Accelerated Reply Injection for Removing NoC Bottleneck in GPGPUs

Yunfan Li, and Lizhong Chen
School of Electrical Engineering and Computer Science
Oregon State University, Corvallis, OR 97229
{liyunf, chenliz}@oregonstate.edu

Abstract

The high level of parallelism in GPGPUs has resulted in significantly changed on-chip data traffic behaviors. This demands new research to identify and address the limiting factors of networks-on-chip (NoCs) in the context of GPGPUs. In this paper, we quantitatively analyze the performance of on-chip networks in GPGPUs, and address a serious NoC bottleneck where the reply data from memory controllers experience large contention when being injected to the reply network. To remove this reply injection bottleneck, we propose Accelerated Reply Injection (ARI)*, a very effective scheme that can supply a fast rate of data traffic from memory controllers to feed the reply injection points, and accelerates the consumption of the injected packets by quickly transferring the packets out of the injection points, thus increasing both supply and consumption of reply traffic injection. Simulation results on a wide range of benchmarks show that the proposed ARI reduces the data stall time in memory controllers by 67.8% on average, and increases IPC by more than 15.4% on average, with less than 1% area overhead.*

1. Introduction

Continuing innovations in General-Purpose Graphic Processing Units (GPGPUs) have spurred their increasing use in high-performance computing (HPC) systems and data centers to accelerate many scientific, economic and social computing applications. With a high degree of parallelism in contemporary GPGPU architectures, a significant challenge is how to provide effective on-chip networks that can efficiently transfer a large amount of data to and from memory to feed potentially thousands of concurrently running threads. This issue will only get more important as the number of cores integrated on a chip continues to escalate, such as in the NVidia GeForce series where the core count increases from 480 in GTX 480 to 3072 in GTX Titan X in less than 5 years. Thus, it is imperative to investigate the major limiting factors of on-chip networks in the current and future GPGPUs and devise effective approaches to address the limitations.

While on-chip networks (a.k.a. OCNs or NoCs) traditionally in the context of multi-core CPUs have been actively studied, GPGPUs are fundamentally different from CPUs in the way which they handle the long latency of memory subsystems (e.g., NoC latency, DRAM access time). Unlike CPUs that mitigate the impact of memory latency by employing complex cache structures to increase hit rate, GPGPUs hide memory latency by swapping out stalled threads and swapping in independent threads. This fundamental difference in the design principle of GPGPUs has led to many distinct architecture choices, including the ability and need to integrate a large number of simple and small cores to maintain a high throughput of thread execution. As a result, the changes in the volume, intensity and patterns of on-chip data traffic (more on this in Sections 2 and 3) demand different NoC designs to meet the new requirements and constraints. Despite the growing importance of GPGPU NoCs, only a few works have looked at this issue so far [2, 3, 10, 15, 20, 39, 40]. While these works all offer valuable insights, more research along this line is much needed to explore many other unaddressed challenges.

In this paper, we start with quantitatively analyzing the bottleneck of on-chip networks in GPGPUs through extensive detailed simulations. A couple of observations are made that progressively identify the exact place in the NoC where the bottleneck occurs. First, we observe that, although the request network that carries read and write requests has a higher average packet latency than the reply network, the reply network is the actual part that limits the overall flow due to heavier reply traffic. Second, for the reply network, a serious bottleneck happens around the injection points, rather than within the reply network, where the data from memory controllers (either returned from L2 or from DRAM) is injected into the reply network. Furthermore, we identify two key reasons for this reply injection bottleneck: 1) the network interface (NI) does not have a matching architecture that can supply a fast rate of traffic to the injection points, even if the injection links are widened; 2) for packets that arrive at the injection point of a router, the current router architecture cannot consume the injected packets fast enough by transferring these packets out of the router. Without removing the reply injection bottleneck, the throughput and latency of the GPGPU NoC are greatly affected.

To address the injection bottleneck, in this work we propose *Accelerated Reply Injection (ARI)*, an effective scheme that removes the reply injection bottleneck by providing fast and high-throughput injection from both supply and consumption aspects. In the supply aspect, the network interface is augmented with a split NI injection queue structure and widened interconnects, allowing data to reach the injection points at the raw injection rate of memory controllers. In the consumption aspect, ports in crossbar switches are allocated to injection ports in a more balanced way, and a prioritization technique is also proposed to help quickly transfer the injected data packets out of the "hot regions" around memory controllers. The proposed ARI accelerates the reply injection process without increasing the network bisection bandwidth or the off-chip memory bandwidth.

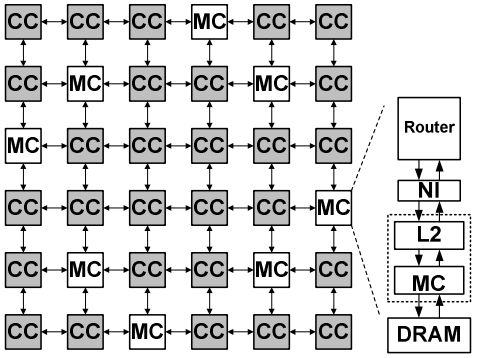

Figure 1: An example GPGPU architecture.

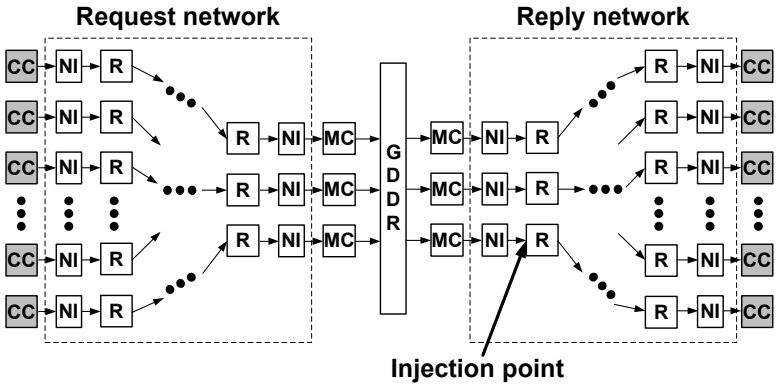

Figure 2: End-to-end traffic flow (CCs on both sides are the same).

The proposed ARI is implemented in RTL as well as in cycle-accurate simulators, and is evaluated using a mix of 30 representative workloads. Simulation results show that ARI is able to reduce the stall time that data waits at memory controllers by 67.8%, on average, and achieve an average IPC improvement of 15.4%, with less than 1% area overhead. This research increases our understanding of on-chip networks in GPGPUs and provides insights on how to design effective techniques to address the reply injection bottleneck.

2. Background and Motivation

2.1 Preliminaries

Without loss of generality, Figure 1 depicts an example of a tile-based GPGPU architecture, and Figure 2 illustrates the end-to-end data traffic flow from sending request packets all the way to receiving reply packets. There are two types of nodes in the system, namely compute nodes and memory controller nodes, as shown in Figure 1. A compute node (CC) usually consists of a cluster of small processing cores. For example, a CC can be a streaming multiprocessor (SM) in NVIDIA's GPUs containing 16 scalar processors (SPs). A memory controller node (MC) often includes a L2 cache bank and a memory controller that interfaces with off-chip GDDR memory. Each compute node or memory controller node connects to a router through a network interface (NI). The NI is responsible for encapsulating the messages sent from CC and MC nodes into packets, and for extracting messages from the packets that are delivered to the CC and MC nodes.

To provide a basis for studying the bottleneck of on-chip networks in GPGPUs, Figure 2 illustrates the end-to-end flow of request and reply packets in the network. When any of the CC nodes has a miss in its local L1 cache, a data request message is generated by the CC and encapsulated by the connected NI into a short request packet. The request packet is sent to the connected router and may be subsequently forwarded by a couple of other routers in the *request network* to reach the router and NI that are associated with an MC node. The NI then extracts the request message from the packet and, if the request misses in the L2 cache bank on the MC node, the memory controller schedules the request and forwards it to the GDDR based on the employed scheduling policy. Later, after the requested data is returned from the main memory to the same MC node[1], the connected NI formats the replied data into a long reply packet and waits for a chance to inject the packet into the *reply network*. The reply packet may be forwarded along several routers in the reply network, and eventually arrives at the router and NI of the CC node that initiates the data request. During the above request-reply process, if the request hits in the L2, the MC node would skip the DRAM step and directly injects the replied data from the L2 cache to the reply network through the NI.

Write request and reply are similar to the above read request and reply, except that write request packets are long packets containing data whereas write reply packets are short. As can be seen, the request and reply on-chip networks play important roles in completing non-local data accesses. As many GPGPU applications have working set that well exceeds the capacity of local cache, there are frequent accesses to non-local data. Thus, efficient on-chip network designs are much need to achieve high performance GPGPUs.

2.2 Related Work

While a large body of related research has been dedicated to improving on-chip networks in the CPU domain, research on GPGPU NoCs is still in its infancy with only a few works. In the CPU domain, prior research has improved various aspects of NoCs, such as topology (e.g., [4, 12, 22]), routing algorithm (e.g., [11, 14, 23, 27, 38]), flow control (e.g., [7, 17, 25, 29, 34, 37]), router microarchitecture (e.g., [16, 21, 31]), resource efficiency (e.g., [9, 13, 30]), power/energy (e.g., [6, 8]), and so on. These works lay a solid foundation for the general designs of on-chip networks. However, due to considerably different constraints, requirements, traffic patterns, and load rate conditions in GPGPUs, new research is much needed to make NoC designs effective in GPU environments.

To date, there are only a couple of works that look at the NoC component in (GP)GPUs [2, 3, 10, 15, 20, 39, 40]. An

[1] This MC node is the same one shown on the left side of the GDDR in Figure 2. We duplicate the MC nodes in the figure to illustrate better the decomposition of the end-to-end process. Similarly, the CC nodes on the leftmost and rightmost of the figure are the same ones.

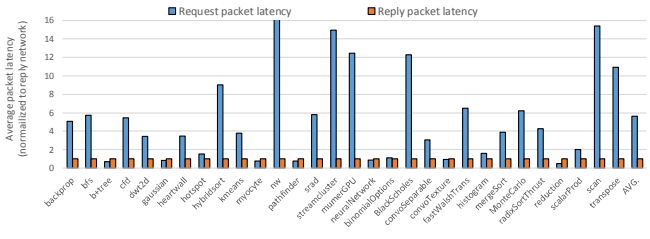

Figure 3: Request vs. reply packet latency.

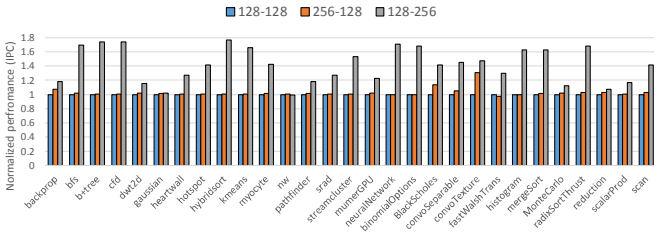

Figure 4: Impact of changing request-request link widths.

initial evaluation of the impact of interconnection networks on GPGPUs is conducted in [2]. The remaining works can be classified in two categories, those optimizing the components inside a network and those optimizing the components on the edge of a network (e.g., injection/ejection). In the first category, a "checkboard" design is proposed in [3] to replace some of the routers in a network with simpler half routers, thereby reducing the cost of GPU NoC. The cost can also be reduced by employing asymmetric designs that remove the unused routers and links in a network based on traffic patterns [39]. It is also possible to use a specific combination of routing algorithms, memory controller placement, and a virtual channel monopolization technique to increase bandwidth utilization [15]. Another design is proposed in [20] that provides a direct all-to-all overlay (DA2mesh) network using multiple dedicated, narrow networks and higher frequency. While the above pioneering works in this category provide useful insights on the design of NoCs for GPGPUs, they mostly optimize the components inside a network.

The second category of prior research targets the components on the edge of a network. Besides our work, the only related work to our knowledge is [3] where a router with multiple injection ports is proposed to accelerate injection. However, as shown later, this technique addresses only some factors of the injection process and has limited effectiveness. We compare it to our scheme quantitatively in the evaluation.

Another related line of research is to leverage silicon photonic interconnects to provide high bandwidth for GPGPUs [10, 40]. While this approach is promising, it is quite different from our targeted electrical NoCs. Outside the GPU NoC domain, almost all the proposed techniques more or less affect L1, L2 and memory accesses, which may potentially change the NoC traffic (e.g., likely increased NoC traffic in cache bypassing [18], and likely reduced NoC traffic in WarpPool [24]). We address this issue by evaluating benchmarks that have varying NoC traffic intensity and sensitivity (high, medium and low) to approximate the effect, although it would be interesting to compare with some of them in future work.

3. Understanding GPGPU NoC Bottleneck

In this section, we present a set of experiments which, collectively, study the interactions among NoC components in GPGPUs to increase the understanding of potential bottlenecks in the network. We use GPGPU-Sim [2] and BookSim 2.0 [19] with a wide range of representative workloads from the Rodinia [5] benchmark suite and CUDA SDK [35] (more details in the Evaluation Methodology section).

First, on the high level, the NoCs in the end-to-end flow in Figure 2 can be divided into a request network and a reply network (denoted by the dotted squares in the figure). We use results in Figures 3, 4 and 5 to show that, due to the backpressure from the reply network to the request network, the bottleneck of GPGPU NoC is on the reply network side.

Figure 3 compares the average packet latency of the request and reply network for a typical GPGPU configured with 28 compute nodes, 8 MC nodes, and 128-bit NoC link width. At the first glance, with the request packet latency being 5.6X of the reply packet latency, on average, it seems that the bottleneck is on the request network side. However, when we double the link width of the request network from 128-bit to 256-bit, the average IPC only increases by 0.8%, as shown in Figure 4. In contrast, if the reply network link width is doubled, a 25.6% increase in the average IPC is observed. This indicates that the reply network is the actual limiting factor.

To explain why the reply network is more congested, Figure 5 examines the relative percentage of four packet types that coexist in GPGPU NoCs, taking into account the number of flits each packet type has to accurately reflect the network traffic load. Among the four packet types, read-request and write-reply are typically short packets; whereas read-reply and write-request are long packets with multiple flits, as they contain large chunk of data. Although each read (write) request packet in the request network corresponds to exactly one read (write) reply packet in the reply network, there are considerably more read transactions than write transactions in most of the benchmarks as shown in the figure. As a result, the reply network, which predominantly carries long read-reply packets, has much more traffic load than the request network (e.g., 72.7% vs. 27.3% of the total NoC traffic in Figure 5), thus being more prone to congestion. When the reply network is congested, its injection buffer queues in NIs would be gradually filled up (the NIs in the right half of Figure 2). This slows down the processing and forwarding of data at the MC nodes to the NIs which, in turn, slows down the processing of requests at the MC nodes on the request network side. Consequently, request packets start to be queued up backward across the routers in the request network, until the backpressure eventually propagates all the way back to the source CC nodes. Analogous to the parking lot problem with a congested exit point, the cars that are the farthest from the exit experience the longest waiting time. Similarly, packets

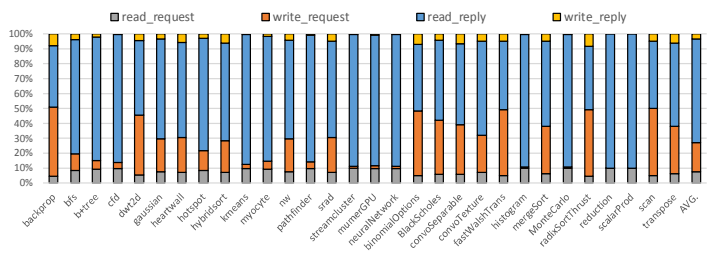

Figure 5: Relative percentage of 4 packet types.

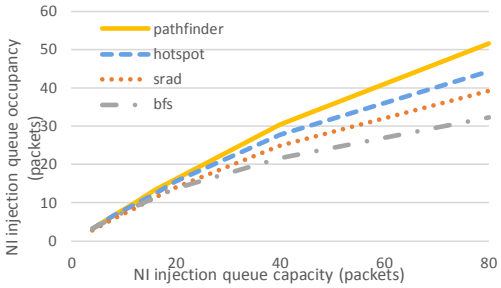

Figure 6: NI injection queue occupancy.

in the request network experience longer NoC latency even though the congestion happens in the reply network.

It is important to note that our simulation does *not* artificially create the congestion in Figure 3 by using 128-bit NoC links. The 128-bit link width is sufficiently wide to match the memory traffic. We show this at two levels. First, at the per-MC level, each connected GDDR5 is modeled after GTX980. According to the specification [36], it operates at 1.75GHz with 32-pin and quadruple data rate. This offers up to 28GB/s of incoming data to an MC (1.75GHz × 32b × 4 = 28GB/s). Meanwhile, each NoC link can transfer up to 128b × 1GHz = 16GB/s of data from an MC. This leads to a total outgoing data from an MC to be 48GB/s (i.e., 3 links from an MC that is located on the edge) or 64GB/s (4 links for a non-edge MC), which is more than the incoming data to the MC. Second, at the aggregate level, the total incoming data from all the 8 MCs is 28GB/s × 8 = 224GB/s (again, matched with the product specification [36]). Prior work has shown that the bisection bandwidth of the NoC need to be around 80% of the total MC bandwidth [3], which is 224GB/s × 0.8 = 179.2GB/s in this case. With 128-bit links, the bisection bandwidth of the simulated NoC is 128b × 1GHz × 12 (12 uni-directional links in the bisection of a 6×6 mesh) = 192GB/s, which is larger than the needed NoC bisection bandwidth. The above calculations indicate that there must be other factors that cause the congestion in the reply network.

Next, we examine the reply network more closely to identify which part of the network is the bottleneck that limits the traffic flow. Simulation results of running a mix of 30 benchmarks show that, the average link utilization of the reply network is actually very low, with only 0.084 flit/cycle. However, the average link utilization of the injection links (i.e., the links from NIs to the connected routers in Figure 2) is 0.39 flits/cycle which is more than 4.5X the utilization of the links within the reply network. This indicates that the injection points can be the potential bottleneck. To verify this is indeed the case, we intentionally increase the capacity of the injection buffer queues in the NIs and record the queue occupancy. Essentially, if the injection point is the bottleneck and limits the traffic flow, as the queue capacity increases, more reply packets would be buffered in the injection queues, waiting to be injected. Figure 6 plots the results and confirms that the queue occupancy closely tracks the queue capacity as it increases from the size of 4 long read-reply packets to 80 long packets. Other benchmarks exhibit similar characteristic and are omitted in the figure for clarify.

The main reason why the injection point is the bottleneck is that the reply network has a few-to-many traffic pattern caused by the high CC-to-MC ratio (e.g., 28 vs. 8). All the reply data that needs to be distributed back to the many CC nodes is forwarded through only a few injection points. This is further worsened by the large number of long read-reply packets in the reply network, thus concentrating the already heavy traffic to only a few paths from the NIs to the connected routers. The reply injection bottleneck of GPGPU NoCs may seriously limit the maximum achievable throughput of the traffic flow. It also increases the packet latency by blocking critical reply data at the injection points that is needed by programs to make forward progress. Nevertheless, simply increasing the width of injection links is not enough to remove this bottleneck, as the injection links are not isolated components but are closely interacting with NIs on the one end and routers on the other end. Both ends are not fully capable of handling such an increased traffic from wide injection links, if conventional CPU NoC designs are used. What is needed is a matching NI architecture that can *supply* a fast rate of traffic to the injection points, and a matching router architecture that can *consume* the injected packets by quickly transferring packets out of the injection points, as proposed in this work.

4. Removing Reply Injection Bottleneck

In this section, we propose *Accelerated Reply Injection (ARI)*, an effective scheme that removes the reply injection bottleneck by providing fast and high-throughput injection process from both supply and consumption aspects.

4.1 Accelerating Injection Traffic Supply

To fully utilize the reply network capacity, it is important to provide a fast supply rate of injection traffic to the reply network. To accelerate this injection supply, the key is to remove the mismatch among the width of various links and interconnects along the datapath. Figure 7(a) zooms in on the region around the reply injection in Figure 2. Because the to-be-injected data in the MC can be acquired by fetching from the memory (the "GDDR" shown in the figure) or by hitting in the L2 cache bank (not shown), it is possible to have multiple back-to-back ready data in the MC in consecutive cycles. The default baseline in GPGPU-Sim assumes a narrow link (e.g., 128-bit) between the MC and the NI, which causes unnecessary slowdown since it takes multiple cycles to transfer

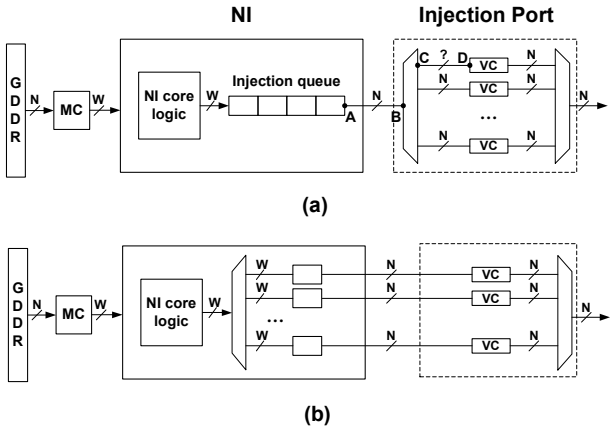

Figure 7: NI architecture: (a) enhanced baseline; (b) proposed.

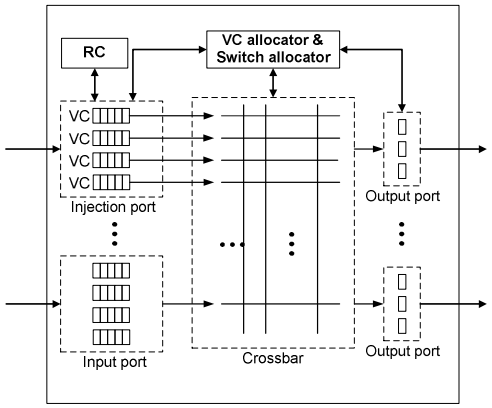

Figure 8: Injection port crossbar speedup.

a data (e.g., 1024-bit) from the MC to the NI. Therefore, we fix this issue by using an enhanced baseline where the link between the MC and the NI and the link between the NI and the injection buffer queue are wide links (e.g., 1024-bit). Note that this does not change the number of pins between the GDDR and the MC. It only widens the link between the MC and the NI, both of which are on the same on-chip tile with abundant intra-tile wiring resource. With this width adjustment, when a requested data is ready in the MC node, the data can be forwarded to the NI through a wide link in just one cycle. In the figure, we use *W* to denote the width of wide links (e.g., 1024-bit) that can transfer the entire data in one cycle, and use *N* to denote the width of narrow links that have the same width as the main reply network (e.g., 128-bit). For the remainder of this paper, all our analysis and evaluation are based on this enhanced baseline, in order to avoid giving unfair advantage to our proposed design.

Once a data is forwarded to the NI, the core logic in the NI formats the data into a standard packet which is then divided into flits. We use a wide link *W* to connect the core logic and the injection buffer queue, so the multiple flits belonging to the same packet can be transferred to the queue in just one cycle. This allows the core logic to continue processing a new data from the MC every cycle, assuming the injection buffer queue is not full. However, the bottleneck of the design in Figure 7(a) is the link *AB* between the NI and the connected router injection port. In conventional NoC designs, this link typically has the same width as the links in the reply network and is thus a narrow link. As a result, each cycle only one flit can be forwarded from the NI injection queue to one of the available virtual channels (VCs) in the injection port of the router. This rate is not fast enough as a data packet contains multiple flits. Therefore, additional changes are needed to increase the supply rate of injection traffic.

A natural thought would be to increase the width of the bottleneck link *AB* from *N* to *W* bits. While more wires will be needed to address this bottleneck (explained shortly), directly increasing the width may cause the syncing issue between link *AB* and link *CD* in the router injection port. This is because, if link *CD* is a normal narrow link, it has to be clocked at a much faster frequency to avoid losing information from the wide link *AB*. Implementing such a fine-grained clocking at per-link level is not practical around the simple multiplexer between point *BC*. Alternatively, if link *CD* is a wide link, the whole injection port that includes multiple of these wide links would be prohibitively larger than a normal input port and would be extremely difficult for efficient wire routing (e.g., considering a typical case with *W* being 1024-bit and *N* being 128-bit). Furthermore, regardless of whether link *CD* is narrow or wide, directly widening link *AB* requires the VC size to be at least as large as a data packet, as the *W* bits of a packet from link *AB* all come in one cycle and the VC has to be large enough to accept all the bits in case downstream routers are not available. This is a strong restriction as smaller VC sizes may be preferred for wormhole switching in area-constrained on-chip environment.

To circumvent these issues in widening link *AB*, we propose a revised design as depicted in Figure 7(b). First, the NI injection queue is split into multiple smaller injection queues, and a multiplexer is added to distribute data packets. The links from the core logic to the injection queues are wide links, so a data packet from the core logic can be forwarded to an injection queue in one cycle. Second, multiple narrow links are used to connect injection queues directly to the VCs in the injection port of the router. The multiplexer in the injection port prior to the VCs is removed since separate links are used instead of having to multiplex on one link (this does not change the fact that the NI decides which VC to use for every injecting packet at the source node). In this design, it is safe to use narrow links owing to the buffering functionality of injection queues. Specifically, the entire data packet can be written into an injection queue through a wide link in one cycle, and later individual flits can be read out through a narrow link in multiple cycles, thus removing the need for different clocks. Each split injection queue needs to be minimally one-packet-sized large, which can be easily met under the same total NI buffer size, because the original NI injection queue is usually large enough to hold multiple packets.

With the above modifications, if there are [*W*/*N*] number of narrow links, the design is able to process a long data per

cycle from the MC node and supply that same rate of injection traffic to the router. In practice, as the MC node likely does not generate a data every cycle, fewer narrow links can be used without blocking any reply data from the MC node.

4.2 Accelerating Injection Traffic Consumption

With more packets being supplied to the VCs of the router injection port, the next step is to accelerate the consumption of injected packets, i.e., quickly transferring these packets out of the VCs and the router, so as to keep accepting new packets from the connected NI.

We first illustrate the potential issues in timely forwarding injected packets in a conventional router design. To facilitate discussion, an *MC-router* is defined to be the router that directly connects to an MC node through an NI. In the example of a 2D mesh network, the 5×5 crossbar switch in an MC-router assigns one input port of the switch (referred to as a *switch-port* hereafter) to the router injection port and assigns 4 switch-ports to the remaining four router input ports. This creates two serious mismatches. First, the injection port and the regular input ports are treated equally in this way with similar physical switch resource. However, as analyzed previously in Section 3, the injection port has 4.5X as much traffic load as a regular input port and thus should be matched with more switch resource. Second, with the enhancement on the supply side, the probability of having multiple flits ready in the same cycle in the injection port VCs greatly increases. When connects to only one switch-port, the injection port can send at most one flit to traverse the switch in each cycle, leading to a mismatch between the injection traffic supply rate and consumption rate.

We address these mismatch issues by using an appropriately determined crossbar speedup for the injection port. The crossbar speedup is traditionally defined to be the number of switch-ports per router input port. Here, we extend the definition in the sense that only the injection port uses the crossbar speedup instead of all the router input ports. As exemplified in Figure 8, multiple switch-ports can be used for an injection port, e.g., directly connecting one switch-port to one VC if the speedup is the same as the number of VCs, N_{VC}. If the crossbar speedup is smaller, de-multiplexers can be added between the VCs and the switch-ports. There is no need to provide switch-ports more than N_{VC}, as the injection port has at most N_{VC} flits ready per cycle.

More of a challenge is to determine how much injection port speedup is needed. A too small speedup may not be sufficient to consume the injected packets in time, and a too large speedup may unnecessarily incur crossbar overhead without bringing substantial performance benefit. In general, in order to have enough capacity to consume injected packets, the speedup S should satisfy:

$$S \geq InjRate_{pkt} \times \bar{N}_{flits\_per\_pkt} \quad (1)$$

where the packet injection rate corresponds to an ideal rate that would have been achieved if the consumption side is perfect and does not block any injection packet (which can be approximated by simulating a reply network with unlimited bandwidth). The average number of flits per packet is calculated by taking into account the size and frequency of read and write packets in the reply network. If the speedup meets (1), the crossbar will have the capacity to consume the injected packets supplied by the NI and injection links, provided that there is no contention from other ports.

Meanwhile, the injection port speedup does not need to be larger than the number of non-local output ports, as the latter is an upper-bound on the number of packets that can be forwarded out of a router in a given cycle. Together with the fact that the speedup does not need to be larger than the number of virtual channels in the injection port, we have

$$S \leq min\{N_{out}, N_{VC}\} \quad (2)$$

Combining (1) and (2) and the need to minimize resource overhead, the general guideline is to choose the minimal integer S_{min} that meets (1), if the S_{min} also meets (2). Otherwise, if the S_{min} does not satisfy (2), the bound in (2) has to be used for the speedup, which implies that some injected packets may be delayed because of the limited consumption capacity.

In the 2D mesh example, if all the non-local output ports and VCs are fully utilized, we are targeting at a bound of 4 in (2). Across the benchmarks evaluated in this paper, we observed that 95% of the peak packet injection rates (calculated using per 100-cycle intervals under perfect consumption) can be satisfied by this bound, indicating that this is a good trade-off point. Therefore, an injection port speedup of 4 is used in our main evaluation, and the impact of speedup on different number of VCs are studied in sensitivity analysis. Note that, only the crossbars in the MC-routers are changed, while non-MC-routers (majority of all the routers) and all the routers in the request networks are not affected.

5. Exploiting Traffic Load Disparity

The architectural changes in the previous section not only allow reply data to reach the injection points at the raw injection rate of the MC nodes, but are also able to forward the data packets from the injection points to downstream routers at a matching rate. More importantly, this is achieved without increasing the bisection bandwidth of the network or the number of pins to the off-chip memory. However, although the injection port speedup allows up to S flits to be transferred out of an MC-router, the packets in other input ports of the router may also compete with the packets in the injection port for the same output ports.

To mitigate this type of contention, we propose to exploit the traffic load disparity between injecting packets and in-network packets. Notice that the reply network, on average, has relatively low traffic load. This means that although packets may experience a long delay during injection, the in-network packets are likely to experience less contention, particularly at non-MC-routers. In this regard, in order to accelerate the forwarding of injecting packets at the MC-routers, it may be beneficial to give higher priority to injecting packets and lower priority to in-network packets. Generalizing on this intuition, we propose a multi-level prioritization technique. A packet has the highest priority when it is being injected at the

Table I. Key Parameters for Evaluation.

Parameters	Value	Parameters	Value
Compute Nodes	28, 1126 MHz	GDDR5 Memory Timing	t_{RP}=12, t_{RC}=40, t_{RRD}=6, t_{RAS}=28, t_{RCD}=12, t_{CL}=12
Memory Controllers	8, FR-FCFS		
Warp Size	32	Memory Clock	1.75 GHz (GTX980)
SIMD Pipeline Width	8	Topology	2D Mesh 6x6 (4x4, 8x8)
# of CTAs / Core	8	Routing	XY, Min. adaptive
Shared Memory / Core	48KB	Interconnect & L2 Clock	1 GHz
L1 Cache Size / Core	16KB	Virtual channel	4 per port, 1 pkt per VC
L2 Cache Size / MC	128KB	Allocator	Separable Input First
Warp Scheduling	Greedy-then-oldest	Link bandwidth	128 bit/cycle
MC placement	Diamond	NI injection queue	36 flits

MC-routers, and the priority gradually decreases as the packet is forwarded further into the network.

To implement this technique, a priority field is added to the packet header. This field is initially set to the highest level, *L*, when a packet is generated. Then, the route computation unit will decrease the value of the priority field by 1 for every packet that it performs route computation for. Over time, the priority of any injected packet will be decreased to 0.

To determine how many levels of priority are needed, we compare the performance improvement by varying the number of priority levels. Figure 9 shows the results for the bfs and mummerGPU benchmarks. As can be seen, most of the performance benefits can be reaped by using just two levels (i.e., 0 and 1). In fact, too many levels of priority may not necessarily lead to higher improvement. This is because, as the packets are forwarded far away from the injection nodes, it is not really needed to differentiate among the in-network

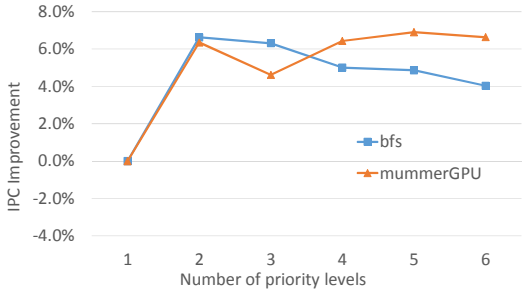

Figure 9: Performance of different priority levels.

packets. Based on this observation, a two-level prioritization is employed and a 1-bit priority field is used. Starvation can be avoided by setting the priority bit to 0 for packets in the injection port if packets in other input ports have been waiting for longer than a pre-determined threshold. A threshold of 1k cycles is used in our evaluation, but starvation of this kind is rare, and our further simulation shows that the overall performance is very insensitive to the threshold value.

6. Evaluation Methodology

6.1 RTL Implementation and Cost

While the proposed ARI removes the bottleneck in supplying and consuming injection traffic, it is also important to understand the cost of ARI. To estimate the overhead more accurately, we implement ARI in Verilog HDL. All the key modified or additional components are modeled, including various widened links, added multiplexers, split NI injection queues (with the same total buffer size but split structures), extra ports in crossbar switch, newly needed control logics, and so on. A standard VLSI design flow is followed. Synthesis is based on Synapsis Design Compiler, using 45nm NanGate open cell libraries [32] and FreePDK45 process from NCSU [33]. P&R is performed in Cadence Encounter. Other important parameters are listed in Table I.

Simulation results show that a revised pair of NI and MC-router incurs an area overhead of 5.4%, and the final amortized area overhead is 0.7% when taking into account the whole network. The small amortized overhead is owing to the fact that the proposed ARI does not need to change any routers in the request network or any of the non-MC-routers in the reply network. The design also does not increase NoC bisection bandwidth or off-chip bandwidth.

6.2 Architecture Level Simulation

The proposed ARI design is also evaluated quantitatively at the architecture level. Cycle-accurate GPGPU-Sim [2] integrated with BookSim 2.0 [19] are used for detailed functional and timing simulation of various components in the GPGPU and on-chip network. The power and energy estimation of the GPGPU part is based on GPUWattch [26], while the NoC part is replaced with more accurate power results from RTL simulation in Cadence Encounter, based on network activity statistics collected from runtime. Significant effort has been made to modify GPGPU-Sim and BookSim to implement the key additional components mentioned in Section 6.1 and to be consistent with the Verilog HDL implementation. Table 1 lists the main configuration parameters. A 6×6 mesh network is used for most of the simulations while other network sizes are also evaluated in scalability analysis. A diamond memory controller placement [1] is used to make a competitive baseline. A wide range of 30 benchmarks from Rodinia [5] and NVidia CUDA SDK [35] are evaluated. To provide a good coverage, we include a mix of benchmarks that have varying sensitive to the NoC (9 highly sensitive, 11 medium, and 10 low).

We compare the following schemes: (1) XY-Baseline: XY routing with the enhanced baseline described in Section 4.1; (2) XY-ARI: XY routing with the proposed ARI design;

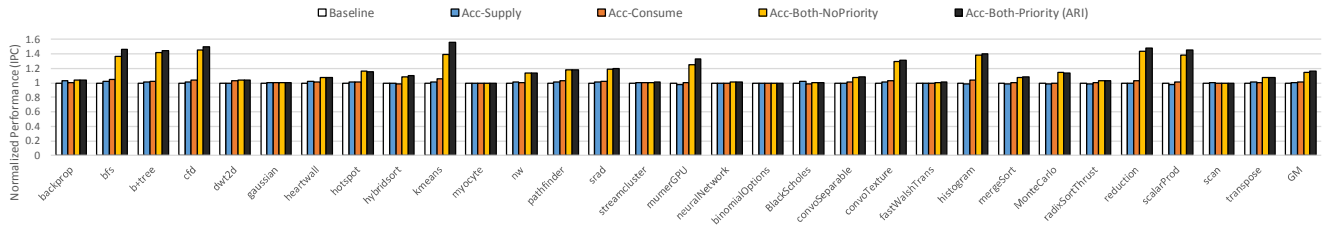

Figure 10: Effects of accelerating injection supply and consumption separately and combined.

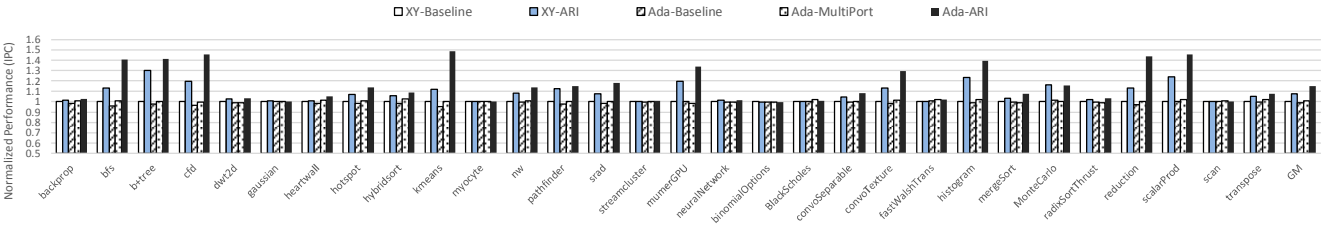

Figure 11: Performance comparison for different schemes.

(3) Ada-Baseline: minimal adaptive routing with the above baseline; (4) MultiPort [3]: an existing scheme with multiple injection ports between an NI and the connected router. This scheme increases injection traffic supply in a different way than ARI, but does not address injection traffic consumption. (5) Ada-ARI: minimal adaptive routing with the ARI design.

To provide fair comparison, the total size of the split NI injection queues in schemes (2) and (5) is the same as the single NI injection queue size in schemes (1), (3) and (4). Also, non-atomic buffer allocation [7] is used for both XY and adaptive routing (enabled by the WPF technique [28]) to utilize the VC resource better.

7. Results and Analysis

7.1 Effect of Accelerating Supply and Consumption

Before comparing with other schemes, it is important to study the relative impact of the injection supply and consumption processes and the effectiveness of the individual components in our proposed ARI. Figure 10 compares the normalized IPC of the evaluated benchmarks under several scenarios, all having adaptive routing. As can be seen, accelerating injection supply or consumption alone (Acc-Supply and Acc-Consume, respectively) has only minimal impact on performance. Interestingly, Acc-Supply actually slightly reduces the performance for 12 out of the 30 benchmarks. This is because allowing more traffic to be supplied at the injection points but not accelerating the consumption causes more packets to be stalled at injection points of the reply network, thus exacerbating the congestion and increasing packet delay.

Significant IPC improvement is achieved when both injection supply and consumption are accelerated. The no priority version improves performance by 13.5%, on average (geometric mean). The proposed binary priority technique also reaps additional performance benefits in most benchmarks, some having larger IPC improvement (e.g., additional 9% in bfs and 16.1% in kmeans on top of Acc-Both-NoPriority) and some having smaller increase. The varying impact is mainly due to the difference in the severity of contention between injecting packets and in-network packets in MC-routers among the benchmarks. In the remainder of this section, the proposed ARI is evaluated as a whole and compared with other schemes.

7.2 Impact on Performance

To assess the effectiveness of the proposed ARI and its compatibility with different routing algorithms, Figure 11 plots the IPC of five schemes normalized to XY-Baseline. Under XY routing, XY-ARI achieves an average of 8% performance increase compared with XY-Baseline, and the specific degree of improvement for a benchmark greatly depends on its sensitivities to on-chip networks.

When adaptive routing is used in the baseline, it is observed that the performance actually decreases a bit (Ada-Baseline vs. XY-Baseline). This is because, on the consumption side, without increasing the switch-ports in crossbar for injection ports, only one flit can be forwarded out of the injection port per cycle, even though adaptive routing may allow multiple flits to go to different output ports without conflict. Meanwhile, on the supply side, using adaptive routing in request network results in more packets to be delivered to the reply injection points. The two sides create a similar situation to the previous Acc-Supply case where the injection contention is intensified rather than alleviated.

With MultiPort, multiple injection ports allows more packets to be injected (i.e., consumed), but the scheme is still limited by the NI capability in supplying traffic. Therefore, the performance is improved but not by a lot, with an average of 2% over Ada-Baseline. In contrast, the proposed ARI increases both injection traffic supply and consumption, thereby providing enough traffic in switch traversal to fully utilize the increased output port options offered by adaptive routing. Compared with Ada-Baseline, Ada-ARI is able to

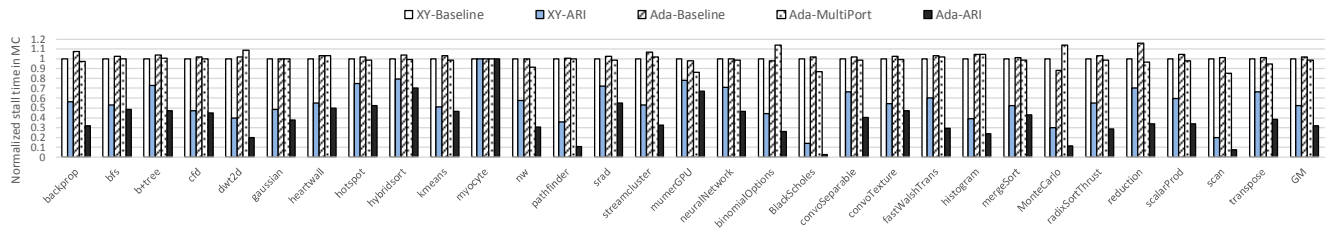

Figure 12: Data stall time in memory controllers due to NI injection queue full.

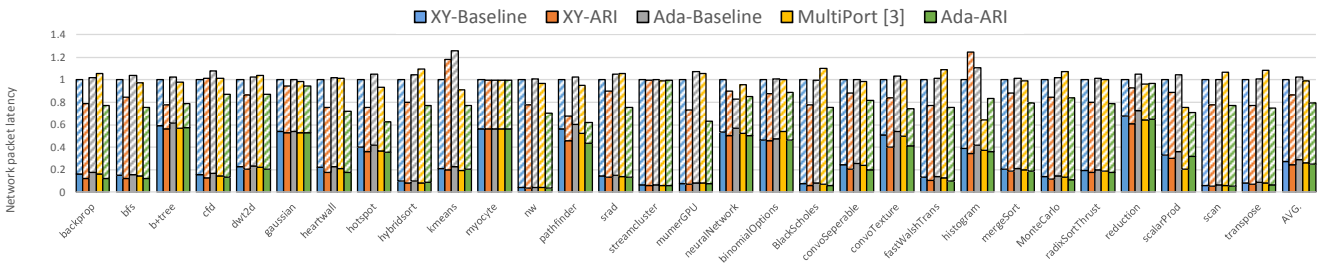

Figure 13: Average packet latency (striped: request packet latency; solid: reply packet latency).

achieve an average IPC improvement of 15.4%, with around a third of the benchmarks having nearly 1.4X IPC increase.

7.3 Effect on Reducing Stalling in MC

To gain more insight on why ARI can achieve such a large overall performance improvement, Figure 12 examines the amount of time that reply data is stalled in MC nodes, waiting to be forwarded to NIs. This happens when the injection queues in the NIs are full and can no longer accept data from the MC nodes. As can be seen from the figure, while Multi-Port reduces the stall time, particularly in mummerGPU and blackScholes, the reduction is not substantial in general. On the other hand, the proposed ARI reduces the stall time remarkably, with an average of 47.5% reduction comparing XY-ARI and XY-Baseline, and an average of 67.8% reduction comparing Ada-ARI and Ada-Baseline.

7.4 Effect on Packet Latency

The previous MC stalling does not include the time that an encapsulated packet waits in the NI injection queue. To demonstrate the effectiveness of ARI in improving this waiting time and in improving NoC latency, Figure 13 shows the average packet latency decomposed into the request and reply parts. The NI injection waiting time is accounted for in the reply packet latency. As ARI removes the reply injection bottleneck, the reply packet latency is reduced as expected. More importantly, the request packet latency also decreases considerably in ARI. Note that ARI does not change anything in the request network. The drop in the request latency further confirms that the GPGPU NoC bottleneck is on the reply side.

7.5 Discussions

(1) Energy. The energy impact of ARI is mostly resulted from the difference in execution time. Figure 14 plots the energy consumption acquired from an appropriately configured GPUWattch, with NoC results from Cadence tools added. The dynamic energy has negligible difference between ARI and baseline, and the static energy of ARI is reduced because of the reduced execution time. Due to the low static energy percentage modeled in the current simulation tools, the overall energy is reduced by around 4%, on average, when comparing ARI with the baseline.

(2) Scalability. As the proposed ARI modifies the NI and router architecture "locally", it does not have any particular design element that limits scalability. In fact, our evaluation shows that, ARI increases the IPC improvement by 3.7%, 15.4% and 24.7% for 4×4, 6×6, and 8×8 networks, respectively. The increased improvement is partly because of the larger impact of NoC latency and throughput in larger chips.

(3) Utilizing Virtual Channels. Figure 15 compares the performance of having 2 VCs and 4 VCs, with and without ARI, for four benchmarks (other benchmarks have similar trends and are omitted here due to space consideration). Injection port crossbar speedup is set to be the same as the number of VCs. Two observations can be made: 1) compared with the baseline, ARI improves performance under the same VC count, i.e., 3rd bar vs. 1st bar, and 4th bar vs. 2nd bar within each bar group; 2) when the VC count increases from 2 to 4, the improvement with ARI is considerably larger than without ARI, i.e., improvement from 3rd to 4th bar vs. from 1st to 2nd bar. This shows that, with the injection bottleneck removed, ARI can fill VCs with more traffic and make full use of the VC resource that would otherwise go underutilized.

(4) Applicability. The proposed ARI has broader applicability than the scenarios evaluated above. To illustrate this further, we apply ARI on top of the DA2mesh [20] scheme mentioned previously in related work. DA2mesh exploits the few-to-many traffic pattern to improve GPU NoC designs, but it does not address the reply injection issue, thus being complementary to ARI. As shown in Figure 16, when ARI is

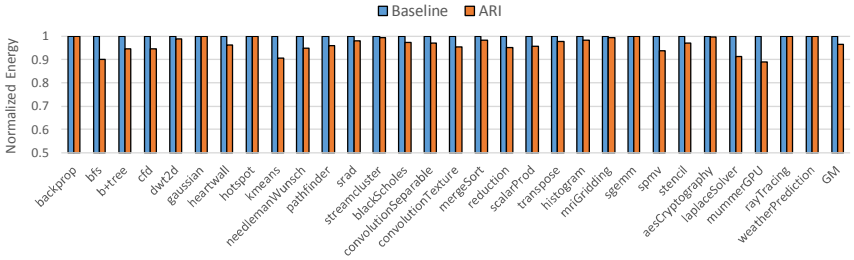

Figure 14: Impact on energy consumption.

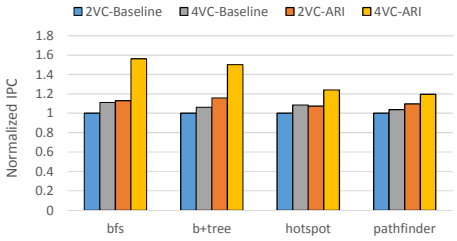

Figure 15: ARI with different virtual channels.

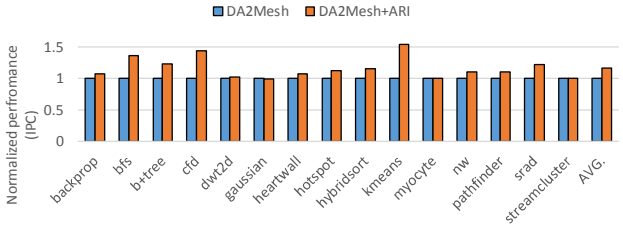

Figure 16: Performance of ARI on top of DA2mesh [20].

used in combination with DA2mesh, an average of 16.4% additional performance improvement can be achieved.

8. Conclusion

With potentially thousands of threads running concurrently in GPGPUs, it is critical to provide effective on-chip networks that can transfer a large amount of data from memory to feed those threads. In this paper, we analyze the performance bottleneck of GPGPU NoCs, and propose an effective scheme, *Accelerated Reply Injection (ARI)*, to remove the reply injection bottleneck. By accelerating both injection traffic supply and consumption through enhanced NI and router architecture, the proposed ARI significantly reduces the data stall time in memory controllers and improves overall performance, while incurring minimal area overhead.

References

[1] D. Abts, N. D. E. Jerger, J. Kim, D. Gibson, and M. H. Lipasti, "Achieving predictable performance through better memory controller placement in many-core CMPs," in *ISCA*, 2009.

[2] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt, "Analyzing CUDA workloads using a detailed GPU simulator," in *ISPASS*, 2009.

[3] A. Bakhoda, J. Kim, and T. M. Aamodt, "Throughput-effective on-chip networks for manycore accelerators," in *MICRO*, 2010.

[4] J. Balfour and W. J. Dally, "Design tradeoffs for tiled CMP on-chip networks," in *ICS*, 2006.

[5] 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 *IISWC*, 2009.

[6] L. Chen and T. M. Pinkston, "NoRD: Node-Router Decoupling for Effective Power-gating of On-Chip Routers," in *MICRO*, 2012.

[7] L. Chen and T. M. Pinkston, "Worm-bubble flow control," in *HPCA*, 2013.

[8] Y. Demir and N. Hardavellas, "SLaC: Stage laser control for a flattened butterfly network," in *HPCA*, 2016.

[9] C. Fallin, C. Craik, and O. Mutlu, "CHIPPER: A low-complexity bufferless deflection router," in *HPCA*, 2011.

[10] N. Goswami, Z. Li, R. Shankar, and T. Li, "Exploring silicon nanophotonics in throughput architecture," *IEEE Design and Test,* vol. 31, pp. 18-27, 2014.

[11] P. Gratz, B. Grot, and S. W. Keckler, "Regional congestion awareness for load balance in networks-on-chip," in *HPCA*, 2008.

[12] B. Grot, J. Hestness, S. W. Keckler, and O. Mutlu, "Express cube topologies for on-chip interconnects," in *HPCA*, 2009.

[13] B. Grot, S. W. Keckler, and O. Mutlu, "Preemptive virtual clock: A flexible, efficient, and cost-effective QOS scheme for networks-on-chip," in *MICRO*, 2009.

[14] M. Hayenga, N. E. Jerger, and M. Lipasti, "SCARAB: A single cycle adaptive routing and bufferless network," in *MICRO*, 2009.

[15] H. Jang, J. Kim, P. Gratz, K. H. Yum, and E. J. Kim, "Bandwidth-Efficient On-Chip Interconnect Designs for GPGPUs," in *DAC*, 2015.

[16] S. Jeloka, R. Das, R. G. Dreslinski, T. Mudge, and D. Blaauw, "Hi-Rise: A High-Radix Switch for 3D Integration with Single-Cycle Arbitration," in *MICRO*, 2015.

[17] N. E. Jerger, L.-S. Peh, M. Lipasti, "Virtual circuit tree multicasting: A case for on-chip hardware multicast support," in *ISCA*, 2008.

[18] W. Jia, K. A. Shaw, and M. Martonosi, "MRPB: Memory request prioritization for massively parallel processors," in *HPCA*, 2014.

[19] N. Jiang, J. Balfour, D. U. Becker, B. Towles, W. J. Dally, G. Michelogiannakis, and J. Kim, "A detailed and flexible cycle-accurate Network-on-Chip simulator," in *ISPASS*, 2013.

[20] H. Kim, J. Kim, W. Seo, Y. Cho, and S. Ryu, "Providing cost-effective on-chip network bandwidth in GPGPUs," in *ICCD*, 2012.

[21] J. Kim, W. J. Dally, B. Towles, and A. K. Gupta, "Microarchitecture of a high-radix router," in *ISCA*, 2005.

[22] J. Kim, J. Balfour, and W. J. Dally, "Flattened butterfly topology for on-chip networks," in *MICRO*, 2007.

[23] M. Kinsy, M. H. Cho, T. Wen, E. Suh, M. Van Dijk, and S. Devadas, "Application-aware deadlock-free oblivious routing," in *ISCA*, 2009.

[24] J. Kloosterman, J. Beaumont, M. Wollman, A. Sethia, R. Dreslinski, T. Mudge, and S. Mahlke, "WarpPool: sharing requests with inter-warp coalescing for throughput processors," in *MICRO*, 2015.

[25] A. Kumar, L.-S. Peh, P. Kundu, and N. K. Jha, "Express virtual channels: Towards the ideal interconnection fabric," in *ISCA*, 2007.

[26] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt, and V. J. Reddi, "GPUWattch: Enabling energy optimizations in GPGPUs," in *ISCA*, 2013.

[27] S. Ma, N. E. Jerger, and Z. Wang, "DBAR: an efficient routing algorithm to support multiple concurrent applications in networks-on-chip," in *ISCA*, 2011.

[28] S. Ma, N. E. Jerger, and Z. Wang, "Whole packet forwarding: Efficient design of fully adaptive routing algorithms for networks-on-chip," in *HPCA*, 2012.

[29] G. Michelogiannakis, J. Balfour, and W. J. Dally, "Elastic-buffer flow control for on-chip networks," in *HPCA*, 2009.

[30] T. Moscibroda and O. Mutlu, "A case for bufferless routing in on-chip networks," in *ISCA*, 2009.

[31] R. Mullins, A. West, and S. Moore, "Low-latency virtual-channel routers for on-chip networks," in *ISCA*, 2004.

[32] NanGate. http://www.nangate.com/, 2016.

[33] NCSU. http://www.eda.ncsu.edu/wiki/FreePDK45:Contents, 2016.

[34] C. A. Nicopoulos, D. Park, J. Kim, N. Vijaykrishnan, M. S. Yousif, and C. R. Das, "ViChaR: a dynamic virtual channel regulator for network-on-chip routers," in *MICRO*, 2006.

[35] NVIDIA. *CUDA SDK*. https://developer.nvidia.com/cuda-code-samples, 2015.

[36] NVIDIA. http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-980/, 2016.

[37] L.-S. Peh, W. J. Dally, "Flit-reservation flow control," in *HPCA*, 2000.

[38] A. Singh, W. J. Dally, A. K. Gupta, and B. Towles, "GOAL: A load-balanced adaptive routing algorithm for torus networks," *ISCA*, 2003.

[39] A. K. Ziabari, J. L. Abellán, Y. Ma, A. Joshi, and D. Kaeli, "Asymmetric NoC Architectures for GPU Systems," in *NOCS*, 2015.

[40] A. K. K. Ziabari, J. L. Abellán, R. Ubal, C. Chen, A. Joshi, and D. Kaeli, "Leveraging silicon-photonic noc for designing scalable gpus," in *ICS*, 2015.