Bandwidth-Efficient On-Chip Interconnect Designs for GPGPUs [*]

Hyunjun Jang[1], Jinchun Kim[2], Paul Gratz[2], Ki Hwan Yum[1], and Eun Jung Kim[1]
[1]Department of Computer Science and Engineering, Texas A&M University, {*hyunjun, yum, ejkim*}*@cse.tamu.edu*
[2]Department of Electrical and Computer Engineering, Texas A&M University, {*cienlux, pgratz*}*@tamu.edu*

ABSTRACT

Modern computational workloads require abundant thread level parallelism (TLP), necessitating highly-parallel, many-core accelerators such as General Purpose Graphics Processing Units (GPGPUs). GPGPUs place a heavy demand on the on-chip interconnect between the many cores and a few memory controllers (MCs). Thus, traffic is highly asymmetric, impacting on-chip resource utilization and system performance. Here, we analyze the communication demands of typical GPGPU applications, and propose efficient Network-on-Chip (NoC) designs to meet those demands. We show that the proposed schemes improve performance by up to 64.7%. Compared to the best of class prior work, our VC monopolizing and partitioning schemes improve performance by 25%.

Categories and Subject Descriptors

C.1.2 [**Computer Systems Organization**]: Multiprocessors-Interconnection architectures

Keywords

Network-on-Chip, Bandwidth, GPGPU

1. INTRODUCTION

General Purpose Graphics Processing Units (GPGPUs) have emerged as a cost-effective approach for a wide range of high performance computing workloads which have a high thread level parallelism (TLP) [10]. GPGPUs are characterized by numerous programmable computational cores which allow for thousands of simultaneous active threads to execute in parallel. The advent of parallel programming models, such as CUDA and OpenCL, makes it easier to program graphics/non-graphics applications, making GPGPUs an excellent computing platform. The growing quantity of parallelism and the fast scaling of GPGPUs have fueled an increasing demand for performance-efficient on-chip fabrics finely tuned for GPGPU cores and memory systems [3, 11].

Ideally, the interconnect should minimize blocking by efficiently exploiting limited network resources such as virtual channels (VCs) and physical channels (PCs) while ensuring deadlock freedom. Networks-on-Chip (NoCs) have been useful in chip-multiprocessor (CMP) environments due to their scalability and flexibility. Although NoC design has matured in this domain [9, 14], NoC design for GPGPUs is still in its infancy. Only a handful of works have examined the impact of NoC design in GPGPU systems [3, 11, 13, 15].

Unlike CMP systems, where traffic tends to be uniform across the cores communicating with distributed on-chip caches, the communication in GPGPUs is highly asymmetric, mainly between many compute cores and a few memory controllers (MCs). Thus the MCs often become hot spots [3], leading to skewed usage of the NoC resources such as wires and buffers. Specifically, heavy reply traffic from MCs to cores potentially causes a network bottleneck, degrading the overall system performance. Therefore, when we design a bandwidth-efficient NoC, the asymmetry of its on-chip traffic must be considered. In prior work [3, 4, 11], the on-chip network is partitioned into two independent, equally divided (logical or physical) subnetworks between different types of packets to avoid cyclic dependencies that might cause protocol deadlocks. Due to the asymmetric traffic in GPGPUs skewed heavily towards reply packets, however, such partitioning can lead to imbalanced use of NoC resources given in each subnetwork. Thus, it fails to maximize the system throughput, particularly for memory-bound applications requiring a high network bandwidth to accommodate many data requests. The throughput-effectiveness is a crucial metric for improving the overall performance in throughput-oriented architectures, thus designing a high bandwidth NoC in GPGPUs is of primary importance. In the GPGPU domain, this is the first study evaluating and analyzing the mutual impacts of different MC placements and routing algorithms on system-wide performance. We observe that the interference from disparate types of GPGPU traffic can be avoided by adopting the *bottom* MC placement with proper routing algorithms, obviating the need of physically partitioned networks.

The contributions of this work are as follows: First, we quantitatively analyze the impact of network traffic patterns in GPGPUs with different MC placements and dimension order routing algorithms. Then, motivated by this detailed analysis, we propose VC monopolizing and partitioning schemes which dramatically improve NoC resource utilization without causing protocol deadlocks. We also investigate the impact of XY, YX, and XY-YX routing algorithms under diverse MC placements. Simulation results show the proposed NoC schemes improve overall GPGPU performance by up to 64.7% over baseline. Compared to the top performing prior work, our VC monopolizing and partitioning schemes achieve a performance gain of 25% with a simple MC placement policy.

2. BACKGROUND

In this section, we describe the baseline GPGPU architecture and NoC router microarchitecture in detail.

2.1 Baseline GPGPU Architecture

A GPGPU consists of many simple cores called streaming multiprocessors (SMs), each of which has a SIMT width of 8. The baseline GPGPU architecture consists of 56 SMs and 8 MCs as shown in Figure 1. Each SM is associated with a private L1 data cache, read-only texture/constant caches, and register files along with a low latency shared memory.

[*]This work was partially supported by NSF CCF-1423433.

DAC '15, June 07 - 11, 2015, San Francisco, CA, USA
Copyright 2015 ACM 978-1-4503-3520-1/15/06 ...$15.00
http://dx.doi.org/10.1145/2744769.2744803.

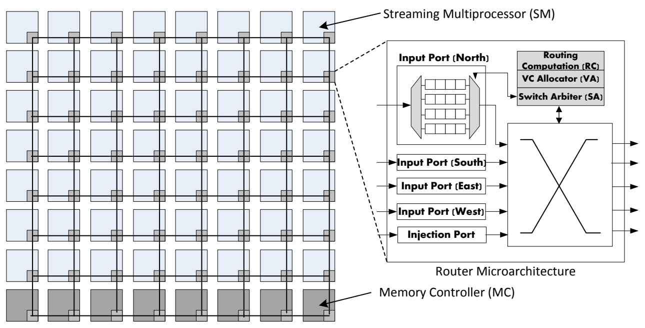


Figure 1: GPGPU NoC Layout and Router Microarchitecture. (The NoC layout consists of many SMs and a few MCs, each of which contains an NoC router.)

Every MC is associated with a slice of the shared L2 cache for faster access to the cached data. We assume write-back polices for both L1 and L2 caches [4], and minimum L2 miss latency is assumed to be 120 cycles. We assume a 2D mesh to connect cores and MCs as in Figure 1 due to its advantages of scalability, simplicity and regularity [3].

2.2 Baseline NoC Router Architecture

Figure 1 shows the baseline NoC router, which has 5 I/O ports to connect the SMs to L2 cache and MCs in a GPGPU. The router is similar to that used by Kumar *et al.* [12] employing several features for latency reduction, including speculation and lookahead routing. Each arriving flit goes through 2 pipeline stages in the router: routing computation (RC), VC allocation (VA), and switch arbitration (SA) during the first cycle, and switch traversal (ST) during the second cycle. Each router has multiple VCs per input port and uses flit-based wormhole switching. Credit-based VC flow control is adopted to provide the backpressure from downstream to upstream routers, which controls flit transmission rate to avoid buffer overflows.

3. DESIGNING BANDWIDTH-EFFICIENT NOCS IN GPGPUS

Here, we analyze the GPGPU workload NoC traffic characteristics and their impact on system behavior. Based on this analysis, we propose VC monopolization and asymmetric VC partitioning to achieve higher effective bandwidth.

3.1 GPGPU On-Chip Traffic Analysis

3.1.1 Request and Reply Traffic

Prior work shows on-chip data access patterns to be more performance critical than data stream size in GPGPUs [7]. Further, these traffic patterns are inherently *many-to-few* (in the request network, from the many cores to the few MCs) and *few-to-many* (in the reply network, from the MCs back to the cores) [3]. As shown in Figure 2 *MC-to-core*, the reply network sees much heavier traffic loads than *core-to-MC*, the request network. This is because the request network consists of many short packets (read requests) mapped into a single flit and fewer long packets (write requests) mapped into 3~5 flits. The reply network consists of many long packets (read reply) mapped into 5 flits and relatively a few short packets (write reply) mapped into a single flit. Figure 3 shows that on average around 63% of packets are read replies. Exceptionally, *RAY*, contains more request packets than reply packets, due to a write demand in this application.

In general, the ratio between request and reply traffic can

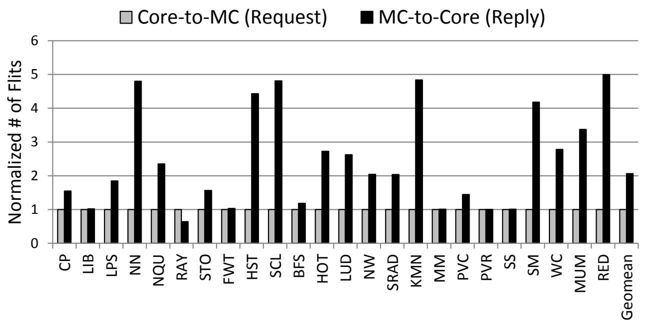


Figure 2: Normalized Traffic Volumes Between Cores and MCs.

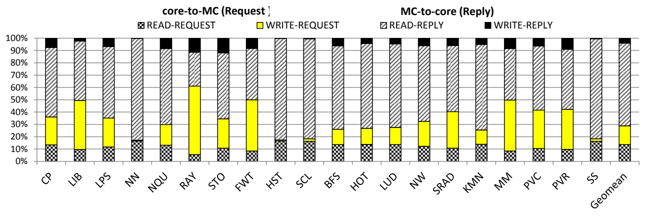


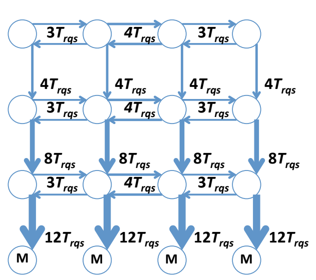
Figure 3: Packet Type Distribution for GPGPU Benchmarks

be derived as follows. Considering the overall injection rate as λ at each node, we denote the ratio of read and write requests by r and w, respectively, and the sum of r and w equals one because request consists of only two types: read and write. The length of each packet can be divided into two groups: a short packet (L_s) representing read request and write reply, and a long packet (L_l) including read reply and write request. The amount of request traffic T_{rqs} is the sum of read and write requests and likewise, the amount of reply traffic T_{rep} is the sum of read and write replies.

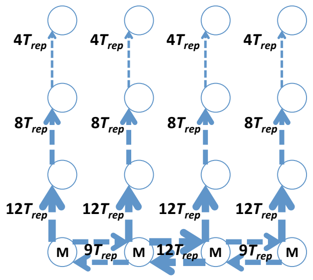
$$\begin{aligned} T_{rqs} &= \lambda \cdot r \cdot \ L_s + \lambda \cdot w \cdot \ L_l \\ T_{rep} &= \lambda \cdot r' \cdot \ L_l + \lambda \cdot w' \cdot \ L_s \end{aligned} \tag{1}$$

where r' and w' denote ratios of replies for read (r) and write (w) requests, respectively. Since a single request is always followed by a single reply, the ratio between r and w is identical to that of r' and w'. Here, the ratio of reply to request (R) is derived by dividing T_{rep} by T_{rqs}. Thus, according to Figure 2, R equals around two since the traffic volume of reply packets is two times higher than that of request packets.

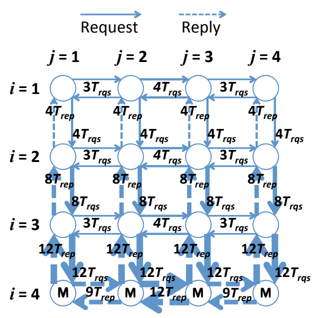
Figure 4 illustrates the traffic, T_{rqs} and T_{rep}, in an (NxN) mesh network with k MCs. In this figure, we take an example of $N = k = 4$, and XY routing with the bottom MC placement. Each arrow represents the direction of traffic and the associated coefficient number denotes the link utilization toward that direction. By multiplying the coefficient with either T_{rqs} or T_{rep}, we can approximate the amount of traffic towards a specific direction. For vertical links, a coefficient value is determined by the row location. For example, each core located at the 1st row (*i=1*) uses its south output port 4 times. If a core sends request packets to N MCs at the bottom row, the south port of a router associated with the core is utilized by N times. Thus, for a core located at the i^{th} row in an (NxN) mesh, the request coefficient towards south direction becomes $N \cdot i$. Similarly, we can utilize the core's column in deriving coefficient values for horizontal links. If a core j is located at the j^{th} column in the mesh network, $N - j$ MCs are on the east side of this



(a) XY Request



(b) XY Reply



(c) XY Request + XY Reply

Figure 4: Network traffic example with XY routing. *(Note that request (a) and reply (b) traffic take different paths, thus traffic does not mix on horizontal and vertical links.)*

core. To access these $N - j$ MCs, all cores located from the 1st to the j^{th} columns must use the east output port of core j. Therefore, the coefficient for east port of j^{th} core is set by $j \cdot (N - j)$. In a similar way, we analyze the coefficient values for all directions as follows.

$$\begin{aligned} C_{south} &= N \cdot i \\ C_{north} &= N \cdot (i-1) \\ C_{east} &= j \cdot (N-j) \\ C_{west} &= (N-j+1) \cdot (j-1) \end{aligned} \tag{2}$$

The quantitative analysis above proves three important facts about the characteristics of GPGPU network traffic. First, reply traffic is much heavier than request traffic as empirically shown in Figures 2 and 3. Second, both request and reply traffic get congested as they approach to the MCs located at the bottom. The growing coefficient values of C_{south} and C_{north} from Equation 2 support this argument. Third, request and reply traffic do not get mixed on horizontal or vertical links as shown in Figures 4(a) and 4(b).

3.1.2 *Memory Controller Placement*

One way of alleviating traffic congestion is to move the MCs. Different MC placements help improve network latency and bandwidth by spreading the processor-memory traffic, balancing the NoC load. While prior work on MC placement shows the performance improvement to be gained from MCs placement in CMPs [2], this work does not analyze how MC placement affects the average hop count for GPGPUs. Here, we conduct a detailed quantitative analysis of GPGPU MC placement policies and show how distributed MC location can improve the NoC efficiency in GPGPUs.

The most clear advantage of distributed MC placement is the reduced average number of hops from cores to MCs, as shown in Figure 5. Comparing with other MC placements, specifically, the diamond MC placement shown in

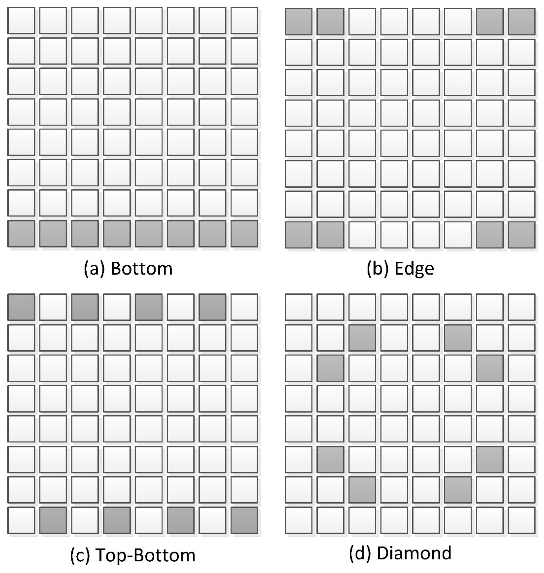


Figure 5: Different MC Placements. Shaded tiles represent MCs co-located with GPGPU cores

Figure 5(d) allows more cores to access MCs with fewer hops. Since we are using dimension order routing, there is only one unique path from each core to a given MC. The row and column number of i^{th} MC are assumed to be $row_{m,i}$ and $col_{m,i}$, respectively. In the same way, the row and column number of j^{th} core are $row_{c,j}$ and $col_{c,j}$. With this notation, for an $(N\text{x}N)$ mesh network with N number of MCs and $(N^2 - N)$ cores, the average number of hops from cores to MCs can be estimated as follows:

$$\begin{aligned} H_{avg} &= \frac{\sum(\text{Vertical + Horizontal hops})}{\text{Total number of paths}} = \frac{H_{vert} + H_{hori}}{(N^2 - N)N} \\ &= \frac{\sum_{j=1}^{N^2-N} \sum_{i=1}^{N} |row_{m,i} - row_{c,j}| + |col_{m,i} - col_{c,j}|}{N^2(N-1)} \end{aligned} \tag{3}$$

Note that Equation 3 is a general form of the equation applicable to any MC placement. H_{vert} and H_{hori} represent the aggregated number of hops for vertical and horizontal directions. We summarize H_{vert} and H_{hori} of different MC placements in Table 1.

Based on Table 1, we find that sorting the MC placement diagrams in the order of decreasing average number of hops yields the following order: *bottom, edge, top-bottom,* and *diamond*. This analysis also corresponds with results from prior work [2], which reported the best performance improvement with *diamond* MC placement. Although the *diamond* placement shows the least number of hops, we show that other MC placement policies can outperform the *diamond* placement by adopting VC monopolizing and different routing algorithms in Section 4.2.

3.2 Proposed Design

3.2.1 *VC Monopolizing and Asymmetric VC Partitioning*

Request and reply packets in GPGPUs compete for NoC resources such as VCs and PCs. When the resources are naïvely shared by both packets, avoiding protocol deadlock requires that reply packets must not compete for the same resources as request packets. To avoid this, prior studies [4, 3, 11] suggest partitioning NoCs equally into two

MC placement	H_{vert}	H_{hori}
Bottom	$\frac{N^3(N-1)}{2}$	$\frac{N(N+1)(N-1)^2}{3}$
Edge	$\frac{N^2(N-1)^2}{2}$	$\approx \frac{N(N+1)(N-1)^2}{3}$
Top-Bottom	$\frac{N^2(N-1)^2}{2}$	$\frac{N(N+1)(N-1)^2}{3}$
Diamond	$\approx \frac{N^2(N+1)(N-2)}{8}$	$\approx \frac{N^2(N+1)(N-2)}{8}$

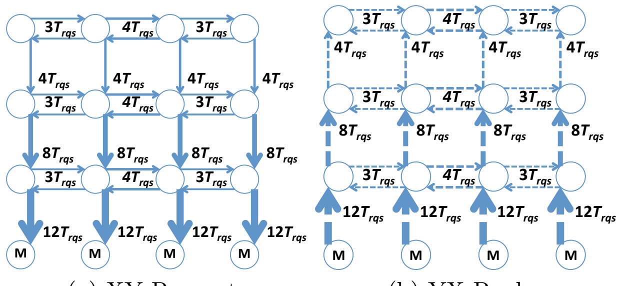
Table 1: The average number of vertical/horizontal hops under different MC placements in an (NxN) mesh

parts for the different types of traffic: one network carries request packets and the other network reply packets. Creating two parallel physical networks [11] incurs significant hardware overheads due to the twofold increase in the number of routers and wire resources. To this overhead, we employ a virtual network partitioning, where the network is divided virtually by two separate sets of VCs dedicated for request-reply traffic under one physical network.

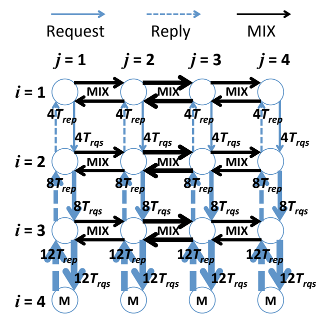
However, when all MCs are located at the bottom, request and reply traffic are not overlapped with dimension ordered routing as shown in Figure 4. Therefore, there is no need to split networks to avoid protocol deadlock. Thus, all the VCs can be *fully monopolized* by either request or reply packets, providing more buffer resources for each type of traffic, thus helping improve overall system performance. On the other hand, VC monopolizing is not feasible when VCs have mixed request and reply traffic, as shown in Figure 6(c). These mixed VCs must be partitioned into request and reply packets to avoid protocol deadlock. In this case, we propose *asymmetric VC partitioning* which assigns more VCs to reply traffic. Since reply traffic generally requires much more network bandwidth than request traffic, moving VC resources from the request to the reply improves the overall system performance while maintaining the same overall NoC area and power budget. The detailed evaluation is described in Section 4.2.

3.2.2 *Routing Algorithms*

A routing algorithm is one of the critical factors in achieving bandwidth-efficient NoC, influencing the amount of traffic each link will carry. Routing contributes to the reduction in network contention (hot spots) when combined with an appropriate MC placement. To find the performance-optimal combination of a routing algorithm and an MC placement, we analyze the impact of different dimension order routing algorithms (XY, YX, and XY-YX [2]) under the different MC placements shown in Figure 5. For example, under our baseline MC placement, *bottom MC*, shown in Figure 5(a), XY routing incurs increased network contention mainly due to the high volume of reply traffic between MCs, thus degrading overall system performance. Alternatively, XY-YX routing which leads request packets to follow XY routing, while reply packets to follow YX routing, helps achieve significant performance improvement because heavy traffic between MCs due to reply packets is entirely eliminated as shown in Figure 6. Since the request traffic in YX routing still generates contention between MCs, the performance improvement of YX routing is less than that of XY-YX routing. However, the reply traffic in YX routing does not cause any communication between MCs since the



(a) XY Request (b) YX Reply



(c) XY-YX Routing

Figure 6: Network traffic example with XY-YX routing. (*Note, request/reply traffic is mixed on horizontal links.)*

reply traffic always traverses to the Y direction first. Therefore, XY-YX or YX routing is effective in load-balancing the processor-memory traffic in a 2D mesh topology with the *bottom* MC placement scheme[1]. On other MC placement schemes, we find routing algorithms have little impact on overall performance. Simulation results for different MC placements and routing algorithms are detailed in Section 4.

4. PERFORMANCE EVALUATION

In this section, we evaluate schemes proposed in Section 3 with the aim of developing a high performance NoC, optimized for use in GPGPUs. We also analyze simulation results in detail using a wide variety of GPGPU benchmarks.

4.1 Methodology

The NoC designs and MC placement schemes examined here are implemented in GPGPU-Sim [5]. The simulator is flexible enough to capture the internal design of GPGPU and our target architecture has similarities to NVIDIA's FermiGTX 480. Figure 1 shows the NoC router microarchitectures modeled in GPGPU-Sim. A 2D mesh network is used to connect SMs, caches, and MCs. To prevent protocol deadlock, the baseline NoC (Table 2) is built with a single physical network with two separate VCs for handling request and reply traffic. We evaluate our schemes with a wide range of GPGPU workloads such as CUDA SDK [1], ISPASS [4], Rodinia [6], and MapReduce [8]. Each benchmark suite is structured to span a range of parallelism and compute patterns, providing feature options that help identify architectural bottlenecks and fine tune system designs.

4.2 Performance Analysis

[1]Note, we do not consider adaptive routing because of the increased critical path delays in a router [2] and degraded row buffer locality caused by not preserving the order of request packets to MCs [15].

System Parameters	Details
Shader Core	56 Cores, 1400 MHz, SIMT width = 8
Memory Model	8 MCs, 924 MHz
Interconnect	8 x 8 2D Mesh, 1400 MHz, XY Routing
Virtual Channel	2 VCs per Port
VC Depth	4
Warp Scheduler	Greedy-then-oldest (GTO)
MC placement	Bottom
Shared Memory	48KB
L1 Inst. Cache	2KB (4 sets/4 ways LRU)
L1 Data Cache	16KB (32 sets/4 ways LRU)
L2 Cache	64KB per MC (8-way LRU)
Min. L2 / DRAM Latency	120 / 220 cycles

Table 2: System Configuration

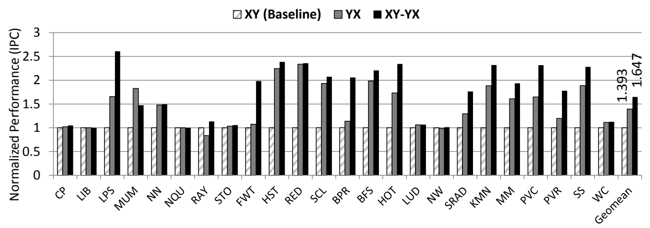


Figure 7: Speed-up with routing algorithms (Normalized to baseline XY)

Impact of Network Division. As described in Section 3.2.1, we advocate for a single physical network with separate virtual networks for request and reply packets. To avoid protocol deadlock, we increase the number of VCs per port, where different types of packets traverse on-chip networks via different VCs. It is noted that additional VCs employed to avoid a protocol deadlock can affect the critical path of a router since VC allocation is the bottleneck in the router pipeline [11]. However, we observe that two separate VCs under a single physical network degrades system performance less than 0.03% in geometric mean across 25 benchmarks. This observation leads us to use separate VCs with a single physical network instead of two physical networks requiring more hardware resources.

Impact of Routing Algorithms. While maintaining the same number of VCs with reduced network resources, we observe that alternative routing algorithms can significantly improve the overall system performance. Figure 7 shows the speed-up obtained with YX and XY-YX, normalized against the baseline XY. YX and XY-YX with the *bottom* MC placement scheme achieve a speedup of 39.3% and 64.7%, respectively. As discussed in Section 3.2.2, the improvement mainly comes from mitigated traffic congestions between MCs. The heavy reply traffic generated from MCs is the main factor causing performance bottlenecks in NoCs. In this context, XY-YX routing outperforms YX routing by more than 25%. This is because unlike YX routing, XY-YX completely removes the resource contention between MCs by providing different routing paths for request and reply packets as illustrated in Figure 6.

VC Monopolizing. As illustrated in Figure 4, request and reply packets never overlap with each other in any dimension under XY or YX routing with the *bottom* MC placement, allowing VC monopolization as described in Section 3.2.1. Figure 8 shows the impact of VC monopolizing on system performance under different routing algorithms. Monopolized VCs lead XY and YX to achieve 43.8% and 88.9% (= 39.3% from YX + 49.6% from monopolization) of speed-up in geometric mean, respectively. Note that unlike XY and YX routing, XY-YX routing still requires separate VCs in horizontal links to prevent protocol deadlock because different types of packets get potentially mixed while moving along the horizontal links as illustrated in Figure 6. This limits the number of VCs that can be monopolized (*partial monopolizing*) because only the VCs located in the vertical links can be fully monopolized in XY-YX routing. Accordingly, partially monopolized XY-YX routing shows less performance improvement at 85.4% (= 64.7% from XY-YX + 20.7% from monopolization), compared to that of the fully monopolized scheme with YX routing. Furthermore, across different MC placements (*edge*, *top-bottom*, and *diamond*), VC monopolizing is effective in achieving better performance improvement, as detailed below.

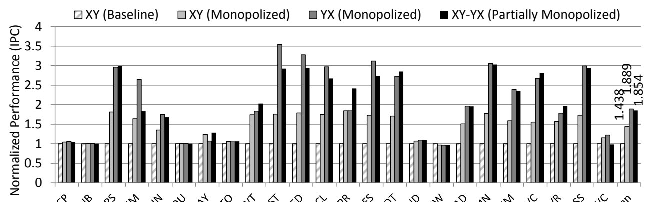


Figure 8: Speed-up with VC monopolized scheme (Normalized to XY routing with VC separated for each traffic)

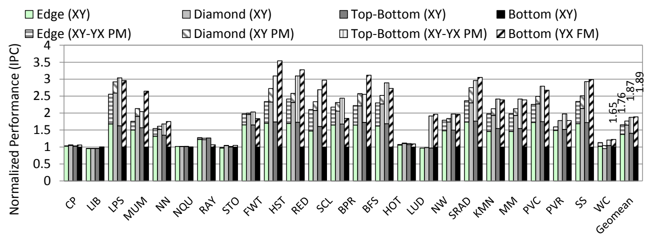


Figure 9: Speed-up with different MC placements with routing algorithms (PM: Partial Monopolizing, FM: Full Monopolizing, Normalized to bottom MC+XY routing)

Impact of MC Placement Scheme. In our baseline, we simply put all MCs at the bottom row in 8 × 8 2D mesh. As the network traffic in GPGPUs gets skewed towards the MCs in this scheme, the *bottom* MC placement causes high network congestion near the MCs, thus degrading performance. One way to alleviating such traffic congestion is to locate MCs sparsely across the network. Figure 9 shows the impact of different MC placements on overall system performance. Note that each MC placement is simulated with three different routing algorithms (XY, YX, and XY-YX). In Figure 9, we pick the routing algorithm showing the highest performance improvement for each MC placement scheme. In the figure we see that MC location has a significant impact on performance. This is because, with distributed MC placements, request and reply packets are spread across multiple locations of the on-chip network rather than converging to the bottom row. Compared to the bottom MC placement, the average performance speedup is 37.3%, 64.4%, and 40.4% for the *edge*, *diamond*, and *top-bottom* placements, respectively. And when applying the VC monopolizing scheme in combination with different MC placements, additional 28.3%, 12%, and 47.3% performance improvement (65.6%, 76.4%, and 87.7% in total) are achieved with the *edge*, *diamond*, and *top-bottom* placements, respectively. Here it is worthwhile to note that our baseline bottom MC placement combined with YX routing and fully monopolized VCs shows the highest performance improvement (89.4%) and even outperforms the prior top-performing work, diamond MC placement, by 25% (= 89.4% - 64.4%), even though the

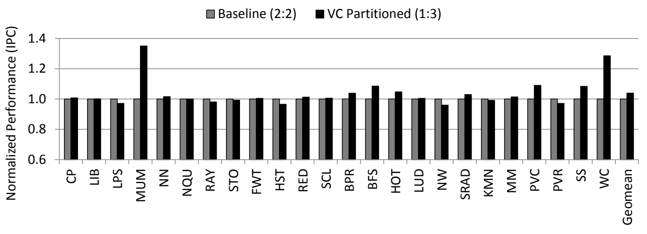


Figure 10: Speed-up with asymmetric VC partitioning (Request:Reply = 1:3)

diamond has the least number of hops as analyzed in Section 3.1.2. This proves the performance effectiveness of the VC monopolizing scheme described earlier in Section 3.2.1. **Asymmetric VC Partitioning.** Mixed request and reply traffic limits the possibility of using the *VC monopolizing* scheme. Different routing algorithms such as XY-YX routing or dispersed MC placements like *diamond* cause the mix of traffic in the middle of the links. In such network configurations, we apply asymmetric VC partitioning between request and reply packets as we detailed in Section 3.2.1. To show the impact of asymmetric VC partitioning, we assume the number of VCs per port are four. Figure 10 shows a speed-up with asymmetric VC partitioning, where only one VC is assigned to request packets and other VCs are assigned to reply packets. For XY-YX routing, the asymmetric partitioning improves the performance by 3.9% in geometric mean. Since reply packets are usually heavier than request packets, assigning more VCs to reply packets is beneficial assuming enough VCs already exist. Note that asymmetric VC partitioning is effective in enhancing performance across all MC placements.

5. RELATED WORK

Yuan et al. [15] proposed a complexity-effective memory scheduler for GPU architectures. NoC routers of the proposed mechanism reorder packets to increase row-buffer locality in the memory controllers. As a result, a simple in-order memory scheduler can perform similar to a much more complex out-of-order scheduler. Bakhoda et al. [3] proposed a throughput-effective NoC for GPU architectures. Due to many cores with a smaller number of memory controllers, a many-to-few traffic pattern is dominant in GPUs. To optimize such traffic, they used a half router, which disallows turns, to reduce the complexity of the network while increasing the injection bandwidth from the memory controllers to provide burst data read. Kim *et al.* propose [11] lightweight high frequency NoC router architectures, which reduces the pipeline delay in routers, achieving high on-chip network bandwidth, and reducing energy consumption due to the simplified architecture. In a heterogeneous system, Lee *et al.* [13] propose feedback based VC partitioning between CPU and GPU applications. The proposed technique dedicates a few VCs to CPU and GPU applications with separate injection queue. VC partitioning in heterogeneous architectures is effective at preventing interference between the CPU and GPU applications. However, it requires a feedback mechanism to dynamically partition VCs on each router which might be additional overhead in a heavily loaded network such as GPGPUs. In addition, GPGPUs concurrently execute a massive number of threads within a single application rather than multiple applications. We believe that static VC partitioning between request and reply is enough to provide a reasonable performance improvement.

6. CONCLUSIONS

In this paper, we analyze the unique characteristics of on-chip communication within GPGPUs under a wide range of benchmark applications. We find that the *many-to-few* and *few-to-many* traffic patterns between cores and MCs create a severe bottleneck, leading to the inefficient use of NoC resources in on-chip interconnects. We show the improved system performance based on VC monopolizing and asymmetric VC partitioning under diverse MC placements and dimension ordered routing algorithms.

7. REFERENCES

[1] NVIDIA CUDA SDK. `https://developer.nvidia.com/gpu-computing-sdk`.
[2] 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 *Proceedings of ISCA*, 2009.
[3] A. Bakhoda, J. Kim, and T. M. Aamodt. Throughput-effective on-chip networks for manycore accelerators. In *Proceedings of MICRO*, 2010.
[4] 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*, pages 163–174, 2009.
[5] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt. Analyzing CUDA Workloads Using a Detailed GPU Simulator. In *Proceedings of ISPASS*, 2009.
[6] 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 *Proceedings of IISWC*, 2009.
[7] N. Goswami, R. Shankar, M. Joshi, and T. Li. Exploring GPGPU Workloads: Characterization Methodology, Analysis and Microarchitecture Evaluation Implications. In *Proceedings of IISWC*, 2010.
[8] B. He, W. Fang, and Q. Luo. Mars: A MapReduce Framework on Graphics Processors. In *Proceedings of PACT*, 2008.
[9] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar. A 5-GHz Mesh Interconnect for a Teraflops Processor. *IEEE Micro*, 27:51–61, 2007.
[10] O. Kayiran, A. Jog, M. T. Kandemir, and C. R. Das. Neither More Nor Less: Optimizing Thread-level Parallelism for GPGPUs. In *Proceedings of PACT*, 2013.
[11] H. Kim, J. Kim, W. Seo, Y. Cho, and S. Ryu. Providing Cost-effective On-Chip Network Bandwidth in GPGPUs. In *Proceedings of ICCD*, 2012.
[12] A. Kumar, L.-S. Peh, and N. Jha. Token Flow Control. In *Proceedings of MICRO*, 2008.
[13] J. Lee, S. Li, H. Kim, and S. Yalamanchili. Adaptive virtual channel partitioning for network-on-chip in heterogeneous architectures. *ACM Trans. Des. Autom. Electron. Syst.*, 18:48:1–48:28, 2013.
[14] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C.-C. Miao, J. F. Brown III, and A. Agarwal. On-Chip Interconnection Architecture of the Tile Processor. *IEEE Micro*, 27:15–31, 2007.
[15] G. L. Yuan, A. Bakhoda, and T. M. Aamodt. Complexity Effective Memory Access Scheduling for Many-core Accelerator Architectures. In *Proceedings of MICRO*, 2009.