CD-Xbar: A Converge-Diverge Crossbar Network for High-Performance GPUs

Xia Zhao, Sheng Ma, Zhiying Wang, *Member, IEEE*,
Natalie Enright Jerger, *Senior Member, IEEE*, and Lieven Eeckhout, *Fellow, IEEE*

Abstract—Modern GPUs feature an increasing number of streaming multiprocessors (SMs) to boost system throughput. How to construct an efficient and scalable network-on-chip (NoC) for future high-performance GPUs is particularly critical. Although a mesh network is a widely used NoC topology in manycore CPUs for scalability and simplicity reasons, it is ill-suited to GPUs because of the many-to-few-to-many traffic pattern observed in GPU-compute workloads. Although a crossbar NoC is a natural fit, it does not scale to large SM counts while operating at high frequency. In this paper, we propose the converge-diverge crossbar (CD-Xbar) network with round-robin routing and topology-aware concurrent thread array (CTA) scheduling. CD-Xbar consists of two types of crossbars, a local crossbar and a global crossbar. A local crossbar converges input ports from the SMs into so-called converged ports; the global crossbar diverges these converged ports to the last-level cache (LLC) slices and memory controllers. CD-Xbar provides routing path diversity through the converged ports. Round-robin routing and topology-aware CTA scheduling balance network traffic among the converged ports within a local crossbar and across crossbars, respectively. Compared to a mesh with the same bisection bandwidth, CD-Xbar reduces NoC active silicon area and power consumption by 52.5 and 48.5 percent, respectively, while at the same time improving performance by 13.9 percent on average. CD-Xbar performs within 2.9 percent of an idealized fully-connected crossbar.
We further demonstrate CD-Xbar's scalability, flexibility and improved performance per Watt (by 17.1 percent) over state-of-the-art GPU NoCs which are highly customized and non-scalable.

Index Terms—Graphics processing unit (GPU), network-on-chip (NoC), crossbar

1 Introduction

Graphics Processing Units (GPUs) are widely deployed in high-performance computing systems and data centers for massive data processing. A GPU-compute application typically consists of a number of kernels that are composed of (up to hundreds of) thousands of threads. These threads are organized into cooperative thread arrays (CTAs) and are scheduled on streaming multiprocessors (SMs). To continuously boost raw computational power in modern high-performance GPUs, the number of SMs keeps increasing. For example, while the Nvidia Fermi GPU integrated 16 SMs, the latest Nvidia Pascal [1] and Volta GPUs [2] feature 60 and 80 SMs, respectively.

The increasing number of SMs puts critical pressure on the Network-on-Chip (NoC) that connects the SMs to the last-level cache (LLC) slices and memory controllers (MCs). How to design a scalable area- and power-efficient GPU NoC is a major challenge. As reported by previous work, the NoC in manycore processors incurs substantial chip area and power consumption [3], [4], [5]; for example, network power accounts for 19 percent of total chip power for a recent manycore processor [6].

Scalable NoC topologies have been proposed, including mesh, Clos and butterfly. These network topologies were conceived for CPU systems in which the different CPUs need to communicate with each other to support cache coherence. However, these networks are poorly suited to GPUs because of the many-to-few-to-many traffic pattern in which communication only exists between SMs on the one side and LLC slices (and memory controllers) on the other side [7]. There is no communication between SMs, i.e., coherence at the SM-side L1 cache is achieved through software issuing flush operations to the shared last-level cache. Scalable CPU topologies lead to underutilized links and are thus both power- and area-inefficient when deployed in a GPU. A crossbar NoC on the other hand is a natural fit by only providing links to connect the SMs to the LLC slices and vice versa; there are no links to connect the SMs among themselves. However, scaling a crossbar NoC to large SM counts at high clock frequency is problematic because of long propagation delays [8], [9], [10], [11].

In this paper, we propose the *Converge-Diverge Crossbar (CD-Xbar)*, a scalable, area- and power-efficient GPU NoC. CD-Xbar consists of two types of switch nodes, a local crossbar and a global crossbar. Instead of directly connecting the SMs to the LLC slices, CD-Xbar uses local crossbars to connect

- X. Zhao and L. Eeckhout are with the Department of Electronics and Information Systems, Ghent University, Gent 9000, Belgium. E-mail: {xia.zhao, lieven.eeckhout}@ugent.be.
- N. Enright Jerger is with the Edward S. Rogers Sr. Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON M5S, Canada. E-mail: enright@ece.utoronto.ca.
- S. Ma and Z. Wang are with the School of Computer, National University of Defense Technology, Changsha 410073, China. E-mail: {masheng, zywang}@nudt.edu.cn.

Manuscript received 5 Sept. 2018; revised 28 Jan. 2019; accepted 13 Mar. 2019. Date of publication 24 Mar. 2019; date of current version 15 Aug. 2019.
(Corresponding author: Xia Zhao and Sheng Ma.)
Recommended for acceptance by D. Gizopoulos.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TC.2019.2906869

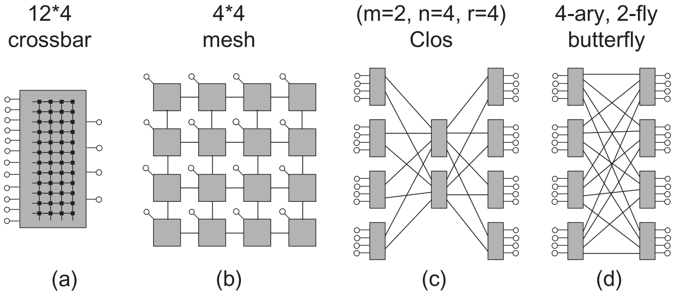


Fig. 1. NoC topologies: (a) crossbar, (b) mesh, (c) Clos and (d) butterfly.

the SMs to a set of *converged ports*. These converged ports are then diverged to the LLC slices through a global crossbar. For example, instead of having one monolithic fully-connected 80×16 crossbar to connect 80 SMs to 16 LLC slices, CD-Xbar features 8 local 10×3 crossbars that each connect 10 SMs to 3 converged ports and a global 24×16 crossbar to connect the 24 converged ports (8 local crossbars × 3 converged ports per local crossbar) to the 16 LLC slices. By doing so, CD-Xbar enables high-frequency operation while preserving area- and power-efficiency.

The converged ports pose a routing challenge as different input nodes have several converged ports to choose from. Existing routing policies such as source-based routing and adaptive routing are ineffective because they make the routing decision locally without considering the requests from the different nodes. We instead propose *round-robin routing* which assigns converged ports to incoming packets in a round-robin way across the different nodes to reduce flit contention. A key benefit of the round-robin routing policy is that it provides a simple mechanism to guarantee path diversity through the converged ports.

CD-Xbar achieves similar performance as a fully-connected crossbar when network traffic is balanced among the different local crossbars, which is typically the case as CTAs exhibit similar execution characteristics [12]. However, unbalanced traffic may exist in two scenarios: (i) a small GPU kernel may not occupy all SMs, and (ii) a spatially multitasking GPU may co-execute a memory-intensive and a compute-intensive kernel on different sets of SMs. In both scenarios, some local crossbars would receive most of the network traffic while other local crossbars would remain idle, which could severely hurt overall system performance. To solve the unbalanced NoC problem, we propose *topology-aware CTA scheduling* to balance network traffic among the different local crossbars.

We make the following contributions in this paper:

- We show that because of a GPU's unique traffic pattern, a crossbar topology is inherently more area- and power-efficient than other NoCs including mesh, Clos and butterfly, however it faces scalability challenges with increasing SM count.
- We propose CD-Xbar, a converge-diverge crossbar that delivers scalable performance at low chip area and power cost. CD-Xbar provides routing path diversity through the converged ports.
- We devise round-robin routing, a critical component to mitigate contention in converged ports, and topology-aware CTA scheduling to balance network traffic among the local crossbars.
- We report that CD-Xbar reduces NoC active silicon area by 52.5 percent and power consumption by 48.5 percent while improving performance by 13.9 percent compared to a mesh network. CD-Xbar performs within 2.9 percent of an idealized fully-connected crossbar. Finally, CD-Xbar improves performance per Watt by 17.1 percent on average over state-of-the-art GPU NoCs while being more flexible and better scalable.

2 Motivation

How to construct a high-performance cost-effective GPU NoC with an increasing number of SMs is a major challenge. In this section, we evaluate the area- and power-efficiency as well as the maximum operating frequency for four different NoC topologies that are well explored in the CPU domain, including mesh, Clos, butterfly and crossbar, plus one NoC topology that is optimized for the GPU, namely S-mesh. This will allow us to more clearly describe the goal of this work.

2.1 NoC Topologies

We consider the following five NoC topologies.

Crossbar. A crossbar (Fig. 1a) connects m inputs (the SMs) to n outputs (the LLC slices) via an $m \times n$ crosspoint switch.

Mesh. An $n \times n$ mesh network (Fig. 1b) consists of n^2 routers with each router consisting of five input ports (i.e., four neighbors and the node itself). Each router consists of a 5×5 crossbar.

Clos. The Clos network (Fig. 1c) is a multistage network. Here, we assume a 3-stage Clos network characterized by the triple (m, n, r): m denotes the number of middle-stage switches; n is the number of input and output ports of the first and last stage switches, respectively; r is the number of the switches in the first and last stage. An (m, n, r) Clos network can support $r \times n$ SMs and memory nodes.

Butterfly. The butterfly network (Fig. 1d) is another multistage interconnection network. A k-ary n-fly is implemented by using n stages of switches. Each stage consists of k^{n-1} switches and each switch has a radix of k which means it has k input and output ports. The k-ary n-fly network can support k^n SMs and memory nodes. Here, we assume a 2-stage butterfly.

S-Mesh. S-Mesh is a mesh NoC optimized for GPUs [13]. Due to a GPU's unique traffic pattern, i.e., there is no communication between SMs, some links and buffers in a traditional mesh are not used. As a result, the 5×5 crossbar architecture of a router can be simplified. S-mesh removes these unused resources to reduce hardware cost.

2.2 Area Analysis

To understand how NoC topology affects chip area, one has to consider its constituting components. Links and routers are the two important components of a NoC. Links connect the input and output ports of neighboring routers. For each router, the input buffers and the crossbar switch are the major components consuming chip area (and power). The overhead of these components is affected by the node degree (the number of input and output ports of the router). As the number of ports increases, the associated buffers, allocator logic and crossbar area also need to increase. We assume input-buffered routers in this work.

TABLE 1
NoC Configurations

Topology	20 SMs	80 SMs	120 SMs	180 SMs
Mesh	6×6	10×10	12×12	14×14
Butterfly	(6-ary, 2-fly)	(10, 2)	(12, 2)	(14, 2)
Clos	(4, 6, 6)	(8, 10, 10)	(8, 12, 12)	(8, 14, 14)
Crossbar	20×16	80×16	120×16	180×16
S-Mesh	6×6	10×10	12×12	14×14

We compare the specific NoC configurations shown in Table 1. We assume a channel width of 32 bytes across all configurations, and every input port has 4 virtual channels (VCs) and 4 flits per VC. We further assume 8 memory controllers and 2 LLC slices per memory controller, for a total of 16 LLC slices. We use DSENT [14] to compute NoC area, power consumption (assuming a 0.1 injection rate) and clock frequency. (See Section 4 for further details about the experimental setup.) Table 2 provides a detailed component breakdown for the five topologies. $\#Routers$ is the total number of routers in the request and reply networks. $\#Buffers$ is the total number of VC buffers located in the input ports across all routers (in both networks). $\#Crossbars$ is the number of crossbar switches and their configurations. $\#Links$ is the total number of links to connect the routers. To provide a fair comparison between the different topologies, we take the input/output links into consideration.

We assume two physically separate networks, a request network and a reply network, to avoid protocol deadlock. This is sufficient as GPUs exploit software-based coherence and network packets only constitute of read/write requests and replies. Note that we assume two physically separate networks for the mesh topology as well, as done in prior work [13], [15], [16], [17]. This allows for separate and asymmetric optimizations for the different networks based on their respective traffic characteristics and load, see the S-Mesh proposal [13] in particular. The alternative approach of exploiting two virtual networks on one physical network can reduce the hardware cost of a mesh network considerably. However, it loses the opportunity for asymmetric optimization opportunities. Moreover, even in the ideal case where two virtual networks would yield similar performance as two physical networks, performance would still be worse than CD-Xbar, as the evaluation section in this paper shows that CD-Xbar significantly outperforms a mesh topology with two physical networks. We further assume one network port per SM and do not consider concentration [18], which is a one-time solution that can be exploited by all networks that we evaluate here.

TABLE 2
NoC Component Breakdown in Terms of the Number of Routers, Buffers, Crossbar Switches and Links

Topology	#Routers	#Buffers	#Crossbars	#Links
Crossbar	1×2	384	2 (80×16)	192 (L)
Mesh	100×2	4,000	200 (5×5)	912 (S)
Clos	28×2	2,080	40 (10×8), 16 (10×10)	512 (L)
Butterfly	20×2	1,600	40 (10×10)	392 (L)
S-Mesh	100×2	3,000	200 (4×4)	732 (S)

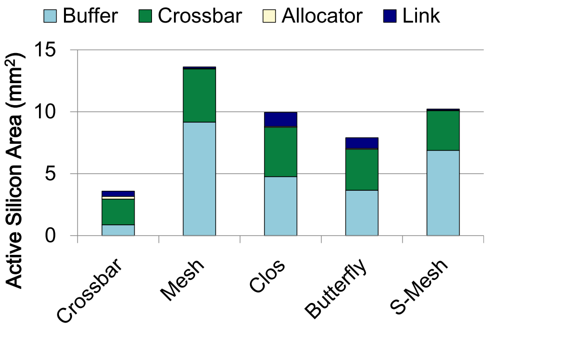


Fig. 2. NoC area breakdown for the five network topologies assuming 80 SMs and 16 LLCs. *A crossbar topology is inherently more area-efficient.*

Fig. 2 breaks down the area cost for the five topologies for a GPU architecture with 80 SMs and 16 LLC slices. The breakdown includes network area due to buffers, crossbars, allocators and links. Interestingly, the crossbar is much more area-efficient than the (S-)mesh, Clos and butterfly networks. In particular, the mesh topology requires 4,000 input buffers and 200 5 × 5 crossbars (Table 2). This leads to more than 4 times the active silicon area cost of the crossbar topology. The Clos, butterfly and S-mesh topologies exhibit similar area inefficiencies. The fundamental reason is that a fully-connected crossbar only provides connections from the SMs to the LLC slices and vice versa, i.e., there are no connections in-between SMs nor in-between LLC slices. In contrast, other topologies provide connections between SMs, by construction. This leads to a large number of under-utilized routers, which induces unnecessary chip area overhead.

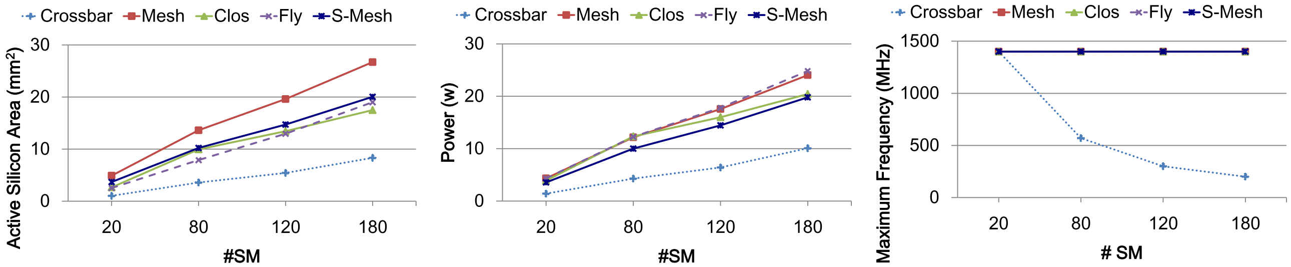
2.3 Scalability Analysis

Although the crossbar is more area-efficient, the other topologies are known to be better scalable with node count. Fig. 3 quantifies chip area, power consumption and operating frequency for the five topologies as we scale the number of SMs. (We keep the number of LLC slices constant but we scale memory bandwidth and LLC capacity proportionally with SM count.) The key observation is that while the crossbar is substantially more area- and power-efficient across SM count, operating it a high frequency becomes problematic as we scale the number of SMs.

As port count increases, a crossbar cannot operate at high frequency. A typical matrix-style crossbar consists of a collection of switches that route the data, plus an arbiter to configure the crossbar. The physical size of the crossbar grows quadratically with port count. This not only increases propagation delay, it also complicates the arbiter logic [8], [9], [11]. Although designs such as swizzle-switch [11] distribute the arbitration logic across the crossbar crosspoints—swizzle-switch was shown to operate at high frequency with 64 ports and a 16-byte channel width—the propagation delay problem caused by the crossbar size still persists. For GPUs that require high bandwidth, the channel width is typically higher and GPUs are expected to grow to hundreds of SMs in the next few years [19]. These considerations all preclude a fully-connected crossbar as a scalable GPU NoC.

2.4 Goal

The mesh, Clos and butterfly topologies offer good scalability, however, they do not fit a GPU's unique traffic pattern



(a) Chip area (b) Power consumption (c) Clock frequency

Fig. 3. Comparing GPU NoC topologies as a function of the number of SMs. *A crossbar is fundamentally more area- and power-efficient than a mesh, Clos or butterfly network, however, maintaining high clock frequency at high SM counts is impossible.*

and are therefore power- and area-inefficient. For GPUs, the crossbar is the most cost-efficient NoC by only supporting communication between SMs and LLC slices, and not among SMs nor LLC slices. Unfortunately, a crossbar faces a major scalability problem with increasing SM count.

One possible solution is to tailor multi-stage NoCs, i.e., butterfly and Clos, to GPUs by only providing communication paths between SMs and LLC slices. The idea of removing unused paths is similar to the previously proposed partial cascaded crossbar network [20]. Unfortunately, such solutions pose significant problems. In particular, a tailored 2-stage butterfly as well as a partial cascaded crossbar do not provide path diversity, which severely degrades performance when network traffic is imbalanced. Although a tailored Clos network can provide path diversity, it leads to a high hardware cost as it relies on middle-stage switches to provide different routing paths. Moreover, it also requires a complex adaptive routing policy to choose the least congested path.

Our goal is to devise a novel GPU NoC topology that achieves the best of both worlds. We want the GPU NoC topology to provide path diversity without relying on middle-stage routers. Moreover, instead of making the routing adaptive, we want a simple routing policy that can still exploit path diversity. In the next section, we show how we achieve this goal by proposing CD-Xbar with converged ports to provide path diversity. Converged ports bridge the gap between the large number of SMs and the relatively small number of LLC slices.

3 CD-XBAR

In this section, we first discuss the inherent limitation of a fully connected GPU crossbar, which provides an opportunity that we exploit in this work. We then introduce the CD-Xbar NoC, after which we propose a solution for the routing and load balancing problem.

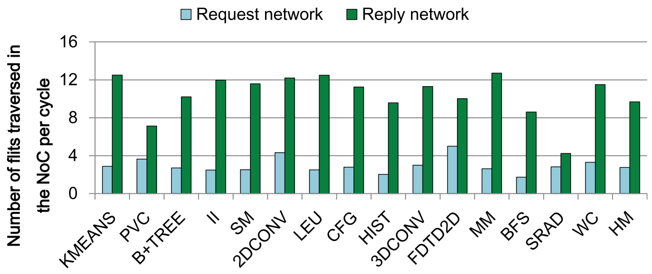


Fig. 4. Average number of flits transferred per cycle in a full crossbar. *The number of utilized links is small.*

3.1 Opportunity

In a conventional crossbar GPU NoC, the request network connects all the SMs to all the LLC slices through a fully-connected crossbar; the reply network does the inverse: it connects all the LLC slices to all the SMs. The fully-connected crossbar here only provides a communication path between SMs and LLC slices. Due to the huge gap between the SM count and LLC slice count, a full crossbar exhibits the inherent limitation that a significant fraction of the network is underutilized—only a limited number of links are effectively used in each cycle. For example, in a GPU architecture with 16 LLC slices and 80 SMs, at most 16 links are used at any given point in time. In reality, the number of active links, or the number of flits transferred per cycle is even smaller (Fig. 4). For the request network, we observe an average of only 2.8 flits per cycle. For the reply network, we observe an average of 9.7 flits per cycle. (The number of flits transferred over the reply network is higher than the request network because a reply typically consists of a long data block transmitted as several flits.) The key take-away message is that many links in a crossbar NoC are underutilized. This provides an opportunity to devise a converge-diverge NoC topology that achieves much better hardware utilization while achieving similar performance as a fully-connected crossbar.

3.2 CD-Xbar NoC

The key idea of the Converge-Diverge Crossbar is to first converge and then diverge NoC traffic to maximize bandwidth efficiency. CD-Xbar consists of several local crossbars and a global crossbar (Fig. 5). SMs are connected to the local crossbars whereas LLC slices are connected to the global crossbar. The key feature of the converge-diverge topology

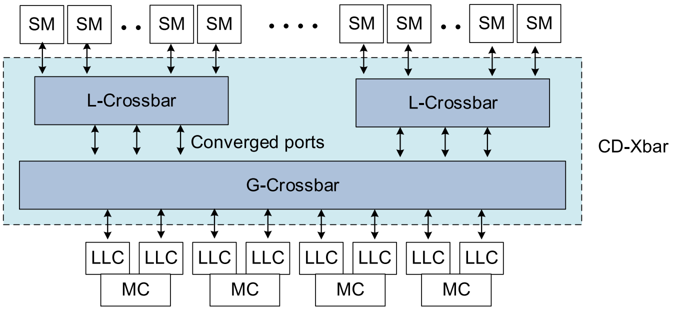


Fig. 5. CD-Xbar architecture. *In the request network, CD-Xbar converges requests from the SMs through a local crossbar to a number of converged ports which are then diverged to the LLCs and memory controllers through the global crossbar. The inverse happens in the reply network.*

is the use of *converged ports* as intermediate ports. The total number of converged ports is a balance between the number of SMs and LLC slices: there are fewer converged ports than SMs to improve hardware utilization; and there are more converged ports than LLC slices to avoid congestion.

CD-Xbar clusters SMs into several groups and all SMs in a single group use a local crossbar to connect to the global crossbar. The local and global crossbars are input-queued crossbar switches. In the request network, a converged port is the output port of a local crossbar and the input port to the global crossbar. We use uni-directional links to connect the converged ports of the local and global crossbars. The link direction is from SM to LLC in the request network. In the reply network, a converged port connects an output port of the global crossbar to an input port of a local crossbar; the link direction is from LLC to SM.

CD-Xbar applies wormhole switching where packets are subdivided into a number of flits of fixed length equal to the channel width. Credit-based flow control guarantees that there is always available buffer space in the downstream crossbar before sending a packet. Note that each flit needs to travel exactly two hops from source to destination. This increases the transfer latency per packet compared to an idealized full crossbar but we will show that this has a negligible impact on overall performance. GPU-compute workloads consist of thousands of concurrent threads, hence the GPU can easily switch to a ready warp when the current one is stalled. Finally, note that CD-Xbar has the same bisection bandwidth as a fully-connected crossbar.

There are different ways to scale CD-Xbar to larger SM counts. In this paper, we scale CD-Xbar by increasing the number of SMs per local crossbar while keeping the number of local crossbars and the number of converged ports per local crossbar unchanged. Alternative scaling solutions would be to increase the number of local crossbars (while keeping the number of converged ports per local crossbar constant) and/or increase the number of converged ports per local crossbar as each local crossbar groups more SMs. Either alternative scaling solution would increase the total number of converged ports which may lead to scaling issues for the global crossbar. The former scaling approach—increasing the number of SMs per local crossbar while keeping the number of local crossbars and the number of converged ports per local crossbar constant—thus is the better option. As shown in the evaluation section, CD-Xbar, with this scaling solution, scales to GPUs with up to 180 SMs.

We want to emphasize that CD-Xbar exhibits a unique feature compared to multi-stage NoCs such as the butterfly and Clos networks. In particular, butterfly does not offer path diversity whereas Clos relies on the middle-stage switch to provide different paths between each pair of nodes. Unlike these two designs, CD-Xbar does not have middle-stage switches, yet it provides path diversity through the converged ports. Moreover, the converged ports also bridge the gap between the SMs and the LLC slices by first shrinking many SM ports to a few converged ports through a local crossbar which are then diverged through the global crossbar to the LLC slices.

It is also worth noting that the converged ports not only provide path diversity, they are all productive ports. This is unlike a mesh network where some router ports are not productive. We will exploit this feature when describing our newly proposed round-robin routing scheme.

CD-Xbar does pose a number of new challenges in terms of routing and CTA scheduling, which we discuss in the next section.

3.3 CD-Xbar Routing Problem

How to route packets is important for load balancing and for minimizing network latency. In particular, if different memory nodes want to send packets to different SMs connected to the same local crossbar, how we route packets to the converged ports can have a significant impact on performance. Possible routing algorithms include oblivious routing and adaptive routing. However, both face challenges.

Oblivious routing makes the routing decision based on the source or destination node [21], [22]. Each node makes its routing decisions locally and independently from other nodes, which makes it a simple and fast routing policy. One possible implementation of oblivious routing in CD-Xbar is source-based routing, which deterministically selects the converged port based on the source node ID, as shown in the formula below:

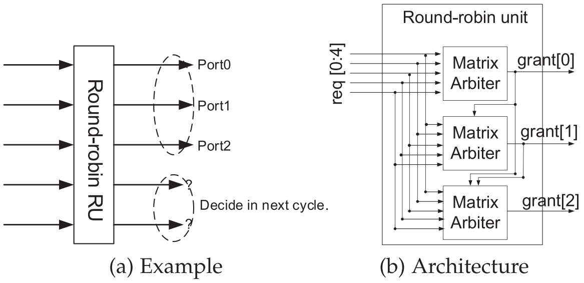
$$PortID = SourceID \ \% \ \# \ Ports.$$

Although oblivious routing is easy to implement, it faces the obvious shortcoming that when a node makes its routing decision, it does not consider whether other nodes also want to send packets to the same port or not, which may lead to congestion. In addition, oblivious routing does not consider the status of the network. A network port that is already over-utilized may still be chosen even though other less-congested ports might be a better option.

Adaptive routing [21], [23] takes network contention into account by choosing the least contended port, e.g., the port with the most free VC buffer entries. A problem arises if different nodes want to send packets to the same group of ports in the same cycle as adaptive routing would choose the same least-contended port for all requests which leads to severe contention on that particular port. To solve this problem, previous work [23] proposed a randomized version of adaptive routing which, for each input port, first randomly chooses several outputs and then adaptively chooses the one with the least congestion. When evaluating randomized adaptive routing in CD-Xbar, we randomly select two out of the three converged ports, and then send the packet to the least congested port out of these two ports. Randomization reduces congestion to some extent by spreading requests across the different ports, although there is still a significant probability that the same congested port is selected for two input ports. Round-robin routing reduces congestion beyond randomized adaptive routing by assigning network ports in a round-robin fashion, as we describe in the next section.

3.4 Round-Robin Routing

To avoid converged port contention, we propose *round-robin (RR) routing* which strives at routing packets to different converged ports. Round-robin routing considers all packets that need to be sent and routes packets so as to minimize contention on the converged ports. Note that sending a packet to any of the converged ports brings the packet closer to its



(a) Example (b) Architecture

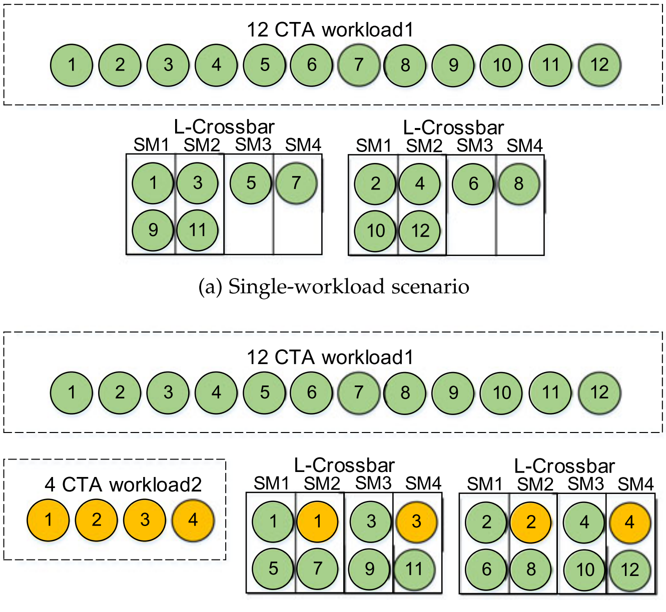
Fig. 6. Round-robin routing. *Round-robin routing minimizes contention on the converged ports.*

destination—as mentioned before, all converged ports are productive ports. For each crossbar, we consider a round-robin unit that calculates the routing path for all the inputs that send packets to the converged ports. Fig. 6 provides an example to illustrate how the round-robin routing unit works along with its microarchitecture. In Fig. 6a, there are five inputs sending packets to three converged ports. Round-robin routing first chooses three out of the five packets in a first-come, first-served (FCFS) way, and then assigns them to the three converged ports in a round-robin way. Fig. 6b shows the microarchitecture of the round-robin unit which consists of several arbiters, where arbiter count equals the number of converged ports. Each arbiter can be implemented as a low-cost/high-frequency matrix arbiter and the number of arbiters is small. The round-robin routing unit incurs negligible hardware cost and can operate at high frequency. In fact, the round-robin policy can be integrated with the switch allocator, which makes our design low-cost by integrating multiple functions within a single unit. We provide a round-robin unit in the local crossbars in the request network, and we provide a round-robin unit in the global crossbar in the reply network.

3.5 Topology-Aware CTA Scheduling

An implicit assumption underlying the CD-Xbar proposal is that network traffic is balanced among the different local crossbars. This is typically the case as CTAs of a kernel exhibit similar execution characteristics [12]. Conventional round-robin CTA scheduling [24], [25] first distributes CTAs across all SMs in a round-robin way, and once all SMs have one CTA assigned, it repeats the assignment process until an SM runs out of hardware resources to accept more CTAs. When more than one kernel is scheduled on a multitasking GPU, the SMs are evenly partitioned among the co-executing kernels.

Round-robin CTA scheduling may lead to imbalanced execution in two specific scenarios. First, a small kernel, due to algorithmic limitations or due to a small input data set [26], [27], [28], may occupy only a subset of the SMs and may lead to imbalance across the SMs, i.e., some SMs are assigned more CTAs than others. Second, when co-executing multiple kernels through spatial multitasking, some local crossbars may be over-utilized while others are under-utilized. Consider for example the case where a memory-intensive kernel is assigned to one half of the SMs while a compute-intensive kernel is assigned to the other half. The former would suffer from heavy contention while the other is under-utilized.



(a) Single-workload scenario

(b) Two-workload scenario

Fig. 7. Topology-aware CTA scheduling. *Balancing network traffic in single- and multi-workload scenarios.*

We propose topology-aware CTA scheduling to balance network traffic across the different local crossbars. For a single workload, topology-aware CTA scheduling first distributes CTAs across local crossbars and then across SMs within a crossbar. Once all local crossbars are assigned one CTA, the next batch of CTAs gets assigned to the second SM in each local crossbar, etc., until all SMs are assigned one CTA. This process repeats until the SMs run out of resources to accept more CTAs, or the workload runs out of CTAs. This is illustrated in Fig. 7a.

For a multitasking GPU, topology-aware CTA scheduling first assigns CTAs of the first kernel to the first SM in each local crossbar, and then switches to the second kernel and assigns CTAs to the second SM in each local crossbar. This process continues until all SMs across all local crossbars are occupied, or until one kernel has no more CTAs to assign. In the latter case, if the other kernel still has CTAs to schedule, the remaining CTAs are assigned as in the single-workload case which distributes CTAs across local crossbars and then across SMs within a local crossbar. This is illustrated in Fig. 7b.

In summary, topology-aware CTA scheduling aims at distributing network traffic across all crossbars and SMs, irrespective of the characteristics of the particular workload.

3.6 CD-Xbar Layout

We rely on a state-of-the-art and widely used architectural NoC analysis tool, namely DSENT [14], to evaluate the effectiveness and feasibility of the CD-Xbar architecture. We motivate and complement DSENT results with a preliminary chip layout, as well as a discussion of DSENT's validity based on previously published hardware validation studies of high-radix NoC designs. Fig. 8 provides a preliminary chip layout of CD-Xbar while considering the placement of SMs and LLC slices based on a die photo of the recent Nvidia Pascal GPU [29]. The local crossbars (LC) are distributed across the chip and are located near a respective cluster of

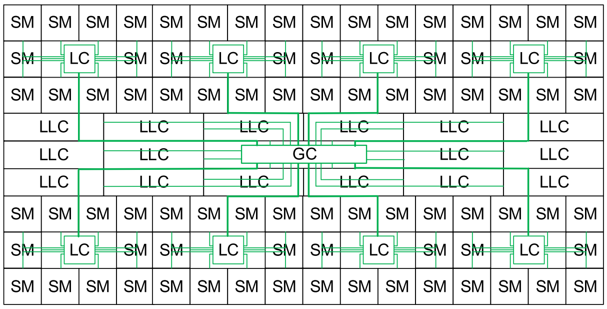


Fig. 8. CD-Xbar preliminary chip layout. *Local crossbars (LCs) connect to neighboring SMs while the global crossbar (GC) connects the local crossbars with the LLC slices.*

SMs. The global crossbar (GC) is located in the center of the GPU die and is connected to the local crossbars on the one hand and the LLC slices on the other hand. This layout suggests that wire congestion can be avoided by spreading out the local crossbars across the chip while locating the global crossbar at the center. This figure also illustrates that assuming the long links in the high-radix topologies to be half the die size, as we will describe in Section 4, is a reasonable (even conservative) assumption for our design; in fact, many links are shorter than half the die.

To further increase our confidence in the results obtained using DSENT, we now compare DSENT's timing predictions against real hardware implementations of high-radix routers. Passas et al. [9], [10] propose a 128-radix crossbar with a 32-bit width that operates at 750 Mhz in a 90 nm technology. Relating this design against our results obtained using DSENT is non-trivial, however, we can make some first-order approximations as follows. A crossbar's critical path is determined by the propagation delay and allocator delay [11]. For large crossbars that are highly optimized, see for example [9], [10], [11], it is reasonable to assume that propagation delay occupies a large portion of the critical path. For wires with repeaters, propagation delay scales approximately linearly with wire length. As a result, propagation delay scales with the square root of crossbar size. Meanwhile, crossbar size scales exponentially with technology node [30], radix and channel width [7]. Hence, as a first-order approximation, we assume that timing scales with the ratio of technology nodes, channel width and radix. For the Passas et al. design, this means that—by shrinking the radix to 80, increasing the channel width to 32 byte and considering a 22 nm technology node—the operating frequency of an 80-radix full crossbar is approximated to be 613 MHz $(= 750\ \mathrm{MHz} \times \frac{128}{80} \times \frac{32\ \mathrm{bits}}{32\ \mathrm{bytes}} \times \frac{90\ \mathrm{nm}}{22\ \mathrm{nm}})$. This is similar to the DSENT results as reported in Fig. 3c for the full crossbar with 80 SMs.

Swizzle switch [11] is an another highly-optimized high-radix crossbar design to scale to high radices. Swizzle switch with a radix of 64 and 16-byte channel width was shown to operate at 1.5 GHz in a 32 nm technology. Assuming the above first-order approximation, we estimate an 80-radix full crossbar with a 32-byte channel width in a 22 nm technology to operate at a frequency of 872.7 MHz, which is somewhat higher than DSENT's predictions for an 80-radix full crossbar with a 32-byte channel width in a 22 nm technology. This discrepancy can be explained by the fact that swizzle switch is specifically designed and optimized to operate at high frequencies.

TABLE 3
Baseline GPU Architecture

Parameter	Value
SM	80 SMs, 1400MHz, 1536 threads/SM, 32768 registers/SM, 2 GTO schedulers/SM, 48KB shared memory/SM, SIMD width 32
L1 data cache/SM	16KB, 4-way, LRU, 128B line
2 L2 slices/MC (8 MCs total)	128KB, 8-way, LRU, 128B line
Interconnection network	Crossbar, 32B channel width 4-stage router, VC/switch allocator – Islip 4 VCs per port – 4 flits/VC
DRAM model and bandwidth	FR-FCFS, 16 banks/MC, 700.0GB/s
GDDR5 timing	t_{CL}=12, t_{RP}=12, t_{RC}=40, t_{RAS}=28, t_{RCD}=12, t_{RRD}=6, t_{CCD}=2, t_{WR}=12

4 Experimental Setup

Simulated System. We use GPGPU-sim v3.2.2 [35] to evaluate the proposed CD-Xbar NoC architecture. Table 3 lists the configuration for our baseline GPU architecture. We consider 4-stage pipeline routers for all the NoCs evaluated in the paper. We use DSENT v0.91 [14] to evaluate the area and power cost for the different NoC designs, assuming a $22\ \mathrm{nm}$ technology node. For power, we collect activity counters through timing simulation using GPGPU-sim which we then use to estimate power using DSENT. For area, we consider active silicon area versus global wire area. For the links, repeater area is included in active silicon area whereas the wires are attributed to the global wire area as they can be routed in the higher metal layers. In the evaluation, we assume that short links measure $3.3\ \mathrm{mm}$ (SM size is estimated to be $10.9\ \mathrm{mm}^2$ from the Pascal die size [1]); the long links in the high-radix topologies are assumed to be $12.3\ \mathrm{mm}$ long (half the die size).[1]In CD-Xbar, unless mentioned otherwise, we assume 8 local crossbars with 3 converged ports each; a local crossbar connects to 10 SMs. A sensitivity analysis regarding CD-Xbar's configuration is provided in the evaluation. We assume topology-aware CTA scheduling throughout the evaluation unless mentioned otherwise.

GPU NoCs. We compare CD-Xbar against three state-of-the-art GPU NoC proposals, namely S-Mesh [13], cfNoC [16] and HRCnet [17]. All three topologies exploit the notion that, in a GPU, communication only exists between SMs and LLC slices. S-Mesh removes unused links and input buffers in a traditional mesh. cfNoC and HRCnet focus on the request and reply network, respectively. cfNoC provides exclusive subnets of SMs to eliminate packet conflicts through a token-based mechanism in the many-to-few request network. HRCnet proposes a ring-like topology to eliminate conflicts in the few-to-many reply network; a high channel width is assumed in the ring-like reply network to improve performance. Conflict elimination enables a simplified router design which reduces hardware cost. It is important to note that cfNoC and HRCnet require special (physical) placement of memory nodes in the network. Because cfNoC optimizes the request network and HRCnet optimizes the reply network, these two NoC optimizations are incompatible. As we will observe in the evaluation,

[1] The length for a long link is a conservative estimate which can be reduced through optimized floorplanning [36], [37], [38].

TABLE 4
Benchmarks Considered in This Paper

Benchmark	Abbr.	NoC Demand
K-means [31]	KMEANS	High
PageViewCount [32]	PVC	High
B+TREE Search [31]	B+TREE	High
InvertedIndex [32]	II	High
StringMatch [32]	SM	High
2DConvolution [33]	2DCONV	High
Leukocyte [31]	LEU	High
Euler3d [31]	CFG	High
Histogram [34]	HIST	High
3DConvolution [33]	3DCONV	High
Fdtd2d [33]	FDTD2D	High
MatrixMultiply [28]	MM	High
Breadth First Search [31]	BFS	High
SRAD [31]	SRAD	High
WordCount [32]	WC	High
Two Point Angular Corre-Function [34]	TPACF	Low
DXTC [28]	DXTC	Low
CP [35]	CP	Low
Pathfinder [31]	PF	Low
N-Queens Solver [35]	NQU	Low
Magnetic Resonance Imaging - Q [34]	MRI-Q	Low
QuasiRandomGenerator [28]	QRG	Low
MergeSort [28]	MS	Low

cfNoC and HRCnet are area-efficient; S-Mesh, although it significantly reduces chip area compared to a traditional mesh, still incurs a significant hardware cost. A major issue with the cfNoC and HRCnet proposals though is that they impose very strict requirements on the number of SMs and LLC slices, and their placement in the network, which severely limits their flexibility.

Workloads. We consider a broad set of CUDA GPU-compute benchmarks from a range of application domains. These benchmarks are selected from Rodinia [31], Parboil [34], CUDA SDK [28], GPGPU-sim [35] and other two sources [32], [33] (Table 4). We divide these applications into two categories depending on their NoC intensity. In particular, we classify an application as a high-NoC demand workload if performance degrades by more than 5 percent when halving the NoC bandwidth. In the evaluation section, we consider the high-NoC bandwidth benchmarks unless mentioned otherwise. When evaluating different CTA scheduling policies, we pair high-NoC demand applications with low-NoC demand applications to construct heterogeneous multi-program workloads; and we use small data sets as input to obtain small kernels.

Performance Metrics. We use instructions per cycle (IPC) to quantify single-application performance; and we simulate one billion instructions, or to completion, whichever occurs first [39]. System throughput (STP) and average normalized turnaround time (ANTT) are used to evaluate multiprogram performance [40]. For the multiprogram workloads, we simulate for two million cycles, which is in line with prior GPU multitasking research [41], [42].

TABLE 5
Mesh-Based GPU NoCs

Network	Configuration
Mesh	Request network: Mesh / Reply network: Mesh
S-Mesh	Request network: S-Mesh / Reply network: S-Mesh
cfNoC	Request network: cfNoC / Reply network: S-Mesh
HRCnet	Request network: S-Mesh / Reply network: HRCnet

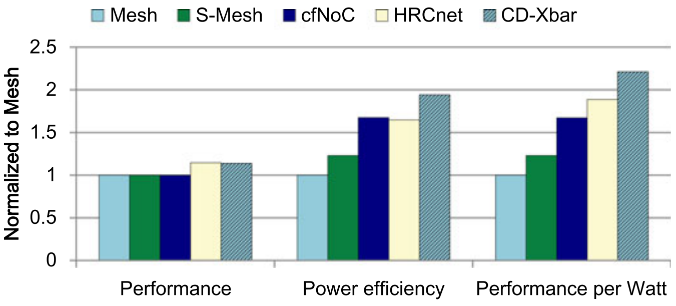


Fig. 9. Performance, NoC power efficiency, and performance/Watt relative to a mesh NoC. *CD-Xbar achieves the highest performance per Watt compared to state-of-the-art GPU NoCs. HRCnet achieves similar performance as CD-Xbar but is not scalable.*

5 Evaluation

The evaluation is done in a number of steps. We first compare against state-of-the-art GPU NoC proposals and evaluate performance, power and chip area (Sections 5.1 through 5.3). This comparison assumes a specific number of SMs (56) and LLC slices (8) because of the limitations in the previously proposed GPU NoCs that we compare against. The second half of the evaluation (Sections 5.4 through 5.6) compares CD-Xbar against idealized and realistic fully-connected crossbar designs. We also perform various sensitivity and scalability analyses assuming the default SM count (80) and LLC slices (16).

5.1 Performance per Watt

Previous work has assumed a baseline mesh network upon which it provides various optimizations to exploit the unique GPU traffic pattern [7], [15], [16], [17], [43]. In this section, we compare CD-Xbar against three state-of-the-art mesh-based GPU NoC proposals, namely S-Mesh [13], cfNoC [16] and HRCnet [17], see also Table 5. As the latter two designs require a specific number of memory nodes versus compute nodes, we assume 56 SMs and 8 memory controllers (MC) with one LLC slice per MC. cfNoC and HRCnet focus on the request and reply networks, respectively. As they both require specific placement of memory nodes, which is different for both topologies, these designs are incompatible. To construct the best possible baseline networks to compare CD-Xbar against, we use S-Mesh for cfNoC's reply network; similarly, we assume S-Mesh for HRCnet's request network. We complement cfNoC and HRCnet with an S-mesh network, and deliberately do not compare against cfNoC and HRCnet complemented with a conventional mesh network, in order to compare against the best possible NoC configuration based on prior work in the literature. For CD-Xbar, we assume 4 local crossbars with 3 converged ports each; each local crossbar connects to 14 SMs. To enable a fair comparison, we keep the bisection bandwidth unchanged across all the NoCs in the comparison.

Fig. 9 reports performance, power efficiency (1 / NoC power) and performance per Watt for the various NoC designs, normalized to the baseline mesh network. CD-Xbar and HRCnet are the best performing NoCs, i.e., CD-Xbar

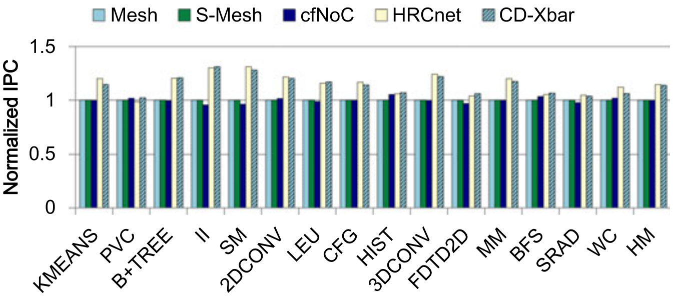


Fig. 10. Per-application performance. *CD-Xbar and HRCnet improve performance by 13.9 and 14.6 percent on average, respectively.*

improves performance by 13.9 percent whereas HRCnet improves performance by 14.6 percent on average over a mesh network. CD-Xbar consumes the least NoC power, followed by cfNoC and HRCnet. CD-Xbar achieves the highest performance per Watt: an average improvement of 17.1 percent over HRCnet, the best performing state-of-art GPU NoC. Note that in addition to not being as power-efficient, HRCnet is less flexible and less scalable than CD-Xbar. To achieve a conflict-free design, HRCnet can only operate with a specific number of memory nodes and compute nodes [17]. Moreover, scalability is limited by HRCnet's ring-like topology.

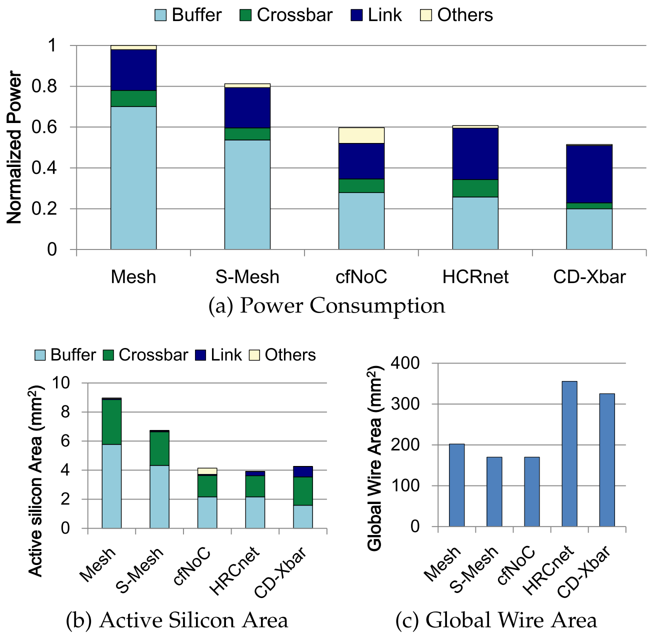
5.2 Per-Application Performance

Fig. 10 reports per-benchmark performance results. S-Mesh and cfNoC achieve similar performance as the default mesh network. S-Mesh removes unused links, input buffers and simplifies the crossbar, with no measurable impact on performance. cfNoC also achieves similar performance to a mesh network. cfNoC uses exclusive subnets for each column of compute nodes to eliminate packet conflicts. Subnets with sliced channel width increase the serialization latency of the request packets; however, most request packets are short read requests that can be transferred as a single flit in the cfNoC subnetwork. Moreover, cfNoC reduces the per-hop latency in the request network by exploiting a conflict-free router design.

HRCnet and CD-Xbar significantly improve performance due to the efficient utilization of the available network bandwidth. Because of the GPU's unique traffic pattern, the actual achieved bandwidth is (much) less than the bisection bandwidth for a mesh topology. In particular, in the reply network, only few memory nodes can inject packets, which leaves many links under-utilized. In contrast, HRCnet and CD-Xbar can achieve the maximum bisection bandwidth when all memory nodes are injecting packets into the network. We observe variable performance benefits across different applications in Fig. 10; this is because of the NoC bottleneck occupying different fractions of the SM stall cycles.

5.3 Power Efficiency and Chip Area

Fig. 11a breaks down NoC power consumption into its four components: buffer, crossbar, links and other. Buffer power is the largest contributor. Compared to a conventional mesh network, S-Mesh reduces power consumption by 18.7 percent by removing the input buffers of unused ports. Compared to S-Mesh, cfNoC and HRCnet reduce power consumption by removing all input buffers from the request and reply networks thanks to the conflict-free design. In cfNoC, the larger 'other' component is caused by the large ejection-buffer size. HRCnet has much higher link power consumption compared to the other three designs as it uses high channel width links for the ring-like network. Compared to the mesh network, CD-Xbar reduces power consumption by 48.5 percent as many input buffers are removed. Link power increases because of the long links to connect the SMs and LLC slices to the local and global crossbars, respectively.



(a) Power Consumption

(b) Active Silicon Area

(c) Global Wire Area

Fig. 11. NoC comparison in terms of (a) power consumption, (b) active silicon area and (c) global wire area. *cfNoC, HRCnet and CD-Xbar are low-power, low-cost designs, although HRCnet and CD-Xbar increase global wire area but these long wires can be routed in the upper metal layers.*

Figs. 11b and 11c report chip area for the different designs in terms of NoC active silicon area and global wire area, respectively. cfNoC, HRCnet and CD-Xbar incur similar active silicon area. cfNoC and HRCnet reduce active silicon area by featuring a simplified router architecture in the request and reply networks, respectively. Compared to cfNoC and HRCnet, although CD-Xbar reduces the buffer area, this is offset by the increase in crossbar area. CD-Xbar and HRCnet both consume more global wire area than the other three designs (Fig. 11c). In HRCnet, high channel width links are used to construct the ring-like reply network. In CD-Xbar, long links are used to connect the SMs and LLC slices to the NoC. Note that long wires are typically routed in the upper metal layers, hence they do not contribute to chip area [44] (apart from the repeaters which we account for as active silicon area, as mentioned before).

Conclusion. CD-Xbar achieves similar performance, similar area cost and higher power efficiency compared to HRCnet (the best performing prior work). However, a key problem with the HRCnet proposal is that it imposes restrictions on the number of SMs, LLC slices and their placement, which severely limits HRCnet's flexibility. In contrast, CD-Xbar can be implemented for any arbitrary number of SMs and LLC slices. Moreover, the ring network in HRCnet has limited scalability with increasing SM and LLC slice count.

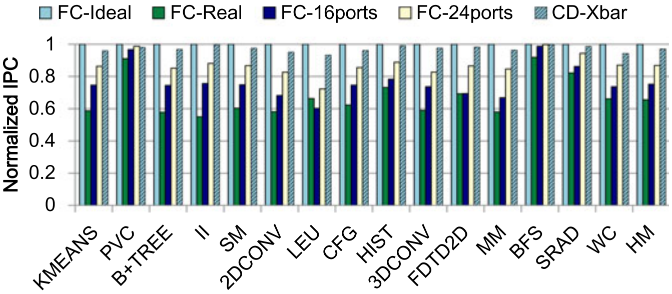


Fig. 12. Comparing CD-Xbar against idealized and realistic crossbar designs. *CD-Xbar performs within 2.9 percent on average of an idealized fully connected crossbar, while significantly outperforming realistic alternative crossbar designs.*

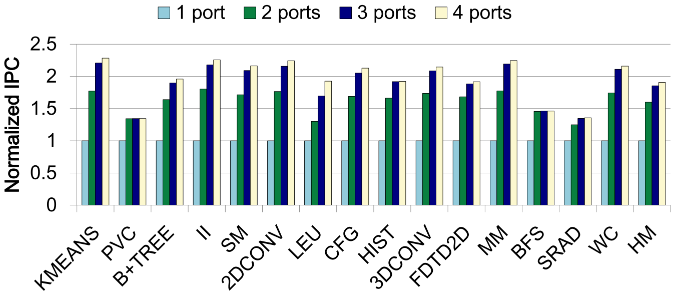


Fig. 13. Performance as a function of the number of converged ports in CD-Xbar. *Performance improves with increasing port count and saturates around 3 converged ports per local crossbar.*

In the evaluation, we adopt 56 SMs and 8 LLC slices as in the original HRCnet paper [17], which is a sweet spot for HRCnet. Even in its best case, CD-Xbar still outperforms HRCnet in performance per Watt by 17.1 percent.

5.4 Crossbar Alternatives

So far, we benchmarked CD-Xbar against mesh-based networks. We now compare CD-Xbar against a number of crossbar NoC alternatives. We compare against a fully-connected crossbar (FC), which is commonly assumed in research studies considering GPUs with a relatively small number of SMs [45], [46], [47]. Note though that a full crossbar incurs a substantially higher hardware cost (26.9 percent higher active silicon area cost) than CD-Xbar. We also compare against crossbar networks that exploit external concentration [18] by grouping several SMs to share one network port. External concentration can be regarded as a special case of the CD-Xbar design in which there is only a single converged port per local crossbar. Concentration reduces the number of ports to the crossbar, hence it enables operating the NoC at high frequency without incurring extra latency. In this section we assume 80 SMs and 16 LLC slices and consider the following fully connected crossbar configurations:

- *FC-Ideal*: Full 80×16 crossbar is assumed to operate at high frequency with no extra latency. This is an idealized fully connected crossbar.
- *FC-Real*: Full 80×16 crossbar incurs a one cycle extra latency. This is an optimistic assumption given that, according to Fig. 3, an 80×16 crossbar can operate at approximately 600MHz compared to the rest of the GPU that is assumed to operate at 1.4GHz. This optimistic assumption for our baseline puts our proposed solution, CD-Xbar, at a disadvantage.
- *FC-16ports*: 5 SMs are grouped to share one concentrated network port of a full 16×16 crossbar.
- *FC-24ports*: 3 (or 4) SMs are grouped to share one concentrated network port of a full 24×16 crossbar.

Fig. 12 reports performance for the different crossbar NoCs. Compared to the idealized full crossbar, the realistic crossbar degrades performance by 34.4 percent on average. The extra cycle latency in the switch and VC allocators increases the number of cycles that a flit occupies the switch and VC resources, which in its turn increases NoC contention substantially. External concentration with 16 ports and 24 ports leads to a performance gap of 24.8 and 13.2 percent, respectively, compared to the idealized full crossbar. Decreasing the number of SMs per local crossbar and only assigning one converged port to connect to the global crossbar increases contention when network traffic is unbalanced. Increasing the number of converged ports per local crossbar reduces port contention. This enables CD-Xbar to achieve performance that is within 2.9 percent of an idealized crossbar, while outperforming a realistic, fully connected crossbar with 24 ports by 10.3 percent.

5.5 Sensitivity Analyses

We now perform sensitivity analyses with respect to the number of converged ports, the routing policy, and CTA scheduling policy.

Number of Converged Ports. The number of converged ports poses a performance-cost trade-off. Increasing the number of converged ports per local crossbar reduces port contention, improving performance. On the other hand, this also incurs higher hardware cost. Fig. 13 evaluates CD-Xbar performance as a function of the number of converged ports assuming 8 local crossbars. Initially, performance improves steeply as we increase the number of converged ports, e.g., performance improves by 60 percent on average as we increase from 1 to 2 converged ports. However, the performance improvement gradually decreases as we further increase converged port count, e.g., from 3 to 4 ports, performance improves by only 5.2 percent. Performance saturates around 3 converged ports per local crossbar, which is what we assume throughout the paper.

Routing Policy. We previously discussed different routing policies including source-based routing, randomized adaptive routing and the newly proposed round-robin routing. Here, we evaluate their performance impact, see Fig. 14. Round-robin routing improves performance by 10.0 and 8.5 percent compared to source-based and randomized adaptive routing, respectively. When multiple packets traverse through a local crossbar, both source-based and adaptive routing may route packets to the same converged port even though other converged ports remain under-utilized. This incurs flit contention as only one flit can be transferred per converged port per cycle. For example, as there are only three converged ports per local crossbar, the randomized version of adaptive routing first randomly chooses two converged ports and then chooses the least congested port. However, with only three converged ports per local crossbar, even with randomization, two incoming packets may still be sent to the same port (probability of $2/3$), as previously

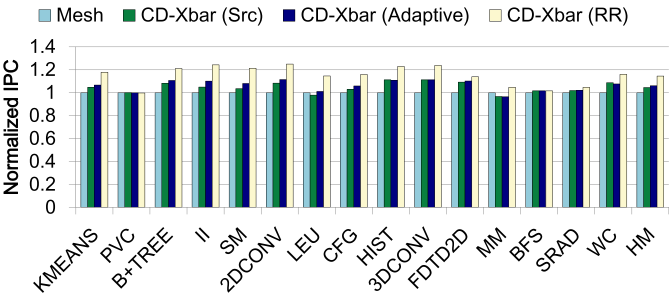
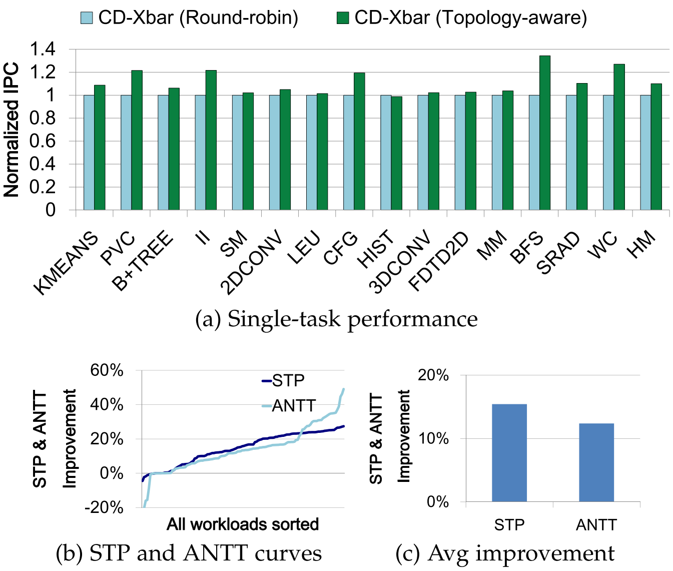


Fig. 14. Performance for different routing policies. *Round-robin (RR) routing outperforms source-based and randomized adaptive routing.*

described in Section 3.3. Such contention does not happen in round-robin routing as it assigns the converged ports to incoming packets in a round-robin way.

CTA Scheduling. Fig. 15a compares topology-aware scheduling against traditional round-robin CTA scheduling. Topology-aware scheduling improves single-task performance by 10.1 percent on average by balancing NoC traffic among the different local crossbars. Fig. 15b reports STP and ANTT improvement curves for the multitasking workloads. STP improves by 15.4 percent on average. The highest improvements are obtained for mixed workloads, i.e., workload mixes in which a high-NoC demand application co-executes with a low-NoC demand application. Topology-aware CTA scheduling for such mixed workloads balances the NoC traffic by the high-NoC demand application across the different converged ports. When co-running two high-NoC demand applications in a multiprogram setup, topology-aware CTA scheduling yields similar performance as traditional round-robin CTA scheduling because network traffic is balanced across all local crossbars for both scheduling policies. We note that for some workloads, per-application performance suffers under topology-aware CTA scheduling. This is the case when a high-NoC demand application puts significant pressure on the converged ports, which decreases the network bandwidth utilization and the performance of a co-executing application with limited TLP to hide memory access latency. Although some applications experience some performance degradation, we report a significant average ANTT improvement of 12.4 percent.



(a) Single-task performance

(b) STP and ANTT curves

(c) Avg improvement

Fig. 15. Evaluating topology-aware CTA scheduling performance for (a) single-task and (b and c) multi-tasking workloads relative to round-robin CTA scheduling. *Topology-aware CTA scheduling outperforms traditional round-robin scheduling.*

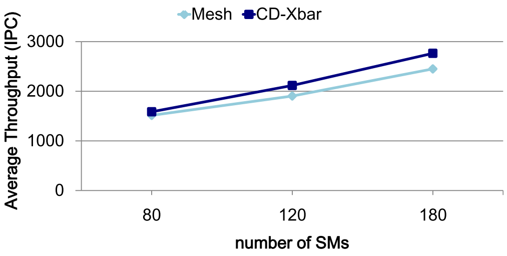


Fig. 16. Scalability analysis. *CD-Xbar scales better than a mesh NoC with increasing SM count.*

5.6 Scalability Analysis

We now evaluate CD-Xbar's scalability with the number of SMs. While increasing SM count, we keep the number of converged ports unchanged and increase the number of SMs per local crossbar. For example, with 120 SMs, we maintain 8 local crossbars (same number as for the default 80 SMs) and we maintain 3 converged ports per local crossbar; hence the number of SMs per local crossbar increases from 10 to 15. Note that scaling the number of SMs per local crossbar does not affect the timing of the overall NoC. In particular, the critical component in CD-Xbar is the global crossbar which is a 24×16 full crossbar, i.e., this crossbar connects 24 ports (8 local crossbars and 3 converged ports per local crossbar) to 16 ports (16 LLC slices). The global crossbar remains the critical component in CD-Xbar as we scale the number of SMs. The reason is that we keep the number of local crossbars constant at 8 and the number of converged ports per local crossbar at 3, hence the total number of ports to connect to the global crossbar is constant at 24. So, even for the largest configuration with 180 SMs, the global crossbar is a 24×16 full crossbar, whereas the local crossbars include four 22×3 full crossbars and four 23×3 full crossbars. Timing for CD-Xbar is determined by the size of the largest crossbar, i.e., the global crossbar. In other words, timing is not affected when scaling the number of SMs.

To demonstrate the performance potential with increasing SM count, we report raw performance across all applications including the NoC bandwidth-limited and bandwidth-unlimited applications. Fig. 16 reports performance for CD-Xbar and the default mesh network as we scale SM count. (We assume the same bisection bandwidth for the mesh and CD-Xbar networks.) There are two take-away messages. First, increasing the number of SMs leads to a significant performance improvement [48], [49]. For example, increasing the SM count from 80 to 180 improves performance by 62 percent, i.e., IPC increases from 1514 to 2,450. Second, CD-Xbar achieves good scalability and in fact the performance improvement over the mesh network increases with increasing SM count as a result of better NoC bandwidth utilization.

6 Related Work

We now discuss related work in GPU NoCs, CPU NoCs and CTA scheduling.

GPU NoC. Prior work in GPU NoCs primarily focuses on how to optimize a mesh network because of its inherent simplicity and scalability. In particular, Bakhoda et al. [7] propose checkboard routing and a simplified crossbar structure to reduce the router area. Kim et al. [15] propose the DA2mesh network to achieve a low-cost conflict-free design by exploiting the few-to-many traffic pattern in a GPU's reply network. Ziabari et al. [13] explore different GPU NoC designs and show that an asymmetric concentrated mesh provides the highest power efficiency. Jang et al. [43] explore MC placement and combine the reply and request network into one network. They further propose asymmetric VC partitioning by assigning more VCs to reply packets. Two recent works, cfNoC [16] and HRCnet [17], optimize the request and reply network, respectively, as previously described in the paper. We propose the CD-Xbar NoC which better fits a GPU's unique traffic pattern while being more area- and power-efficient than and similarly scalable as mesh-based NoCs. Moreover, CD-Xbar is better scalable and more flexible than the state-of-the-art cfNoC and HRCNet topologies while improving performance per Watt by 17 percent.

CPU NoC. A GPU NoC bears essential differences and opportunities compared to a multicore CPU NoC. Next to connecting all CPU cores with the LLC slices and memory controllers, a CPU NoC also needs to connect all CPU cores with each other to support cache coherence, memory consistency and synchronization [50]. Scalable CPU NoC solutions, such as mesh, Clos or butterfly, are suboptimal solutions for GPUs in terms of chip area and power consumption because of the unique GPU traffic pattern, as detailed in this paper. Other CPU NoCs such as CNoC [36], Slim NoC [38], Flattened Butterfly [51], Kilo-Core [52], concentrated mesh [53] and Kilo-NoC [54], all face similar issues in the context of a GPU. The many-to-few-to-many GPU traffic suggests a crossbar network topology, however, scaling a crossbar to large SM counts and a large number of memory nodes is problematic. Buffered and pipelined crossbars have been proposed [10], [55]; swizzle-switch improves crossbar scalability by distributing the centralized arbiter to each crosspoint [11]. However, these proposals assume a fully connected crossbar which GPUs do not need to support the many-to-few-to-many traffic pattern. Various routing policies have been proposed for CPUs [23], [51], however round-robin routing exploits the characteristics of the CD-Xbar topology and is shown to outperform previously proposed routing policies including source-based and (randomized) adaptive routing.

CTA Scheduling. Recent work focuses on increasing the number of SMs beyond a single chip. The Multi-Chip Module GPU design (MCM-GPU) aggregates several GPU modules in a single package [48]. NUMA-aware GPUs achieve performance scalability by exploiting a multi-socket GPU design [49]. How to connect the SMs to the LLC slices and memory controllers within a chip in a scalable way remains unexplored in this prior work. Several proposals optimize CTA scheduling to control thread-level parallelism per SM [25], to fit the multi-GPU system [56] or to exploit inter-CTA locality [24], [57]. None of these prior proposals however notice that traditional CTA scheduling policies may cause unbalanced network traffic in GPUs.

7 Conclusion

The increasing number of SMs in modern-day GPUs poses a major challenge for the network-on-chip that connects the SMs to the LLC slices and memory controllers. In this paper, we propose the converge-diverge crossbar network by exploiting the observation that, because of the many-to-few-to-many traffic pattern, there is no need to directly connect all SMs to the LLC slices as done in a fully-connected crossbar. CD-Xbar features local crossbars that connect a group of SMs to a smaller number of converged ports, which are then connected to the LLC slices through a global crossbar. Converged ports provide routing path diversity; round-robin routing is employed to reduce flit contention on the converged ports. Topology-aware CTA scheduling balances network traffic among the different local crossbars. Our experimental results report that CD-Xbar improves performance by 13.9 percent on average compared to a mesh with the same bisection bandwidth, while at the same time reducing NoC active silicon area and power consumption by 52.5 and 48.5 percent, respectively. We find that CD-Xbar performs within 2.9 percent of an idealized fully-connected crossbar. In addition, we demonstrate CD-Xbar's scalability, flexibility, and improved performance per Watt (by 17 percent on average) compared to state-of-the-art, highly-customized GPU NoCs.

Acknowledgments

We thank the anonymous reviewers for their valuable feedback. This work is supported by the European Research Council (ERC) Advanced Grant agreement No. 741097, FWO projects G.0434.16N and G.0144.17N, Natural Science and Engineering Research Council of Canada, NSFC under Grant No. 61572508, 61672526 and 61802427, NUDT Research Project No. ZK17-03-06, Science and Technology Innovation Project of Hunan Province under Grant 2018RS3083. Xia Zhao is supported through a CSC scholarship and UGent-BOF co-funding.

References

[1] Nvidia. NVIDIA GP100 Pascal Architecture. White paper. 2016. [Online]. Available: http://www.nvidia.com/object/pascal-architecture-whitepaper.html

[2] Nvidia. NVIDIA Tesla V100 GPU Architecture The Worlds Most Advanced Data Center GPU. White paper. 2017. [Online]. Available: http://www.nvidia.com/object/volta-architecture-whitepaper.html

[3] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar, "A 5-GHz mesh interconnect for a teraflops processor," *IEEE Micro*, vol. 27, no. 5, pp. 51–61, Sep. 2007.

[4] R. Das, S. Narayanasamy, S. K. Satpathy, and R. G. Dreslinski, "Catnap: Energy proportional multiple network-on-chip," in *Proc. Int. Symp. Comput. Archit.*, Jun. 2013, pp. 320–331.

[5] S. Borkar, "Thousand core chips: A technology perspective," in *Proc. Des. Autom. Conf.*, Jun. 2007, pp. 746–749.

[6] B. K. Daya, C. H. O. Chen, S. Subramanian, W. C. Kwon, S. Park, T. Krishna, J. Holt, A. P. Chandrakasan, and L. S. Peh, "SCORPIO: A 36-core research chip demonstrating snoopy coherence on a scalable mesh NoC with in-network ordering," in *Proc. Int. Symp. Comput. Archit.*, Jun. 2014, pp. 25–36.

[7] A. Bakhoda, J. Kim, and T. M. Aamodt, "Throughput-effective on-chip networks for manycore accelerators," in *Proc. Int. Symp. Microarchitecture*, Dec. 2010, pp. 421–432.

[8] G. Chen, M. Anders, and H. Kaul, "Scalable crossbar apparatus and method for arranging crossbar circuits," U.S. Patent 9,577,634, Feb. 21, 2017. [Online]. Available: https://www.google.com/patents/US9577634

[9] G. Passas, M. Katevenis, and D. Pnevmatikatos, "VLSI microarchitectures for high-radix crossbar schedulers," in *Proc. Int. Symp. Netw.-Chip*, May 2011, pp. 217–224.

[10] G. Passas, M. Katevenis, and D. Pnevmatikatos, "A 128 x 128 x 24Gb/s crossbar interconnecting 128 tiles in a single hop and occupying 6% of their area," in *Proc. Int. Symp. Netw.-Chip*, May 2010, pp. 87–95.

[11] K. Sewell, R. G. Dreslinski, T. Manville, S. Satpathy, N. Pinckney, G. Blake, M. Cieslak, R. Das, T. F. Wenisch, D. Sylvester, D. Blaauw, and T. Mudge, "Swizzle-switch networks for many-core systems," *IEEE J. Emerging Sel. Topics Circuits Syst.*, vol. 2, no. 2, pp. 278–294, Jun. 2012.

[12] J. Lee and H. Kim, "TAP: A TLP-aware cache management policy for a CPU-GPU heterogeneous architecture," in *Proc. Int. Symp. High Perform. Comput. Archit.*, Feb. 2012, pp. 1–12.

[13] A. K. Ziabari, J. L. Abellán, Y. Ma, A. Joshi, and D. Kaeli, "Asymmetric NoC architectures for GPU systems," in *Proc. Int. Symp. Netw.-Chip*, Sep. 2015, pp. 25:1–25:8.

[14] C. Sun, C. H. O. Chen, G. Kurian, L. Wei, J. Miller, A. Agarwal, L. S. Peh, and V. Stojanovic, "DSENT - A tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling," in *Proc. Int. Symp. Netw.-Chip*, May 2012, pp. 201–210.

[15] H. Kim, J. Kim, W. Seo, Y. Cho, and S. Ryu, "Providing cost-effective on-chip network bandwidth in GPGPUs," in *Proc. Int. Conf. Comput. Des.*, Sep. 2012, pp. 407–412.

[16] X. Zhao, S. Ma, Y. Liu, L. Eeckhout, and Z. Wang, "A low-cost conflict-free NoC for GPGPUs," in *Proc. Des. Autom. Conf.*, Jun. 2016, pp. 34:1–34:6.

[17] X. Zhao, S. Ma, C. Li, L. Eeckhout, and Z. Wang, "A heterogeneous low-cost and low-latency ring-chain network for GPGPUs," in *Proc. Int. Conf. Comput. Des.*, Oct. 2016, pp. 472–479.

[18] P. Kumar, Y. Pan, J. Kim, G. Memik, and A. Choudhary, "Exploring concentration and channel slicing in on-chip network router," in *Proc. Int. Symp. Netw.-Chip*, May 2009, pp. 276–285.

[19] N. Agarwal, D. Nellans, E. Ebrahimi, T. F. Wenisch, J. Danskin, and S. W. Keckler, "Selective GPU caches to eliminate CPU-GPU HW cache coherence," in *Proc. Symp. High Perform. Comput. Archit.*, Mar. 2016, pp. 494–506.

[20] M. Jun, D. Woo, and E. Chung, "Partial connection-aware topology synthesis for on-chip cascaded crossbar network," *IEEE Trans. Comput.*, vol. 61, no. 1, pp. 73–86, Jan. 2012.

[21] L. M. Ni and P. K. McKinley, "A survey of wormhole routing techniques in direct networks," *Comput.*, vol. 26, no. 2, pp. 62–76, Feb. 1993. [Online]. Available: http://dx.doi.org/10.1109/2.191995

[22] M. H. Cho, M. Lis, K. S. Shim, M. Kinsy, T. Wen, and S. Devadas, "Oblivious routing in on-chip bandwidth-adaptive networks," in *Proc. Int. Conf. Parallel Archit. Compilation Techn.*, Sep. 2009, pp. 181–190.

[23] J. Kim, W. J. Dally, and D. Abts, "Adaptive routing in high-radix clos network," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, Nov. 2006, pp. 7:1–7:11.

[24] M. Lee, S. Song, J. Moon, J. Kim, W. Seo, Y. Cho, and S. Ryu, "Improving GPGPU resource utilization through alternative thread block scheduling," in *Proc. Int. Symp. High Perform. Comput. Archit.*, Feb. 2014, pp. 260–271.

[25] O. Kayıran, A. Jog, M. T. Kandemir, and C. R. Das, "Neither more nor less: Optimizing thread-level parallelism for GPGPUs," in *Proc. Int. Conf. Parallel Archit. Compilation Techn.*, Sep. 2013, pp. 157–166.

[26] Y. Ukidave, C. Kalra, D. Kaeli, P. Mistry, and D. Schaa, "Runtime support for adaptive spatial partitioning and inter-kernel communication on GPUs," in *Proc. Int. Symp. Comput. Archit. High Perform. Comput.*, Oct. 2014, pp. 168–175.

[27] J. T. Adriaens, K. Compton, N. S. Kim, and M. J. Schulte, "The case for GPGPU spatial multitasking," in *Proc. Symp. High-Perform. Comput. Archit.*, Feb. 2012, pp. 1–12.

[28] NVIDIA CUDA SDK code samples, [Online]. Available: https://developer.nvidia.com/cuda-downloads. NVIDIA Corporation, May 2011.

[29] AnandTech. Hot Chips 2016: Nvidia GP100 Die Shot Released. 2016. [Online]. Available: https://www.anandtech.com/show/10588/hot-chips-2016-nvidia-gp100-die-shot-released

[30] A. Ceyhan, M. Jung, S. Panth, S. K. Lim, and A. Naeemi, "Impact of size effects in local interconnects for future technology nodes: A study based on full-chip layouts," in *Proc. IEEE Int. Interconnect Technol. Conf.*, May 2014, pp. 345–348.

[31] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *Proc. Int. Symp. Workload Characterization*, Oct. 2009, pp. 44–54.

[32] B. He, W. Fang, Q. Luo, N. K. Govindaraju, and T. Wang, "Mars: A MapReduce framework on graphics processors," in *Proc. Int. Conf. Parallel Archit. Compilation Techn.*, Oct. 2008, pp. 260–269.

[33] S. Grauer-Gray, L. Xu, R. Searles, S. Ayalasomayajula, and J. Cavazos, "Auto-tuning a high-level language targeted to GPU codes," in *Proc. Innovative Parallel Comput.*, May 2012, pp. 1–10.

[34] J. A. Stratton, C. Rodrigues, I.-J. Sung, N. Obeid, L.-W. Chang, N. Anssari, G. D. Liu, and W.-M. W. Hwu, "Parboil: A revised benchmark suite for scientific and commercial throughput computing," *Center for Reliable and High-Performance Computing*, vol. 127, 2012.

[35] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt, "Analyzing CUDA workloads using a detailed GPU simulator," in *Proc. Int. Symp. Perform. Anal. Syst. Softw.*, Apr. 2009, pp. 163–174.

[36] Y. H. Kao, N. Alfaraj, M. Yang, and H. J. Chao, "Design of high-radix clos network-on-chip," in *Proc. Int. Symp. Netw.-Chip*, May 2010, pp. 181–188.

[37] L. Chen, L. Zhao, R. Wang, and T. M. Pinkston, "MP3: Minimizing performance penalty for power-gating of Clos network-on-chip," in *Proc. Symp. High Perform. Comput. Archit.*, Feb. 2014, pp. 296–307.

[38] M. Besta, S. M. Hassan, S. Yalamanchili, R. Ausavarungnirun, O. Mutlu, and T. Hoefler, "Slim NoC: A low-diameter on-chip network topology for high energy efficiency and scalability," in *Proc. 23rd Int. Conf. Archit. Support Program. Lang. Operating Syst.*, Mar. 2018, pp. 43–55.

[39] G. Koo, Y. Oh, W. W. Ro, and M. Annavaram, "Access pattern-aware cache management for improving data utilization in GPU," in *Proc. Int. Symp. Comput. Archit.*, Jun. 2017, pp. 307–319.

[40] S. Eyerman and L. Eeckhout, "System-level performance metrics for multiprogram workloads," *IEEE Micro*, vol. 28, no. 3, pp. 42–53, May/Jun. 2008.

[41] Z. Wang, J. Yang, R. Melhem, B. Childers, Y. Zhang, and M. Guo, "Simultaneous multikernel GPU: Multi-tasking throughput processors via fine-grained sharing," in *Proc. Symp. High Perform. Comput. Archit.*, Mar. 2016, pp. 358–369.

[42] Q. Xu, H. Jeon, K. Kim, W. W. Ro, and M. Annavaram, "Warped-slicer: Efficient intra-SM slicing through dynamic resource partitioning for GPU multiprogramming," in *Proc. 43th Int. Symp. Comput. Archit.*, Jun. 2016, pp. 230–242.

[43] H. Jang, J. Kim, P. Gratz, K. H. Yum, and E. J. Kim, "Bandwidth-efficient on-chip interconnect designs for GPGPUs," in *Proc. Des. Autom. Conf.*, Jun. 2015, pp. 9:1–9:6.

[44] P. Lotfi-Kamran, B. Grot, and B. Falsafi, "NOC-Out: Microarchitecting a scale-out processor," in *Proc. Int. Symp. Microarchitecture*, Dec. 2012, pp. 177–187.

[45] W. W. L. Fung, I. Sham, G. Yuan, and T. M. Aamodt, "Dynamic warp formation and scheduling for efficient GPU control flow," in *Proc. Int. Symp. Microarchitecture*, Dec. 2007, pp. 407–420.

[46] R. Ausavarungnirun, J. Landgraf, V. Miller, S. Ghose, J. Gandhi, C. J. Rossbach, and O. Mutlu, "Mosaic: A GPU memory manager with application-transparent support for multiple page sizes," in *Proc. Int. Symp. Microarchitecture*, Oct. 2017, pp. 136–150.

[47] A. ElTantawy and T. Aamodt, "Warp scheduling for fine-grained synchronization," in *Proc. Symp. High Perform. Comput. Archit.*, Feb. 2018, pp. 375–388.

[48] A. Arunkumar, E. Bolotin, B. Cho, U. Milic, E. Ebrahimi, O. Villa, A. Jaleel, C.-J. Wu, and D. Nellans, "MCM-GPU: Multi-chip-module GPUs for continued performance scalability," in *Proc. Int. Symp. Comput. Archit.*, Jun. 2017, pp. 320–332.

[49] U. Milic, O. Villa, E. Bolotin, A. Arunkumar, E. Ebrahimi, A. Jaleel, A. Ramirez, and D. Nellans, "Beyond the socket: NUMA-aware GPUs," in *Proc. Int. Symp. Microarchitecture*, Oct. 2017, pp. 123–135.

[50] N. E. Jerger, T. Krishna, and L. Peh, *On-Chip Networks: Second Edition*. San Mateo, CA, USA: Morgan and Claypool Publishers, 2017.

[51] J. Kim, W. J. Dally, and D. Abts, "Flattened butterfly: A cost-efficient topology for high-radix networks," in *Proc. Int. Symp. Comput. Archit.*, Jun. 2007, pp. 126–137.

[52] N. Abeyratne, R. Das, Q. Li, K. Sewell, B. Giridhar, R. G. Dreslinski, D. Blaauw, and T. Mudge, "Scaling towards kilo-core processors with asymmetric high-radix topologies," in *Proc. Int. Symp. High Perform. Comput. Archit.*, Feb. 2013, pp. 496–507.

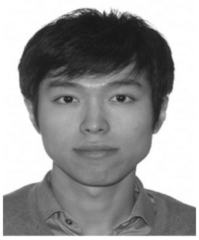
[53] J. Balfour and W. J. Dally, "Design tradeoffs for tiled CMP on-chip networks," in *Proc. Int. Conf. Supercomput.*, Jun. 2006, pp. 187–198.

[54] B. Grot, J. Hestness, S. W. Keckler, and O. Mutlu, "Kilo-NOC: A heterogeneous network-on-chip architecture for scalability and service guarantees," in *Proc. 38th Annu. Int. Symp. Comput. Archit.*, Jun. 2011, pp. 401–412.

[55] G. Kornaros, "BCB: A buffered CrossBar switch fabric utilizing shared memory," in *Proc. EUROMICRO Conf. Digital Syst. Des.: Archit. Methods Tools*, Aug. 2006, pp. 180–188.

[56] G. Kim, M. Lee, J. Jeong, and J. Kim, "Multi-GPU system design with memory networks," in *Proc. Int. Symp. Microarchitecture*, Dec. 2014, pp. 484–495.

[57] A. Li, S. L. Song, W. Liu, X. Liu, A. Kumar, and H. Corporaal, "Locality-aware CTA clustering for modern GPUs," in *Proc. Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, Apr. 2017, pp. 297–311.



Xia Zhao received the MS degree in computer science from the National University of Defense Technology (NUDT), Changsha, China, in 2015. He is working toward the fourth year PhD degree at Ghent University, Belgium. His research interests include GPGPU architecture in general, and multi-program execution and Network-on-Chip (NoC) design more in particular.

Sheng Ma received the BS and PhD degrees in computer science and technology from the National University of Defense Technology (NUDT), in 2007 and 2012, respectively. He visited the University of Toronto, from 2010 to 2012. He is currently an associate professor with the College of Computer. His research interests include microprocessor architecture, on-chip networks, SIMD architectures, and arithmetic unit designs. He was a recipient of the Young Talent Development Program of the China Computer Federation, in 2016.

Zhiying Wang received the PhD degree in electrical engineering from the National University of Defense Technology (NUDT), Changsha, China, in 1988. He is a professor with the School of Computer, NUDT. He has contributed more than 10 invited chapters to book volumes, published 240 papers in archival journals and refereed conference proceedings, and delivered more than 30 keynotes. His main research fields include computer architecture, computer security, VLSI design, reliable architecture, multicore memory system, and asynchronous circuit. He is an IEEE member.

Natalie Enright Jerger received the PhD degree in electrical engineering from the University of Wisconsin-Madison. She is the Percy Edward Hart professor of Electrical and Computer Engineering, University of Toronto. Her research interests include computer architecture, approximate computing, interconnection networks, and hardware acceleration of machine learning. She is a senior member of the IEEE and a distinguished member of the ACM.

Lieven Eeckhout received the PhD degree in computer science and engineering from Ghent University, in 2002. He is a full professor at Ghent University, Belgium. His research interests are in the area of computer architecture, with specific interests in performance analysis, evaluation and modeling, and dynamic resource management. He is the recipient of the 2017 Maurice Wilkes Award and the 2007 OOPSLA Most Influential Paper Award. He served as editor-in-chief of the *IEEE Micro* (2015-2018), program chair of HPCA 2015, CGO 2013 and ISPASS 2009, associate editor of the *IEEE Transactions on Computers*, the *IEEE Computer Architecture Letters* and the *ACM Transactions on Architecture and Code Optimization*. He is an IEEE Fellow.