Packet Pump: Overcoming Network Bottleneck in On-Chip Interconnects for GPGPUs*

Xianwei Cheng[1], Yang Zhao[2], Hui Zhao[1], Yuan Xie[2]
[1]Department of Computer Science and Engineering, University of North Texas
[2]Department of Electrical and Computer Engineering, University of California, Santa Barbara
xianweicheng@my.unt.edu,yang_zhao@ucsb.edu,hui.zhao@unt.edu,yuanxie@ucsb.edu

ABSTRACT

In order to fully exploit GPGPU's parallel processing power, on-chip interconnects need to provide bandwidth efficient data communication. GPGPUs exhibit a many-to-few-to-many traffic pattern which makes the memory controller connected routers the network bottleneck. Inefficient design of conventional routers causes long queues of packets blocked at memory controllers and thus greatly constrained the network bandwidth. In this work, we employ heterogeneous design techniques and propose a novel decoupled architecture for routers connected with memory controllers. To further improve performance, we propose techniques called Injection Virtual Circuit and Memory-aware Adaptive Routing. We show that our scheme can effectively eliminate NoC bottleneck and improve performance by 78% on average.

CCS CONCEPTS

• **Computer systems organization** → **Interconnection architectures**;

KEYWORDS

Network-on-Chip, GPGPU, Bandwidth

1 INTRODUCTION

As we are approaching the exascale era, GPUs have been widely adopted by the high-performance computing community as "throughput processors". GPUs are able to exploit massive degree of parallelism at thread level which makes them especially cost effective [8, 17]. Parallel programming models such as CUDA and OpenCL reduce the complexity of programming on GPU platforms and further propelled the development of GPU systems. In order to fully take advantage of GPU's parallel computing power, on-chip networks must be able to deliver large amount of data between the compute cores and main memory in a timely manner.

Network-on-Chips have been widely accepted as the backbone for on-chip communication in CMPs [6, 7, 13, 18]. However, except for a handful of works [2, 10–12, 14, 19], the design of NoC for GPUs has not been well examined. Data communication in GPUs has unique features that require special design considerations. To avoid protocol deadlocks, GPU NoCs are usually designed as two physical networks (request and reply network), handling read and write messages separately. In the request network, all compute cores send requests to the few memory controllers and generate a many-to-few traffic pattern. In the reply network, memory controllers send reply messages to compute cores in a few-to-many pattern. The traffic load between these two networks is heavily unbalanced, with reply network carrying about 70% of total traffic [10]. It has been shown that improved design in the reply networks can lead to significant performance gain in the overall NoC [12, 19]. Thus enhancing the reply network performance becomes a research focus.

Due to GPU's many-to-few-to-many traffic pattern, routers connected to memory controllers often get congested and become network bottlenecks. Such bottlenecks cause compute cores to stall because the data they need cannot be injected into the network, even if rest of the routers have very low traffic load [2]. Prior work proposed techniques to reduce network confliction or decrease the cost [10, 12, 19]. However, these techniques are proposed to solve the problem at network level, not at router level. Bottleneck routers connected with memory controllers are not provided with more resource or special tailored design to alleviate congestion. As a result, these routers are not capable to meet the injection demand from memory, while routers connected with compute cores get over provisioned regarding their small injection load.

In this work, we propose to overcome the on-chip interconnect bottleneck by employing heterogeneous router architectures for GPUs. We made the following contributions:

(1) We propose to design routers using heterogeneous architectures and allocate more resource to memory connected routers which are the network bottleneck. To improve injection bandwidth, we propose a decoupled router architecture that divides a memory router into two functional modules: Routing Module and Injection Module. The proposed router architecture can effectively eliminate blocking among injected packets. As far as we know, we are the first one to propose heterogeneous router architecture to eliminate bottleneck in GPGPU NoCs.

(2) We propose a scheme called Injection Virtual Circuit (IVC) to further improve injection bandwidth of memory connected routers. IVC can accelerate packet traversal and thus reduce congestion near memory controllers.

(3) We develop a light-weight adaptive routing algorithm called Memory-aware Adaptive Routing. The proposed algorithm routes

*Yang Zhao and Yuan Xie are supported in part by NSF 1500848.

DAC '18, June 24–29, 2018, San Francisco, CA, USA

ACM ISBN 978-1-4503-5700-5/18/06…$15.00
https://doi.org/10.1145/3195970.3196087

packets based on memory controller activity and can improve network performance by reducing traffic around hotspots.

Our simulation results show the proposed schemes can improve system performance by 78% on the average without increasing network area.

2 MOTIVATION

In order to achieve higher performance, NoC designs need to be tailored to work with GPU's unique traffic pattern. Traffic load of the two networks is unbalanced, with the reply network carrying most of the data packets. This is because there are much more read messages than write messages and read reply messages have much larger payload than other types of messages. In this work, we focus on improving the NoC design by reducing bottlenecks in the reply network. Our baseline GPU architecture consists of 56 streaming multiprocessors (SMs) and 8 memory controllers (MCs). We employ staggered MC placement which is shown to have better performance [10]. The detailed configuration of SMs and MCs can be found in Table 1. We employ a 2D mesh as the baseline architecture for the NoC to connect all the SMs and MCs. Each router has 5 input/output ports equipped with VC buffers for credit-based flow control and employs wormhole switching. The router has two pipeline stages: look-ahead routing computation (RC), VC allocation (VA) and switch arbitration (SA) are performed in the first stage; and switch traversal (ST) is executed in the second stage.

First we quantitatively analyzed the severity of congestion at MC routers. We ran multiple benchmarks on GPUGPU-Sim simulator [2]. Figure 1 shows the average number of packets blocked at the injection queue at MCs in every cycle. On average, there are 13 packets waiting at a MC router's inject queue. This shows current MC router design is not capable to meet the demand of MCs in packet injection. However, it also indicates that there is ample room for performance improvement if we can effectively overcome this bottleneck.

We then investigated output link usage of MC routers and the result is shown in Figure 2. Here we only calculate the usage of an output that has credit to receive a new flit. It is interesting to find that contrary to the long queue of blocked packets, the average output link usage is only about 20%. One major cause is the Head Of Line (HOL) blocking in the injection port. If all VCs in the injection port stall due to blocked downstream routers, flits in the injection queue cannot move to an output port even if it is usable. As shown in Figure 2, if all HOL blockings can be removed as in an ideal case, the output link usage can improve to 39% on average.

An intuitive solution to this problem is to add more injection ports as proposed in [2]. However, conventional router design causes a lot of blockings among the injection ports. As a result, multiple injection ports does not effectively improve injection efficiency due to these blockings. One category of blocking is caused by switch arbitration (SA) when the router selects a VC in the injection ports. Figure 3 shows an example that has 4 injection ports. Without losing generality, we assume no flits from the cardinal input ports are requesting any output link and there are credits available in all four output links. As shown in Figure 3(a), there are flits waiting in VCs of the multiple injection ports heading to all four directions. Ideally, the router should be able to pick a flit heading to each output direction so all output links can be utilized. However, as shown in Figure 3(a), the router can only send flits in the W and S direction. The cause is the SA pipeline stage which arbitrates and grants requests for flits requesting an output port. SA is implemented in two stages: first select a VC from each input port; then for output port, choose one input port to access it. In the first stage, SA usually follows Round Robin to select a VC. SA does not know if flits in the selected VCs are contending for a same output. In this case, two pairs of flits are selected by the SA heading for the W and S output. However, only one from each pair can win the second stage of the SA. As a result, the flits heading for N and E output ports are blocked even these outputs are available. If this example, the real output link usage is only half of the ideal case. This problem can be solved if we organize the flits from each injection port by their desired output as is shown in Figure 3(b). Using this method, blocking is removed because all flits in the same queue head for a same output.

There exists a second type of blocking even after we reorder injectied packets into output mapped queues. This is due to the biased selection of output direction by the routing algorithm. Because of their simplicity, DOR routings are usually employed for NoCs, such as XY routing. However, when the injected packets use DOR routings to select direction of their next hop, one dimension is always heavily favored than others. For example, using XY routing, most packets will be sent to the W and E inject queues in Figure 3. The N and S queues only have packets with their destination in the column of the MC router. This causes low usage of output links in the Y dimension while many packets are blocked in the X dimension at the same time. In our analysis, we found that the output link usage can be further improved to 62% if routing caused blocking is removed. This result is shown in Figure 2.

Through our analysis, we found current router architecture is not efficient in providing high injection bandwidth demanded by GPGPUs. To improve the injection bandwidth, specific mechanisms to remove blocking caused by SA and RC need to be applied to injection ports. However, in a generic router architecture, the inject port is tightly coupled with other the input ports. Changing pipeline design to optimize injection can lead to increased pipeline cycle time and thus affect traffic from other input ports. This necessitates a decoupled design for routing and injection functions in MC routers.

3 PROPOSED PACKET PUMP NOC ARCHITECTURE

3.1 Decoupled Memory Controller Router

In this work, we propose heterogeneous design for routers in GPU NoCs and employ a decoupled router architecture to connect to MCs. A decoupled router is divided into two distinct, independent functional modules: Routing Module and Injection Module. Routing Module routes packets coming from cardinal inputs while the Injection Module handles packets injection. Figure 4(a) depicts the major components of such a decoupled router.

Routing Module: This module is responsible for routing packets from non-local inputs and is depicted in Figure 4(b). Similar to a baseline router, packets routing is performed in 2 pipeline stages. The first stage performs the function of look-ahead routing (RC), virtual channel allocation (VA) and speculative switch allocation (SA), with all three operations executing in parallel. The second stage is crossbar traversal (ST). Because we move injection to the

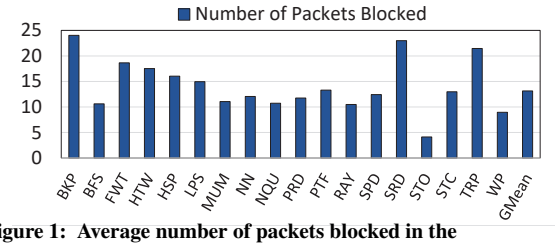


Figure 1: Average number of packets blocked in the inject queue of MC routers.

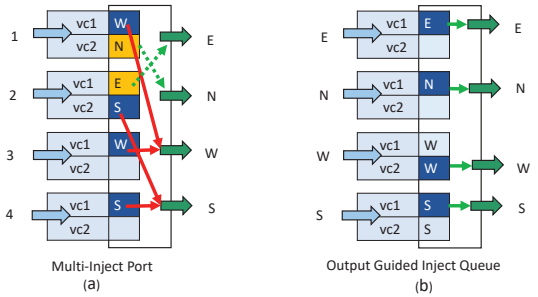


Figure 3: Blocking caused by SA in multiple inject ports. There are 4 injection ports, each with 2 VCs. The dark colored VC is selected by the first stage of SA.

Injection Module, there are 4 inputs and 5 outputs (including local ejection). To simplify the crossbar design, we employ *Early Ejection* technique to remove the local output from the crossbar. Using this technique, a flit destined for a MC is immediately ejected upon arrival to an input port. This mechanism works because look-ahead routing already calculated outport information before a flit arrives. Now the 5x5 crossbar is reduced to a 4x4 crossbar which not only saves area but also shortens the time in switch traversal.

Injection Module: The Injection Module is designed to maximize injection bandwidth to reduce MC bottleneck. Figure 4 (c) depicts the major components of this module. Injection Module takes a more significant role rather than the injection port in a conventional router. We employ *output-mapped queuing* to organize injected flits into a queues mapped to their outputs. There are 4 output-mapped queues each corresponding to a cardinal direction. Instead of multiple injection ports, there is only one injection port in our design. However, we increase the injection channel bandwidth to 4 times of the baseline in order to match the maximum output link usage (one injection flit on each output). All flits in an output-mapped queue head to a same output. Output mapped queues are also divided into VCs. A header flit uses look-ahead routing result to select an output-mapped queue. It also gets assigned a VC by the VA unit in the Injection Module. Then the flits can pass through the demux and head to their output mapped queues. This queuing mechanism amounts to a preliminary switching operation which we call "output-mapped queuing". It has two advantages: (1) The contention in the crossbar switch can be significantly alleviated because now injection flits do not compete with other flits; (2) blocking as described in Figure 3 is greatly reduced. By pre-arranging injection flits to their desired outputs, now flits in a given queue all destine to a same output. The Injection Module undertakes the function similar to VA/Buffer Write stage of the base line router. Because both SA and ST stages are removed now, it takes one cycle for a flit to be moved to its output-mapped queue.

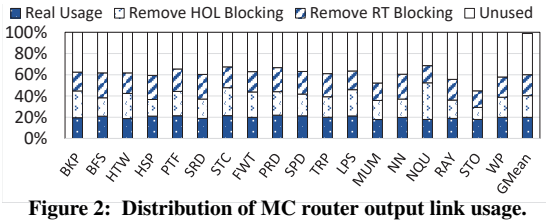


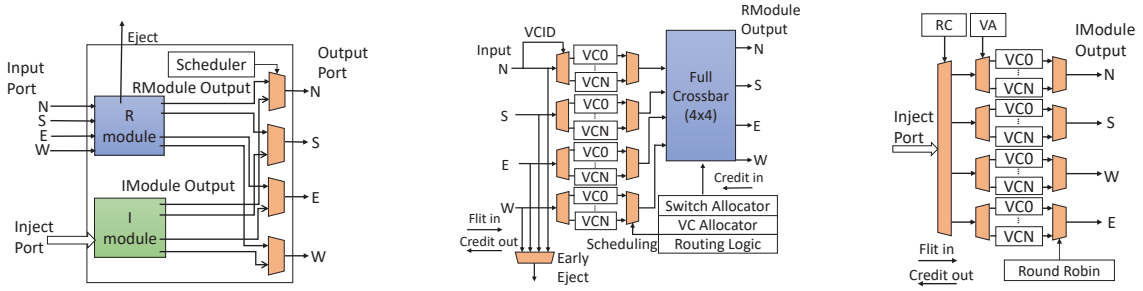
Figure 2: Distribution of MC router output link usage.

Load Balancing in Output-Mapped Injection Queue: Injection Module relies on routing algorithm to assign an output queue to a packet. If MC routers use DOR routing such as XY routing, there will be severe load imbalance in the injection queues. This is because DOR routing heavily favor some dimension over the others. This problem is inherent to the routing algorithm and exists in both the generic router and the proposed MC router. To solve this problem, we developed a scheduling mechanism that takes into account output-mapped injection queue occupancy when choosing an output for a packet. In this scheme, the routing algorithm generates two directions on the minimum-path for a packet. The scheduler selects the output queue with lower occupancy. Since we modified the DOR routing to adaptively send flits to output-mapped injection queues, there is possibility for deadlock. Adding extra VCs is a candidate technique to avoid deadlock in adaptive routing. However, this will increase area overhead in the Injection Module. Instead, we developed an adaptive routing algorithm to solve this problem. Our routing algorithm is based on XY routing but forbids certain turns similar to the odd-even routing [5].

Scheduling between Routing and Injection Modules: Since the routing and injection functions are decoupled in the proposed MC routers, it is possible that both modules send flits to a same output in one cycle. A scheduler is needed to avoid collision. With different design goals, the scheduler can assign different priority to the competing candidates. We use a simple scheduling policy and always give higher priority to the routing module. This is based on two considerations: (1) Considering the long queues of flits waiting for injection, giving higher priority to injection could lead to starvation in the routing module; (2) The probability of routed flits to take over all output links is low. As long as there is credit available for an untaken link and an injection flit is waiting, the Injection Module can send the flit to that link. The Routing and Injection Modules work independently and the critical path of a MC router is determined by the Routing Module. It needs to be noted that there is extra delay at the end of the ST stage which is caused by the mux selecting outputs from the two modules. However, the simplified crossbar in the Routing Module reduces ST latency, thus the added mux delay will not change pipeline cycle time.

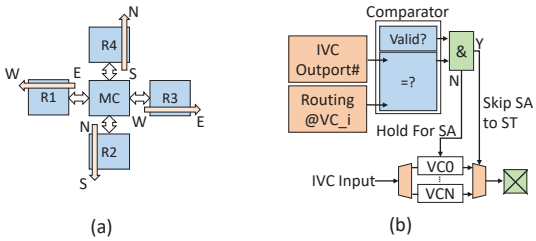
3.2 Injection Virtual Circuit

The decoupled architecture for MC routers can effectively improve injection efficiency by removing blocking among the injected flits. However, packets can still be congested at MC routers if downstream routers have not enough credits to receive new flits. We propose a technique called Injection Virtual Circuit (IVC) to accelerate flit traversal away from MC routers similar to [1]. The idea is to take advantage of communication locality in routers around MCs. This



(a) MC Router. (b) R Module. (c) I Module.

Figure 4: Architecture of decoupled MC router. There are two modules: R Module is responsible for packet routing; I Module is responsible for packet injection.



(a) (b)

Figure 5: IVC direction and control path.

is based on the observation that flits newly injected by a MC tend to share some common path before separately heading to their destinations. Routers on the shared path have more frequent crossbar traversal between certain pair of input and output. Thus the crossbar connection can be reused by later flits to skip the SA stage. We propose to speculatively create virtual circuits on frequently used path around MCs so some flits can directly enter the ST stage, just like traversing a single cycle router. This allows recurrent packets in the IVC to move faster, thereby reducing the probability of chained blocking around MC routers. Adjacent IVC routers have their reused crossbar switch path connected so flits passing through these paths just like traversing on a virtual circuit.

IVC Location: We choose a static mechanism to design IVC, i.e. routers and input/output connection are fixed in IVC. Instead of a dynamic mechanism that all routers need to have IVC support, we only select part of the routers to support IVC. This is due to two reasons:(1) The goal of IVC is to reduce congestion caused by injected packets in routers around MCs. The closer a router to a MC, the higher chance its switch connection is shared by injected packets. As a router's distance from a MC router increases, the probability for this router to be on shared path significantly decreases due to the random destination of packets. Accordingly, the benefit of IVC's congestion alleviation becomes smaller. (2) dynamic setting up of IVC can bring more performance gain, but also incurs more hardware cost and design complexity. Figure 5(a) shows routers 1 hop away in the X or Y dimensions that is equipped with IVC. In such an IVC router, the crossbar connection to be reused is also fixed and only one such pair is used to reduce cost.

IVC Creation: Since the IVC is between one input and output pair, all flits in the same input can reuse the same swtich connection. We only need one bit in the input port to indicate if the switch connection is valid to be used as IVC. Because the output for a router's IVC is fixed, there is no need to save the output number. This mechanism is speculative because we predict a flit coming into the IVC input port will likely traverse to the IVC output port. This prediction is based on the communication locality of the injected flits. Every flit traversal in a router creates a crossbar connection from an input port to an output port. The connection from the IVC designated input and output remains connected for future uses until it is terminated. A flit entering a vc of an IVC router's input first compares its output with the designated IVC output of that router. If they do not match, meaning the flit is not travelling on the IVC, then the flit resets the valid bit and enters the SA stage just like in a normal router. If the comparison returns a match and the IVC is valid, then the flit can enter ST stage directly. Otherwise, the flit set the valid bit to indicate the IVC is created between the designated input and output pair. Later flits arriving in the same input's can reuse the crossbar connection. Figure 5(b) shows the control path of an IVC.

IVC Termination: The reused switch connect in an IVC router is terminated when another flit claim either the input port or the output port. For instance, if another flit from a different input port claims the same output of IVC, then the current IVC is terminated by resetting its valid bit. After termination, flits coming to the IVC input of a router need to go through the SA stage. IVC does not assign the flits high priority in VC arbitration than other flits. Neither does it change the way that a flit is selected for crossbar traversal. It only allows some flits to skip the SA stage by reusing earlier switch connections. If some other flits going through the SA stage claims the output port, then current IVC will be terminated. Thus IVC is able to achieve starvation freedom.

3.3 Memory-aware Adaptive Routing

Prior adaptive routing algorithms are mainly proposed for CMP based NoCs [9, 15]. Traffic pattern in CMPs is random uniform [2] so congestion hotspots are dynamic and hard to predict. Thus very complicated adaptive routing algorithms are needed in order to accurately predict traffic hotspots. Very little research has been proposed on adaptive routing for GPUs. In contrast to CMPs, GPU traffic hotspots are usually generated around memory controllers and are easy to detect. To further improve NoC performance, we propose a light weight adaptive routing algorithm called Memory-aware Adaptive Routing (MAR). Our routing algorithm does not need complicated mechanisms to predict congestion and use memory controller activity to predict congestions. When using MAR to select a route for a packet, we first check if there is a MC router nearby.

MAR uses a pre-defined distance threshold value *H* to search for MC routers. If there is a MC router within *H* hops from current router, then MAR continues to evaluate if the MC is actively injecting packets. This operation is performed by comparing the occupancy of the MC router's injection queue with a threshold value *Th_O*. If the occupancy is greater than *Th_O*, then MAR will select an output direction for the packet that can avoid bumping into the MC. Otherwise, the packet can move on toward the MC. MAR relies on these two parameters in making routing decisions: *H* and *Th_O*. In our current design, we use emperial values to set up these parameters. In MAR, MC routers send to their neighbor information about their inject queue occupancy and this information is piggy backed with regular flits. So no extra control network is needed. Thus proposed MAR is very cost effective in implementation compared to other adaptive routing algorithms.

3.4 Reducing Cost in SM Routers

We have proposed decoupled micro-architecture for MC routers. Compared with MCs, SM routers inject very small amount of write reply messages in the reply network. These messages are very small (one byte) and infrequent. However, conventional designs allocate same amount of buffers to SM routers' injection port as other input ports. We propose to further reduce network cost by removing some injection buffers from SM routers and allocate them to MC routers' injection ports. Our experiment results show this technique has negligible impact on performance while yield savings in network resource at the same time.

4 EVALUATION

4.1 Methodology

We use GPGPU-Sim [3] to simulate our proposed Packet Pump NoC. Our baseline network consists of an 8x8 2D mesh with 56 SMs and 8 MCs. Routers in the baseline network employs conventional 5 ports VC based micro-architecture. Table 1 shows the configuration used in our evaluation. To evaluate the area cost of proposed design, we use Synopsis Design Compiler to evaluate the area of various router architectures. We used UMC LP/RVT (lower power/regular Vth) low-k standard Cell library in 28 nm. MC placement significantly impacts the effectiveness of GPU NoC design. Placing MCs at the edge or bottom will reduce the performance gain of IVC and MAR. However, it has been shown that staggered MC placement achieves better performance [2, 10]. So we choose this type of MC placement in our evaluation similar to these works. We used GPU workloads from Ispass [3], Rodinia [4] and Cuda SDK [16] to evaluate our design and execute the whole applications in our evaluation.

Shader Core	56 cores, 1.4GHz, SIMT width=8
Warp Scheduler	Greedy-Then-Oldest
Shared Memory	48 KB
Cache	2KB L1 I-Cache (4 sets/4 ways LRU), 16KB L1 D-Cache (32 sets/4 ways LRU), 64KB L2 Cache per MC (8 way LRU)
Memory Model	8 MCs, 924 MHz
NoC	128-bit channel width, 2-stage pipeline, 16-byte flits, 1-cycle link latency, X-Y routing, vc buffer depth=4
subnet	2

Table 1: System configuration.

4.2 Performance Analysis

The performance evaluation of our proposed Packet Pump NoC architecture is shown in Figure 6. Most of the benchmarks benefit from our proposed techniques except *NQU* and *STO*. These two benchmarks have very low network activity so they are not sensitive to our optimizations. Among the single techniques applied, IVC can achieve an average of 8% performance gain, with a maximum of 14% for benchmark *LPS*. MAR can bring more benefit in performance with an average improvement of 19%. The maximum IPC improvement is achieved by benchmark *HW*. The most effective single technique is the heterogeneous design using decoupled MC routers. Half of the benchmarks enhanced their performance by two times. On average, system performance is improved by 69%. Combining the three single techniques together, the average performance can achieve a gain of 78%. Performance of some benchmarks gets small amount degradation compared with Decoupled MC routers only, such as *MUM* and *PF*. This is because MAR is based only on local information. These benchmarks have very active memory injections. Without accurate prediction of global congestion, MAR created interference with Decoupled MC router design technique so the overall performance is worse than using Decoupled MC routers only. It can be observed that removing buffers from the injection ports of SM routers has no impact on performance when we compare the last two schemes in Figure 6. This is because SM routers only inject write reply packets with very small amount and they are not on the critical path.

Figure 7 shows the average packet latency compared with baseline. On average, Decoupled MC routers can reduce latency by 30%. The reason is the re-designed MC routers can inject packets in all directions. It is interesting to see that some benchmarks have increased packet latency when Decouplec MC routers are applied, such as *NN*. However, this benchmark's system performance get improved by 70%. This is because this benchmark is more sensitive to throughput than latency [2]. The Decoupled MC router design can significantly improve network throughput by eliminating injection bandwidth constraints. As a result, the overall system performance can improve even though average packet latency is longer.

Next we evaluate the injection efficiency of MC routers and the result is shown in Figure 8. Injection efficiency is calculated as the number of injected flits per cycle per MC router. When applied alone, all techniques can improve injection efficiency. IVC improves injection efficiency by accelerating traversal of newly injected flits. MAR leads to better injection efficiency because contention near MC routers is alleviated by routing packets away. Decoupled MC routers achieve best injection efficiency because blockings are removed by the Injection Module. Combined together, the three techniques can improve injection efficiency by more than two times.

We also evaluate energy consumption of the proposed techniques. It can be observed in Figure 9 that the proposed NoC architecture yields significant energy savings. On average, the network energy is decreased by 22% when all three techniques are combined. Some benchmarks almost save energy consumption by half, such as *SCP* and *BFS*. This is because our techniques can improve network throughput and reduce blockings of data inside the network buffers.

To compare with the state-of-art GPU NoCs, we evaluate our decoupled router design against the work proposed in [2] which has MC router injection ports doubled. The result is shown in Figure 10. Doubling ports can improve performance by 10% averagely while our scheme can achieve 50% performance gain. This is because

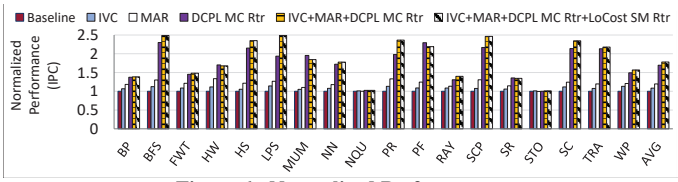


Figure 6: Normalized Performance.

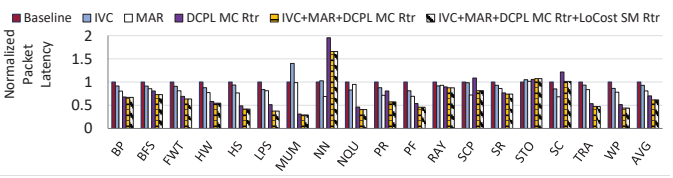


Figure 7: Normalized Packet Latency.

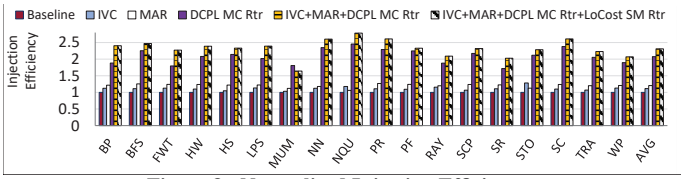


Figure 8: Normalized Injection Efficiency.

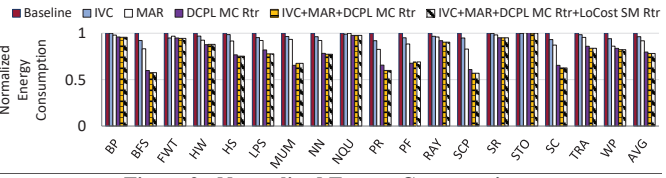


Figure 9: Normalized Energy Consumption.

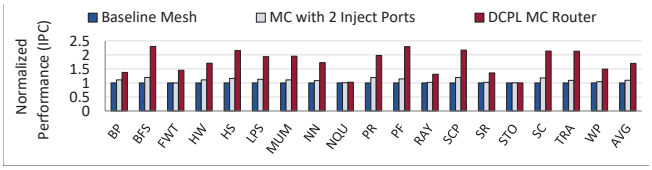


Figure 10: Performance of Decoupled MC Router vs. Double Injection Ports.

our decoupled routers can achieve higher injection efficiency by removing flit blocking.

Finally, we compare the area overhead of proposed techniques with baseline as shown in Table 2. Decoupled MC routers increased buffer size due to the Injection Module. However, the crossbar (4x4) area is smaller in the R Module compared to baseline (5x5). Overall, Decoupled MC routers increases the size of a MC router by 50% and sum of total router area by 8.6%. Using Decoupled MC router and low cost SM router design, we remove half of the buffers in injection ports of SM routers and increase buffers in MC router injection ports. Because there are much more SM routers than MC routers, the overall router area becomes even smaller than the baseline architecture.

	Crossbar Area	Buffer Area	Allocator Area	SM Router Area	MC Router Area	Router Area Sum
Base Line	826.59	14.29×10^3	153.66	16.38×10^3	16.38×10^3	1.05×10^6
Decoupled MC baseline SM	452.28 826.59	22.86×10^3 14.29×10^3	153.66	16.38×10^3	24.58×10^3	1.11×10^6
Decoupled MC LowCost SM	452.28 826.59	22.86×10^3 12.86×10^3	153.66	14.95×10^3	24.58×10^3	1.03×10^6

Table 2: Area Comparison (in um^2).

5 RELATED WORK

Through VC monopolization and employing asymmetric request and reply networks, Jang et al. proposed a bandwidth efficient NoC design [10]. Kim et.al proposed a conflict-free design for the reply network called DA2mesh [12] which assigns each memory node a dedicated channel-sliced network. There are other GPGPU NoC schemes such as asymmetric cmesh [11]. However, these techniques employ homogeneous architectures in designing routers connected with MCs and SMs. Schemes to reduce GPU NoC costs have also been proposed [2, 10]. Their schemes target on minimizing performance degradation when less resource is allocated. In comparison, our scheme improves performance through heterogeneous router design. By carefully provisioning resource, we are able to achieve performance gain without increasing the overall cost.

6 CONCLUSION

In this work, we propose a NoC architecture called Packet Pump to reduce bottleneck caused by MC routers in GPGPUs. We developed a decoupled router micro-architecture for MCs that significantly improves injection efficiency. Two techniques called IVC and Memory-aware Adaptive Routing are also proposed to further improve network performance. Our evaluation shows that Packet Pump can significantly improve system performance without increasing network cost.

REFERENCES

[1] M. Ahn and E. J. Kim. 2010. Pseudo-Circuit: Accelerating Communication for On-Chip Interconnection Networks.. In *MICRO*.

[2] A. Bakhoda, J. Kim, and T.M. Aamodt. 2010. Throughput-Effective On-Chip Networks for Manycore Accecerators. In *MICRO*.

[3] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt. 2009. Analyzing CUDA Workloads Using a Detailed GOU Simulator. In *ISPASS*.

[4] S. Che, M. Boyer, J. Meng, D. Tarjan, J. Sheaffer, S. H. Lee, and K. Skadron. 2009. Rodinia: A Benchmark Suite for Heterogeneous Computing. In *IISWC*.

[5] G. M. Chiu. 2004. The odd-even turn model for adaptive routing. In *IEEE Trans. Parall. Distrib. Syst. 11*. 729–738.

[6] W. Choi, K. Duraisamy, R. Kim, J. Doppa, P. Pande, R. Marculescu, and D. Marculescu. 2016. Hybrid network-on-chip architectures for accelerating deep learning kernels on heterogeneous manycore platforms. In *CASES*.

[7] W. Dally and B. Towles. 2003. Principles and Practices of Interconnection Networks. In *Morgan Kaufmann Publishers Inc*.

[8] N. Goswami, R. Shankar, M. Joshi, and T. Li. 2010. Exploring GPGPU Workloads: Characterization Methodology, Analysis and Microarchitecture Evaluation Implications. In *IISWC*.

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

[10] H. Jang, J. Kim, P. Gratz, K. Yum, and E. Kim. 2015. Bandwidth-Efficient On-Chip Interconnection Designs for GPGPUs. In *DAC*.

[11] A. Kavyan, J. L. Abellan, Y. Ma, A. Joshi, and D. Kaeli. 2015. Asymmetric NoC Architectures for GPU Systems. In *NoCs*.

[12] H. Kim, J. Kim, Wong. Seo, Y. Cho, and S. Ryu. 2012. Providing Cost-effective On-Chip Network Bandwidth in GPGPUs. In *ICCD*.

[13] J. Kim, C. Nicopoulos, D. Park, V. Narayanan, M. S. Yousif, and C. R. Das. 2006. A gracefully degrading and energy-efficient modular router architecture for on-chip networks. In *ISCA*.

[14] Kyung Hoon Kim, Rahul Boyapati, Jiayi Huang, Yuho Jin, Ki Hwan Yum, and Eun Jung Kim. 2017. Packet coalescing exploiting data redundancy in GPGPU architectures. In *ICS*.

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

[16] NVIDIA. 2011. CUDA C/C++ SDK Code Samples. In *http://developer.nvidia.com/cuda-cc-sdk-code-samples*.

[17] G. L. Yuan, A. Bakhoda, and T. M. Aamodt. 2009. Complexity Effective Memory Access Scheduling for Many-Core Accelerator Architectures. In *MICRO*.

[18] H. Zhao, O. Jang, W. Ding, Y. Zhang, M. T. Kandemir, and M. J. Irwin. 2012. A hybrid NoC design for cache coherence optimization for chip multiprocessors. In *DAC*.

[19] X. Zhao, S. Ma, Y. Liu, L. Eeckhout, and Z. Wang. 2016. A low-cost conflict-free NoC for GPGPUs. In *DAC*.