The Design of NoC-side Memory Access Scheduling for Energy-Efficient GPGPUs

Wenjie Liu, Sheng Ma, Libo Huang, and Zhiying Wang

College of Computer, National University of Defense Technology, Changsha, China
{liuwenjie15,masheng,libohuang,zywang}@nudt.edu.cn

Abstract. Memory access scheduling schemes, often performed in memory controllers, have a marked impact on alleviating the heavy burden placed on memory systems of GPGPUs. Existing out-of-order scheduling schemes, like FR-FCFS, improve memory access efficiency by reordering memory request sequences at the destination. Their effectiveness, however, is at the expense of complex logics and high power consumption. In this paper, we propose a NoC-side memory access scheduling based on the key insight that the transmission of on-chip networks is the dominating factor in destroying the row access locality and causing poor memory access efficiency. With appropriate NoC-side optimization, straight-forward in-order scheduling can be used in memory controllers to simplify scheduling logics and alleviate the tight power envelope. Moreover, we introduce several light-weight optimizations to further improve system performance. Experimental results on memory-intensive applications show that, comparing with FR-FCFS, our proposed scheme increases the overall system performance by 10.5%, reduces the power consumption by 20% and improves the energy efficiency by 36.9%.

Keywords: GPGPU · Networks-on-Chip · Memory Access Scheduling

1 Introduction

Communication in GPGPUs mainly happens between many compute cores and a few memory controllers (MCs). This many-to-few-to-many traffic places heavy burden on memory systems[2]. To be specific, thousands of simultaneous active threads executing in GPGPUs will generate massive memory requests, which often creats a memory access bottleneck. Therefore, maximizing memory access efficiency is crucial to alleviate this bottleneck and then obtain high overall performance.

Going through current designs of MCs, we find that most of them employ out-of-order scheduling to maximize the row access locality and in turn maximize the memory access efficiency. For example, First-Ready First-Come First-Service (FR-FCFS)[17] is a widely-used scheduling scheme due to its effectiveness in exploiting row access locality among multiple pending requests in MCs. However, the high performance from FR-FCFS and its extensions [13–16] is obtained at the expense of high area and power overheads. These overheads are acceptable now but will become a more severe problem with the increasing amount of concurrently running threads in GPGPUs [6]. Thus, the mistuning between current memory access scheduling and fast-scaling GPGPUs fuels an increasing demand for an effective alternative to keep high memory access efficiency and reduce power consumption at the same time.

Yuan G L et al. [19] have observed that memory requests from GPGPU compute nodes usually have sufficient row access localities. This observation inspires us to design memory scheduling from a brand-new perspective: Prevalent out-of-order scheduling schemes, like FR-FCFS, maximize the memory access efficiency by reordering memory requests *at the destination*. However, after analyzing the whole transmission procedure of memory requests from source nodes via NoC to destination nodes, we find that the problem actually stems from the NoC. In other words, it is the transmission of NoC that destroys the row access locality originally high *at the source*, finally leading to poor memory access efficiency at the destination. Therefore, the key to an effective alternative is to maintain the row access locality *during the transmission on the on-chip networks*. With proper NoC-side optimization, simplified in-order scheduling can be used again in MCs to reduce hardware overhead and power consumption.

The key contributions made in this work are:

- We provide a new perspective of designing memory scheduling—that is, protecting the row access locality from the interference of NoC transmission is much more important than reordering memory requests at the destination.
- Based on the aforementioned observation, we propose a novel NoC-side memory access scheduling. This scheme can maintain the row access locality more effectively and help to simplify memory-side scheduling at the same time.
- We introduce two light-weight optimizations to hide performance loss caused by simplified memory-side scheduling.

The reminder of this paper is organized as follows. Section 2 details the proposed NoC-side memory access scheduling and light-weight memory-side optimizations. Section 3 introduces the experimental methodology, followed by an in-depth analysis of experimental results in Section 4. In Section 5, we review the related work and Section 6 concludes this paper.

2 The Design of NoC-side Memory Access Scheduling

Section 2.1 analyzes the influence of NoC on the row access locality. Then, we propose a Same Source First (SSF) NoC Arbitration and a Destination-oriented Virtual Channel Partitioning (DVCP) in Section 2.2. In Section 2.3, we optimize the memory-side architectures to improve the system performance.

2.1 Row Access Locality Analysis

GPGPU used in this paper includes 28 compute nodes and 8 memory nodes, all of which compose a 6×6 Mesh NoC. We choose ten applications to explore

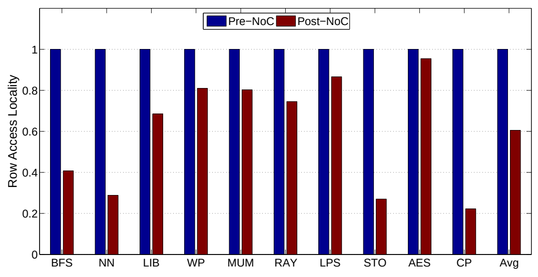

Fig. 1. The proportion of row access locality after on-chip networks transmission comparing with that at the source compute nodes.

the influence of on-chip networks on the row access locality. Figure 1 illustrates the proportion of remaining row access locality at the destination to that at the source. As shown in this figure, the row access locality of all applications is damaged from the on-chip networks transmission, especially CP, NN and STO. The remaining row access locality of those three applications drops to less than 30% comparing with that at the source. On average, only 60% of original row access locality is remained after arriving at the destination.

Significant reduction of the row access locality at the destination verifies that the interference of NoC transmission is the main reason for low memory access efficiency. Therefore, memory access scheduling should focus on optimizing the NoC transmission.

2.2 NoC-side memory access scheduling

The proposed NoC-side memory access scheduling includes two complementary strategies. In the first strategy, we introduce the source node information of memory requests into the NoC arbitration. In the second strategy, the destination node information of memory requests is taken into consideration. Therefore, both the source and destination node information are leveraged to optimize the NoC transmission, thus maintaining the row access locality.

Same Source First (SSF) NoC Arbitration. Existing arbitrations hold fairness to provide equal services and avoid network starvation. The problem of fair arbitrations is that memory requests may obtain different priorities of resource allocation at different intermediate nodes. As a result, the inherent row access locality of memory requests will be significantly destroyed after an interleaved transmission. Therefore, to maintain the row access locality during transmission, we introduce the source node information of memory requests into the proposed NoC arbitration. To avoid network starvation, the proposed arbitration still keeps some fairness based on the round-robin arbitration.

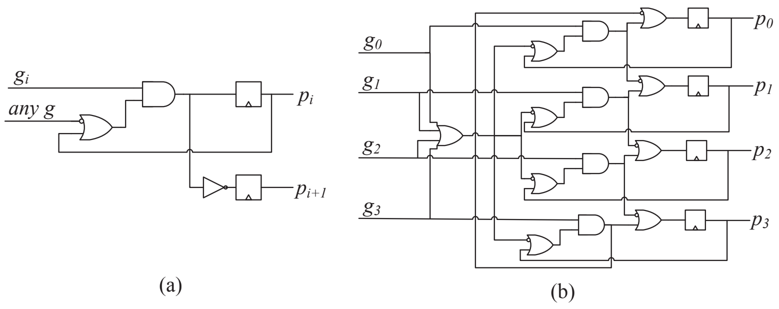

Fig. 2. The logic circuit of SSF arbitration. (a) The basic logic of a single input i. (b) The complete logic of a four-input SSF Arbiter.

To be specific, the proposed SSF Arbitration is designed as follows: *If request* $i+1$ *is generated from the same compute node as the winning request i in the previous arbitration cycle, request* $i{+}1$ *will be granted the access to corresponding resource again in current cycle. Otherwise, arbitration in* $i{+}1$ *cycle will be carried on according to round-robin policy.* Taking a certain input i as an example, Figure 2(a) illustrates its basic arbitration logic. Suppose input i got the access to output o in the last cycle—that is, line of p_i goes high. If input i still competes for the access to output o in current cycle, line of g_i will be high. Once these two conditions satisfied, line of p_i will keep high in current cycle, which means input i will use output o again. Otherwise, line of p_{i+1} will be high and the request next to i will be served in this cycle according to round-robin arbitration. Figure 2(b) shows the complete logic of a four-input SSF arbiter.

Arbiters often function as basic subcomponents of allocators in on-chip networks. Throughout existing allocators, separable allocators are widely-used for their great resource matching and high allocation efficiency. Therefore, SSF arbitration will be used in the design of separable allocators to form SSF-based input-first separable allocators in this paper.

Destination-oriented Virtual Channel Partitioning (DVCP). Known as 'swiss-army knife' of on-chip networks, virtual channels (VCs) were first proposed to avoid network deadlock and extend the bandwidth [7]. Current methods of VC partitioning, no matter in static or dynamic manners, always intend to assign the first channel of idle VC queues to the pending request. This mechanism causes path diversity and further does damage to the row access locality because of its underlying randomness of resource partitioning. Therefore, the second strategy of NoC-side scheduling is to redesign the VC partitioning according to the destination node information of various memory requests. Suppose there are n_M memory nodes in NoC and n_{VC} virtual channels per input port for a router. The proposed DVCP mainly consists of the following two compute stages.

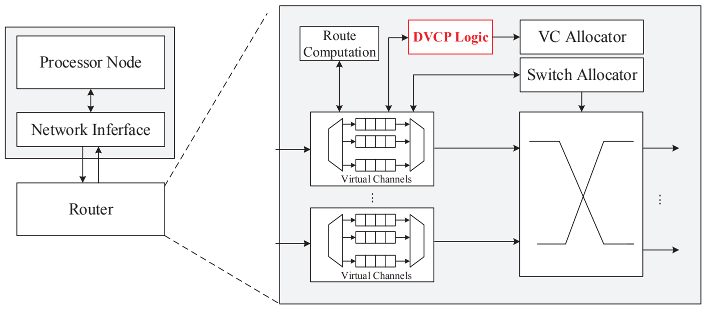

Fig. 3. The router microarchitecture with newly-added Destination-oriented Virtual Channel Partitioning Logic (marked in red).

1) Compute the total number of VCs every memory node can use according to Formula (1):

$$t_M = \frac{n_{VC}}{n_M} + 1 \tag{1}$$

2) For all memory requests with memory node m as their destination, compute the serial number of their accessed VCs according to Formula (2):

$$V_{m_k} = (m + n_M \times k)\%(n_{VC} - 1) \quad k = 0, 1, ..., t_M - 1 \tag{2}$$

where the set composed of all V_{m_k} is a collection of VCs used to transfer memory requests to memory node m. The sequent allocation for those memory requests can only works on this set.

The baseline router in this paper is the wormhole flow control router. The head flit of the packet goes through four logical pipeline stages: Route Computation, VC Allocation, Switch Allocation and Switch Traversal. After introducing DVCP (shown in Fig.3), VC allocation is divided into two stages: the DVCP logic firstly computes the available set of VCs according to the destination information of current inputs. Output of the first stage is then fed into the VC allocator as the input of the second stage. Calculations within DVCP logic can be implemented into bit operations because the number of VCs and memory nodes are generally set as powers of two in actual hardware. Our scheme indicates that VC allocation can still be finished in one clock cycle after introducing DVCP.

2.3 Memory-side optimizations

After moving memory scheduling into NoC, scheduling at memory side can be simplified into FIFO policy. The FIFO-like policy can effectively reduce power overhead but meanwhile, causes performance loss. In this subsection, we will introduce two performance-enhanced strategies implemented at memory side—the batched-FIFO policy and the multi-port router microarchitecture.

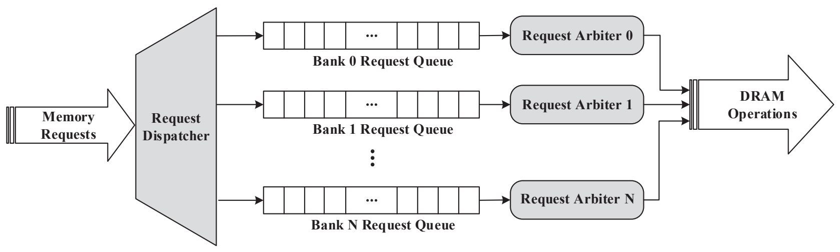

Fig. 4. The architecture of MC with a batched-FIFO policy.

Batched-FIFO policy. The bank-level parallelism (BLP) is crucial to DRAM performance. To explore the BLP of memory request streams, we add a buffered queue for each bank to support memory access in batches. As shown in Fig. 4, the batched-FIFO policy mainly consists of three components, which are Request Dispatcher, Request Queue and Request Arbiter. When memory requests arrive at MCs, the request dispatcher buffers them into corresponding queues according to their accessed banks. Buffered requests are then scheduled based on FIFO policy. Once requests leaving buffered queues, the request arbiter will resolve the accessed memory addresses and generate memory access operations like *precharge* according to the current memory status. With this policy, requests buffered in different queues can be arbitrated concurrently in the same clock cycle, thus improving the DRAM access efficiency and further achieving higher system performance.

Multi-port Router Microarchitecture. Heavy traffic between memory-side routers and memory nodes often causes a bottleneck and degrades the system performance. A simple solution to this problem is to increase resources needed by this traffic. As shown in Fig. 5, the traffic between routers and memory nodes includes two procedures called ejection and injection respectively. The typical architecture of a router only contains one ejection port and one injection port, which is far from enough for the many-to-few-to-many traffic pattern. So the first step of our design is to double both of the ejection and injection ports. These additional ports can provide more ejection and injection bandwidth so as to help MCs serve requests more efficiently.

The second part of our design is to configure the buffer capacity. Figure 5 shows the buffers involved in ejection/injection procedures, which are Ejection Buffers, Boundary Buffers and Injection Buffers. In the procedure of ejection, memory requests are pushed into and out of ejection buffers at the NoC clock frequency (1 flit/cycle). Boundary buffers are designed to help cross from one clock domain to the other—memory requests are pushed into boundary buffers at the NoC clock frequency (1 flit/cycle) and removed from them at the memory clock frequency (1 packet/cycle). In the procedure of injection, there is no disturbance from clock domain change. So it only needs injection buffers to travel memory replies at the rate of 1 flit/cycle. Overall, both ejection and injection

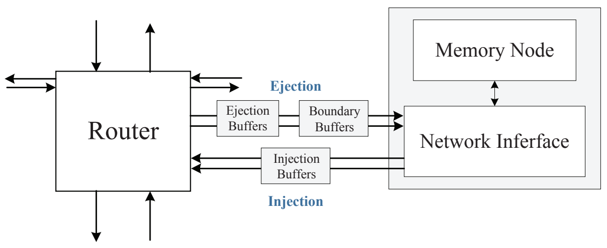

Fig. 5. Multi-port router microarchitecture. Five-port router is adopted in this design. The I/O ports of four directions (East, West, South and North) has no change and the local injection/ejection ports are doubled to alleviate the memory access bottleneck.

Table 1. Baseline configuration of GPGPU-Sim and the interconnection network

GPGPU-Sim		**Interconnection Network**	
Number of Shader Cores	28	Topology	6×6 Mesh
Number of Memory Controllers	8	Routing	Dimensional-order
Number of Memory Banks	64	Number of VCs	1,2,4
Number of Warps/SM	32	VC Buffer Size	8
Warp Size	32	VC Allocator	Input-first Separable
Number of Threads/Core	1024	SW Allocator	PIM
Number of Shader Registers	16384	Traffic	Uniform
DRAM Scheduling Policy	FR-FCFS, FIFO	Flit Size	32(default)

buffers work at the NoC clock frequency, thus they always have space for coming flits and require no additional resources. For boundary buffers, the gap between high input rate and low output rate often causes a traffic jam. Hence we double the storage capacity of boundary buffers to alleviate the heavy traffic.

3 Methodology

We evaluate our scheme in GPGPU-Sim [1], a cycle-level performance simulator that focuses on general purpose computation on GPUs. The power consumption is evaluated through GPUWattch [11], a power model integrated with GPGPU-Sim version 3.2.0 and later.

The left part of Table 1 describes the architectural parameters of GPGPU-Sim. It includes 28 compute nodes (shader cores) and 8 memory nodes (MCs). Every compute node can hold 1024 threads at most and it schedules threads at the size of warp. Every memory node contains two memory channels and 64 memory banks. The configuration of NoC is shown in the right part of Table 1. We adopt a 6×6 2D mesh topology and the dimensional-order routing with 4 virtual channels. The proposed SSF arbitration will be used in VC allocation, so input-first separable allocators are selected as the baseline. No changes happening on SW allocators, so we perform PIM, a simple but efficient allocation, in the design.

We evaluate our design with 11 benchmarks from Parboil [18] and ISPASS [1], as shown in Fig. 6. According to the proportion of memory access instructions

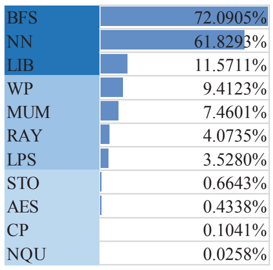

Fig. 6. Benchmarks and their classifications. From top to bottom, they are 3 heavy memory-intensive (>10% memory access instructions of all) applications, 4 light memory-intensive (1%~10% memory access instructions of all) applications and 4 compute-intensive (<1% memory access instructions of all) applications, marked in different colors.

to the whole instructions, we classify these benchmarks into three groups which are heavy memory-intensive, light memory-intensive and compute-intensive respectively.

4 Experimental Results

We compare the proposed design with FR-FCFS, a state-of-the-art memory access scheduling scheme. Section 4.1 shows the evaluation on memory-level metrics including the row access locality and the memory fetch latency. Section 4.2 analyzes influences of our design on system-level metrics including the system performance, total power consumption and the energy efficiency.

4.1 Memory Access Evaluation

We firstly present the row access locality of FR-FCFS and the proposed scheduling SSF-DVCP-Batched-Multiport-FIFO (SDBM-FIFO for short) at the destination. As shown in Fig. 7, SDBM-FIFO maintains higher row access locality for most benchmarks except WP and LPS. Specifically, row access locality at the destination for all memory-intensive applications increases by 11.7% and the average improvement of all applications achieves 7%. We introduce both the source and destination node information into the NoC-side memory access scheduling, which contributes the most to optimize the NoC transmission and maintain the row access locality.

Figure 8 illustrates the memory fetch latency of SDBM-FIFO normalized to FR-FCFS. Among all benchmarks, the memory fetch latency of NN and LIB drops the most—12.7% and 12.4% respectively. On average, SDBM-FIFO can reduce the memory fetch latency by 4.63% for memory-intensive benchmarks and 3.1% for all benchmarks. The memory fetch latency defined in this paper refers to the round trip latency from compute nodes to memory nodes and back,

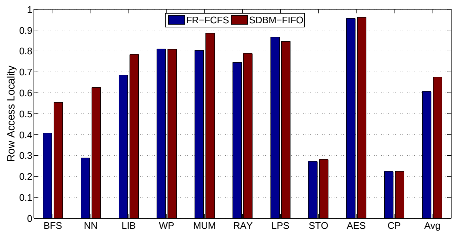

Fig. 7. Row access locality at the destination

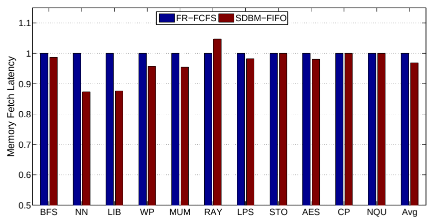

Fig. 8. Memory fetch latency normalized to FR-FCFS

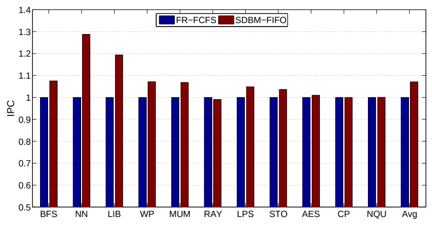

Fig. 9. System Performance of SDBM-FIFO normalized to FR-FCFS

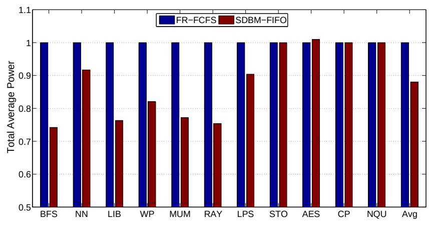

Fig. 10. System power of SDBM-FIFO normalized to FR-FCFS

so NoC transmission latency and memory response latency all contribute to this metric. NoC transmission latency may be increased as a result of optimizing arbitration and VC partitioning but it can be hidden from the improvement of memory access efficiency for most applications. The RAY application, however, is an exception. Its reduction of memory response latency cannot compensate for the increase of NoC latency, resulting in 5% rise of overall memory fetch latency.

4.2 System-Level Metrics Evaluation

We use the IPC to evaluate the system performance of our scheme and FR-FCFS. As shown in Fig. 9, NN and LIB still achieve the greatest performance improvements (28.8% and 19.4%, respectively). However, performance of RAY experiences a slight drop influenced by its rise of memory fetch latency. On average, the IPC rises by 10.5% for all memory-intensive applications and 7.1% for the whole applications. We evaluate our scheme on both memory-intensive and compute-intensive applications. For compute-intensive applications, the proposed scheme can protect them from performance loss or power increase at least. The results show that our design can be suitable for various applications.

Results in Fig. 10 indicate that comparing with FR-FCFS, SDBM-FIFO can significantly reduce the power consumption of memory-intensive applications to 80%. The power reduction of most memory-intensive applications exceeds 20% (e.g., 25.8% for BFS). Even the least reduction of power consumption reaches 8.3% (achieved by NN).

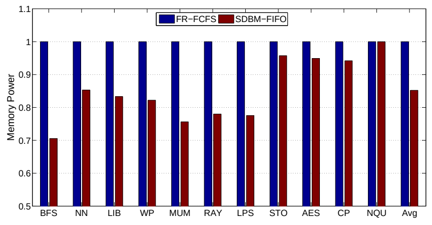

Fig. 11. Memory power of SDBM-FIFO normalized to FR-FCFS

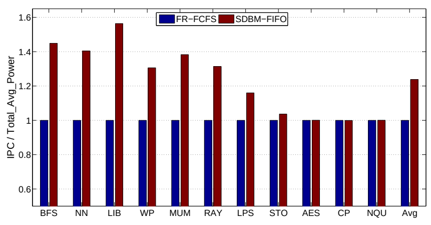

Fig. 12. Energy efficiency of SDBM-FIFO normalized to FR-FCFS

To explain the significant power consumption after adopting SDBM-FIFO, we analyze the components of system power. We find that execute unit power, constant access power and memory (including MC and DRAM) power are three main parts of system power. Our research has nothing to do with the first two parts but optimizes the third part, which is the one accounting for the largest proportion of total system power. Specifically, FR-FCFS needs to perform fully-associative comparisons of all waiting requests every scheduling cycle, which requires a large number of comparators and causes considerable memory power. SDBM-FIFO, on the contrary, can simplify the scheduling at memory nodes to a FIFO-based policy. This policy only needs a simple FIFO queue, therefore saves most comparators and becomes the main contributor of system power reduction. Figure 11 presents the statistics of memory power. The memory power of all applications drops by 14.8% using SDBM-FIFO. For memory-intensive applications, their memory power declines to 78.9% on average and four of them reach a reduction over 20%.

The *energy efficiency* in this paper is defined as the ratio of system performance to system power consumption—that is $IPC/Total\_Average\_Power$. Normalized results in Fig. 12 clearly present that SDBM-FIFO improves the energy efficiency of all applications by 23.8% compared with FR-FCFS. This figure reaches up to 36.9% for all memory-intensive applications. Moreover, the energy efficiency of BFS, NN and LIB increases by more than 40%. Such considerable improvements on the energy efficiency mainly benefit from the high performance and low power consumption achieved by our scheme.

5 Related Work

Networks-on-Chip (NoCs) [5] have emerged as a popular communication fabric in multi-core chips for their scalability and high bandwidth. To make NoCs applicable to chip-multiprocessor (CMP) environments, many researchers have proposed a variety of mechanisms involving topologies, routing algorithms and arbitration policies. For example, S. Bourduas and Z. Zilic [3] proposed a hybrid ring/mesh topology to reduce communication radius. S. Ma et al. [12] designed an efficient routing algorithm named DBAR to support multiple concurrent applications in on-chip networks. Although designs of NoCs have matured in CMPs, corresponding designs for GPGPUs are still in their infancy. Only a handful of works have explored design space of NoCs in GPGPUs [19, 8, 10, 2]. Unlike uniform core-to-core communications in CMPs, the many-to-few-to-many traffic in GPGPUs puts heavy demands on the memory systems, leading to limited off-chip throughput. Therefore, out-of-order scheduling schemes like FR-FCFS [17] and its extensions [13–16] have been applied extensively to maximize the row access locality and in turn maximize the memory access efficiency. Unfortunately, high performance of out-of-order scheduling accompanies with complex logic and high power consumption. Therefore, several schemes have been proposed trying to fill the growing gap between performance demand and power constraints in GPGPUs. For example, Yuan G L et al. [19] have proposed an alternative based on in-order scheduling to reduce the complexity of MCs. Kim, Y. et al. [9] performed memory access scheduling at the source by grouping multiple requests into a single *superpacket*. The work in [4] also focused on merging memory request packets but it exploited the coalescing opportunity across from different compute cores. Even though these works alleviated power constraints to some extent, they all achieved lower performance than FR-FCFS. We solve this problem by moving memory access scheduling from memory side to NoC side and meanwhile, introducing several performance-enhanced designs.

6 Conclusion

In this paper, we propose a novel NoC-side memory access scheduling to achieve low power consumption and keep high system performance for GPGPUs. Our scheme consists of two main parts. We firstly move memory access scheduling from traditional memory side to NoC side, because it is the NoC that interleaves the transmission of memory requests and then reduces the memory access efficiency. After that, we do some optimizations at memory side to improve system performance. Evaluation results indicate that our work increases the system performance by 10.5% and reduces the power consumption by 20%. A 36.9% increase in the energy efficiency means our scheme can obtain higher performance per unit power consumption than FR-FCFS.

References

1. Bakhoda, A., Yuan, G.L., Fung, W.W.L., Wong, H., Aamodt, T.M.: Analyzing cuda workloads using a detailed gpu simulator. In: 2009 IEEE International Symposium on Performance Analysis of Systems and Software. pp. 163–174 (2009)
2. Bakhoda, A., Kim, J., Aamodt, T.M.: Throughput-effective on-chip networks for manycore accelerators. In: Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture. pp. 421–432. IEEE Computer Society (2010)
3. Bourduas, S., Zilic, Z.: A hybrid ring/mesh interconnect for network-on-chip using hierarchical rings for global routing. In: Proceedings of the First International Symposium on Networks-on-Chip. pp. 195–204. IEEE Computer Society (2007)

4. Chen, C.T., Huang, Y.S.C., Chang, Y.Y., Tu, C.Y., King, C.T., Wang, T.Y., Sang, J., Li, M.H.: Designing Coalescing Network-on-Chip for Efficient Memory Accesses of GPGPUs, pp. 169–180. Springer Berlin Heidelberg (2014)
5. Dally, W.J., Towles, B.: Route packets, not wires: on-chip interconnection networks. In: Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232). pp. 684–689 (2001)
6. Jang, H., Kim, J., Gratz, P., Yum, K.H., Kim, E.J.: Bandwidth-efficient on-chip interconnect designs for gpgpus. In: Proceedings of the 52Nd Annual Design Automation Conference. pp. 9:1–9:6. ACM (2015)
7. Jerger, N.E., Peh, L.S.: On-chip networks. Synthesis Lectures on Computer Architecture 4(1), 1–141 (2009)
8. Kim, H., Kim, J., Seo, W., Cho, Y., Ryu, S.: Providing cost-effective on-chip network bandwidth in gpgpus. In: 2012 IEEE 30th International Conference on Computer Design (ICCD). pp. 407–412 (2012)
9. Kim, Y., Lee, H., Kim, J.: An alternative memory access scheduling in manycore accelerators. In: 2011 International Conference on Parallel Architectures and Compilation Techniques. pp. 195–196 (2011)
10. Lee, J., Li, S., Kim, H., Yalamanchili, S.: Adaptive virtual channel partitioning for network-on-chip in heterogeneous architectures. ACM Trans. Des. Autom. Electron. Syst. 18(4), 48:1–48:28 (2013)
11. Leng, J., Hetherington, T., ElTantawy, A., Gilani, S., Kim, N.S., Aamodt, T.M., Reddi, V.J.: Gpuwattch: Enabling energy optimizations in gpgpus. In: Proceedings of the 40th Annual International Symposium on Computer Architecture. pp. 487–498. ISCA '13, ACM (2013)
12. Ma, S., Enright Jerger, N., Wang, Z.: Dbar: An efficient routing algorithm to support multiple concurrent applications in networks-on-chip. In: Proceedings of the 38th Annual International Symposium on Computer Architecture. pp. 413–424. ACM (2011)
13. Mutlu, O., Moscibroda, T.: Stall-time fair memory access scheduling for chip multiprocessors. In: Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture. pp. 146–160. IEEE Computer Society (2007)
14. Mutlu, O., Moscibroda, T.: Parallelism-aware batch scheduling: Enhancing both performance and fairness of shared dram systems. In: Proceedings of the 35th Annual International Symposium on Computer Architecture. pp. 63–74. IEEE Computer Society (2008)
15. Nesbit, K.J., Aggarwal, N., Laudon, J., Smith, J.E.: Fair queuing memory systems. In: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture. pp. 208–222. IEEE Computer Society (2006)
16. Rafique, N., Lim, W.T., Thottethodi, M.: Effective management of dram bandwidth in multicore processors. In: 16th International Conference on Parallel Architecture and Compilation Techniques (PACT 2007). pp. 245–258 (2007)
17. Rixner, S., Dally, W.J., Kapasi, U.J., Mattson, P., Owens, J.D.: Memory access scheduling. In: Proceedings of the 27th Annual International Symposium on Computer Architecture. pp. 128–138. ACM (2000)
18. Stratton, J.A., Rodrigues, C., Sung, I.J., Obeid, N., Chang, L.W., Anssari, N., Geng, D., Liu, W.M., Hwu, W.: Parboil: A revised benchmark suite for scientific and commercial throughput computing (2012)
19. Yuan, G.L., Bakhoda, A., Aamodt, T.M.: Complexity effective memory access scheduling for many-core accelerator architectures. In: 2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). pp. 34–44 (2009)