Improving GPU NoC Power Efficiency through Dynamic Bandwidth Allocation

Xianwei Cheng[1], Hui Zhao[1], Saraju P. Mohanty[1], Juan Fang[2]
[1]Computer Science and Engineering Department, University of North Texas
xianweicheng@my.unt.edu, {hui.zhao,saraju.mohanty}@unt.edu
[2]Faculty of Information Technology, Beijing University of Technology
fangjuan@bjut.edu.cn

Abstract—**High throughput in data communication is of great significance for GPU accelerated systems in order to fully exploit thread level parallelism. Different traffic patterns between GPU NoCs and CPU NoCs lead to suboptimal performance in GPU NoCs that directly adapt from CPU NoCs. Moreover, for GPU NoCs, two networks are usually employed to avoid deadlocks between requests and reply messages. Another important feature of GPU NoCs is the unbalanced traffic load between request network and reply network. This feature often causes the reply network to be congested while the request network is idle. Based on these features of GPU NoCs, this paper proposes a technique called Stop Request Network (SRN). SRN works by stopping request network to reduce energy cost when congestion occurs in the reply network. Our evaluation results show that SRN can save power by 10% with negligible performance degradation.**

I. INTRODUCTION

Increasing core numbers in GPU accelerated computing systems provide opportunities to process a large number of threads in parallel to achieve high performance. However, this also brings challenges for researchers to solve the accompanying cost problems. On-chip networks which serve as GPUs' communication backbone affect GPUs' performance and cost significantly [1], [2], [3], [6]. On the one hand, high throughput is need for data communication to deeply exploit thread level parallelism. On the other hand, network cost such as area and power consumption needs to be reduced in order to incorporate more cores. GPUs have been shown to be power hungry and it is imperative to develop techniques which can reduce power cost with minimal performance degradation.

Topology is the way routers are connected with each other. Typical topologies used in NoCs include mesh, ring, concentrated mesh or torus. Topology plays a critical role in network performance and cost. Different from CPU-based multi-processor systems, GPUs exhibit a many-to-few-to-many traffic pattern which defines how data flows between many computing cores and a few memory controllers (MCs). Due to the simplicity in design, meshes are most widely used in NoCs. However, hot spots are often created in memory controller connected routers due to the many-to-few-to-many traffic pattern. Because the majority of messages are read messages and read reply messages usually have larger payload size, the reply network accounts for most of the network traffic load. When congestion occurs at the reply network hotspots (i.e. MCs), memory controllers cannot service request messages anymore because their buffers are fully occupied. As a result, the request messages are piled up in the network buffers, consuming a large amount of static energy.

This paper proposes a technique for power savings by stopping the request network when we detect that the reply network is congested. After some time interval, the request network is restarted if the congestion is relieved. Our objective is to reduce power cost without performance degradation. Our simulation results show that our technique helps to reduce power consumption by 10% on the average, with minimal negative impact on system performance. Our results demonstrate the effectiveness of the proposed technique in exploring performance and cost tradeoffs for GPGPU NoCs.

The rest of this paper is organized as follows: Section II introduces background and motivation; Section III describes our proposed design in detail; Section IV presents our experimental results and related analysis and we conclude this paper in Section V.

II. BACKGROUND AND MOTIVATION

A. GPU Traffic Pattern

Figure 1 shows a typical mesh NoC for GPGPUs. There are 28 compute cores and 8 memory controllers. Data packets are transmitted between the compute cores and memory controllers. A memory controller and a L2 cache are co-located together and are connected to the network through a MC router. If a L1 cache miss occurs, a request message is sent to L2 cache. Requested data will be returned to the computing core if it is found in L2. Otherwise, the request will be directed to the off-chip SDRAM to retrieve the data. There is limited communication between computing cores (L1 to L1) in GPUs and this is mostly cache coherency signals between L1 caches. Because the amount of this type of traffic is small, L1-to-L1 communication in GPU is often removed and replaced by L1-to-L2 transfer followed by L2-to-L1 transfer [5] [10]. Therefore, GPU network traffic usually consists of two categories: (1) requests received by memory controllers from computing cores; and (2) reply messages received by computing cores from memory controllers. GPU NoCs exhibit a many-to-few (request) and few-to-many (reply) traffic pattern because the number of computing cores is much larger compared with that of memory controllers.

Request and reply networks in the GPU NoCs exhibit imbalance in their traffic loads. There are much more read messages than write messages. In addition, the size of read reply messages is much larger than that of read request messages. Prior studies have shown that 70% of the total on-chip traffic are reply messages [12, 13]. Because the sources of reply messages are a few memory controllers, network

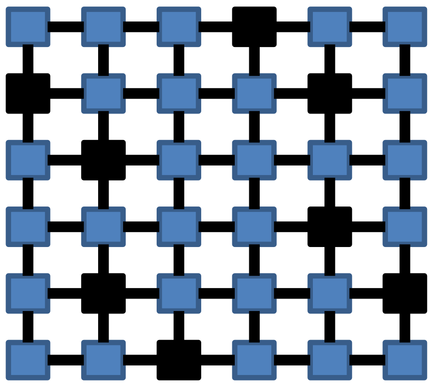

Figure 1. A typical GPU NoC with a mesh topology.

bottlenecks are usually created in routers connected with those memory controllers. Our goal for improving GPU NoC design in this paper is to improve network power efficiency without performance degradation. We propose to stop transmitting request messages if reply messages are blocked at the memory controllers due to congestion. By power gating the routers in the request network, we can save power in request network when the reply network is congested and then restart the request network when congestion in the reply network is relieved.

B. *Limitations in Current GPU NoCs*

Crossbars are mostly employed by industry in designing GPU NoCs. Although crossbars provide good throughput by allowing no-blocking connectivity between each pair of nodes, they are not scalable to large networks. The cost of crossbars grow exponentially as the number of cores increases and this limits the application of crossbars to large GPU systems. Mesh topology has been most widely proposed for designing NoC for GPUs [4], [7], [9], [10], due to its regularity, scalability, and simplicity in layout. To avoid protocol deadlock between request and reply messages, almost all mesh-based NoCs need two physical networks except a logically positioned single network [4].

However, mesh topology exhibits inherent drawbacks with regard to performance and cost despite its wide application: (1) At MC connected routers, which is usually the network bottleneck, there is severe interference between injection/ejection packets and bypassing packets. Packet injection and ejection is only part of functions that MC connected router need to implement. These routers also need to route bypassing packets to their neighbors. Therefore, these different types of packets compete for the router resource and link bandwidth. As a result, congestion is generated especially in the reply network; (2) The position of MCs significantly influences network performance. That is due to the interaction between the many-to-few-to-many traffic pattern and DOR routing. Trying to find an optimal MC placement increases the NoC design complexity; (3) The utilization of network resource in a mesh network is inefficient. Mesh networks' evenly distribute resources to all routers, such as routers ports and links to simplify the design. However uneven traffic pattern of GPUs causes over provisioning of resource and power. Some regions tend to be much more congested than the rest of the network. As a result, static power is wasted in regions that are not utilized. All these limitations demand more power efficient design for mesh-based GPU NoCs.

III. Design of SRN

A. *Opportunities and Challenges*

We propose to take advantage of two inherent features of GPU NoCs: (1) Imbalance between request and reply network traffic and the reply network carries most of the traffic load. (2) In reply network, the traffic pattern is few-to-many, i.e., few memory-controllers send reply packets to many compute cores. Both of these features tend to cause congestions in the reply network, with the request network being idle. We propose a technique to stop the request network to save power when congestion is detected in the reply network. When the reply network congestion is relieved, we immediately restart the request network. This technique can minimally affect the network performance but improve the power efficiency.

GPUs achieve high performance executing parallel applications because all cores can run threads in parallel and there is a large pool of runnable threads to hide communication delay. If the communication delay cannot overlap with computation, the performance will degrade due to stalled cores waiting for data from memory controllers. Therefore, one challenge is to appropriately determine stop and restart time in order to avoid unnecessary stall caused by our mechanism. Another challenge is keeping the frequency of stop and restart at an appropriate frequency. Stop and restart the request network incurs extra overhead in network control and synchronization. Frequent stop and restart may result in thrashing and have a negative effect on performance and power efficiency. Therefore, we need to find proper metrics to determine the appropriate time to stop and restart of request network.

B. *Design of SRN*

As has been shown by prior work [12, 13], 70% of GPU traffic load is reply messages. We observe that when congestion occurs, a large number of reply messages are blocked in MC output queues, waiting to be injected into the network. In case that a MC output queue is full, the MC will stall. Therefore, the performance of the whole network can be improved by increasing reply network throughput. On the other hand, blocked reply messages provide the request network some slack to tolerate delay. Network performance will not suffer from degradation in this case because no incoming request messages will be served since the MC already stalls. However, the request network needs to be promptly restarted when the congestion in the reply network is relieved. Otherwise, stopped request network will incur extra stalls in the computing cores and leads to performance degradation. Therefore, we need to first find proper metrics and mechanisms to monitor network status and determine the duration for stopping the request network.

(i) Stop Metrics

Selecting a metric to evaluate the benefit of stopping the request network in epochs of time is a critical part of our design. The desired metric needs to be able to reflect network slack. The slack time represents reply packets' congestion status at output queues of memory controllers. We experimented with several metrics, such as all MCs' maximum output queue occupancy and the average output queue occupancy. However, we found that neither of these metrics can accurately capture the uneven occupancy of the output queues. The maximum output queue occupancy reflects only one of the MCs local congestion status but cannot show the global congestion situation. It is not an accurate metric because the whole network may not be congested even if one of the MC is very congested. Similarly, the average MC output queue occupancy is not accurate because the whole network may not be busy even if multiple output queues are full. To accurately measure the network congestion, we choose a combination of two metrics that can avoid bias caused by unbalanced output occupancy. The first metric is individual MC's output queue occupancy. We label one MC to be congested if its occupancy is greater than a threshold value *Th_queue*. The second metric is the number of congested MCs. Only when the total number of congested MCs is above a threshold value *Th_MC*, the reply network is considered to be congested and the actions are taken to stop the request network. By using two metrics, we can detect global congestion status of MCs instead of local ones which ensures the accuracy. We set up both thresholds *Th_queue* and *Th_MC* using empirical values.

(ii) Restart Metrics

We stop the request network when slack is detected. However, to ensure that our mechanism does not introduce extra stall time in computing cores, the request network needs to be restarted immediately once congestion is relieved. Therefore, in this work, the metric of deciding the time for network restart is also needed. Different from the network stop metrics, we evaluate the input queue of MCs for network restart. We consider both the individual input occupancy and the occupancy of all input queues. Similarly, we use empirical values to set thresholds for the evaluation. Once it is determined that restart needs to be initiated, a RESTART signal is sent to cores from MCs. To avoid the overhead caused by thrashing that stop and restart are too frequently called, we use a *HOLD TIME* to enforce that the request network does not transit to a new state unless certain time has elapsed.

IV. EXPERIMENTAL RESULTS

Our proposed SRN technique is evaluated regarding performance and power consumption. GPGPU-Sim [7] is used to simulate our proposed technique. The simulated GPGPU system consists of an 8x8 2D mesh connecting 56 computing cores and 8 MCs. Routers in the network employ conventional 5 ports VC-based micro-architecture.

TABEL I. System Configuration

Number of Computing Cores	56 cores
Number of Memory Controllers	8
MSHR per Core	64
Warp Size	32
SIMD Pipeline Width	8
Number of Threads per Core	1024
Number of CTAs/Core	8
Constant Cache Size/Core	8KB
Texture Cache Size/Core	8KB
L1 Cache Size/Core	16KB
L2 Cache Size/Core	128KB
Number of Registers/Core	16384
Warp Scheduler	Greedy-Then-Oldest
Shared Memory	48 KB
Memory Scheduler	FR-FCFS
Memory Model	8 MCs, 924 MHz
NoC Channel Width	128 bit
NoC Router Pipeline Stage	2
Number of VC per Port	2
VC Buffer Depth	3
Subnet	2

A. *Impact on Performance*

As shown in Figure2, the performance of all the benchmarks are more than 90% of the baseline system. There are several benchmarks, such as WP and BP, that achieve better performance than the baseline. This is because after we stop request network, the contention of reply network has reduced and the network throughput gets improved. As a result, the SRN can achieve better performance than baseline. On the average, SRC achieves performance that is 99% of that of the baseline. This means our technique has negligible performance degradation. We also compare our work with the state-of-the-art GPU NoC design of Checker Board [3] and our design achieves 12% better performance than the Checker Board.

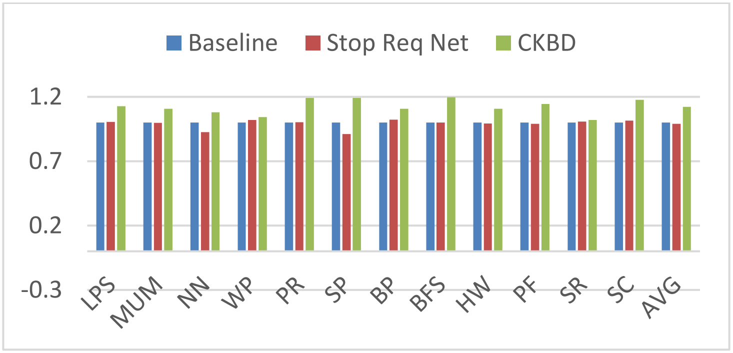

Figure 2. Performance.

TABLE II. Benchmark Description

Abbreviation	Description	Injection Rate
LPS	3D Laplace Solver	2.14%
MUM	MUMer GPU	1.31%
NN	Neural Network	1.30%
WP	Weather Prediction	1.38%
PR	Parallel Reduction	1.16%
SP	Scalar Product	1.43%
BP	Back Propagation	3.55%
BFS	Breadth First Search	1.30%
HW	Heart Wall	2.16%
PF	Path Finder	1.74%
SR	Srad	1.34%
SC	Stream Cluster	2.98%

B. *Impact on Energy*

The SRN aims to save network power by stopping the request network when the reply network is congested. As shown in Figure 3, our technique can reduce energy cost by about 10% averagely compared with baseline. For benchmark SC, the energy savings can be 35%. The result shows that the SRN can effectively reduce energy cost in the GPU network.

C. *Impact on Packet Latency*

We also analyzed network packet latency. As shown in Figure 4, the packet latency gets reduced for most of the benchmarks. This is because after we stop the request network, the congestion of the reply network gets decreased, so the latency gets reduced as a result. On the average, the packet latency gets reduced by about 9%. Benchmark PF and SC achieve the largest latency reduction with their packet latency reduced by 31% and 44% respectively.

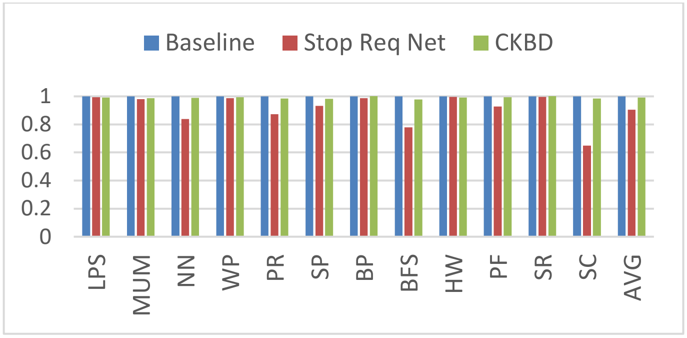

Figure 3. Energy Consumption.

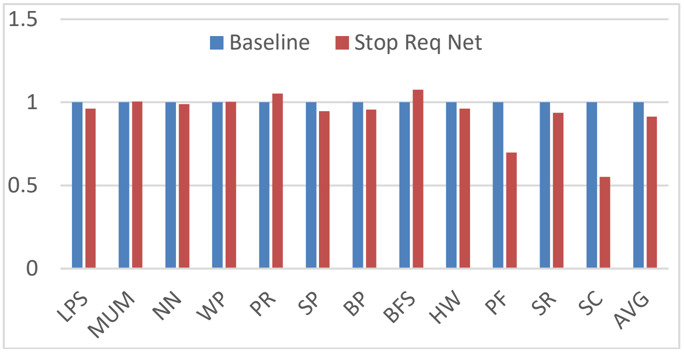

Figure 4. Packet Latency.

D. *Impact on EDP*

Finally, we evaluate the Energy Delay Product (EDP), the result is shown in Figure 5. On average, the EDP reduction is about 17%. The best result is achieved by benchmark SC which is about 64%. The result shows that the proposed SRN technique can effectively reduce the power consumption and packet latency.

V. Conclusion

In this work, we propose a technique called SRN by stopping the request network when network slack is detected to save energy based on the special characteristics of GPU NoCs. Our evaluation results show that the technique proposed can effectively reduce network energy with negligible performance degradation.

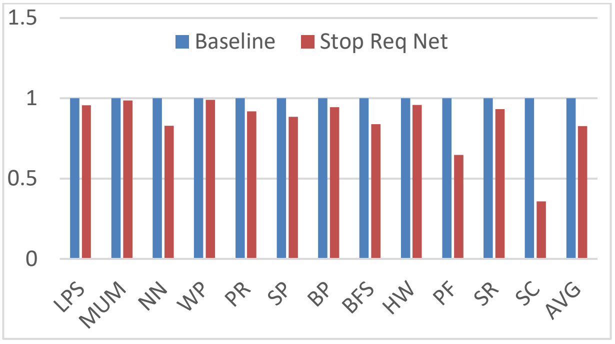

Figure 5. Energy Delay Product.

References

[1] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture", in *ACM SIGCOMM Computer Communication Review*, 2008.

[2] M. Ahn and E. J. Kim, "Pseudo-Circuit: Accelerating Communication for On-Chip Interconnection Networks", in *MICRO*, 2010.

[3] A. Bakhoda, J. Kim, and T. M. Aamodt, "Throughput-Effective On-Chip Networks for Manycore Accelerator", in *MICRO*, 2010.

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

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

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

[7] N.Goswami, R. Shankar, M. Joshi, and T. Li. "Exploring GPGPU Workload: Characterization Methodology, Analysis and Microarchitecture Evaluation Implications.", in *IISWC*, 2010.

[8] W. Dally and B. Towles. "Principles and Practices of Interconnection Networks", in *Morgan Kaufmann Publishers Inc.*, 2003.

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

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

[11] H. Zhao, X. Cheng, S. P. Mohanty, and J. Fang, "Designing Scalable Hybrid Wireless NoC for GPGPUs", in *Proceedings of the 17th IEEE Computer Society Annual Symposium on VLSI*, 2018, pp. 703--708.

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

[13] X. Cheng, Y. Zhao, H. Zhao and Y. Xie, "Packet Pump: Overcoming Network Bottleneck in On-Chip Interconnects for GPGPUs", In DAC, 2018.