallel version of the *MPEG-4* decoder is executed on an MPSoC with seven processors with local L1 I/D Cache (16KB, 4-way set associative, 16B block size), four L2 cache banks (1MB, 16-way set associative, 32B block size) and five memory controllers. The NoC is configured as a 4×4 mesh overlaid with one shortcut. In order to maintain the manually performed task partition, we leverage MC-Sim [19] to run the *MPEG-4* decoder and collected network message injection traces with two frames decoded, and then executed these traces on Garnet.

For *auto_industry* and *telecom*, the tasks are mapped onto a 4×4 mesh NoC overlaid with one shortcut. Simulation annealing is used to minimize the distance of the heavily communicated task nodes. Networks message injection traces are developed based on the communication pattern described in their task graphs.

5.3 Comparison Results

Figure 5 shows the NoC performance comparison results in terms of average network latency. To make a clear demonstration, the bars in the figure depict the normalized value to DDR and the absolute values are shown above the corresponding bars. By restricting the routing without considering the application-specific communication pattern, SLR consistently performs the worst among the three methods. Compared to DDR, ACES only introduces 1.9% degradation on average. In particular, when the injection rate goes high in *hotspot* and *biDF*, ACES outperforms DDR since many packets move to deadlock-free VN in DDR. In relatively small size NoC (i.e., the *MPEG4*, *auto_industry* and *telecom*), ACES performs almost the same as DDR.

Figure 6 shows the NoC power results. Compared to DDR, ACES can achieve a significant power reduction (11%~35%) since only a small set (0%~16%) of the network channels need to be split (The number above the third bar of each benchmark shows the percentage of channels that are split by ACES). Compared to SLR, although a small set of channels are split, by reducing the distance between the most frequently communicated nodes, ACES also reduces the dynamic power, and thus has power similar to SLR.

Overall, ACES provides the best trade-off of NoC performance and power. It improves the performance by 10% ~36% with a power overhead of -5%~7% compared to SLR. It reduces the power by 7%~35% on average while maintaining approximately the same network performance compared to DDR.

6. CONCLUSIONS

An application-specific cycle elimination and splitting (ACES) method is proposed in this paper to provide an efficient deadlock-routing method in application-specific NoC designs. For power-critical designs where only 1~2 VCs can be used for each router-port, ACES can achieve the best power-performance trade-off. It should be noted that, for performance-critical designs with

plentiful VCs for each router-port, ACES can still be used to optimize the deadlock-free virtual network (VN) based on [6], by ensuring plentiful shortest-paths between heavily communicated nodes, which is important because latency in the deadlock-free VN is crucial to the whole NoC latency when the network traffic is heavy since packets moved to the deadlock-free VN can not go back to the fully-adaptive VN to avoid livelock.

Future work includes extending ACES to the fault-tolerant NoC designs since regular NoC with faults can also be considered as irregular networks. Moreover, the scheme to dynamically and incrementally adjust the routing table to the communication variation is also an interesting topic for future work.

7. ACKNOWLEDGEMENTS

This research is partially supported by the SRC Contract 2009-TJ-1984, the NSF Expeditions in Computing Award CCF-0926127 and the NSF grant CCF-0903541. We also thank Mishali Naik and Karthik Gururaj for interesting discussions and helps in the experiments throughout the course of this work.

8. REFERENCES

[1] A. Pinto, L. P. Carloni and A. L. Sangiovanni-Vincentelli, "Efficient synthesis of networks on chip," In *Proc. ICCD*, pp. 146-150, Oct. 2003.

[2] U. Y. Ogras and R. Marculescu, "Energy- and performance-driven NoC communication architecture synthesis using a decomposition approach," In *Proc. DATE*, pp. 352-357, Mar. 2005.

[3] U. Y. Ogras and R. Marculescu, "It's a small world after all: NoC performance optimization via long-range link insertion," In *IEEE Trans. VLSI Syst. - Special Section Hardware/Softw. Codesign Syst. Synthesis*, vol. 14, no. 7, pp. 693 - 706, Jul. 2006.

[4] M. F. Chang, J. Cong, A. Kaplan, M. Naik, G. Reinman, E. Socher and S.W. Tam, "CMP network-on-chip overlaid with multi-band RF-Interconnect," In *Proc. HPCA*, pp. 191-202, Feb. 2008.

[5] M. F. Chang, J. Cong, A. Kaplan, C. Liu, M. Naik, J. Premkumar, G. Reinman, E. Socher, and R. Tam, "Power reduction of CMP communication networks via RF-Interconnects," In *Proc. MICRO*, pp. 376-387, Nov. 2008.

[6] J. Duato and T. M. Pinkston, "A general theory for deadlock-free adaptive routing using a mixed set of resources," In *IEEE Trans. Parallel and Distributed Systems*, Vol. 12, Issue 12, pp. 1219-1235, Dec 2001.

[7] D. Starobinski, M. Karpovsky, L. A. Zakrevski, "Application of network calculus to general topologies using turn-prohibition," In *IEEE/ACM Trans. Networking*, vol. 11, issue 3, pp. 411-421, 2003.

[8] M. Palesi, R. Holsmark, S. Kumar and V. Catania, "Application specific routing algorithms for networks on chip," In *IEEE Trans. Parallel and Distributed Systems*, vol. 20, issue 3, pp. 316 - 330, March 2009.

[9] J. Flich and J. Duato, "Logic-Based Distributed Routing for NoCs," In *IEEE Computer Architecture Letters*, vol. 7, Issue 1, pp. 13-16, Jan. 2008

[10] M. Kinsky, M. H. Cho, T. Wen, E. Suh, M. Dijk, S. Devadas, "Application-Aware Deadlock-Free Oblivious Routing," in *Proc. ISCA*, pp. 208-219, 2009.

[11] M. R. Garey and D. S. Johnson, *Computers and intractability: a guide to theory of NP-completeness*, W. H. Freeman, 1979.

[12] C. Demetrescu, I. Finocchi, "Combinatorial algorithms for feedback problems in directed graphs," In *Information Processing Letters*, Vol. 86, Issue 3, pp. 129-136, May 2003.

[13] N. Agarwal, L.-S. Peh, and N. Jha, "Gamet: A detailed interconnection network model inside a full-system simulation framework," Technical Report CE-P08-001, Dept. of Electrical Engineering, Princeton University, 2007.

[14] A. Kahng, B. Li, L.-S. Peh and K. Samadi, "ORION 2.0: A Fast and Accurate NoC Power and Area Model for Early-Stage Design Space Exploration" In *Proc. DATE*, pp. 423 - 428, April 2009.

[15] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. "The SPLASH-2 Programs: Characterization and Methodological Considerations," In *Proc. ISCA*, pp. 24-36, June 1995.

[16] P. Schumacher and W. Chung, "FPGA-based MPEG-4 codec," In *DSP Magazine*, pp. 8-9, 2005.

[17] R. Dick, Embedded system synthesis benchmarks suites. Available: <http://www.ece.northwestern.edu/~dickrcp/e3s>

[18] P. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner. "Simics: A full system simulation platform," In *IEEE Computer*, vol. 35, pp. 50-58, Feb 2002.

[19] J. Cong, K. Gururaj, G. Han, A. Kaplan, M. Naik and G. Reinman, "MC-Sim: An efficient simulation tool for MPSoC designs," In *Proc. ICCAD*, pp. 364-371, Nov. 2008.

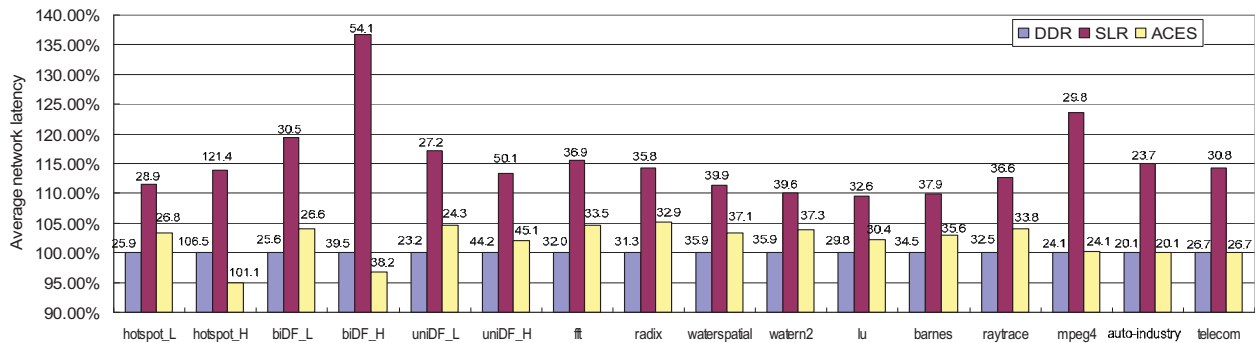


Figure 5. Comparison results for NoC performance

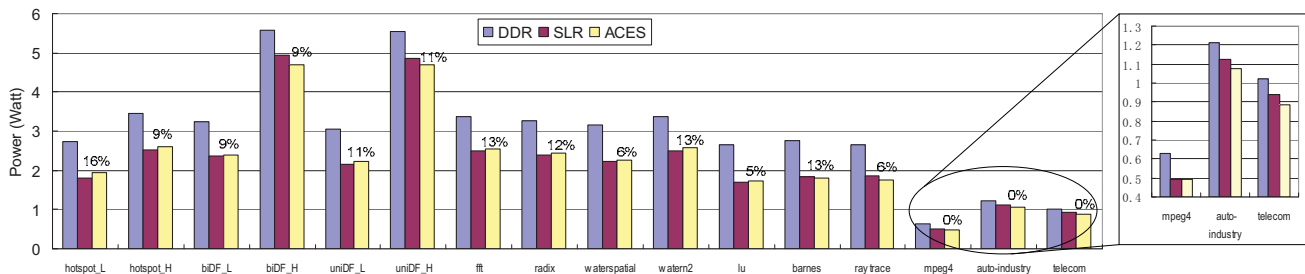


Figure 6. Comparison results for NoC power