Traffic-Aware Reconfigurable Routing Algorithm Postprint

Authors: Fu Binzhang, Han Yinhe, Li Huawei, Li Xiaowei

Date: 2016-06-08T00:00:00+00:00

Abstract

In many-core processor systems, Network-on-Chip (NoC) is commonly employed to provide high-bandwidth, low-latency, and highly reliable on-chip communication. To mitigate network congestion and enhance network performance, traffic-balanced routing algorithms have garnered extensive attention from researchers. Such algorithms typically leverage fully adaptive routing algorithms to provide path diversity; however, existing fully adaptive routing algorithms either require a substantial number of virtual channels or assume a conservative flow control strategy. On the one hand, virtual channels represent relatively expensive resources; on the other hand, conservative flow control strategies may result in degraded network performance. Consequently, researchers have proposed exploiting application traffic information to improve routing performance. These algorithms can be reconfigured for diverse traffic characteristics without employing virtual channels, thereby achieving on-demand allocation of routing adaptivity. Based on the type of traffic information utilized, traffic-aware reconfigurable routing algorithms can be categorized into offline and online algorithms. Offline algorithms necessitate prior knowledge of application traffic characteristics and are therefore primarily targeted at application-specific multicore Systems-on-Chip (SoCs). Online algorithms, conversely, reconfigure based on traffic information collected at runtime and can thus be applied to general-purpose processor systems. This paper will discuss two prominent offline algorithms recently proposed in the international community, and will emphasize the online reconfigurable routing algorithm based on the abacus turn model, which was presented by the authors of this paper at the 2011 International Symposium on Computer Architecture (ISCA 11).

Full Text

Traffic-Aware Reconfigurable Routing Algorithms

Binzhang Fu, Yinhe Han, Huawei Li, Xiaowei Li

Abstract: In many-core processor systems, Networks-on-Chip (NoC) are commonly employed to provide high-bandwidth, low-latency, and reliable on-chip communication. To reduce network congestion and improve performance, load-balancing routing algorithms have attracted significant research attention. These algorithms typically leverage fully adaptive routing to provide path diversity. However, existing fully adaptive routing algorithms either require numerous virtual channels or assume conservative flow control strategies. Virtual channels are expensive resources, while conservative flow control may degrade network performance. Consequently, researchers have proposed utilizing application traffic information to enhance routing performance. Such algorithms can be reconfigured for different traffic characteristics without using virtual channels, thereby enabling on-demand allocation of routing adaptivity. Based on the type of traffic information used, traffic-aware reconfigurable routing algorithms can be classified as offline or online. Offline algorithms require prior knowledge of program traffic characteristics and are primarily designed for application-specific multicore SoCs. Online algorithms, by contrast, perform reconfiguration based on traffic information collected at runtime and can thus be applied to general-purpose processor systems.

This paper discusses two prominent offline algorithms recently proposed internationally and focuses on the online reconfigurable routing algorithm based on the Abacus Turn Model, presented by the authors at the 2011 International Symposium on Computer Architecture (ISCA' 11).

Keywords: Network-on-Chip, routing algorithm, routing reconfiguration, load balancing

1 Introduction

Constrained by the memory wall, ILP wall, and power wall, traditional approaches relying on single-processor performance improvements no longer yield satisfactory results. In this context, multi-core and many-core processors that enhance application performance through parallelization are considered viable solutions [1]. In many-core processor systems, NoC is widely expected to become the mainstream interconnect solution due to its excellent performance [2]. Since NoC provides communication between processors or between processors and caches, its performance significantly impacts overall system performance. Generally, NoC performance depends primarily on network topology, flow control strategy, and routing algorithm.

Network topology determines the shortest distance between any two nodes and the network' s bisection bandwidth, thereby defining its peak performance. Topology design must consider various factors, including port count, bandwidth per port, wiring density allowed by the process technology, and signal rates. Common network topologies include crossbar matrices, Clos Networks, butterfly networks, and Tori networks (including mesh, torus, and hypercube) [3]. Among these, the two-dimensional mesh' s planar structure facilitates manufac-

turing and has been widely adopted in NoC systems [4][5][6]. This paper focuses primarily on two-dimensional mesh networks.

Flow control mechanisms allocate network resources such as channel bandwidth, buffer space, and state information to data packets. Effective flow control should accurately and efficiently allocate resources to packets that need them most. Common flow control mechanisms in packet-switched networks include store-and-forward, virtual cut-through, wormhole, and virtual channel mechanisms. Wormhole switching is widely adopted in NoC systems because it effectively reduces buffer requirements and packet latency. Wormhole switching divides packets into multiple flow control digits (flits). Each packet consists of a head flit, several body flits, and a tail flit. The head flit establishes the path, subsequent flits follow the same route, and the tail flit releases the path. The discussions in this paper are based on NoC employing wormhole switching.

Routing algorithms determine the transmission path of packets through the network. Effective routing algorithms should specify paths that enable packets to reach their destination nodes as quickly as possible [3]. Based on transmission distance, routing algorithms can be classified as minimal-path or non-minimal-path. Minimal-path routing requires each hop to select a direction that brings the packet closer to its destination. This approach offers simple rules and guarantees freedom from livelock. Non-minimal-path routing, while more complex, can enhance network tolerance to congestion and faults. For clarity, this paper focuses primarily on minimal-path routing algorithms.

Based on whether network state is considered during routing, algorithms can be classified as oblivious or adaptive. Oblivious routing ignores network state, resulting in simpler implementation but lower tolerance to congestion, router failures, and link failures. Adaptive routing algorithms can dynamically select paths based on current network conditions, thereby avoiding congestion.

According to the number of usable paths, adaptive routing algorithms can be categorized as partially adaptive or fully adaptive. Fully adaptive routing allows packets to use any path between source and destination nodes, while partially adaptive routing permits only a subset of paths.

Since sufficient path diversity is needed to alleviate congestion, most load-balancing routing algorithms currently utilize a fully adaptive routing algorithm to provide candidate output ports [7][8][9][10][11][12]. However, common fully adaptive routing algorithms either require numerous virtual channels [13][14] or assume very conservative flow control strategies [15][16][17].

Because virtual channels are relatively expensive to implement in NoC systems, routing algorithms requiring many virtual channels, such as those in [13][14], are unsuitable for NoC. Duato' s theory-based routing algorithms require fewer virtual channels but demand conservative flow control. According to Duato' s theory, router buffer queues cannot store flits from different packets simultaneously [15]. A buffer queue can only be reallocated after the tail flit of the previous packet has departed. This constraint ensures that when a packet expe-

riences congestion, its head flit remains at the head of the buffer queue, allowing it to select an escape channel. However, this wastes buffer space. For networks transmitting short packets, this constraint degrades network performance [18].

Compared with fully adaptive routing algorithms, partially adaptive routing algorithms without virtual channels incur lower implementation overhead and can employ more advanced flow control strategies. Examples include the Turn Model by Glass and Ni [19] and the Odd-Even Turn Model by Chiu [20]. The Turn Model designs partially adaptive routing algorithms without virtual channels by dividing all possible turns in a network into two abstract cycles: clockwise and counterclockwise. By prohibiting one turn in each abstract cycle, deadlock freedom is guaranteed. However, Chiu discovered that the adaptivity provided by Turn Model-based algorithms is non-uniform. To address this, he proposed the Odd-Even Turn Model. Nevertheless, the Odd-Even Turn Model cannot provide full adaptivity for any node pair with a distance greater than two.

Non-uniform or insufficient routing adaptivity may cause network congestion under bursty traffic conditions. To solve this problem, researchers have proposed reconfigurable routing algorithms that provide on-demand routing adaptivity for applications. Overall, reconfigurable routing algorithms are partially adaptive, thus requiring neither virtual channels nor conservative flow control. These algorithms analyze application traffic characteristics to provide more path diversity for packets in heavily loaded directions.

Based on how traffic information is obtained, reconfigurable routing algorithms can be divided into offline and online categories. Offline reconfigurable routing algorithms assume that application traffic information can be obtained in advance and determine routing strategies through traffic analysis. Representative offline algorithms include the Application-Specific Routing Algorithm published in IEEE TPDS Issue 3, 2009 [21] and the Application-Aware Oblivious Routing Algorithm presented at ISCA' 09 [22].

Compared with offline algorithms, online reconfigurable routing algorithms can reconfigure themselves based on traffic information collected online. Therefore, online reconfigurable routing algorithms can be applied to general-purpose processor systems. This paper introduces the reconfigurable routing algorithm based on the Abacus Turn Model, presented by the State Key Laboratory of Computer Architecture at ISCA' 11 [23].

2 Offline Reconfigurable Routing Algorithms

The core challenge in designing offline reconfigurable routing algorithms is maximizing network performance while guaranteeing deadlock freedom. Most offline algorithms follow this basic approach: first construct a channel dependency graph, then ensure deadlock freedom by removing all cycles from the graph, and finally improve network performance by balancing traffic across channels.

The two offline algorithms discussed in this paper follow this general approach but with different emphases.

2.1 Application-Specific Routing Algorithm [21]

Application-Specific Routing Algorithms (APSRA) exploit the characteristics that certain nodes may not communicate or certain node pairs may not communicate simultaneously to improve routing performance. APSRA first abstracts an application as a task graph and maps tasks to different processor nodes. It then transforms the task graph into a channel dependency graph by referencing the network topology. Finally, it analyzes the channel dependency graph and ensures deadlock freedom by guaranteeing it is acyclic. Additionally, during cycle removal, it enhances routing performance by balancing traffic across channels. We illustrate its basic working principle using an example from [21]; detailed algorithms and implementation specifics can be found in [21].

[Figure 1: see original paper] shows the example. The communication graph in Figure 1(a) indicates that the application contains six tasks. Communication between tasks is represented by arrows, where double arrows indicate bidirectional communication and single arrows indicate unidirectional communication. Figure 1(b) shows the network topology, where circles represent processors and arrows represent channels connecting processors. In this example, we assume task Ti is mapped to processor Pi, i.e., M(Ti)=Pi, i=1, 2, ···, 6, where M is the mapping function. To generate the channel dependency graph, we first assume a fully adaptive minimal-path routing algorithm. The resulting channel dependency graph is shown in Figure 1(c). Since this graph contains six dependency cycles, according to Dally' s theory [14], the network cannot use fully adaptive routing [3]. However, not all tasks actually communicate. The application channel dependency graph after removing non-existent dependencies is shown in Figure 1(d). For example, dependency l12→l23 exists in the channel dependency graph (Figure 1(c)) but not in the application channel dependency graph (Figure 1(d)). By analyzing the topology, we find that only communications T1→T3, T1→T6, and T4→T3 could introduce dependency l12→l23. According to the communication graph, none of these communications exist. Therefore, dependency l12→l23 never actually occurs and can be removed from the application channel dependency graph. The final application channel dependency graph contains two cycles, so according to Dally' s theory, fully adaptive routing still cannot be used. To remove cycles, dependencies l41→l21 and l14→l45 are prohibited.

While this eliminates deadlock, routing adaptivity is compromised. If the timing of inter-task communications is known in advance—for example, if communications T1→T5 and T2→T4 do not overlap in time—then although cycles exist in the application channel dependency graph, they never actually form in the network. Therefore, we need not remove these cycles, and the resulting routing algorithm can remain fully adaptive.

2.2 Application-Aware Oblivious Routing Algorithm [22]

Unlike APSRA, [22] argues that adaptive routing increases implementation complexity and therefore proposes using oblivious routing. To address different application characteristics, [22] presents an application-aware oblivious routing generation flow consisting of five steps:

1. Select the best path set.

Step 1: Removing all cycles from the channel dependency graph guarantees that the generated oblivious routing algorithm is deadlock-free. However, finding and removing all cycles in a directed graph is complex, so [22] proposes using the Turn Model [19] to construct acyclic channel dependency graphs. For example, assuming the North-Last routing algorithm [19], the resulting channel dependency graph is guaranteed to be acyclic. Readers can refer to [14] for channel dependency graph construction methods. For the 3×3 mesh network shown in Figure 2(a), the channel dependency graph corresponding to the North-Last routing algorithm is shown in Figure 2(b). When routing a packet from node E to node G using North-Last, the packet can travel along path E→D→G, so channel ED depends on channel DG. Correspondingly, an edge from ED to DG exists in Figure 2(b). Clearly, Figure 2(b) contains no cycles.

[Figure 2: see original paper] Channel dependency graph and flow graph construction example

Step 2: Build a flow graph by inserting "dummy nodes" into the acyclic channel dependency graph from Step 1. Dummy nodes are actually routers in the network. For example, if the application has a communication flow from node H to node D, nodes H and D are inserted as dummy nodes into the graph. The source node H (labeled S1) is connected to all its output channels, and the destination node D (labeled D1) is connected to all its input channels. As shown in Figure 2(c), edges connected to dummy nodes are represented by dashed lines. Figure 2(c) only adds one communication flow H→D; the actual application may contain multiple flows, and the final flow graph should include all communication flows.

Step 3: Select a path for each communication flow in the constructed flow graph. Path allocation should satisfy certain constraints, such as minimizing maximum channel load. For small-scale networks, [22] formulates path selection as a mixed integer linear programming problem; for large-scale networks, it proposes a greedy algorithm that first sorts communication flows in descending order of bandwidth demand, then selects paths for each flow sequentially using a weighted Dijkstra shortest-path algorithm.

Step 4: Check whether all communication flows have been assigned paths and whether the network state meets optimization objectives. If satisfied, return to Step 1 to construct a new acyclic channel dependency graph using a different turn-model routing algorithm and select paths for each communication flow.

Step 5: Compare the communication flow paths generated for different turn-model routing algorithms and select an optimal path set according to predefined criteria.

The selected path set is then written into the chip via routing tables, allowing the application to route communication flows along optimized paths. [22] also discusses optimization methods for networks with multiple virtual channels. For example, different turn-model routing algorithms can be used in different virtual channels to construct different flow graphs, thereby partitioning communication flows into different virtual networks to better balance network traffic. Details can be found in [22].

3 Online Reconfigurable Routing Algorithm

This section discusses the Abacus Turn Model and its corresponding reconfigurable routing algorithm proposed in [23].

3.1 Basic Idea of the Abacus Turn Model

The original Turn Model proved that a network is deadlock-free if all allowed turns do not form abstract cycles [19]. [20] further proved that a network is deadlock-free if neither the rightmost clockwise nor rightmost counterclockwise cycles exist. As shown in Figure 3(a), the rightmost clockwise cycle consists of two turns (East-to-South (ES) and South-to-West (SW)) and several North-to-South (NS) channels.

To eliminate all rightmost cycles, [20] requires all nodes in odd columns to prohibit South-to-West turns (Figure 3(b)) and all nodes in even columns to prohibit East-to-South turns (Figure 3(c)). The principle for prohibiting counterclockwise rightmost cycles is similar and is omitted here.

[Figure 3: see original paper] Composition of the rightmost clockwise cycle

[23] inherits the core idea from [20] that "if neither clockwise nor counterclockwise rightmost cycles exist, the network is deadlock-free." Unlike the original Turn Model, [23] provides more flexible methods for eliminating rightmost cycles. As shown in Figure 3(a), forming a rightmost clockwise cycle requires an East-to-South turn to be above a South-to-West turn. Therefore, clockwise rightmost cycles can be eliminated by prohibiting East-to-South turns above all South-to-West turns (Figure 3(d)). After this prohibition, each column in the network must contain a point: above this point, all East-to-South turns are prohibited, while below it, all South-to-West turns are prohibited. [23] refers to this type of node as a clockwise bead node. Similarly, each column also contains a counterclockwise bead node to separate counterclockwise turns, preventing the formation of counterclockwise rightmost cycles.

3.2 The Abacus Turn Model

As shown in Figures 4(a) and 4(b), a 4×4 mesh network is analogized to an abacus, with each column assumed to have a rod and two sliding beads: a clockwise bead and a counterclockwise bead. The rectangular blocks with dashed edges represent possible bead node positions, solid ellipses represent clockwise beads, hollow ellipses represent counterclockwise beads, and dashed arrows represent prohibited turns. For clarity, allowed turns are not shown. The clockwise and counterclockwise beads can be controlled independently and are used to determine the distribution of clockwise and counterclockwise turns in each column, respectively.

The Abacus Turn Model is defined by three rules:

1. The clockwise (respectively, counterclockwise) bead node does not prohibit clockwise (respectively, counterclockwise) turns.

Figures 4(c) and 4(d) show the distribution of prohibited turns following the Abacus Turn Model. According to this model, the holder of a clockwise (respectively, counterclockwise) bead node may allow all clockwise (respectively, counterclockwise) turns. All routers above the clockwise (respectively, counterclockwise) bead node allow all clockwise turns except East-to-South (respectively, all counterclockwise turns except North-to-West), while routers below allow all clockwise turns except South-to-West (respectively, all counterclockwise turns except East-to-North). Turns distributed according to these rules do not form clockwise or counterclockwise rightmost cycles. Therefore, designing deadlock-free routing algorithms under the Abacus Turn Model simplifies to determining the positions of clockwise and counterclockwise bead nodes in each network column. Once positions are determined, new routing rules are formed.

For a k×k mesh network, each column contains two bead nodes. Since each bead node has k possible positions, each column has k×k configuration possibilities. For the entire network with k columns, there are (k×k)k possible configurations. Each bead node configuration represents a routing configuration, so routing algorithm reconfiguration becomes a matter of moving beads within each column. The following example illustrates how reconfigurable routing based on the Abacus Turn Model works.

As shown in Figure 5(a), the clockwise bead is initially placed in the bottom row of the network. Therefore, according to the Abacus Turn Model, all nodes above the bead node prohibit East-to-South turns. Suppose a hotspot node 5 is detected, and node 6 needs to send packets to it for a relatively long period. Since only one minimal path is available for this node pair, that path easily becomes congested. To obtain more usable paths for load balancing, node 6 complains to its east neighbor node 7 about the current situation—specifically, that node 7 prohibits the East-to-South turn needed by node 6. Node 7 collects complaints from its neighbors and negotiates with the bead owner node 1. The bead owner evaluates requests from different nodes and ultimately decides

whether to relinquish bead ownership and to whom to pass the bead. In this example, node 1 passes the bead to node 7. As shown in Figure 5(b), after node 7 obtains the clockwise bead, it can allow East-to-South turns. Now node 6 has two paths to node 5 for load balancing. Similar to node 6, node 7 complains to its neighbor node 8 because node 8 also prohibits East-to-South turns. After a similar process, node 8 also obtains clockwise bead ownership and opens the East-to-South turn. As shown in Figure 5(c), all minimal paths between node 6 and node 5 can now be used by packets for traffic balancing.

[Figure 5: see original paper] Example of Abacus Turn Model routing

3.3 Safe Bead Passing

Guaranteeing deadlock freedom while moving bead nodes in the network is challenging. As shown in Figure 6(a), node 1 is the clockwise bead owner and thus allows South-to-West turns. Node 4 can use this South-to-West turn to send packets to node 0. If the bead node is moved upward (Figure 6(b)), node 1 will prohibit South-to-West turns. If node 4 does not detect this change promptly and continues sending packets destined for node 0 to node 1, these packets will be blocked at node 1. Additionally, after node 7 obtains the clockwise bead, it can open the East-to-South turn. However, if this turn is opened too early—for example, while packets still exist in nodes 1 and 4 using the South-to-West turn —a clockwise rightmost cycle may form, violating the Abacus Turn Model rules.

[Figure 6: see original paper] Distribution of prohibited turns before and after bead movement

To address these issues, bead movement must satisfy two rules:

1. If it is certain that no packet is using any turn that could form a rightmost cycle with a given turn, that turn may be allowed.

Generally, moving a bead by h hops requires prohibiting and enabling h turns respectively. For large-scale networks, simultaneously satisfying both requirements is extremely difficult. To simplify this complexity, [23] divides bead movement into h sub-steps, moving the bead only one hop per step. This basic sub-step is treated as a safe atomic operation named "bead-passing." Using this safe atomic operation offers two benefits: first, bead-passing can be completed through local interaction between neighbors, providing good scalability; second, bead-passing itself guarantees that the network will not introduce deadlocks during reconfiguration, allowing designers to develop reconfigurable routing algorithms without considering complex deadlock issues.

In each atomic operation, only the current bead owner needs to prohibit a turn. For example, in Figure 5, to move the bead upward from node 1 to node 4, node 1 must prohibit the South-to-West turn. To achieve this, node 1 first notifies node 4 to stop sending Southwest-bound packets because these packets might need the South-to-West turn. Upon receiving this notification, node 4 sets its South output port to "not accepting Southwest-bound packets." Subsequently,

all Southwest-bound packets at node 4 will be routed through its West output port to node 3. However, node 4 cannot send an acknowledgment to node 1 yet because packets that have completed the routing stage may still exist. These packets were routed without knowing that node 1 would prohibit the South-to-West turn. Therefore, before sending acknowledgment to node 1, node 4 must drain any Southwest-bound packets that might be destined for node 1.

Methods for clearing specific types of packets within routers have been widely studied in the routing reconfiguration domain [24][25][26][27][28][29]. [23] borrows the reconfiguration token concept from [29], but differs in that not all packets need to be drained. In the above example, node 4 only needs to drain Southwest-bound packets. Since node 4 can only receive Southwest-bound packets from its Local, East, and North input ports, reconfiguration tokens need only be inserted at the ends of the corresponding input buffer queues. After the South output port receives tokens from all three input ports, node 4 can confirm that no Southwest-bound packets potentially routable to node 1 remain in its buffers. Node 4 can then send acknowledgment to node 1.

After receiving acknowledgment, node 1 can be confident that its North input port will not receive new Southwest-bound packets. However, Southwest-bound packets may still exist in the buffer queue of its North input port. Therefore, before prohibiting the South-to-West turn, node 1 must also drain Southwest-bound packets from its own North input port. Afterward, node 1 can prohibit the South-to-West turn and pass the bead upward. Upon receiving the bead, node 4 opens the East-to-South turn and notifies its West neighbor that it can now send Southeast-bound packets to it. This completes one bead-passing operation. Figure 7 summarizes the operations for moving clockwise and counterclockwise beads in pseudocode form. The above discussion corresponds to lines 2-5 in Figure 7(a). The principles for moving the clockwise bead downward and moving the counterclockwise bead are similar, differing only in the involved turns.

[Figure 7: see original paper] Pseudocode for bead-passing

3.4 Reconfigurable Routing Algorithm Based on the Abacus Turn Model

The Abacus Turn Model and bead-passing atomic operation simplify reconfigurable routing design to determining rules for bead movement direction.

3.4.1 Arm-Wrestling Routing Algorithm According to the Abacus Turn Model, four non-critical turns—West-to-North, North-to-East, West-to-South, and South-to-East—are always allowed. The other four turns—East-to-South, South-to-West, East-to-North, and North-to-West—require reconfiguration through bead-passing. These four turns can be divided into two groups: the clockwise group (East-to-South and South-to-West) and the counterclockwise group (East-to-North and North-to-West). Generally, moving a bead means

prohibiting a turn at the old bead owner while enabling another turn from the same group at the new owner. Therefore, the demand for different turns naturally becomes the basis for determining bead movement direction. To record demand for different turns, each node maintains three registers: CTxy, CTxy-n, and CTxy-s, representing the demand for turn xy at the current node, its North neighbor, and its South neighbor, respectively, where xy {es, sw, en, nw} (e: East; s: South; n: North; w: West; es: East-to-South, sw: South-to-West, etc.).

With node demand for different turns recorded, moving beads up or down can be analogized to pushing a zero-mass block up or down a vertical wall. As shown in Figure 8, the bead is analogized as a zero-mass block. Without loss of generality, assume this block is a clockwise bead.

In the arm-wrestling algorithm, to move the block upward, the North neighbor of the current bead holder applies an upward force Fup. This force actually reflects the North neighbor's demand for the East-to-South turn, because only by pulling the bead up can this node open the East-to-South turn. Therefore, the greater the North neighbor' s demand for East-to-South, the larger the upward force. Simultaneously, the South neighbor of the bead holder applies a force to open the South-to-West turn. This force reflects the South neighbor' s demand for the South-to-West turn. For the current bead holder, once it loses the bead, it must prohibit the corresponding turn. For example, if the bead moves upward, the current holder will be below the new holder and must prohibit the South-to-West turn. If the bead moves downward, the current holder must prohibit the East-to-South turn. To prevent the bead from being taken by other nodes, the current holder applies a resistance force to bead movement. To prevent upward movement, this resistance is downward and reflects the current holder' s demand for the South-to-West turn. To prevent downward movement, the resistance is upward and reflects the current holder' s demand for the East-to-South turn. The upward force, downward force, and current node' s resistance combine to form a resultant force that determines bead movement direction. To prevent bead oscillation, we set a threshold (Th); the bead moves only when the resultant force exceeds this threshold. For counterclockwise bead movement, the upward force reflects the North neighbor' s demand for the North-to-West turn, while the downward force reflects the South neighbor' s demand for the East-to-North turn.

3.4.2 Tug-War Routing Algorithm In the arm-wrestling algorithm described above, only three nodes participate in determining bead movement direction: the current bead holder, its North neighbor, and its South neighbor. The demands for turns from nodes farther from the bead holder are not considered. To address this limitation, the tug-war algorithm divides nodes above and below the current bead holder into two groups. To move the clockwise bead upward, the total demand for East-to-South turns from all nodes above the holder is treated as the upward force Fup. To pull the clockwise bead down-

ward, the total demand for South-to-West turns from all nodes below the holder is treated as the downward force Fdown. Similar to arm-wrestling, the current bead holder also applies resistance to prevent bead movement. To emphasize the importance of nodes closer to the bead, the "demand" is halved each time it passes through a node during propagation.

3.5 Experimental Evaluation

This section evaluates the reconfigurable routing algorithm based on the Abacus Turn Model by comparing it with commonly used routing algorithms. The Abacus Turn Model provides a "safe" method for dynamic generation and reconfiguration of deadlock-free routing. Here, "safe" means that no deadlocks are introduced during algorithm generation or reconfiguration. Therefore, we select reference routing algorithms that also address the "safety" issue. Compared algorithms include a deterministic routing algorithm: XY routing; two partially adaptive routing algorithms: West-First [19] and Odd-Even [20]; and a minimal-path fully adaptive routing algorithm [15]. Recent routing algorithms such as CQR [30], O1Turn [31], and RCA [32] address load balancing and are not included in the comparison.

Adaptive routing algorithms may produce two candidate output ports. In such cases, the output port with more available buffer credits is selected. Each router contains five input and output ports. Except for [15], all routers assume one virtual channel per virtual network with an input buffer depth of 4 flits per virtual channel. In [15], each virtual network contains 2 virtual channels; for fair comparison, each virtual channel is allocated only a 2-flit-deep input buffer. Additionally, [15] does not allow a virtual channel to be reallocated until the tail flit of the previous packet has left the virtual channel. For other routing algorithms, a virtual channel can be reallocated as soon as it receives the tail flit of the previous packet, regardless of whether that flit has departed.

This subsection first evaluates routing algorithm performance under synthetic traffic patterns, including uniform, transpose, and hotspot. Simulations initially assume a 4×4 mesh network, then repeat on an 8×8 mesh network to demonstrate scalability. All routing algorithms are implemented in a cycle-accurate open-source simulator, Garnet [33]. Garnet provides two simulation modes: flexible and detailed. We use the detailed mode because it allows modification of routing structures. In these experiments, each router implements one virtual network.

Subsequently, we evaluate routing algorithms under application traffic using trace-driven simulation. Application traces are obtained from the open-source full-system simulator GEMS [34]. GEMS adds timing simulation for memory and processor modules to the commercial full-system simulator Simics [35] through the Ruby and Opal modules. Our experiments load only the Ruby module and use the Garnet network within Ruby. The cache coherence protocol used is MSI_MOSI_CMP_directory, which requires five virtual networks

to resolve protocol deadlocks. Two of these virtual networks require in-order packet delivery. To reduce simulation time, we use a 4×4 mesh network in these experiments. The benchmark suites are SPLASH-2 [36] and PARSEC [37].

3.5.1 Network Performance Under Synthetic Traffic Under uniform traffic, each node sends packets to other nodes with equal probability. Simulation results for 4×4 and 8×8 mesh networks are shown in Figure 9. The horizontal axis represents flit injection rate, and the vertical axis represents average packet latency (in router clock cycles). Due to its inherent balance, uniform traffic generally yields better performance with deterministic routing than with adaptive routing. This occurs primarily because adaptive routing algorithms often make decisions based on local information, a short-sighted strategy that typically degrades network performance.

As shown in Figure 9(a), XY routing achieves the best performance, while the two partially adaptive routing algorithms perform similarly. The reconfigurable routing algorithm based on the Abacus Turn Model performs worse than partially adaptive algorithms under uniform traffic. This is mainly because reconfigurable routing always reconstructs based on historical information. This reconstruction strategy is often incorrect for uniform traffic. For example, if Northeast-bound packets have the highest load in the current period, the reconfigurable algorithm will allocate more routing adaptivity to Northeast-bound packets in the next period. However, by definition of uniform traffic, the number of Northeast-bound packets in the next period will likely be very small. The minimal-path fully adaptive routing algorithm performs worst in this experiment for two reasons: (1) it has the least input buffer space, and (2) it uses conservative flow control.

To demonstrate scalability, we repeat the experiment on an 8×8 mesh network. Results are similar to those on the 4×4 network, but the relative performance of the fully adaptive routing algorithm improves. This is mainly because the probability of conflicts increases with network scale. When conflicts occur, [15] requires packets to wait in escape channels. Since escape channels use XY routing, the fully adaptive routing algorithm can also benefit from uniform traffic balance like XY routing.

In the real world, most applications generate non-uniform traffic, such as transpose traffic. In transpose traffic, source node s always sends packets to destination node d, where d = transpose(s). As shown in Figure 10(a), under transpose traffic, the reconfigurable routing algorithm based on the Abacus Turn Model achieves the best network performance because it can provide full adaptivity for all packets.

Transpose traffic easily causes severe network congestion, so XY routing performs worst. West-First routing achieves better performance than XY because it provides full adaptivity for Eastbound packets, but its improvement is lim-

ited because Westbound packets still suffer severe congestion. Odd-Even routing provides relatively balanced adaptivity for all directions, achieving better performance than West-First. However, because its adaptivity is incomplete, it cannot match the performance of the Abacus Turn Model-based reconfigurable routing algorithm. The fully adaptive routing algorithm also provides full adaptivity, but due to smaller buffers and conservative flow control, its performance is lower than that of the Abacus Turn Model-based reconfigurable routing algorithm. Figure 10(b) shows simulation results on an 8×8 mesh network. While the relative performance between routing algorithms remains unchanged, the performance gap between our proposed algorithm and existing algorithms widens, primarily because our algorithm better handles congestion in large-scale networks.

Bursty traffic in applications can create network hotspots, exacerbating congestion. The next two simulations assume four hotspot nodes: nodes 0, 4, 8, and 12. These hotspots have a 20% higher probability of receiving packets than other nodes. We select these four nodes to simulate frequently accessed memory controllers. In this scenario, Westbound packets easily become congested. Simulation results in Figure 11(a) show that the Abacus Turn Model-based routing algorithm achieves the best performance because it provides full adaptivity for all packets. The fully adaptive routing algorithm, due to smaller buffer space and conservative flow control, performs worse than the Abacus Turn Model-based reconfigurable routing algorithm and even worse than Odd-Even routing. West-First and XY routing perform worst because they provide no adaptivity. When network scale increases, congestion becomes more severe. As shown in Figure 11(b), the performance advantage of the Abacus Turn Model-based reconfigurable routing algorithm becomes more pronounced.

3.5.2 Network Performance Under Application Traffic In Figure 12(a), all routing algorithms' average network latency on the SPLASH-2 benchmark suite is normalized to XY routing' s latency. The Abacus Turn Model-based arm-wrestling and tug-war algorithms significantly reduce average network latency by dynamically allocating more adaptivity to bursty traffic. Overall, our two proposed algorithms achieve varying degrees of performance improvement for all applications in the benchmark suite. However, the degree of improvement varies by application type. For applications that easily create conflicts, such as FFT and water-spatial, network performance improvements are significant. For low-conflict applications like raytrace and ocean, improvements are relatively small. For example, ocean generates over 60% local traffic, where arm-wrestling and tug-war achieve only 6% and 7% improvements, respectively. Overall, for SPLASH-2, our algorithms achieve an average 10% network performance improvement, with a maximum of 19%.

To increase confidence in these results, we repeat the experiments on the PARSEC benchmark suite. Simulation results in Figure 12(b) similarly normalize average packet latency to XY routing. The Abacus Turn Model-based reconfig-

urable routing algorithms also significantly improve network performance on PARSEC. Performance improvements are particularly substantial for highly congested applications like canneal (a cache-aware annealing algorithm for chip layout optimization) and freqmine (frequent itemset mining). For freqmine, tug-war reduces average latency by 15%, while arm-wrestling reduces it by 12%. For low-conflict applications like streamcluster (online clustering of input streams), improvements are smaller: arm-wrestling and tug-war reduce average packet latency by 2% and 3%, respectively. Overall, for PARSEC, our reconfigurable routing algorithms reduce packet access latency by up to 15% and an average of 10%.

[Figure 12: see original paper] Average packet latency under application traffic

4 Summary

This paper introduces three traffic-aware reconfigurable routing algorithms: two offline and one online. Offline algorithms use pre-collected application communication characteristics to determine routing rules, achieving the goal of balancing network traffic and improving network performance. This approach benefits from using global traffic information for system optimization but can only be used in application-specific systems. In general-purpose processor systems, where application traffic characteristics cannot be collected in advance, only online reconfigurable routing algorithms can be used. The reconfigurable routing algorithm based on the Abacus Turn Model achieves on-demand allocation of routing adaptivity by dynamically changing prohibited turns at each node. Since packets in heavily loaded directions can dynamically obtain more routing adaptivity for traffic balancing, network performance can be significantly improved. Simulation experiments under non-uniform synthetic traffic, SPLASH-2, and PARSEC benchmark suites demonstrate that compared with existing routing algorithms, the Abacus Turn Model-based reconfigurable routing algorithm achieves noticeable performance improvements.

References

[1] D. Geer, “Chip makers turn to multicore processors” , Computer, vol. 38, no. 5, pp: 11-13, May, 2003. [2] W.J. Dally and B. Towles, “Route packets, not wires: on-chip interconnection networks” , in Proceedings of Design Automation Conference, pp: 684-689, 2001. [3] W.J. Dally and B. Towles, “Principles and practices of interconnection networks” , Morgan Kaufmann, 2004. [4] M.B. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffman, P. Johnson, Lee Jae-Wook, W. Lee, A. Ma, A. Saraf, M. Seneski, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, and A. Agarwal, “The Raw microprocessor: a computational fabric for software circuits and general-purpose programs” , IEEE Micro, vol. 22, no. 2, pp: 25-35, 2002. [5] 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, 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, Jan. 2008. [6] D.R. Fan, N. Yuan, J. C. Zhang, Y. B. Zhou, W. Lin, F.L. Song, X. C. Ye, H. Huang, L. Yu, G. P. Long, H. Zhang, and L. Liu, "Godson-T: An Efficient Many-Core Architecture for Parallel Program Executions" , Journal of Computer Science and Technology, vol. 24, no. 6, 2009. [7] Jongman Kim, Dongkook Park, T. Theocharides, N. Vijaykrishnan, C.R. Das, "A low latency router supporting adaptivity for on-chip interconnects," in Proceedings of Design Automation Conference, pp. 559-564, 2005. [8] A. Singh, W.J. Dally, A.K. Gupta, B. Towles, "GOAL: a load-balanced adaptive routing algorithm for torus networks," in Proceedings of International Symposium on Computer Architecture, pp. 194-205, 2003. [9] J. Brand, C. Ciordas, K. Goossens, T. Basten, "Congestion-Controlled Best-Effort Communication for Networks-on-Chip," Design, Automation & Test in Europe Conference & Exhibition, pp.1-6, 2007. [10] J. Duato, I. Johnson, J. Flich, F. Naven, P. Garcia, T. Nachiondo, "A new scalable and cost-effective congestion management strategy for lossless multistage interconnection networks," International Symposium on High-Performance Computer Architecture, pp. 108-119, 2005. [11] P. Gratz, B. Grot, S.W. Keckler, "Regional congestion awareness for load balance in networks-on-chip," in Proceedings of High Performance Computer Architecture, pp.203-214, 2008. [12] D. Park, R. Das, C. Nicopoulos, J. Kim, N. Vijaykrishnan, R. Iyer, C.R. Das, "Design of a Dynamic Priority-Based Fast Path Architecture for On-Chip Interconnects," in Proceedings of IEEE Symposium on High-Performance Interconnects, pp.15-20, 2007. [13] S.A. Felperin, L. Gravano, G.D. Pifarre, J. Sanz, "Fully-adaptive routing: packet switching performance and wormhole algorithms," in Proceedings of the ACM/IEEE Conference on Supercomputing, pp.654-663, 1991. [14] W.J. Dally, H. Aoki, "Deadlock-free adaptive routing in multicomputer networks using virtual channels" , IEEE Transactions on Parallel and Distributed Systems, vol. 4, no. 4, pp: 466-475, Apr. 1993. [15] J. Duato, "A new theory of deadlock-free adaptive routing in wormhole networks" , IEEE Transactions on Parallel and Distributed Systems, vol. 4, no. 12, Dec. 1993. [16] Y. M. Boura, C.R. Das, "Efficient fully adaptive wormhole routing in n-dimensional meshes" , in Proceedings of International Conference on Distributed Computing Systems, pp: 589-596, 1994. [17] J.H. Upadhyay, V. Varavithya, P. Mohapatra, "Efficient and balanced adaptive routing in two-dimensional meshes" , in Proceedings of IEEE Symposium on High-Performance Computer Architecture, pp: 112 -121, 1995. [18] L.S. Peh and W.J. Dally, "A delay model and speculative architecture for pipelined routers", in Proceedings of International Symposium on High-Performance Computer Architecture, pp: 255 -266, 2001. [19] C.J. Glass and L.M. Ni, "The Turn Model for Adaptive Routing" , in Proceedings of International Symposium on Computer Architecture, pp: 278-287, 1992. [20] G.M. Chiu, "The odd-even turn model for adaptive routing" , IEEE Transactions on Parallel and Distributed Systems, vol. 11, no. 7, pp: 729-738, Jul. 2000. [21] M. Palesi, R. Holsmark, S. Kumar, and V. Catania, "Application Specific Routing Algorithms for Networks on Chip" , IEEE Transactions on Parallel and Distributed Systems, vol. 20, no. 3, Mar. 2009. [22] M.A. Kinsy, M.H. Cho, T. Wen, E. Suh, M. van Dijk, and

S. Devadas, “Application-aware deadlock-free oblivious routing” , in Proceedings of international symposium on Computer architecture (ISCA’ 09), pp: 208 –219, 2009. [23] B. Fu, Y. Han, J. Ma, H. Li, and X. Li, “An abacus turn model for time/space-efficient reconfigurable routing” , in Proceedings of international symposium on Computer architecture (ISCA’ 11), pp: 259-270, 2011. [24] T.L. Rodeheffer and M.D. Schroeder, “Automatic reconfiguration in Autonet” , in Proceedings of ACM symposium on Operating systems principles, pp: 183-197, 1991. [25] O. Lysne and J. Duato, “Fast dynamic reconfiguration in irregular networks” , in Proceedings of International Conference on Parallel Processing, pp: 449-458, 2000. [26] R. Casado, A. Bermudez, F.J. Quiles, J.L. Sanchez, J. Duato, “Performance evaluation of dynamic reconfiguration in high-speed local area networks” , in Proceedings of International Symposium on High-Performance Computer Architecture, pp: 85-96, 2000. [27] R. Casado, A. Bermudez, J. Duato, F.J. Quiles, J.L. Sanchez, “A protocol for deadlock-free dynamic reconfiguration in high-speed local area networks” , IEEE Transactions on Parallel and Distributed Systems, vol. 12, no.2, pp: 115-132, Feb. 2001. [28] D. Avresky and N. Natchev, “Dynamic reconfiguration in computer clusters with irregular topologies in the presence of multiple node and link failures” , IEEE Transactions on Computers, vol. 54, no. 5, pp: 603-615, May 2005. [29] O. Lysne, J.M. Montanana, J. Flich, J. Duato, T. M. Pinkston, T. Skeie, “An efficient and deadlock-free network reconfiguration protocol” , IEEE Transactions on Computers, vol. 57, no. 6, pp:762-779, June 2008. [30] A. Singh, W.J. Dally, A.K. Gupta, and B. Towles, “Adaptive channel queue routing on k-ary n-cubes” , in Proceedings of ACM symposium on Parallelism in Algorithms and Architectures, pp: 11-19, 2004. [31] D. Seo, A. Ali, W.T. Lim, N. Rafique, and M. Thottethodi, “Near-Optimal Worst-Case Throughput Routing for Two-Dimensional Mesh Networks” , in Proceedings of international symposium on Computer Architecture, pp: 432-443, 2005. [32] P. Gratz, B. Grot, S.W. Keckler, “Regional congestion awareness for networks-on-chip” , in Proceedings of IEEE International Symposium on High Performance Computer Architecture, pp:203-214, 2008. [33] N. Agarwal, L.S. Peh, N. Jha, “Garnet: A detailed interconnection network model inside a full-system simulation framework” , TR CE-P08-001, Princeton University, 2007. [34] M.M.K. Martin, D.J. Sorin, B.M. Beckmann, M.R. Marty, M. Xu, A.R. Alameldeen, K.E. Moore, M.D. Hill, D.A. Wood, “Multifacet’s general execution-driven multiprocessor simulator (GEMS) toolset” , ACM SIGARCH Computer Architecture News, vol. 33, no. 4, pp: 92-99, 2005. [35] P.S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner, “Simics: A full system simulation platform” , Computer, vol. 35, no.2, pp: 50-58, 2002. [36] S.C. Woo, M. Ohara, E. Torrie, J.P. Singh, and A. Gupta, “The SPLASH-2 programs: characterization and methodological considerations” , in Proceedings of International Symposium on Computer Architecture, pp:24-36, 1995. [37] C. Bienia, S. Kumar, J.P. Singh, and K. Li, “The PARSEC benchmark suite: Characterization and architectural implications” , in Proceedings of International Conference on Parallel Architectures and Compilation Techniques, pp: 72-81, 2008.