Traffic-Aware Reconfigurable Routing Algorithm Postprint

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

Date: 2017-03-10T00:00:00+00:00

Abstract

In many-core processor systems, Networks-on-Chip (NoC) are commonly employed to provide high-bandwidth, low-latency, and highly reliable on-chip communication. To reduce network congestion and improve network performance, load-balanced routing algorithms have garnered extensive attention from researchers. Load-balanced algorithms typically utilize fully adaptive routing algorithms to provide path diversity; however, current fully adaptive routing algorithms either require a large number of virtual channels or assume a conservative flow control strategy. On the one hand, virtual channels are relatively expensive resources; on the other hand, conservative flow control strategies may lead to degradation in network performance. Therefore, researchers have proposed leveraging application traffic information to enhance routing performance. These algorithms can be reconfigured based on different traffic characteristics without using virtual channels, thereby achieving on-demand allocation of routing adaptivity. According to the type of traffic information utilized, traffic-aware reconfigurable routing algorithms can be classified into offline and online algorithms. Offline algorithms require prior knowledge of application traffic characteristics, and thus they are mostly targeted at application-specific multi-core Systems-on-Chip. Online algorithms, on the other hand, perform reconfiguration based on traffic information collected online, and therefore can be applied to general-purpose processor systems. This paper will discuss two notable offline algorithms recently proposed internationally, and will focus on introducing the online reconfigurable routing algorithm based on the Abacus Turn Model, which was published by the authors at the 2011 International Symposium on Computer Architecture (ISCA' 11).

Full Text

Preamble

Vol. 9 No. 6 Information Technology Letter Nov. 2011

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, traffic-balancing routing algorithms have attracted widespread attention from researchers. These algorithms typically leverage fully adaptive routing to provide path diversity, yet existing fully adaptive routing algorithms either require numerous virtual channels or assume a conservative flow control strategy. On one hand, virtual channels are relatively expensive resources; on the other hand, conservative flow control strategies may degrade network performance. Consequently, researchers have proposed utilizing application traffic information to enhance routing performance. These 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 and online algorithms. Offline algorithms require prior knowledge of program traffic characteristics and are thus primarily targeted at application-specific multi-core SoCs. Online algorithms, by contrast, reconfigure themselves based on traffic information collected at runtime, making them suitable for general-purpose processor systems.

This paper discusses two recently proposed prominent offline algorithms and focuses on introducing an online reconfigurable routing algorithm based on the abacus turn model, which was published by the authors at the 2011 International Symposium on Computer Architecture (ISCA' 11).

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

1 Introduction

Due to constraints such as the memory wall, ILP wall, and power wall, traditional approaches relying on single-processor performance improvements no longer yield satisfactory results. Multi-core and many-core processors that enhance application performance through parallelization are now considered viable solutions [1]. In many-core processor systems, Networks-on-Chip (NoC) are widely expected to become the mainstream interconnect solution due to their 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 requires comprehensive consideration of various factors, includ-

ing port count, bandwidth per port, wiring density allowed by the fabrication process, 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 topology' s planar structure facilitates manufacturing 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 the 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 mechanisms divide packets into multiple flow control digits (flits), comprising 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 systems 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 algorithms. Minimal-path routing requires each hop to select a direction that brings the packet closer to its destination. This approach offers simple, regular rules and prevents livelock in the network. Non-minimal-path algorithms, 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 relatively simple implementations but lower tolerance to network 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, whereas partially adaptive routing permits only a subset of paths.

Since sufficient path diversity is needed to alleviate congestion, most traffic-balancing routing algorithms currently utilize a fully adaptive routing algorithm to provide candidate output ports [7][8][9][10][11][12]. However, existing 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 large numbers of virtual channels, such as those in [13][14], are not suitable for NoC. Routing algorithms based on Duato' s theory require fewer virtual channels but demand conservative flow control strategies. According to Duato' s theory, router buffer queues are not allowed to store flits belonging to 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 experiences congestion, its head flit must be at the front of the buffer queue, enabling it to select an escape channel. However, this leads to wasted 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 proposed by Glass and Ni [19] and the odd-even turn model proposed by Chiu [20]. The turn model can design partially adaptive routing algorithms without virtual channels by dividing all possible turns in the network into two abstract cycles: clockwise and counterclockwise. By prohibiting one turn in each abstract cycle, deadlock can be prevented. However, Chiu discovered that the adaptivity provided by turn model-based routing algorithms is non-uniform. To address this issue, he proposed the odd-even turn model. Nevertheless, the odd-even turn model cannot provide full adaptivity for any node pair with 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 using reconfigurable routing algorithms to provide on-demand routing adaptivity for applications. Overall, reconfigurable routing algorithms are partially adaptive, thus requiring neither virtual channels nor conservative flow control strategies. These algorithms can analyze application traffic characteristics to provide greater path diversity for packets traveling in high-load directions.

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

Compared with offline algorithms, online reconfigurable routing algorithms can reconfigure themselves based on traffic information collected online, making them applicable to general-purpose processor systems. This paper introduces a reconfigurable routing algorithm based on the abacus turn model, published 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 how to maximize network performance improvement while ensuring deadlock-free routing. Most offline algorithms follow this basic approach: first construct a channel dependency graph, then ensure deadlock-free routing 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 both follow this fundamental approach but with different emphases.

2.1 Application-Specific Routing Algorithms [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 into 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-free routing by guaranteeing the graph is acyclic. Additionally, during cycle removal, it enhances routing performance by balancing traffic across channels.

We use an example from [21] to illustrate its basic working principle; detailed algorithms and implementation specifics can be found in [21]. As shown in Figure 1(a), the communication graph indicates that the application contains six tasks. Communications between tasks are 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 the routing algorithm is fully adaptive minimal-path routing. The resulting channel dependency graph is shown in Figure 1(c). Since this channel dependency graph contains six dependency cycles, according to Dally' s theory [14], the network cannot use fully adaptive routing. However, not all tasks actually communicate, so the application channel dependency graph after removing non-existent channel 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, but according to the communication graph, none of these communications exist. Therefore, dependency l12→l23 will never actually occur 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 prevents 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 will never form in the actual network. Therefore, we need not remove these cycles, and the resulting routing algorithm can remain fully adaptive.

2.2 Application-Aware Oblivious Routing Algorithms [22]

Unlike application-specific routing algorithms, [22] argues that adaptive routing algorithms increase implementation complexity and therefore oblivious routing should be used instead. To address characteristics of different applications, [22] proposes an application-aware oblivious routing generation flow consisting of five main steps:

1. Select a turn model routing algorithm
2. Build a flow graph
3. Select paths for each communication flow
4. Check if all flows are assigned paths and if network status meets optimization goals
5. Select the best path set

Step 1: Using a turn model routing algorithm to construct the channel dependency graph ensures it is cycle-free, thereby guaranteeing the generated oblivious routing algorithm is deadlock-free. In fact, searching for and removing all cycles in a directed graph is complex work, so [22] proposes using the turn model [19] to construct acyclic channel dependency graphs. For example, assuming the "north-last" routing algorithm [19] is used, the resulting channel dependency graph is guaranteed to be cycle-free. Readers can refer to [14] for methods to construct channel dependency graphs. For the 3×3 mesh network shown in Figure 2(a), its 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 routing, the packet can travel along path E→D→G, so channel ED depends on channel DG, represented by an edge from ED to DG in Figure 2(b). Clearly, no cycles exist in Figure 2(b).

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 final flow graph should contain all communication flows present in the application.

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] abstracts the path selection problem 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 in sequence using a weighted Dijkstra shortest-path algorithm.

Step 4: Check whether all communication flows have been assigned paths and whether the network status 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. Detailed content can be found in [22].

3 Online Reconfigurable Routing Algorithms

This section discusses the abacus turn model and corresponding reconfigurable routing algorithm proposed in [23].

3.1 Basic Idea of the Abacus Turn Model

The original turn model has proven that if all allowed turns in a network do not form abstract cycles, the network is deadlock-free [19]. [20] further proved that if neither the rightmost clockwise nor rightmost counterclockwise cycles exist in the network, the network is deadlock-free. 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 the south-to-west turn (Figure 3(b)) and all nodes in even columns to prohibit the east-to-south turn (Figure 3(c)). The principle for prohibiting counterclockwise rightmost cycles is similar and is omitted here.

[23] inherits the core idea from [20] that "if a network has no clockwise or counterclockwise rightmost cycles, it is deadlock-free." Unlike the original turn model, [23] employs a more flexible method to remove rightmost cycles. As shown in Figure 3(a), to form a rightmost clockwise cycle, an east-to-south turn must exist above a south-to-west turn. Therefore, rightmost clockwise cycles can be eliminated by prohibiting east-to-south turns above all south-to-west turns

(Figure 3(d)). After adopting this approach, each column in the network must contain a point above which all east-to-south turns are prohibited and below which all south-to-west turns are prohibited. [23] refers to this type of node as a clockwise bead node. Similarly, each column also has a counterclockwise bead node to separate counterclockwise turns, thereby preventing 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.

The abacus turn model can be defined by three rules: 1. Each column has exactly one clockwise bead node and one counterclockwise bead node 2. Clockwise (respectively, counterclockwise) bead nodes do not prohibit clockwise (respectively, counterclockwise) turns 3. The distribution of prohibited turns follows the pattern described below

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 can allow all clockwise (respectively, counterclockwise) turns. All routers above a 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 will not form clockwise or counterclockwise rightmost cycles. Therefore, designing a deadlock-free routing algorithm 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 can be in k possible positions, there are k×k configurations in one column. 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 the reconfigurable routing algorithm 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 duration. Since only one minimal path is available for this node pair, that path can easily become congested. To obtain more usable paths for traffic balancing, node 6 complains to its east neighbor node 7 about the current situation—specifically, that node 7 is prohibiting the east-to-south turn that node 6 needs. 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 the bead should be passed. In this example, node 1 will pass 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. Node 6 then has two paths to node 5 for balancing traffic. Similar to node 6, node 7 also complains to its neighbor node 8 because node 8 similarly 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 provided to packets for traffic balancing.

3.3 Safe Bead Passing

Ensuring deadlock-free properties 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 at this time (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, those 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 in nodes 1 and 4 are still using the south-to-west turn—the network may develop a clockwise rightmost cycle, violating the abacus turn model rules.

To address these issues, bead movement must satisfy two rules: 1. Before a node prohibits a turn, it must ensure no packets will use that turn 2. If it is certain that no packets are 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 allowing 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 node by only one hop in each 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 neighbor interactions, 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 the scenario shown 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 to use 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 the west output port to node 3. However, node 4 cannot send an acknowledgment to node 1 at this point because there may still be packets that have completed the routing stage and were routed before node 1 decided to prohibit the south-to-west turn. Therefore, before sending acknowledgment to node 1, node 4 must also drain any southwest-bound packets within itself 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 idea 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 local, east, and north input ports, reconfiguration tokens need only be inserted at the end 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 that could be routed to node 1 remain within itself. Node 4 can then send acknowledgment to node 1.

After receiving acknowledgment, node 1 can be confident that its north input port will no longer receive new southwest-bound packets. However, southwest-bound packets may still exist in the buffer queue of the 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. After this, 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. As shown in Figure 7, operations for moving clockwise and counterclockwise beads are summarized 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 to the upward movement of the clockwise bead discussed above, differing only in the involved turns.

3.4 Reconfigurable Routing Algorithm Based on the Abacus Turn Model

The combination of the abacus turn model and bead-passing atomic operations simplifies 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 algorithms. 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 allowing another turn from the same group at the new bead owner. Therefore, the demand for different turns naturally becomes the basis for determining bead movement direction.

To record demand for different turns, each node requires 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 the east-to-south turn, the larger the upward force. Simultaneously, the south neighbor of the bead holder also applies a force to the block 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 bead holder will be located below the new bead holder and must prohibit the south-to-west turn. If the bead moves downward, the current bead 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 force is downward and reflects the current holder' s demand for the south-to-west turn. To prevent downward movement, the resistance force 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 together form a resultant force that determines bead movement direction. To prevent bead oscillation, we set a threshold (Th); the bead only moves when the resultant force exceeds this threshold. For moving the counterclockwise bead, 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 of all nodes above the holder for the east-to-south turn is treated as the upward force Fup. To pull the clockwise bead downward, the total demand of all nodes below the holder for the south-to-west turn 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 transfer.

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, where "safe" means that no deadlocks are introduced during algorithm generation and reconfiguration. Therefore, we select reference routing algorithms that also address the "safety" problem. 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]. Recently proposed routing algorithms such as CQR [30], O1Turn [31], and RCA [32] target traffic balancing and are not included in this 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 are assumed to have one virtual channel per virtual network, with each virtual channel containing an input buffer queue of depth 4. In [15], each virtual network contains 2 virtual channels; for fair comparison, each virtual channel is allocated only an input buffer queue of depth 2. 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 left the virtual channel.

This subsection first evaluates routing algorithm performance under synthetic traffic patterns, including uniform, transpose, and hotspot traffic. 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 called Garnet [33]. Garnet provides two simulation modes: flexible and detailed. This simulation uses the detailed mode because it provides the ability to modify routing structures. In these experi-

ments, each router implements one virtual network.

Subsequently, this subsection evaluates routing algorithms for application traffic using trace-driven simulation. Application traces are obtained from the open-source full-system simulator GEMS [34]. GEMS adds timing simulation of memory and processor modules to the commercial full-system simulator Simics [35] through 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 5 virtual networks to resolve protocol deadlocks. Among these 5 virtual networks, 2 require in-order packet delivery. To reduce simulation time, a 4×4 mesh network is used in these experiments. The benchmark suites are SPLASH-2 [36] and PARSEC [37].

3.5.1 Network Performance under Synthetic Traffic In 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). Because uniform traffic is inherently balanced, deterministic routing algorithms generally achieve better performance than adaptive routing algorithms. This occurs primarily because adaptive routing algorithms often make routing decisions based on local information, which is a shortsighted 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 exhibit similar network performance. The reconfigurable routing algorithm based on the abacus turn model performs worse than partially adaptive routing algorithms in this scenario. This is mainly because the reconfigurable routing algorithm always reconfigures routing based on historical information. However, this reconfiguration strategy is often incorrect under uniform traffic. For example, if northeast-bound packets have the highest load in the current time period, the reconfigurable routing algorithm will allocate more routing adaptivity to northeast-bound packets in the next period. Yet 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 a conservative flow control mechanism.

To demonstrate scalability, we repeated the same experiment on an 8×8 mesh network. The results are similar to those in 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 trans-

pose traffic. In transpose traffic, source node s always sends packets to destination node d, where d = (s + b/2) mod b, and b is the bit length needed to index all nodes. 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 routing because it provides full adaptivity for eastbound packets, but its improvement is limited because westbound packets still suffer severe congestion. Odd-even routing provides relatively balanced adaptivity for packets in all directions, achieving better performance than west-first, but 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 its smaller buffers and conservative flow control mechanism, its performance is lower than that of the abacus turn model-based reconfigurable routing algorithm. Figure 10(b) shows simulation results for the same experiment on an 8×8 mesh network. Although 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 hotspots: nodes 0, 4, 8, and 12. These four nodes 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 shown in Figure 11(a) indicate 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 also provides full adaptivity, but due to its smaller buffer space and conservative flow control mechanism, its performance is inferior to 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 for Application Traffic In Figure 12(a), all routing algorithms' average network latency on the SPLASH-2 benchmark suite is normalized to the latency of XY routing. The arm-wrestling and tug-war algorithms based on the abacus turn model 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 for different application types. 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, performance improvements are relatively small. For example, ocean generates over 60% local traffic, resulting in only 6% and 7% performance improvements from arm-wrestling and tug-war algorithms, respectively. Overall, for the SPLASH-2 benchmark suite, our proposed algorithms achieve an average 10% network performance improvement, with a maximum improvement of 19%.

To increase confidence in these results, we repeated the experiments on the PARSEC benchmark suite. Simulation results are shown in Figure 12(b), where average packet latency is also normalized to XY routing. Similar to the previous results, the abacus turn model-based reconfigurable routing algorithms also significantly improve network performance on the PARSEC benchmark suite. Performance improvements are particularly substantial for highly congested applications like canneal (a cache-aware simulated annealing algorithm for chip layout and routing optimization) and freqmine (frequent itemset mining). For example, for freqmine, the tug-war algorithm reduces average latency by 15%, while the arm-wrestling algorithm reduces it by 12%. For low-conflict applications like streamcluster (online clustering of streaming inputs), performance improvements are relatively small. For streamcluster, arm-wrestling and tug-war algorithms reduce average packet latency by 2% and 3%, respectively. Overall, for the PARSEC benchmark suite, our proposed reconfigurable routing algorithms can reduce packet access latency by up to 15% and by an average of 10%.

4 Summary

This paper introduces three traffic-aware reconfigurable routing algorithms: two offline reconfigurable algorithms and one online reconfigurable algorithm. Offline algorithms utilize pre-collected application communication characteristics to determine routing rules, thereby 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 can dynamically change prohibited turns at each node to achieve on-demand allocation of routing adaptivity. Since packets in high-load directions can dynamically obtain more routing adaptivity for traffic balancing, network performance can be significantly improved. Simulation experiments under non-uniform synthetic traffic and SPLASH-2 and PARSEC benchmark suites demonstrate that the abacus turn model-based reconfigurable routing algorithm achieves noticeable performance improvements over existing routing algorithms.

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 multi-stage 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 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.

Note: Figure translations are in progress. See original paper for figures.