LA-LLC: Inter-Core Locality-Aware Last-Level Cache to Exploit Many-to-Many Traffic in GPGPUs

Xia Zhao, Yuxi Liu, Almutaz Adileh, and Lieven Eeckhout

Abstract—The reply network is a severe performance bottleneck in General Purpose Graphic Processing Units (GPGPUs), as the communication path from memory controllers (MC) to cores is often congested. In this paper, we find that instead of relying on the congested communication path between MCs and cores, the unused core-to-core communication path can be leveraged to transfer data blocks between cores. We propose the inter-core Locality-Aware Last-Level Cache (LA-LLC), which requires only few bits per cache block and enables a core to fetch shared data from another core's private cache instead of the LLC. Leveraging inter-core communication, LA-LLC transforms few-to-many traffic to many-to-many traffic, thereby mitigating the reply network bottleneck. For a set of applications exhibiting varying degrees of inter-core locality, LA-LLC reduces memory access latency and increases performance by 21.1 percent on average and up to 68 percent, with negligible hardware cost.

Index Terms—GPGPU, NoC, inter-core locality, LLC

1 Introduction

GPGPUs exploit the single-instruction multiple-thread (SIMT) architecture and leverage thread-level parallelism (TLP) to hide memory access latency. However, many GPGPU applications generate a large number of memory accesses which increases the pressure on the memory system and interconnection network. Solely relying on TLP cannot completely hide memory access latency and as a result, memory operations become a severe performance bottleneck. Although a lot of work has been done to optimize the memory system, few pay attention to network latency, which plays an important role in the overall memory access latency [1], [2].

By exploiting a bulk-synchronous parallel (BSP) programming model, GPGPUs usually execute a large number of independent thread blocks and do not support hardware cache coherence [3]. This yields a unique traffic pattern, called many-to-few-to-many, where communication only exists between compute cores and memory controllers (MCs) [2]. The GPGPU network-on-chip (NoC) typically consists of a request and a reply network. The request network transfers request packets, including write and read requests from cores to MCs, while the reply network transfers read (and write) replies in the opposite direction. The few-to-many traffic from the MCs to the cores causes serious congestion in the reply network. The network bottleneck increases memory access latency and has a detrimental effect on performance [2], [4].

Due to this unique traffic pattern, these is no inter-core communication in current GPGPU networks. In this paper, we find that the unused inter-core communication can be leveraged to mitigate the reply network bottleneck and reduce memory access latency. In particular, we propose a simple yet effective inter-core locality-aware last-level cache (LA-LLC) to record the core where the data may exist. Instead of relying on the already congested communication path between MCs and cores, LA-LLC enables a core to fetch data from a remote L1 and transfer shared data blocks between cores. This transforms few-to-many traffic to many-to-many traffic. To our best knowledge, this is the first study to notice and leverage inter-core communication to mitigate the reply network bottleneck in GPGPUs.

2 Background and Motivation

Fig. 1 illustrates our baseline NoC architecture. Current GPGPUs often use a crossbar as the interconnection network while limiting the number of ports by sharing a single port among several cores. However, to continue improve raw computational power, GPGPUs have seen a rapid increase in the number of compute cores, e.g., the latest Nvidia Pascal GPGPU supports 60 cores [5]. As the number of cores increases, scalability of a crossbar NoC will be limited. Thus, similar to previous work, we choose a 2D mesh topology in our research due to its regularity, simplicity and scalability [2], [4], [6], [7], [8]. We further consider two separate networks, a request and a reply network, to avoid protocol-level deadlock [2], [4], [7], [8].

Due to (i) the ratio of many cores to few MCs, and (ii) a significant portion of memory accesses being read requests and the corresponding replies carrying a long data block, MCs have a much higher injection rate than cores which leads to congestion in the reply network. As a result, the injection port of an MC is often blocked as the reply network can no longer accept packets from the MCs. As shown in Fig. 2a, these blocked cycles account to more than 60 percent of the total cycle count on average.

The network bottleneck increases memory access latency and has a detrimental impact on performance [2], [4]. The simplest way to solve the network bottleneck and reduce memory access latency is to increase the bandwidth in the reply network. As shown in Fig. 2b, with an unchanged request network, doubling bandwidth of the reply network mitigates network congestion and reduces memory access latency by 44 percent. After solving the reply network bottleneck, cores get data replies much faster. This decreases pipeline stalls due to memory operations and increases performance by 60 percent on average as shown in Fig. 2c.

However, solving the network bottleneck by increasing network bandwidth is not a cost-effective solution. Network bandwidth impacts both input buffers and the crossbar, which constitute the dominant portion of the router area. Moreover, network bandwidth also impacts link area. Several prototype chips show that current NoCs already consume a substantial portion of system power and area, and increasing network bandwidth would make the NoC problem become even more serious [9]. Thus, mitigating the network bottleneck without increasing NoC cost becomes a challenge.

Many GPGPU applications exhibit high inter-core locality as different cores may access shared read-only data. Recently, Li and Aamodt analyze sources of inter-core locality and leverage inter-core locality by designing a locality-aware memory controller [1]. In their design, they assign higher priority to memory requests that fetch inter-core shared data. This design does not consider the network bottleneck as in this paper. In fact, after improving the efficiency of the memory controller, the network bottleneck becomes even more serious as MCs now have a higher injection rate of reply packets.

Due to the few-to-many traffic in the reply network, when congestion happens on the transfer path between MCs and cores, some inter-core links are unused as shown in the example of Fig. 1. (Red lines are congested paths between MCs and cores.) This offers the opportunity to mitigate the network bottleneck by leveraging core-to-core communication. Due to inter-core locality, multiple copies of the same data may exist in different cores. Thus, instead of fetching data blocks from the LLC and transferring replies between MCs and cores, inter-core locality enables fetching data from remote L1s and transferring replies between cores. Leveraging the unused core-to-core communication mitigates the reply network bottleneck and reduces memory access latency.

• *The authors are with Ghent University, Belgium.*
E-mail: {xia.zhao, yuxi.liu, almutaz.adileh, lieven.eeckhout}@ugent.be.

Manuscript received 14 July 2016; revised 17 Aug. 2016; accepted 1 Sept. 2016. Date of publication 19 Sept. 2016; date of current version 26 June 2017.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/LCA.2016.2611663

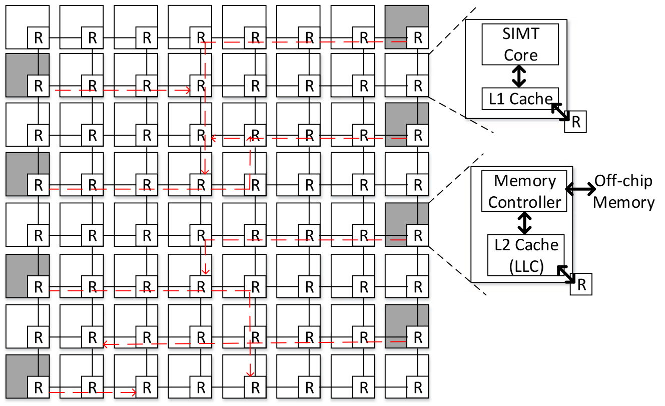


Fig. 1. The baseline architecture of a GPGPU NoC. White nodes are compute cores, and gray nodes are MCs.

3 Architecture of LA-LLC

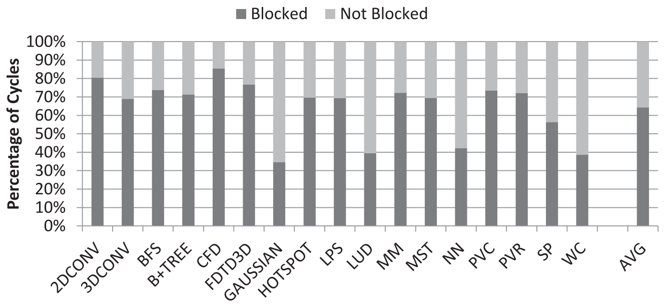
We now present the architecture of the inter-core locality-aware last-level cache (LA-LLC) and describe how it works.

Fig. 3 shows the overall architecture of the LA-LLC design. In particular, we add a core pointer (CoreID) to each cache block. The core pointer records the ID of the core that potentially has the cache block in its L1 cache. In our evaluation containing 56 compute cores, the core pointer only costs 6 bits. For GPGPUs with a 48-bit address space [10] and an L2 cache of 64 sets and eight ways, the extra hardware cost of the LA-LLC design is less than 0.57 percent of the L2 cache size. (Note that L2 is the LLC in our setup.)

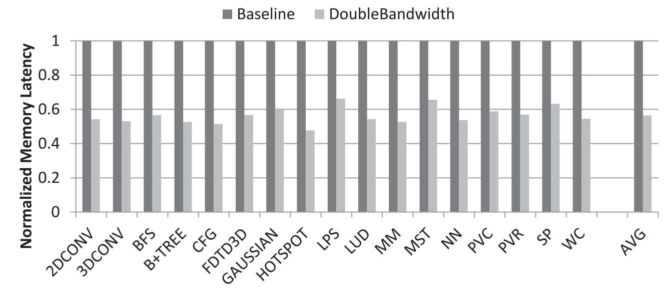
When an L1 cache miss occurs ①, its memory request is sent to the L2 cache through the request network ②. If the memory request cannot find the requesting cache block in the LLC, i.e., a cache miss occurs, a cache block is evicted based on the cache replacement policy. The tag area of the cache block is updated and its status is marked as reserved. At the same time, the core pointer is also updated and now points to the core sending the memory request ⑤.

In case of an LLC hit, the core pointer of the cache block is checked ③. If the core pointer points to a remote core, the memory request is sent to the corresponding core through the request network ④. However, if the core pointer points to the core that sent the memory request, the L2 will send the data to the requesting core ⑪. To be noticed, sending requests to remote cores will likely not cause congestion in the request network, as the request network experiences less traffic and is underutilized [6], [8].

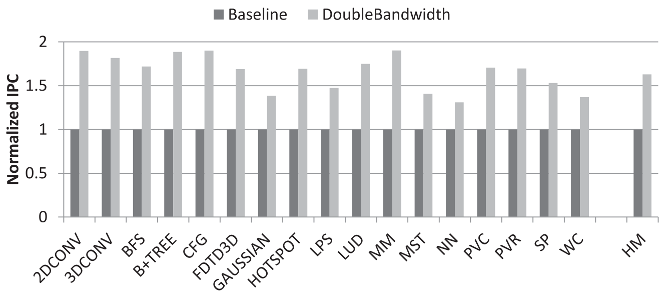
Once a core receives memory requests from the L2 cache ⑥, it injects them into the Request Queue (RQ). The L1 cache prioritizes remote memory requests from other cores in the RQ over local memory requests ⑦. The inverse, prioritizing local requests over remote requests, would stall remote requests and may even lead to a deadlock situation. Consider two cores that both experience a local cache miss causing them to stall on a particular resource (e.g., MSHR entry, cache line) being unavailable. A



(a) Blocked cycles of MCs that cannot inject reply packets, assuming a 8x8 mesh network with 2VCs.



(b) Memory access latency reduction by doubling bandwidth of the reply network.



(c) Performance improvement by doubling bandwidth of the reply network.

Fig. 2. Quantifying the reply network problem.

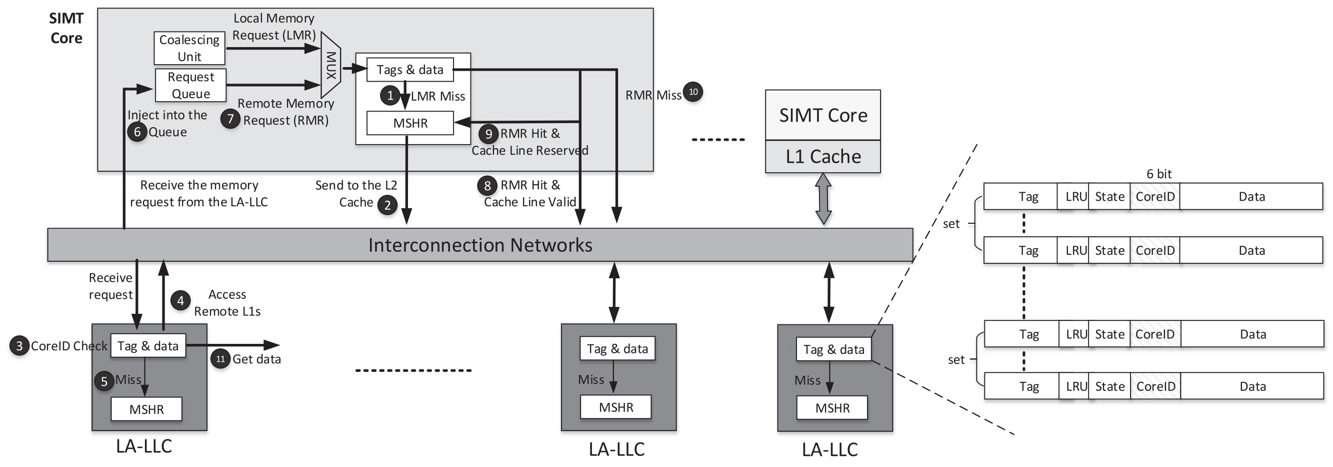


Fig. 3. The LA-LLC architecture.

TABLE 1
Configuration of GPGPU-Sim

Parameters	Value
Number of Cores	56
Schedulers/Core	2 (GTO)
L1 Data Cache/Core	16 KB, four-way, LRU, 128B line
Number of MCs	8
L2 Cache/MC	64 KB, eight-way, LRU, 128B line
DRAM Model	FR-FCFS, 16 banks/MC
GDDR5 Timing	t_{CL}=12, t_{RP}=12, t_{RC}=40, t_{RAS}=28, t_{RCD}=12, t_{RRD}=6, t_{CCD}=2, t_{WR}=12
Total Memory Bandwidth	59.1 GB/s
Topology	8×8 2D Mesh
Routing Function	XY routing
Channel Width	128 bits
Buffer Configuration	2VCs 4flit/VC

deadlock situation may occur if both cache misses lead to remote accesses to the other core.

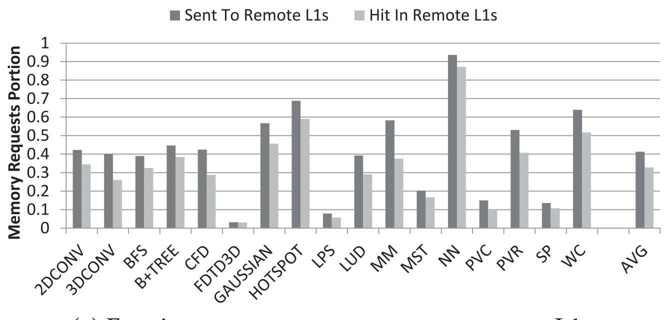
There are three possible outcomes when a local L1 cache serves a remote memory request from another core: (1) a cache hit and the data is available; (2) a cache hit but the data is not available now as the memory request has not yet come back; or (3) a cache miss. In the first case ⑧, the current core will send a memory reply to the requesting core through the reply network directly. In the second case ⑨, the memory request will be added to the list in the MSHR and the core will send the memory reply to the requesting core as soon as it gets the data and processes the MSHR list. In the third case ⑩, memory requests will be re-sent to the L2 cache upon which the L2 will send the data and update the core pointer ⑪. Although the third case does not benefit from the LA-LLC and incurs extra overhead, LA-LLC still reduces overall memory access latency because (i) this case is unlikely to happen, and (ii) the extra overhead can be overlapped by the long latency caused by the blocked MCs.

4 Evaluation

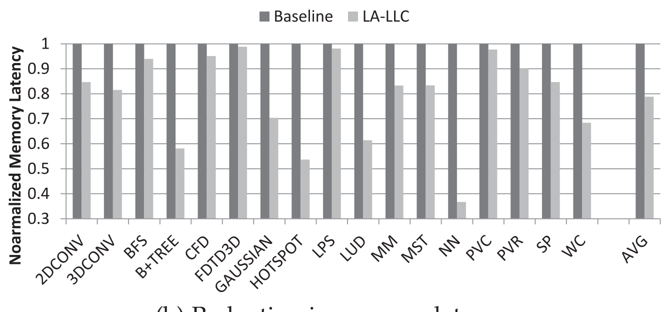
We modify GPGPU-sim v3.2.3 [14] to evaluate our design. The baseline configuration of GPGPU-sim is listed in Table 1 which is similar to previous work [4], [6], [8]. We evaluate LA-LLC using a set of benchmarks that exhibit varying degrees of inter-core locality from CUDA SDK [13], GPGPU-sim [14], Rodinia [12], LonestarGPU [16], Mars [15] and PolyBench [11], see Table 2 along with the benchmarks' injection rates in the reply network, measured in flits per node per cycle.

TABLE 2
Benchmarks

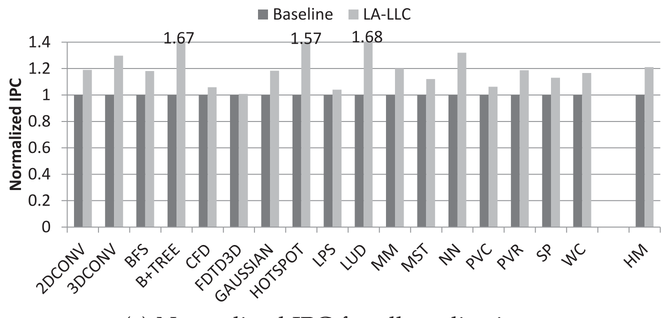
Name	Abbr.	Injection Rate (flits/cycle)
2dconvolution [11]	2DCONV	0.673
3dconolution [11]	3DCONV	0.583
BFS Graph Traversal [12]	BFS	0.622
B+ Tree [12]	B+TREE	0.616
CFD Solver [12]	CFD	0.704
3D Finite Difference Time Domain [13]	FDT3D	0.651
Gaussian Elimination [12]	GAUSSIAN	0.324
Rodinia Hotspot [12]	HOTSPOT	0.591
Laplace Solver [14]	LPS	0.639
LU Decomposition [12]	LUD	0.352
Matrixmul [15]	MM	0.598
Minimum Spanning Tree [16]	MST	0.567
Neural Network [14]	NN	0.446
Pageviewcount [15]	PVC	0.628
Pageviewrank [15]	PVR	0.609
Survey Propagation [16]	SP	0.552
Wordcount [15]	WC	0.331



(a) Fraction memory requests sent to remote L1s.



(b) Reduction in memory latency.



(c) Normalized IPC for all applications.

Fig. 4. Evaluation of LA-LLC.

LA-LLC enables memory read requests to fetch data from remote L1s and transfer data blocks between cores. Fig. 4a quantifies the fraction memory requests sent and hitting in remote L1s. The selected applications have different degrees of inter-core locality. NN exhibits the highest inter-core locality and more than 80 percent of the memory requests can fetch data from remote L1s. On the other hand, FDTD3D and LPS are low inter-core locality applications and less than 10 percent of the memory requests are sent to remote L1s. On average, 40 percent of the memory requests are sent to remote cores and 31 percent of the requests are satisfied by remote L1s. In other words, LA-LLC transforms 31 percent of the few-to-many traffic into many-to-many traffic. The 9 percent requests that cannot find the data in another core's L1 cache are sent back to the L2 cache again. As the request network experiences less traffic and is not the bottleneck [6], [8], sending requests to access remote L1s does not incur noticeable overhead.

By fetching data from remote L1s, the LA-LLC utilizes unused inter-core bandwidth and mitigates the bottleneck of the reply network. This reduces the reply network latency which is an important component of the overall memory access latency. As shown in Fig. 4b, for applications such as B+TREE, HOTSPOT and NN, the reduction in memory access latency exceeds 40 percent, and on average, LA-LLC achieves a 21.2 percent memory access latency reduction.

Fig. 4c shows the performance improvement of LA-LLC over the baseline. By reducing memory access latency, cores can get data replies faster. This reduces pipeline stalls due to memory operations and improves performance by 21.1 percent on average. As for B+TREE, HOTSPOT and LUD, they experience the largest performance improvements and performance improves by more than 50 percent. Although NN has the largest memory latency reduction, its performance improvement is not the highest. The overall performance improvement does not depend only on the reduction in memory latency. Only a fraction of the memory latency is exposed as stall cycles that directly affect performance. Other factors/components (e.g., L1 hit rates and structural bottlenecks) also play a role in the final performance figure. Applications exhibiting low inter-core locality, such as FDTD3D and LPS, get limited benefit from LA-LLC and their performance only improves by 0.76 and 3.98 percent, respectively.

5 Related Work

Previous NoC research leverages the many-to-few-to-many traffic pattern in GPGPUs to achieve a low-cost or high-performance design. To reduce hardware cost, Bakhoda et al. [2] simplify the crossbar structure by exploiting a checkboard routing algorithm. Ziabari et al. [17] propose asymmetric NoC designs for different types of traffic to achieve an energy-efficient network. Zhao et al. [8] eliminate conflicts in the request network and simplify the router architecture using a conflict-free design. To achieve higher performance, Kim et al. [4] propose the DA2mesh network which exploits a simplified router architecture and doubles the NoC frequency to improve performance. Jang et al. [6] assign more VCs to reply packets as the reply traffic generally requires more bandwidth. To maintain performance in high-throughput workloads in a bufferless network, Kim et al. [7] propose clumsy flow control to reduce the amount of deflection caused by network contention. Compared to these designs, LA-LLC is the first study to notice and leverage inter-core communication to mitigate the reply network bottleneck. LA-LLC does not modify the NoC architecture and mitigates the reply network bottleneck by exploiting inter-core locality.

6 Conclusion

GPGPUs have a unique traffic pattern called many-to-few-to-many which causes serious congestion in the reply network. In this paper, we observe that the underutilized core-to-core communication can be leveraged to alleviate this bottleneck. To exploit inter-core locality in GPGPUs, we propose LA-LLC which enables memory requests to fetch data from remote L1s instead of the LLC. By utilizing inter-core communication, LA-LLC transfers few-to-many traffic into many-to-many traffic and thereby mitigates the reply network bottleneck. Our evaluation shows that LA-LLC transforms 31 percent of the few-to-many traffic into communication between cores. This reduces memory access latency and improves performance by 21.1 percent on average.

References

[1] D. Li and T. M. Aamodt, "Inter-core locality aware memory scheduling," *IEEE Comput. Archit. Lett.*, vol. 15, no. 1, pp. 25–28, Jan. 2016.

[2] A. Bakhoda, J. Kim, and T. M. Aamodt, "Throughput-effective on-chip networks for Manycore accelerators," in *Proc. 43rd Annu. IEEE/ACM Int. Symp. Microarchit.*, 2010, pp. 421–432.

[3] L. G. Valiant, "A bridging model for parallel computation," *Commun. ACM*, vol. 33, pp. 103–111, 1990.

[4] H. Kim, J. Kim, W. Seo, Y. Cho, and S. Ryu, "Providing cost-effective on-chip network bandwidth in GPGPUs," in *Proc. IEEE 30th Int. Conf. Comput. Des.*, 2012, pp. 407–412.

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

[6] H. Jang, J. Kim, P. Gratz, K. H. Yum, and E. J. Kim, "A bandwidth efficient on-chip interconnects design for GPGPUs," in *Proc. 52nd Annu. Des. Autom. Conf.*, 2015, pp. 1–6.

[7] H. Kim, Y. Kim, and J. Kim, "Clumsy flow control for high-throughput bufferless on-chip networks," *Proc. IEEE Comput. Archit. Lett.*, 2013, pp. 47–50.

[8] X. Zhao, S. Ma, Y. Liu, L. Eeckhout, and Z. Wang, "A low-cost conflict-free NoC for GPGPUs," in *Proc. 53rd Annu. Des. Autom. Conf.*, 2016, pp. 1–6.

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

[10] T. G. Rogers, M. O'Connor, and T. M. Aamodt, "Cache-conscious wavefront scheduling," in *Proc. 45th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2012, pp. 72–83.

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

[12] S. Che, et al., "Rodinia: A benchmark suite for heterogeneous computing," in *Proc. IEEE Int. Symp. Workload Characterization*, 2009, pp. 44–54.

[13] NVIDIA CUDA SDK code samples. "NVIDIA corporation." (2015). [Online]. Available: https://developer.nvidia.com/cuda-downloads

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

[15] B. He, W. Fang, Q. Luo, N. K. Govindaraju, and T. Wang, "Mars: A MapReduce framework on graphics processors," in *Proc. 17th Int. Conf. Parallel Archit. Compilation Techn.*, 2008, pp. 260–269.

[16] M. Burtscher, R. Nasre, and K. Pingali, "A quantitative study of irregular programs on GPUs," in *Proc. IEEE Int. Symp. Workload Characterization*, 2012, pp. 141–151.

[17] A. K. Ziabari, J. L. Abellán, Y. Ma, A. Joshi, and D. Kaeli, "Asymmetric NoC Architectures for GPU Systems," in *Proc. 9th Int. Symp. Netw.-on-Chip*, 2015, pp. 1–8.