J Supercomput (2015) 71:3419–3439
DOI 10.1007/s11227-015-1439-3

# Per-packet global congestion estimation for fast packet delivery in networks-on-chip

**Pejman Lotfi-Kamran**[1]

Published online: 10 May 2015

**Abstract** Networks-on-chip (NOCs) are becoming the de facto communication fabric to connect cores and cache banks in chip multiprocessors (CMPs). Routing algorithms, as one of the key components that influence NOC latency, are the subject of extensive research. Static routing algorithms have low cost but unlike adaptive routing algorithms, do not perform well under non-uniform or bursty traffic. Adaptive routing algorithms estimate congestion levels of output ports to avoid routing traffic over congested ports. As global adaptive routing algorithms are not restricted to local information for congestion estimation, they are the prime candidates for balancing traffic in NOCs. Unfortunately, destinations of packets are not considered for congestion estimation in existing global adaptive routing algorithms. We will show that having identical congestion estimates for packets with different destinations prevents global adaptive routing algorithms from reaching their peak potential. In this work, we introduce *Fast*, a low-cost global adaptive routing algorithm that estimates congestion levels of output ports on a per-packet basis. The simulation results reveal that *Fast* achieves lower latency and higher throughput as compared to those of other adaptive routing algorithms across all workloads examined. *Fast* increases the throughput of an $8 \times 8$ network by 54, 30, and 16 % as compared to DOR, Local, and RCA on a synthetic traffic profile. On realistic benchmarks, *Fast* achieves 5 % average, and 12 % maximum latency reduction on SPLASH-2 benchmarks running on a 49-core CMP as compared to the state of the art.

**Keywords** Adaptive routing · Chip multiprocessor · Congestion estimation · Network-on-chip

✉ Pejman Lotfi-Kamran
plotfi@ipm.ir

1 School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran

## 1 Introduction

Today's information centric world is powered by processors. The unprecedented growth of processors' performance in the past 30 years has enabled many useful applications and services to emerge. Many of these applications and services have inherent parallelism [4,6,12,46]. Taking advantage of the parallelism in applications, and driven by the need to increase performance of processors, the industry and academia have introduced multi-core processors (e.g., [3,16,18,30,39,43]).

In light of known scalability limitations for crossbar- and bus-based designs, existing many-core chip multiprocessors (CMPs), such as Tilera's Tile series, have featured a network-on-chip (i.e., a mesh-based interconnect fabric) and a tiled organization. Typically, each tile integrates a core, a slice of the shared last-level cache (LLC) with directory, and a router. The resulting organization enables cost-effective scalability to high core counts.

In tiled processors, the network-on-chip is responsible to carry cores' requests for data and instructions to the LLC, and to bring back the responses to the requesting cores. As the delay of delivering instructions and data is critical to the performance of parallel applications, the design of a fast network-on-chip is the subject of many research activities [1,2,22,25,29,33,34,38,44,45].

One of the key components that has a major impact on the speed of a network-on-chip is routing algorithm [15,17,23,26,28,31,42]. At each router, the routing algorithm decides how to route incoming packets to the destinations. The simplest routing algorithm is dimension order routing (DOR) which statically routes packets along one of the dimensions before moving to the next one. Static routing algorithms are simple and have low complexity, but cannot adapt routing decisions to the traffic in the network, and hence, result in high network delay and low performance under non-uniform or bursty traffic.

To address the shortcomings of static routing algorithms, researchers have proposed adaptive routing algorithms. An adaptive routing algorithm associates a congestion estimate to each output port that a packet can be forwarded to. When a packet can be routed over more than one output port, an adaptive routing algorithm picks the port that has the lowest congestion.

A local adaptive routing algorithm uses metrics that are accessible in a router or in the neighboring routers (e.g., number of free virtual channels (VCs), number of free slots in the downstream buffer, etc.) to estimate congestion levels of output ports. Local adaptive routing algorithms reduce latency of the network over static routing algorithms. However, as they only observe the status of a small fraction of the network, local adaptive routing algorithms greedily rely on local information to route packets, and hence, frequently take suboptimal decisions [15].

Global adaptive routing algorithms benefit from global (or regional) metrics to estimate congestion levels of output ports, and hence, can take better decisions. Regional Congestion Awareness (RCA) [15], which is one of the state-of-the-art global adaptive routing algorithms, aggregates the congestion estimates of links in a region (e.g., a row or a column) of a NOC to estimate congestion levels of routers' output ports. While using an aggregated congestion estimate enables RCA to improve NOC performance over local adaptive routing algorithms, existing aggregation methods introduce noise

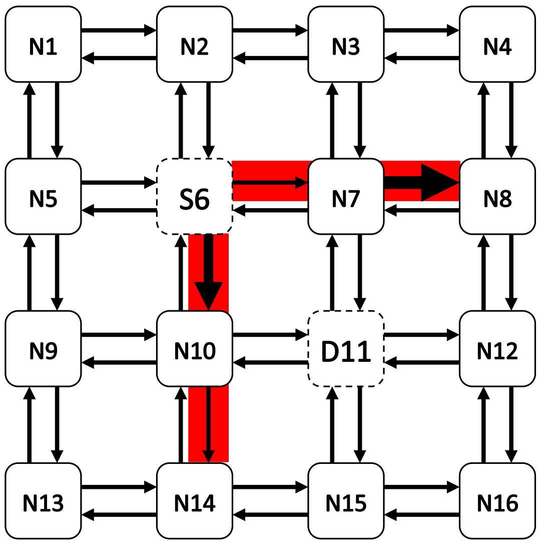

**Fig. 1** Illustration of the inefficiency of existing global adaptive routing algorithms in estimating the congestion levels of output ports. The thickness of a link in the *shaded area* is proportional to the traffic passing through it

into the congestion estimate, and prevent global adaptive routing algorithms to reach their peak potential.

Figure 1 illustrates the inefficiency of existing global adaptive routing algorithms in estimating congestion levels of routers' output ports. Assume that *Node 6* has a packet that needs to be sent to *Node 11*. *Node 6* can either forward the packet on EAST port to *Node 7*, or on SOUTH port to *Node 10*. A global adaptive routing algorithm combines the congestion status of the EAST port of *Node 6* and *7*, and the congestion status of the SOUTH port of *Node 6* and *10* to estimate congestion levels of the EAST and the SOUTH port, respectively. While the destination of the packet is *Node 11*, and definitely, this packet will not pass through the EAST port of *Node 7* or the SOUTH port of *Node 10*, nevertheless, the congestion status of these ports are part of the congestion estimates that *Node 6* will use to pick the favorite port. If one of these ports is congested (e.g., EAST port of *Node 7*), and consequently, contributes significantly to the congestion estimate, the global adaptive routing algorithm's decision will be suboptimal.

In this paper, we introduce *Fast*, a global adaptive routing algorithm that eliminates noise in the process of global congestion estimation. Unlike existing global routing algorithms that use an identical congestion estimate for all packets that may go through an output port at a given time, *Fast* estimates, on a per-packet basis, a congestion level for each port, taking into consideration the destinations of the packets. *Fast* uses a novel mechanism that requires only 1-bit of global information for each router in the row and column. We will show that *Fast* improves network latency by 5 %, on average, and 12 %, at best, over the state of the art on SPLASH-2 benchmarks, while requiring less hardware overhead.

The rest of this paper is organized as follows. Section 2 presents background information on networks-on-chip. Section 3 reviews existing routing algorithms, and Sect. 4 discusses why existing global adaptive routing algorithms cannot reach their peak

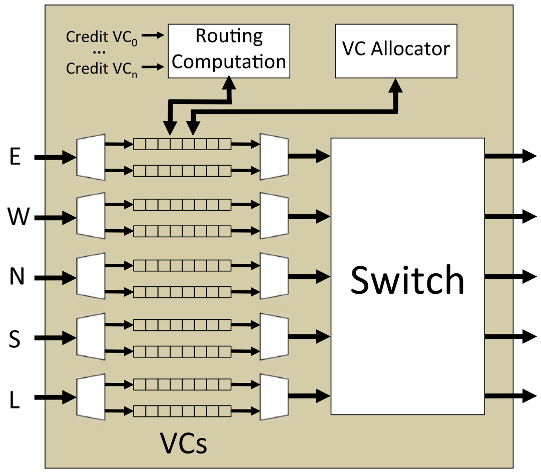

**Fig. 2** Organization of a mesh router

potential. Section 5 introduces *Fast* global adaptive routing algorithm. In Sect. 6, we evaluate our proposal for global adaptive routing. Finally, we conclude in Sect. 8.

## 2 Background

Except for the boundary routers, every router in a 2D mesh has five input/output ports (four connected to the neighboring routers and one for the local core). Main architectural elements of a wormhole router includes the input FIFO for each port, route computation unit, VC allocation unit, crossbar control logic, and the crossbar. The packet transmission is performed by dividing it to smaller pieces called flits. A flit enters into the router through one of the ports and is stored in its FIFO. If the flit is a header, indicating the start of a new packet, it proceeds to the routing unit, which determines the output port that the packet should use. Then, the header flit attempts to acquire a virtual channel for the next hop. Upon a successful VC allocation, the header flit enters the switch arbitration stage, where it competes for the output port with other flits from the other router input ports. Once the crossbar passage is granted, the flit traverses the switch and enters the channel. Subsequent flits belonging to the same packet can proceed directly to the crossbar and go to the output port. The organization of a router is shown in Fig. 2.

To eliminate the delay of routing from the critical path of a router, researchers proposed look-ahead routing, in which routing decisions are performed one hop in advance (i.e., For each packet, a router performs routing decision for the next hop, as the previous router has already performed the routing for this hop) [13]. The delay of the routing being off the critical path, researchers proposed dynamic routing methods, which are more complex, for better distribution of traffic in the network.

The main difference between a static and dynamic router is that, depending on the network condition, in a dynamic router, the route computation unit may select different paths at different times for the same source–destination pair. Various factors may be used for choosing an output port like (1) the number of free VCs, (2) the crossbar pressure, and (3) the number of free buffers at the corresponding input port in the downstream router. We use (2) to estimate network condition for adaptive routers as we experimentally verified that this metric works best across the range of workloads that we examined (Our results also corroborate prior work [15]).

Packets can be routed in both *X* and *Y* directions without restriction, adaptive routing algorithms need a mechanism to guarantee deadlock avoidance. In networks having virtual channels (general case), usually the following method, which is called virtual sub-network, is used to guarantee deadlock avoidance. Virtual channels in *Y* dimensions are divided into two parts. The network is partitioned into two sub-networks called +*X* sub-network and −*X* sub-network each having half of the channels in the Y dimension. If the destination node is to the right of the source, the packet will be routed through the +*X* sub-network. If the destination node is to the left of the source, the packet will be routed through the −*X* sub-network. Otherwise that packet can be routed using either sub-network [36].

In this work, we only consider minimal routing algorithms, as they are by far the most dominant method in the literature and real machine implementations. Investigating the usefulness of the techniques that we present in this work under non-minimal routing is part of our future work.

## 3 Existing routing algorithms

In this section, we review three main classes of routing algorithms: DOR, local adaptive routing, and global adaptive routing.

### 3.1 Dimension order routing (DOR)

The simplest type of routing is static routing, in which the path that a packet travels is determined only based on the source and the destination of the packet. One of the widely used static routing mechanism is DOR. Having a static order of the dimensions (e.g., *X*, *Y*, etc.), with DOR, packets are routed to the correct position in the highest-order dimension before moving to the next dimension.

DOR is easy to implement and has low complexity. However, as network conditions do not have any roles in the routing decisions, this routing algorithm is incapable of balancing non-uniform traffic in the NOC. Consequently, DOR leads to large network latency when traffic is bursty (e.g., traffic of a CMP), and hence, is not well suited for chip multiprocessors.

### 3.2 Local adaptive routing

To address the shortcomings of static routing algorithms, researchers proposed using adaptive routing algorithms. In adaptive routing, the path that a packet travels not only depends on the source and the destination of the packet, but also on the network conditions. For this purpose, adaptive routers estimate congestion level of each output port. When a packet can be routed over two ports, the port with a smaller congestion level is taken.

Local adaptive routing algorithms take advantage of metrics that are accessible in a router to estimate congestion level of an output port. Various metrics have been used in the literature for estimating congestion level of an output port. The number of free VCs [7], the number of free buffer slots in the downstream router [23], and the active demand that each output port experiences (i.e., crossbar demand) [15] are among such metrics. Corroborating prior work [15], we experimentally verify that the crossbar demand works best across all of the workloads that we examined.

A local adaptive routing algorithm distributes traffic just by observing the status of a single router in the downstream path. If a link or router is congested further away, a local adaptive routing algorithm has no knowledge of it, and hence, cannot avoid the congested link or router.

### 3.3 Global adaptive routing

Global adaptive routing algorithms address the shortcoming of local adaptive routing algorithms by estimating congestion level of a router's output port, not just by the status of the router itself, but also by taking into consideration the status of other routers. For this purpose, a global adaptive routing algorithm requires a monitoring network to send congestion status among routers. Every router aggregates (takes weighted average of) its local congestion estimate and that of downstream routers. The resulting global congestion estimate is being used for the output port selection, and also gets propagated to upstream routers. Figure 3 shows how in a global adaptive routing algorithm, congestion status is propagated in the EAST direction [15].

## 4 Why not existing global adaptive routing algorithms?

Global adaptive routing algorithms have the potential to reduce the network latency beyond what is possible by local adaptive routing algorithms. However, as mentioned in the Introduction Section, existing global adaptive routing algorithms cannot reach their peak potential because they do not estimate congestion level of an output port on a per-packet basis: congestion estimate of an output port is independent of packets' destinations. Therefore, it is possible that a further away congested link that is not on the path to the destination contributes significantly to the congestion estimate, preventing global adaptive routing algorithms from balancing traffic in NOCs.

Congestion estimate being independent of a packet's destination, congested links that are outside the path to the destination may influence the routing decision while they ideally should not. To quantify the importance of taking into consideration the path to the destination while estimating congestion level, we count the number of routing decisions where all congested links are outside the path to the destination but still influence the routing decision (these congested links should not affect the routing decision as they are outside the path to the destination but existing global routing algorithms have no way of identifying such links) and normalize it to the number of routing decisions where at least one congested link is on the path to the destination.

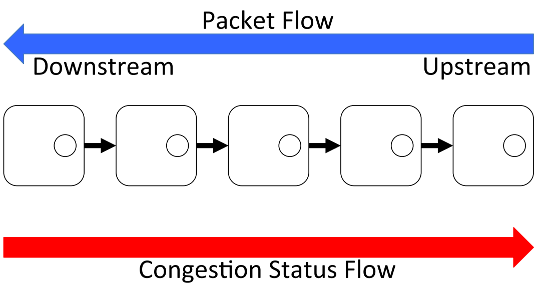

**Fig. 3** Congestion status flow in a global adaptive routing algorithm [15]

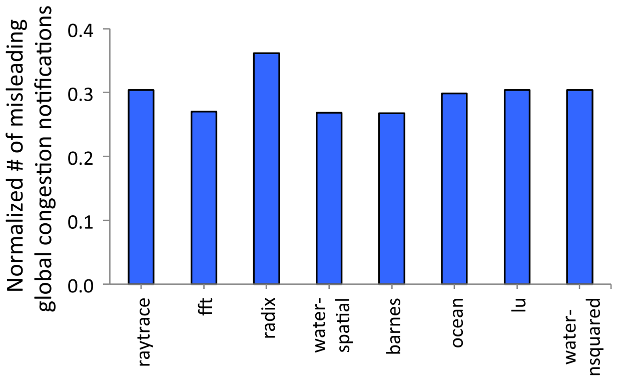

**Fig. 4** Number of misleading global congestion notifications of existing global adaptive routing algorithms, normalized to the number of useful global congestion notifications across SPLASH-2 benchmarks

Figure 4 shows the results of this experiment across all SPLASH-2 benchmarks. We consider a link congested if two or more packets compete for that link in a given cycle. When congested links are NOT on the path to the destination, and links that are on the path to the destination are not congested, the congestion estimate gets heavily influenced by the congested links that are outside the path, misleading global adaptive routing algorithms. Figure 4 shows that the number of such misleading cases is large across SPLASH-2 benchmarks: on average for every three useful congestion notifications, there is a misleading one.

Congestion estimation being independent of packets' destinations, existing global adaptive routing algorithms are incapable of sifting useful congestion notifications from misleading ones. As the number of misleading congestion notifications is high, existing global adaptive routing algorithms cannot fully balance traffic in the network, missing the opportunity to minimize latency.

## 5 Fast global adaptive routing algorithm

To exclusively take into consideration the congestion status of links that are on the path to the destination, a global adaptive routing algorithm should include destinations of packets in the process of estimating congestion levels of output ports. Destina-

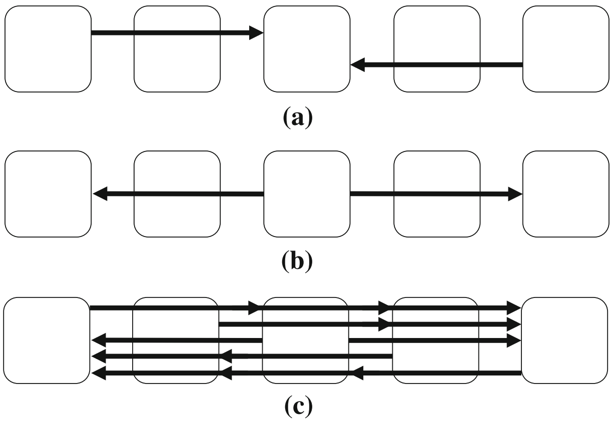

**Fig. 5** Congestion flags of *Fast* global adaptive routing algorithm in a row of a network-on-chip. **a** Congestion flags to the node in the middle. **b** Congestion flags of the node in the middle. **c** All congestion flags in a row of a NOC

tions of packets enable global routing algorithms to dynamically include congestion status of downstream routers that are on the path to the destination in the routing decision.

Unfortunately, existing global routing algorithms are inherently incapable of selectively excluding congestion status of not-on-the-path-to-the-destination downstream routers from global congestion status. This is due to the fact that as congestion status of downstream routers gets mixed with each other (i.e., the weighted average) to form global congestion status, it is no longer possible to extract congestion status of a particular downstream router from the global congestion status.

To address this shortcoming, we rely on concatenation, instead of weighted average, to form global congestion status. As with concatenation, the correspondence between bits in global congestion status and status of downstream routers is preserved, a global routing algorithm can take advantage of packets' destinations to dynamically include congestion status of routers that are on the path to the destination in the routing decision.

In this work, we associate a 1-bit congestion flag per routers' output port. This 1-bit flag is ONE when the port is congested and ZERO otherwise. Each router propagates its congestion flags to downstream routers in the same row and column. As a result, every router receives a congestion flag for each router in the same row and column. Figure 5a shows the congestion flags that a node in the middle of a row receives from other nodes in the same row.

For every output port, a router propagates the congestion flag of the port to the neighbor that is connected to the port. The neighbor has access to the congestion flag (i.e., can use it in the routing process) and also propagates the flag further in the row or column for the use of other upstream routers. Figure 5b shows the wires that carry two of the four congestion flags of a node to the rest of the nodes in the same row (there are other wires propagating the other two congestion flags to the nodes in the column, but are not shown).

Every router requires $n$ 1-bit wire segments (i.e., each wire segment connects two neighboring nodes) to propagate the two congestion flags of EAST and WEST ports to all the nodes in the row, where $n$ is the number of nodes in the row. Likewise, there will be another $n$ 1-bit wire segments to propagate the congestion flags of NORTH and SOUTH ports to all the nodes in the column. In total, we need $n \times n$ 1-bit wire segments in a row (or a column) to propagate all congestion flags of all the nodes in the row (or the column) as shown in Fig. 5c.

In this work, we introduce *Fast*, a simple and low-cost global routing algorithm to demonstrate the potentials of per-packet congestion estimation. *Fast* leverages the following heuristic to minimize NOC delays: as links become further away from the current node, their congestion status becomes less important. The heuristic states that a congested link near the current router is more important (i.e., has more negative impact) than a congested link that is further away.

Having the congestion flags of all the routers in the same row and column, the routing logic can be as simple as the following. If the destination is on the same row or column as the current node (i.e., there is only one option for routing the packet), pick the only possible output port. Otherwise, if local congestion metrics favor a port, pick that port. Otherwise, for every possible output port, calculate the length of the continuously uncongested path to the destination in the row or column using global congestion flags. Pick the port with longer uncongested path to the destination.

As a near congested link has more negative impact on performance than a further away congested link, *Fast* relies on local congestion metrics to pick the favorite port. If local congestion metrics do not favor an output port over the other one, *Fast* uses global congestion flags to break the tie. Using congestion flags and for each possible output port, *Fast* calculates the length of the straight path to the destination that starts from the output port and ends with the first congested link. *Fast* picks the output port that enables a packet to travel more hops before facing a congested link.

The pseudocode of the *Fast* adaptive routing algorithm is shown as Algorithm 1. The inputs of this algorithm are *pos_x*, *pos_y*, *diff_x*, *diff_y*, *local_x*, *local_y*, *g_cong_x*, and *g_cong_x*. *pos_x* and *pos_y* determine the $X$ and $Y$ position of current router. *diff_x* and *diff_y* show both the distance (i.e., the absolute value) to the destination and the direction (i.e., the sign) of the destination in each of the $X$ and $Y$ dimensions, respectively. *local_x* and *local_y* are the congestion estimates of neighboring routers, and *g_cong_x*, and *g_cong_y* are the global congestion flags of routers in the $X$ and $Y$ dimensions, respectively.

Line 2 of the pseudocode checks if there are two options for routing the packet. If the condition does not hold, the control goes to Line 37 and picks the only available option using standard routing techniques. Otherwise if the condition holds and consequently two options are available for routing the packet, Line 3 checks if local congestion metrics (congestion level of neighbors) are different. If they are different, *Fast* relies on these local metrics to route the packet (i.e., no global congestion metric will be used). Otherwise, Lines 6–19 determine the direction and the distance to the destination in each of the $X$ and $Y$ dimensions. Lines 20–25 count how many hops the packet passes through on the path to the destination before encountering a congested link in each of the $X$ and $Y$ dimensions. Finally, Lines 26–35 pick the option (out of the two available options) in which the packet faces a congested link further away.

**Algorithm 1** Pseudocode of Fast Global Adaptive Routing

```
1: procedure FAST
2:     if (($diff\_x <> 0$) & ($diff\_y <> 0$)) then
3:         if $local\_x <> local\_y$ then
4:             Local adaptive routing based on $local\_x$ and $local\_y$
5:         else
6:             if $diff\_x > 0$ then
7:                 $dist\_x \leftarrow +$ diff_x
8:                 $dir\_x \leftarrow +1$
9:             else
10:                $dist\_x \leftarrow -$ diff_x
11:                $dir\_x \leftarrow -1$
12:            if $diff\_y > 0$ then
13:                $dist\_y \leftarrow +$ diff_y
14:                $dir\_y \leftarrow +1$
15:            else
16:                $dist\_y \leftarrow -$ diff_y
17:                $dir\_y \leftarrow -1$
18:            $cnt\_x \leftarrow 0$
19:            $cnt\_y \leftarrow 0$
20:            while (($dist\_x <> 0$) & ($g\_cong\_x[pos\_x + cnt\_x] == 0$)) do
21:                $cnt\_x \leftarrow cnt\_x + dir\_x$
22:                $dist\_x \leftarrow dist\_x - 1$
23:            while (($dest\_y <> 0$) & ($g\_cong\_y[pos\_y + cnt\_y] == 0$)) do
24:                $cnt\_y \leftarrow cnt\_y + dir\_y$
25:                $dist\_y \leftarrow dist\_y - 1$
26:            if $cnt\_x >= cnt\_y$ then
27:                if $dir\_x == +1$ then
28:                    Pick EAST
29:                else
30:                    Pick WEST
31:            else
32:                if $dir\_y == +1$ then
33:                    Pick SOUTH
34:                else
35:                    Pick NORTH
36:    else
37:        Just one option: pick the only possible direction
```

## 5.1 Hardware realization of fast adaptive routing algorithm

In this part, we propose a hardware implementation for *Fast* adaptive routing algorithm, and evaluate its critical path delay. To this goal, we suggest a hardware implementation for computing *cnt_x* and *cnt_y* (see Algorithm 1 for details of *cnt_x* and *cnt_y*), and then provide a complete hardware implementation for *Fast* algorithm using hardware implementation of *cnt_x* and *cnt_y*.

*cnt_x* and *cnt_y* store the distance to the first congested link in the $X$ and $Y$ dimensions, respectively. We use a chain of multiplexers to implement this computation. Figure 6 shows the hardware realization of *cnt_x* (Hardware realization of *cnt_y* is identical). As the chain of multiplexers do not take into consideration the destination of the packet, we take the minimum of the output of the mux-chain and the distance to the destination as the value of *cnt_x* (The same is true for *cnt_y*). The suggested hardware

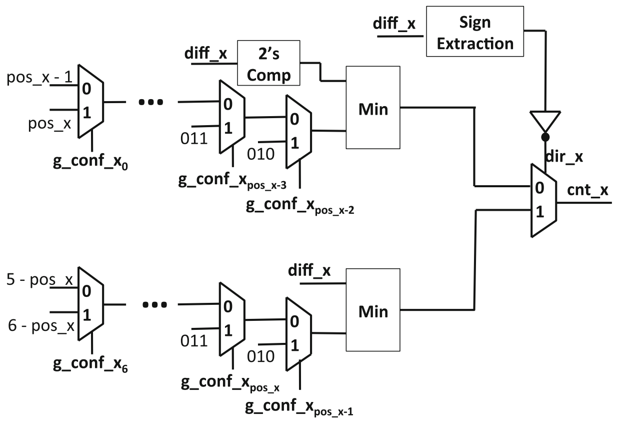

**Fig. 6** Hardware realization of *cnt_x* and *cnt_y*

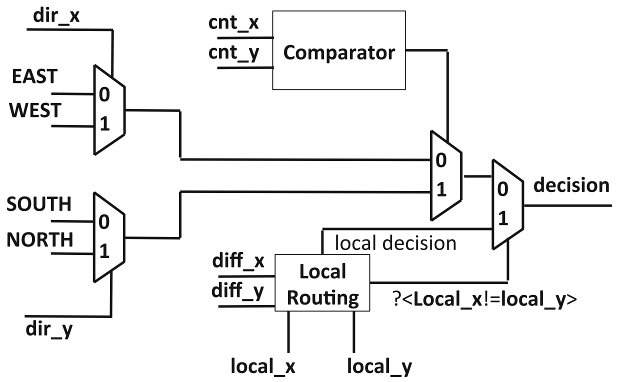

**Fig. 7** Hardware realization of *Fast* adaptive routing algorithm

realization of *cnt_x* is simple, because it only requires standard digital components (2-input multiplexer, 2's complement, comparator, and sign extractor). Moreover, it is fast, because it operates on values with few bits (4–8 bits).

Having a hardware realization for *cnt_x* and *cnt_y*, the hardware implementation of the whole *Fast* routing algorithm is trivial. Figure 7 shows the hardware realization of *Fast* routing algorithm. It essentially checks the condition to determine if local or global routing should be used. In case local routing is appropriate, the output comes from the standard local router. Otherwise, the output is determined using *cnt_x* and *cnt_y* (the one that is larger wins).

We use detailed technology models to derive component latencies. The worst-case delay of the routing algorithm happens in a router in the corner of an $8 \times 8$ network-on-chip that has the highest number of multiplexers in the mux-chain. At 32 nm technology, the routing easily fits in one clock cycle for frequencies up to 5 GHz given the critical-path delay of the *Fast* routing algorithm.

While *Fast* is simple and has low cost, it estimates congestion levels of output ports on a per-packet basis. Consequently, it can showcase the importance of per-packet congestion estimation by global routing algorithms.

## 6 Evaluation

To assess the effectiveness of the proposed routing algorithm (i.e., *Fast*), we compare it against DOR, Local, and RCA (the latter is the state-of-the-art global adaptive routing algorithm). We use synthetic and real workloads for the evaluation.

### 6.1 Methodology

To determine latency-throughput characteristics, we use a detailed cycle-accurate NOC simulator for the virtual channel routers. Each input virtual channel has a buffer (FIFO) depth of six flits. The congestion threshold value for *Fast* routing is set to two, meaning that an output port of a router is considered to be congested when at least two packets compete for the same output port in a given cycle. The per-hop latency for packets is three cycles and congestion flags can pass one hop in a single cycle (as they do not need routing or arbitration). In all of the simulations, the latency is measured by averaging the latency of the packets when each local core generates 10,000 packets. The routers use the minimally fully adaptive virtual sub-network deadlock avoidance technique discussed in [36]. Table 1 shows the baseline network configuration, and the variations used in the sensitivity studies.

#### *6.1.1 Technology parameters*

We use publicly available tools and data to estimate the area and energy of global routing algorithms. Our study targets a 32-nm technology node with an on-die voltage of 0.9 V and a 2-GHz operating frequency.

We use custom wire models, derived from a combination of sources [2,20], to model links and router switch fabrics. For links, we model semi-global wires with a pitch of

**Table 1** Baseline network configuration and variation

| Characteristic | Baseline | Variation |
|---|---|---|
| Topology | $7 \times 7$ 2D mesh | $4 \times 4$, $15 \times 15$ |
| Routing | Minimal, fully adaptive, virtual sub-network deadlock avoidance | – |
| Per-hop latency | 3 | – |
| VC/port | 2 | 4 |
| Flit buffers/VC | 6 | – |
| Packet size (flits) | 5 | 10 |
| Traffic workload | Bit-rotate, hotspot, transpose, uniform | SPLASH-2 |
| Packets/node | 10,000 | – |

200 nm and power-delay-optimized repeaters that yield a link latency of 125 ps/mm. On random data, links dissipate 50 fJ/bit/mm, with repeaters responsible for 19 % of link energy. For area estimates, we assume that link wires are routed over logic or SRAM and do not contribute to network area; however, repeater area is accounted for in the evaluation. Our buffer models are taken from ORION 2.0 [21]. We model flip-flop-based buffers.

#### *6.1.2 Workloads*

We evaluate *Fast* using four standard synthetic traffic patterns: *bit-rotate*, *hotspot 5 %*, *transpose*, and *uniform random*. These synthetic traffic patterns provide insight into the relative strengths and weaknesses of the competing routing algorithms.

Moreover, we evaluate *Fast* using traces of SPLASH-2 benchmarks [46]. The traces are obtained from a 49-core CMP, arranged in a $7 \times 7$ 2D mesh topology [25].

### **6.2 Synthetic traffic patterns**

We evaluate various routing algorithms on a $7 \times 7$ 2D mesh network with two virtual channels using standard synthetic traffic patterns. The packet length is set to five flits. The packet latency for different traffic profiles are shown in Fig. 8.

#### *6.2.1 Uniform random traffic profile*

In this traffic profile, every node sends packets to other nodes while using a uniform distribution to construct the destination set of packets [27]. Figure 8a shows the average communication delay as a function of the average packet injection rate. For this traffic profile, DOR, RCA, and *Fast* routing algorithms lead to lower latencies (uniform traffic is balanced under DOR routing [15]).

#### *6.2.2 Hotspot traffic profile*

The same as uniform random, each node sends packets to other nodes in the network. But nodes at positions (3, 3), (3, 5), (5, 3), and (5, 5) receive 5 % more packets . The average communication delays as a function of average packet injection rate for this traffic pattern are shown in Fig. 8b. For this traffic pattern, adaptive routing algorithms perform much better than DOR, while *Fast* and RCA leading to the lowest latencies. This is due to the non-uniform nature of the traffic pattern.

#### *6.2.3 Transpose traffic profile*

In transpose traffic profile within an $n \times n$ mesh network, a node at position $(i, j)(i, j \in [0, n-1))$ only sends packets to the node at position $(n-i-1, n-j-1)$. This traffic pattern is similar to the concept of transposing a matrix [14]. This traffic profile leads to a non-uniform traffic distribution with heavy traffic in the center of the mesh. As the results shown in Fig. 8c reveal, if the packet injection rate is very low, the routing

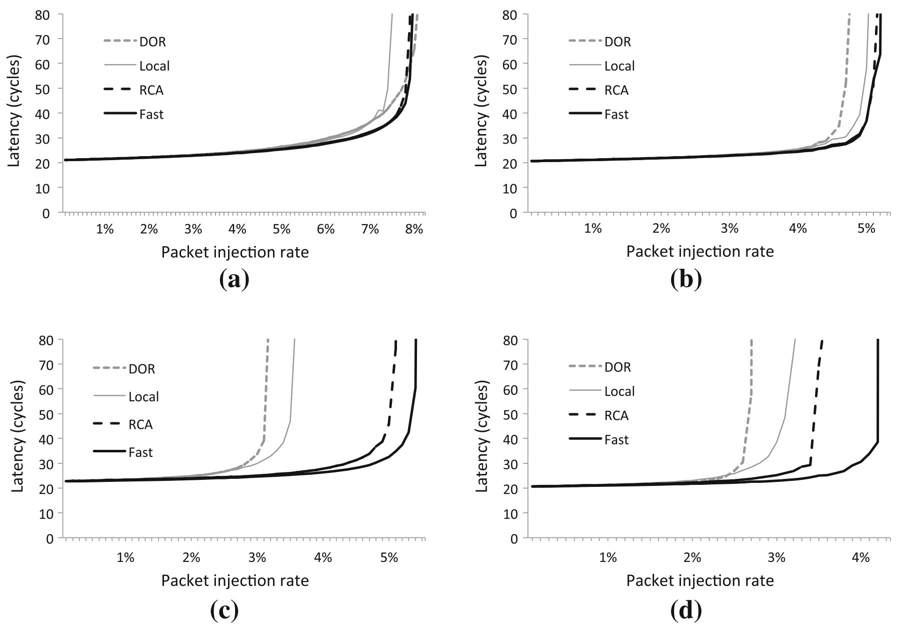

**Fig. 8** Latency versus packet injection rate for DOR, Local, RCA, and *Fast* on a $7 \times 7$ 2D mesh with two virtual channels per port and 5-flit packets. **a** Uniform random traffic, **b** hotspot 5 % traffic, **c** transpose traffic, and **d** bit-rotate traffic

techniques behave similarly. As the injection rate increases and links get congested, the *Fast* and RCA routing algorithms lead to smaller average delays, with *Fast* being the absolute best.

### 6.2.4 Bit-rotate traffic profile

In an $n \times n$ mesh network, a node at position $(i, j)(i, j \in [0, n-1))$ is assigned an address $i * n + j$. In bit-rotate traffic profile, a node with address $s$ sends packets only to the node with address $d$, where $d_i = s_{i+1 \; mod \; n}$. This traffic profile leads to a non-uniform traffic distribution. With this traffic profile, *Fast* routing algorithm leads to the lowest latencies, beating DOR, Local, and RCA.

## 6.3 Sensitivity analysis

In this section, we change some of the NOC parameters to evaluate the sensitivity of the routing algorithms to the change in the network parameters.

### 6.3.1 NOC size

Figures 9 and 10 show the latency versus packet injection rate for a small (i.e., $4 \times 4$) and a large (i.e., $15 \times 15$) mesh NOC under the hotspot 5 % (9a, 10a), and transpose

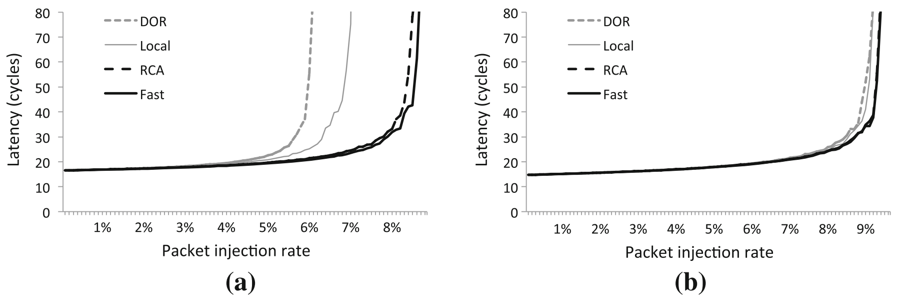

**Fig. 9** Latency versus packet injection rate for DOR, Local, RCA, and *Fast* on a 4 × 4 2D mesh with two virtual channels per port and 5-flit packets. **a** Hotspot 5 % traffic, and **b** transpose traffic

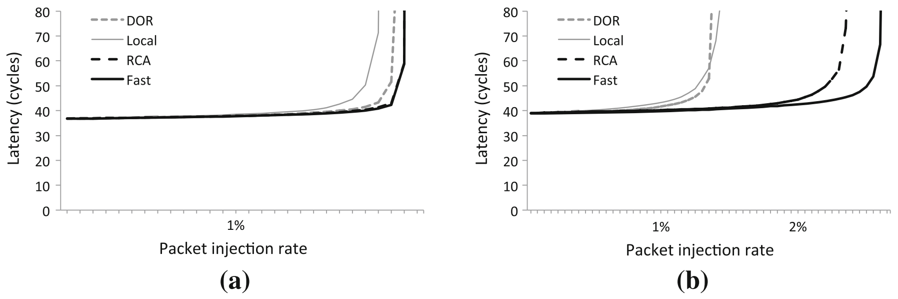

**Fig. 10** Latency versus packet injection rate for DOR, Local, RCA, and *Fast* on a 15 × 15 2D mesh with two virtual channels per port and 5-flit packets. **a** Hotspot 5 % traffic, and **b** transpose traffic

(9b, 10b) traffic profiles with 5-flit packets. For hotspot 5 %, nodes at positions (1, 1), (1, 2), (2, 1), and (2, 2) for the 4 × 4, and nodes at positions (3, 3), (3, 12), (12, 3), and (12, 12) for the 15 × 15 mesh receive 5 % more traffic. For the small network, the results in Fig. 9 show that *Fast* performs very well on Hotspot 5 %, achieving 42 % and 26 % better throughput than DOR and Local and slightly exceeding that of RCA. On transpose traffic profile, *Fast* is 6 % better than Local and DOR and slightly better than RCA. On smaller networks, *Fast* provides excellent visibility into the congestion state of the network.

For the large network (i.e., Fig. 10), adaptive approaches do not perform as well versus DOR. This is due to the fact that noise in congestion estimate is increased. However, under both traffic profiles, *Fast*, which eliminates noise, is the absolute best routing algorithm, with RCA and DOR being the second and third, respectively.

### 6.3.2 Packet length

Figure 11 shows load-latency graphs for long packets (i.e., packets that are 10 flits long) in a 7 × 7 2D mesh under the hotspot 5 % (11a), and transpose (11b), traffic profiles. The average packet latencies for all routing algorithms are higher than those of short packets. The increased latency is a known characteristic of wormhole routing with long packets. The reason is that for long packets, an imbalance in the resource utilization occurs because packets are likely to hold resources over multiple routers.

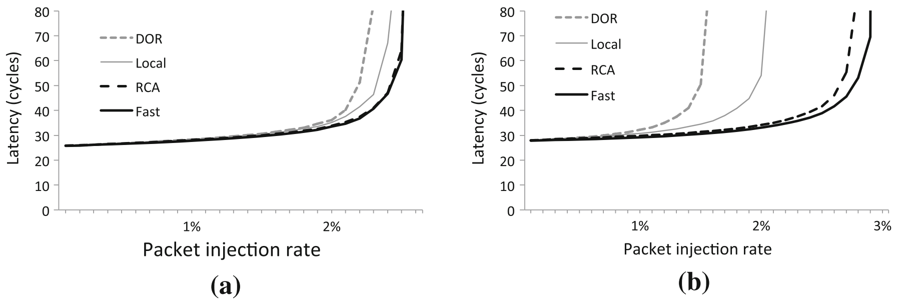

**Fig. 11** Latency versus packet injection rate for DOR, Local, RCA, and *Fast* on a 7 × 7 2D mesh with two virtual channels per port and 10-flit packets. **a** Hotspot 5 % traffic, and **b** transpose traffic

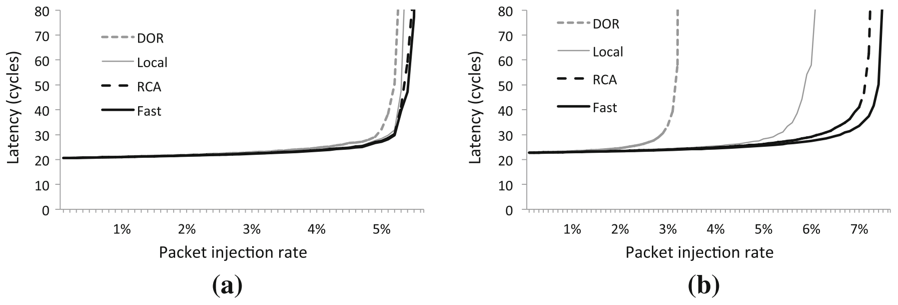

**Fig. 12** Latency versus packet injection rate for DOR, Local, RCA, and *Fast* on a 7 × 7 2D mesh with four virtual channels per port and 5-flit packets. **a** Hotspot 5 % traffic, and **b** transpose traffic

Similar to the case of 5-flit packets, *Fast* routing algorithm performs the best under both traffic profiles, with RCA and Local being the second and third, respectively.

### *6.3.3 Network with four virtual channels*

Figure 12 shows latency versus packet injection rate for DOR, Local, RCA, and *Fast* routers with four virtual channels. In this case, we consider a 7 × 7 2D mesh with 5-flit packets. As the results show, *Fast* continues to perform better than other routing algorithms in a network with four virtual channels.

## 6.4 SPLASH-2 benchmarks

Figure 13 shows the average packet latency of *Fast* routing algorithm across SPLASH-2 benchmarks, normalized to RCA. Contention is the cause of significant packet latency in *barnes*, *lu*, *ocean*, and *water-nsquared*; thus global adaptive routing algorithms have an opportunity to improve performance. Although *Fast* provides equal or lower latency than RCA for all benchmarks, the greatest benefit is on *ocean* with over 12 % reduction in latency. On average, *Fast* provides a latency reduction of 5 % across all benchmarks, and 10 % across contended benchmarks versus the state of the art.

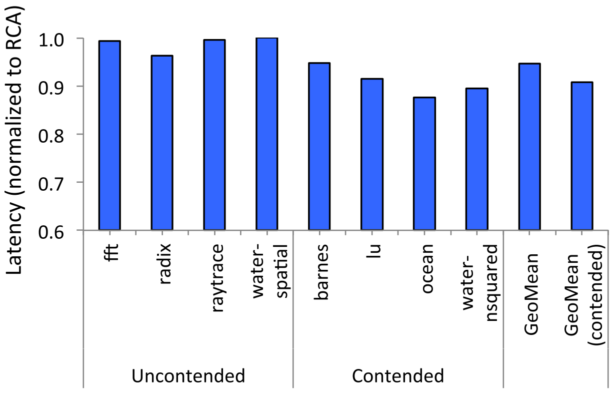

**Fig. 13** Average latency of *Fast* across SPLASH-2 benchmarks normalized to the latency of RCA

## 6.5 Hardware overhead

RCA requires nine bits of information per output port to propagate global congestion status to upstream routers, and *Fast* only requires an average of three bits per port. Consequently, *Fast* requires 3× fewer wires for global congestion propagation, which represents a 8 % reduction in the number of wires for a 64-bit flit network.

The 8 % reduction in the number of wires is important for wire-intensive designs that are wire limited. But, as most of the area of a mesh network is due to the buffers and crossbars, in absolute terms, the difference between the area of RCA and *Fast* is small (1.29 versus 1.27 mm$^2$).

## 6.6 Power dissipation

Our analysis confirms findings of prior work that CMP NOC is not a significant consumer of power at the chip level [24,29]: NOC power is below 1 W under SPLASH-2 workloads. Good L1 cache performance is the main reason for the low power consumption at the NOC level.

## 7 Related work

In [19], a static routing algorithm for two-dimensional meshes which is called *XY* is introduced. In this routing algorithm, each packet first travels along the *X* and then the *Y* direction to reach the destination. For this method, deadlock never occurs but no adaptivity exists in this algorithm. An adaptive routing algorithm named turn-model is introduced in [14] based on which another adaptive routing algorithm called OddEven turn is proposed in [5]. To avoid deadlock, OddEven method restricts the position that turns are allowed in the mesh topology. Another algorithm called DyAD is introduced in [17]. This algorithm is a combination of a static routing algorithm called oe-fix, and an adaptive routing algorithm based on the OddEven turn algorithm. Depending on the congestion condition of the network, one of the routing algorithms is invoked.

Another adaptive routing is hot potato or deflection routing [11,35,37] which is based on the idea of delivering a packet to an output channel at each cycle. If all the channels belonging to minimal paths are occupied, then the packet is misrouted. When contention occurs and the desired channel is not available, the packet, instead of waiting, will pick any alternative available channels (minimal or non-minimal) to continue moving to the next router; therefore, the router does not need buffers. In hot potato routing, if the number of input channels is equal to the number of output channels at every router node, packets can always find an exit channel and the routing is deadlock free. However, livelock is a potential problem in this routing. Also, hot potato increases message latency even in the absence of congestion and bandwidth consumption. Accordingly, performance of hot potato routing is not as good as other wormhole routing methods [8].

There are adaptive routing algorithms for increasing fault tolerance of the on-chip network. Stochastic communication method has been proposed to deal with permanent and transient faults of network links and nodes [9]. This method has the advantage of simplicity and low overhead. The selection of links and of the number of redundant copies to be sent on the links is stochastically done at runtime by the network routers. As a result, the transmission latency is unpredictable and, hence, it cannot be guaranteed. Also, stochastic communication is not efficient in terms of power dissipation and latency.

A local adaptive deadlock-free routing algorithm called Dynamic *XY* (DyXY) has been proposed in [26]. In this algorithm, which is based on the static *XY* algorithm, a packet is sent either to the *X* or *Y* direction depending on the congestion condition. It uses local information which is the current queue length of the corresponding input port in the neighboring routers to decide on the next hop. It is assumed that the collection of these local decisions should lead to a near-optimal path from the source to the destination. The main weakness of DyXY is that the use of the local information in making routing decision could forward the packet in a path which has congestion in the routers farther than the current neighbors. The technique described in [15] overcomes this problem. It uses global information in making a routing decision. The technique requires a mechanism to mix local and global congestion information. The main problem of this work, as is discussed in this paper, is the lack of correspondence between a router in the downstream path and the global congestion value. *Fast* resolves this problem by having a dedicated single-bit congestion wire for each router in the downstream path. Several other techniques also attempted to address this problem [10, 31,32,40,41], but they either use low-cost mechanism which cannot reach the full potential of global adaptive routing [31] or significantly increase the complexity and hence the hardware overhead of the routers [10,32,40,41]. In this work, we showed that by using a few wires in a smart way, a low-cost fast adaptive routing algorithm for networks-on-chip emerges.

## 8 Conclusion

Chip multiprocessors require fast networks-on-chip to maximize performance. A routing algorithm is one of the key components that influences network's latency. A static/local adaptive routing algorithm has no/limited visibility into the network, and hence, cannot minimize network's latency. A global routing algorithm, which has more visibility into the network, is an alternative that has the potential to minimize latency and maximize performance. One of the key components of a global routing algorithm is congestion estimation unit, which measures the likelihood of a packet experiencing congestion if it gets forwarded to a particular router's output port. Unfortunately, existing congestion estimation units ignore destinations of packets, and hence, introduce noise into the congestion estimate, preventing existing global routing algorithms from minimizing network's latency.

This work introduced *Fast*, a global adaptive routing algorithm that does congestion estimation on a per-packet basis. *Fast* benefits from a low-cost congestion status network (1-bit per port) that informs upstream routers of congestion in downstream routers. Taking advantage of the congestion status network, *Fast* dynamically forwards packets to output ports, where the first congested link is further away. We showed that while *Fast* requires fewer wires, and hence silicon area, it reduces network's latency by up to 12 % over the state-of-the-art global adaptive routing algorithm.

## References

1. Bakhoda A, Kim J, Aamodt TM (2010) Throughput-effective on-chip networks for Manycore accelerators. In: Proceedings of the 43rd annual IEEE/ACM international symposium on microarchitecture, USA, NY, NY, pp 421–432
2. Balfour JD, Dally WJ (2006) Design tradeoffs for tiled CMP on-chip networks. In: Proceedings of the 20th annual ACM international conference on supercomputing, Cairns, Queensland, Australia, pp 187–198
3. Barroso LA, Gharachorloo K, McNamara R, Nowatzyk A, Qadeer S, Sano B, Smith S, Stets R, Verghese B (2000) Piranha: a scalable architecture based on single-chip multiprocessing. In: Proceedings of the 27th annual international symposium on computer architecture, Vancouver, British Columbia, Canada, pp 282–293
4. Bienia C, Kumar S, Singh JP, Li K (2008) The PARSEC benchmark suite: characterization and architectural implications. In: Proceedings of the 17th international conference on parallel architectures and compilation techniques, New York, New York, USA, pp 72–81. doi:10.1145/1454115.1454128
5. Chiu GM (2000) The odd-even turn model for adaptive routing. IEEE Trans Parallel Distrib Syst 11(7):729–738
6. Council TPP. http://www.tpc.org/default.asp
7. Dally WJ, Aoki H (1993) Deadlock-free adaptive routing in multicomputer networks using virtual channels. IEEE Trans Parallel Distrib Syst 4(4):466–475
8. Duato J, Yalamanchili S, Lionel N (2002) Interconnection networks: an engineering approach, 1st edn. Morgan Kaufmann Publishers Inc., San Francisco
9. Dumitras T, Marculescu R (2003) On-chip stochastic communication. In: Proceedings of the conference on design, automation and test in Europe, vol 1, p 10790
10. Ebrahimi M, Daneshtalab M, Farahnakian F, Plosila J, Liljeberg P, Palesi M, Tenhunen H (2012) HARAQ: congestion-aware learning model for highly adaptive routing algorithm in on-chip networks. In: Proceedings of the 6th IEEE/ACM international symposium on networks-on-chip, pp 19–26
11. Feige U, Raghavan P (1992) Exact analysis of hot-potato routing. In: Proceedings of the 33rd annual symposium on Foundations of Computer Science, pp 553–562
12. Ferdman M, Adileh A, Kocberber O, Volos S, Alisafaee M, Jevdjic D, Kaynak C, Popescu AD, Ailamaki A, Falsafi B (2012) Clearing the clouds: a study of emerging scale-out workloads on modern hardware. In: Proceedings of the 17th international conference on architectural support for programming languages and operating systems, England, UK, London, pp 37–48
13. Galles M (1997) Spider: a high-speed network interconnect. IEEE Micro 17(1):34–39

14. Glass CJ, Ni LM (1992) The turn model for adaptive routing. In: Proceedings of the 19th annual international symposium on computer architecture, Queensland, Australia, pp 278–287
15. Gratz P, Grot B, Keckler SW (2008) Regional congestion awareness for load balance in networks-on-chip. In: Proceedings of the 14th international symposium on high-performance computer architecture, Salt Lake City, UT, USA, pp 203–214
16. Grot B, Hardy D, Lotfi-Kamran P, Falsafi B, Nicopoulos C, Sazeides Y (2012) Optimizing data-center TCO with scale-out processors. IEEE Micro 32(5):52–63
17. Hu J, Marculescu R (2004) DyAD: smart routing for networks-on-chip. In: Proceedings of the 41st annual design automation conference, San Diego, CA, USA, pp 260–263
18. Intel. Intel Xeon Processor X5670. http://ark.intel.com/products/47920/
19. Intel (1991) A touchstone DELTA system description. In: Technical report. Supercomputer Systems Division, Intel Corporation
20. International Technology Roadmap for Semiconductors (ITRS) 2011 Edition. URL http://www.itrs.net/Links/2011ITRS/Home2011.htm
21. Kahng AB, Li B, Peh LS, Samadi K (2009) ORION 2.0: a fast and accurate NoC power and area model for early-stage design space exploration. In: Proceedings of the conference on design, automation, and test in Europe, Nice, France, pp 423–428
22. Kim J, Dally WJ, Abts D (2007) Flattened butterfly: a cost-efficient topology for high-radix networks. In: Proceedings of the 34th annual international symposium on computer architecture, San Diego, California, USA, pp 126–137
23. Kim J, Park D, Theocharides T, Vijaykrishnan N, Das CR (2005) A low latency router supporting adaptivity for on-chip interconnects. In: Proceedings of the 42nd annual design automation conference, Anaheim, California, USA, pp 559–564
24. Kumar A, Kundu P, Singh AP, Peh LS, Jha NK (2007) A 4.6Tbits/s 3.6GHz single-cycle NoC router with a novel switch allocator in 65nm CMOS. In: Proceedings of the 25th international conference on computer design, pp 63–70
25. Kumar A, Peh LS, Kundu P, Jha NK (2007) Express virtual channels: towards the ideal interconnection fabric. In: Proceedings of the international symposium on computer architecture, San Diego, California, USA, pp 150–161
26. Li M, Zeng QA, Jone WB (2006) DyXY: a proximity congestion-aware deadlock-free dynamic routing method for network on chip. In: Proceedings of the 43rd annual design automation conference, CA, USA, San Francisco, pp 849–852
27. Lin X, Ni L (1993) Multicast communication in multicomputer networks. IEEE Trans Parallel Distrib Syst 4(10):1105–1117
28. Lotfi-Kamran P, Daneshtalab M, Lucas C, Navabi Z (2008) BARP—a dynamic routing protocol for balanced distribution of traffic in NoCs. In: Proceedings of the conference on design. Automation and test in Europe, Munich, Germany, pp 1408–1413
29. Lotfi-Kamran P, Grot B, Falsafi B (2012) NOC-Out: microarchitecting a scale-out processor. In: Proceedings of the 45th annual IEEE/ACM international symposium on microarchitecture, Vancouver, BC, Canada, pp 177–187
30. Lotfi-Kamran P, Grot B, Ferdman M, Volos S, Kocberber O, Picorel J, Adileh A, Jevdjic D, Idgunji S, Ozer E, Falsafi B (2012) Scale-out processors. In: Proceedings of the 39th annual international symposium on computer architecture, Portland, Oregon, USA, pp 500–511
31. Lotfi-Kamran P, Rahmani AM, Daneshtalab M, Afzali-Kusha A, Navabi Z (2010) EDXY—a low cost congestion-aware routing algorithm for network-on-chips. J Syst Archit 56(7):256–264
32. Ma S, Enright Jerger N, Wang Z (2011) DBAR: an efficient routing algorithm to support multiple concurrent applications in networks-on-chip. In: Proceedings of the 38th annual international symposium on computer architecture, pp 413–424
33. Marculescu R, Ogras UY, Peh LS, Jerger NE, Hoskote Y (2009) Outstanding research problems in NoC design: system, microarchitecture, and circuit perspectives. IEEE Trans Comput-Aided Des Integr Circuits Syst 28(1):3–21
34. Michelogiannakis G, Balfour J, Dally WJ (2009) Elastic-buffer flow control for on-chip networks. In: Proceedings of the 15th IEEE international symposium on high-performance computer architecture, Raleigh, NC, USA, pp 151–162
35. Moscibroda T, Mutlu O (2009) A case for bufferless routing in on-chip networks. In: Proceedings of the 36th annual international symposium on computer architecture, pp 196–207

36. Ni LM, McKinley PK (1993) A survey of wormhole routing techniques in direct networks. Computer 26(2):62–76
37. Nilsson E, Millberg M, Oberg J, Jantsch A (2003) Load distribution with the proximity congestion awareness in a network on chip. In: Proceedings of the conference on design, automation and test in Europe, vol 1, p 11126
38. Ogras UY, Hu J, Marculescu R (2005) Key research problems in NoC design: a holistic perspective. In: Proceedings of the 3rd international conference on hardware/software codesign and system synthesis, Jersey City, NJ, USA, pp 69–74
39. Ozer E, Flautner K, Idgunji S, Saidi A, Sazeides Y, Ahsan B, Ladas N, Nicopoulos C, Sideris I, Falsafi B, Adileh A, Ferdman M, Lotfi-Kamran P, Kuulusa M, Marchal P, Minas N (2010) EuroCloud: energy-conscious 3D server-on-chip for green cloud services. In: Proceedings of the workshop on architectural concerns in large datacenters in conjunction with ISCA
40. Ramanujam RS, Lin B (2010) Destination-based adaptive routing on 2D mesh networks. In: Proceedings of the 6th ACM/IEEE symposium on architectures for networking and communications systems, pp 19:1–19:12
41. Ramanujam RS, Lin B (2013) Destination-based congestion awareness for adaptive routing in 2D mesh networks. ACM Trans Des Autom Electron Syst 18(4):60:1–60:27
42. Schonwald T, Zimmermann J, Bringmann O, Rosenstiel W (2007) Fully adaptive fault-tolerant routing algorithm for network-on-chip architectures. In: Proceedings of the 10th Euromicro conference on digital system design architectures. Methods and tools, Lubeck, Germany, pp 527–534
43. Shin JL, Tam K, Huang D, Petrick B, Pham H, Hwang C, Li H, Smith A, Johnson T, Schumacher F, Greenhill D, Leon AS, Strong A (2010) A 40nm 16-Core 128-Thread CMT SPARC SoC processor. In: Proceedings of the IEEE international solid-state circuits conference, CA, USA, San Francisco, pp 98–99
44. Singh A, Dally WJ, Gupta AK, Towles B (2003) GOAL: a load-balanced adaptive routing algorithm for torus networks. In: Proceedings of the 30th annual international symposium on computer architecture, Tel-Aviv, Israel, pp 194–205
45. Vangal SR, Howard J, Ruhl G, Dighe S, Wilson H, Tschanz J, Finan D, Singh A, Jacob T, Jain S, Erraguntla V, Roberts C, Hoskote Y, Borkar N, Borkar S (2008) An 80-Tile Sub-100-W TeraFLOPS processor in 65-nm CMOS. IEEE J Solid-State Circuits 43(1):29–41
46. Woo SC, Ohara M, Torrie E, Singh JP, Gupta A (1995) The SPLASH-2 programs: characterization and methodological considerations. In: Proceedings of the 22nd international symposium on computer architecture, S. Margherita Ligure, Italy, pp 24–36. doi:10.1145/223982.223990