Designing Coalescing Network-on-Chip for Efficient Memory Accesses of GPGPUs

Chien-Ting Chen[1], Yoshi Shih-Chieh Huang[1], Yuan-Ying Chang[1], Chiao-Yun Tu[1], Chung-Ta King[1], Tai-Yuan Wang[1], Janche Sang[2], and Ming-Hua Li[3]

[1] Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan
[2] Department of Computer and Information Science, Cleveland State University, Cleveland, OH, USA
[3] Information and Communications Research Laboratories, Industrial Technology Research Institute, Hsinchu, Taiwan

Abstract. The massive multithreading architecture of General Purpose Graphic Processors Units (GPGPU) makes them ideal for data parallel computing. However, designing efficient GPGPU chips poses many challenges. One major hurdle is the interface to the external DRAM, particularly the buffers in the memory controllers (MCs), which is stressed heavily by the many concurrent memory accesses from the GPGPU. Previous approaches considered scheduling the memory requests in the memory buffers to reduce switching of memory rows. The problem is that the window of requests that can be considered for scheduling is too narrow and the memory controller is very complex, affecting the critical path. In view of the massive multithreading architecture of GPGPUs that can hide memory access latencies, we exploit in this paper the novel idea of rearranging the memory requests in the network-on-chip (NoC), called packet coalescing. To study the feasibility of this idea, we have designed an expanded NoC router that supports packet coalescing and evaluated its performance extensively. Evaluation results show that this NoC-assisted design strategy can improve the row buffer hit rate in the memory controllers. A comprehensive investigation of factors affecting the performance of coalescing is also conducted and reported.

Keywords: Network-on-chip, general-purpose graphic processors unit, memory controller, latency hiding, router design.

1 Introduction

Modern General Purpose Graphic Processors Units (GPGPUs) have over ten to hundred times more computing power than general purpose processors [15,19]. GPGPUs are thus well suited for high performance computing [14,16,20]. A GPGPU typically contains many *streaming multiprocessors* (SMs), sometimes referred to as *shader cores*, each is composed of many small *streaming processors* (SPs).

A *warp*, consisting of multiple threads, is the basic unit of scheduling on a SM. Multiple warps can be assigned to a SM and synchronized only within the

C.-H. Hsu et al. (Eds.): NPC 2014, LNCS 8707, pp. 169–180, 2014.
© IFIP International Federation for Information Processing 2014

SM. When a warp is blocked due to memory access, other ready warps may be executed so that the SM is not idle. The amount of time when a warp is context-switched out until it is scheduled for execution again is known as the *slack time* [21]. If the slack time is longer than memory access time, the latency in memory accesses is effectively hidden.

The massive multithreading architecture of GPGPUs makes them ideal for data parallel computing. However, the many concurrently memory accesses out of data parallel computing also severely stress the memory, particularly the buffers in the memory controllers (MCs). The problem is made even worse due to the *many-to-few-to-many* traffic patterns [4] in GPGPUs, which is from *many* SMs to *few* MCs and then back to many SMs.

As DRAM cells are typically organized in a two-dimensional array and to access a cell, a whole row of cells need to be loaded into the *row buffer* first, previous approaches to improving memory performance in GPGPUs focus on scheduling memory requests in the request queue in MCs to increase the *row buffer hit rate* [11,18]. The problem is that the window of requests that can be considered for scheduling is too narrow and the memory controller becomes complicated, affecting the critical path [22].

In view of the latency-hiding capability of GPGPUs, we exploit in this paper the novel idea of rearranging the memory requests in the network-on-chip (NoC), called *packet coalescing*. The idea is to merge memory requests destined for the same row of the memory in the routers of the NoC. In this way, the *many* memory requests from the different SMs may be merged to a *few* large packets along the path. When they arrive at the MC, memory requests are already in proper order for continuous row buffer hits. Note that packets may be delayed in the NoC for coalescing opportunity. However, the gain in faster memory accesses may more than compensate the delays. To study the feasibility of this idea, we have designed an expanded NoC router that supports packet coalescing and evaluated its performance extensively.

The contributions of this paper are as follows:

- We propose the novel idea of coalescing and reordering memory requests in the NoC in a distributed manner to improve the row buffer hit rate of the memory controllers.
- We design an enhanced NoC that supports packet coalescing.
- We provide an in-depth evaluation of the proposed architecture, investigate possible sources of inefficiency, and study ways to mitigate the problems.

2 System Design

To study the feasibility of packet coalescing, we present in this section an expanded NoC router based on a typical five-stage pipeline, which consists of *Input Buffering* (IB), *Routing Computation* (RC), *Virtual-channel Allocation* (VA), *Switch Allocation* (SA), and *Link Traversal* (LT) [7,8]. To extend the router for packet coalescing, a number of components are added: *Coalescer*, *Scheduler*, *Detector* and *Grant Holder*. They are mainly implemented in the *IB* and *SA* stages.

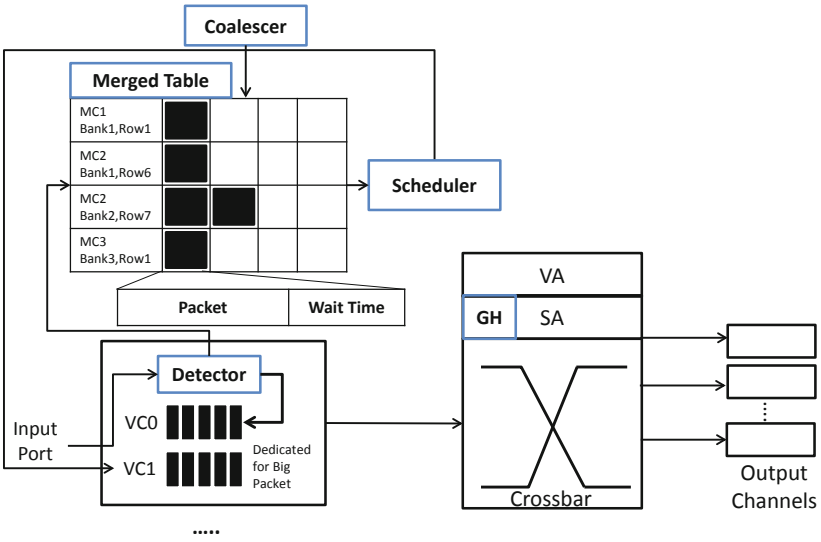


Fig. 1. The extended router microarchitecture for packet coalescing

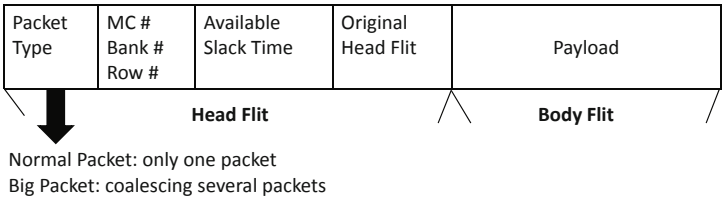


Fig. 2. Extended packet format for packet coalescing

2.1 Router Microarchitecture

Fig. 1 shows the proposed router microarchitecture. To match the extended router, the packet format should also be modified as shown in Fig. 2. The first field, *Packet Type*, indicates whether the packet is a normal packet or a coalesced, long packet. The second field contains information of the destination of the request to the row of a memory bank. This information is used to decide whether two packets can be merged together or not.

– *Detector*: When the router receives a head flit indicating it is a read request, Detector decides for how long the packet should wait for coalescing. The decision is affected by factors such as the router location, the routing algorithmth, available slack time of the packet, number of available warps, and GPGPU architecture. For example, if a router is closer to the memory controller, it will see more packets that could be coalesced, and thus it is

nature to hold a packet longer there. Detailed strategies will be discussed later.

- *Coalescer*: If it is decided that a packet should wait for coalescing, then Coalescer stores this packet to the *Merged Table*. The *Merged Table* is indexed by the MC id, bank id, and row id. When packets are stored in the Merge Table, their packet type is changed to *Big Packet*, as Figure 2 shows. If a new packet arrives that can be merged with another packet already in the Merged Table, the headers of the two packets are merged with an updated packet size and available slack time. Their payloads are concatenated together to form the payload of the new packet.
- *Scheduler*: Scheduler checks in every cycle if there is any packet in the Merged Table whose held time has expired. Such packets contains the memory requests destined to the same row of the same bank in the same memory controller. They will be removed from the Merged Table and placed in the virtual channel to be sent to the next hop. Strategies to schedule these big packets will be discussed in the next section.
- *Grant Holder*: To avoid a Big Packet from being transferred apart due to the fairness mechanism of the *Switch Allocation* pipeline stage, the *Grant Holding* (GH) mechanism is added to let a Big Packet hold the grant until the entire packet is transferred.

2.2 Design Issues

One important design issue is where the packets should wait for opportunities of coalescing and for how long. The simplest idea is to delay every request packet for a fixed time in each router on the path to the MC. However, since all the packets are delayed for a fixed time interval in the routers, the overall traffic pattern will remain the same. A better strategy is to distribute the allowable delays of a packet wisely among the routers along its path to the destination MC. Since packets that can be merged are all destined to the same MC, the routers closer to the MC should have a higher probability of seeing packets that can be merged. Therefore, we can allocate more delays to the routers closer to the MC. This is called the *dynamic slack distribution policy*.

Specifically, we employ a design parameter, called *Fixed Delay per Hop* and multiply it with the number of hops in the path from the requesting SM to the destination MC. This gives the total delay that the packet will experience along the path. The amount of delay in each router on the path is then calculated as shown below. The idea is to allocate one portion of the total delays on the first router, two portions on the second router, and so on.

$$(\textit{Fixed Delay per Hop} \times \textit{Total Hop Count}) \times \frac{\textit{Traversed Hop Count}}{\sum_{i=1}^{\textit{Total Hop Count}_i}} \quad (1)$$

Another issue is the head-of-line problem in the memory request queue that unavoidably arises when memory requests to the same rows are grouped together [10,12]. Modern DRAM chips are usually organized into banks, and memory requests to different banks can be serviced concurrently. These requests are

queued in the memory request queue in the MC and served in a FCFS fashion. Any scheduling scheme, including packet coalescing, that attempts to increase row buffer hits will necessarily group the memory requests to the same DRAM row together in the request queue. However, this would inversely block requests to other banks, reducing the bank level parallelism. Therefore, advanced memory scheduling schemes are often complemented with mechanisms such as *banked FIFO*, in which each bank has its own request queue.

3 Evaluation

We use GPGPU-Sim [5], a cycle-accurate many-core simulator, to evaluate the proposed packet coalescing mechanism. The microarchitecture parameters used in the evaluations are shown in Table 1. Eleven benchmark programs are used in our evaluation, including three from GPGPU-Sim. The benchmark programs are shown in Table 2. The proposed packet coalescing mechanism is compared with two MC-side buffer scheduling methods: FIFO and FR-FCFS [11,18], where FIFO is used as the baseline. The design parameter, *Fixed Delay per Hop*, is set to 5 cycles. Note that, being a NoC-side solution, packet coalescing can be used together with FIFO or FR-FCFS.

3.1 Row Buffer Miss Rate

Packet coalescing aims to reduce the row buffer miss rate by rearranging memory requests in the NoC. A lower row buffer miss rate implies faster memory accesses. Fig. 3 shows the performance in terms of row buffer miss rate.

From the figure, we can see that packet coalescing, when used together with FR-FCFS (denoted FR+dynamic), can reduce the row buffer miss rate by 78.75% over that of pure FIFO. When compared with FIFO and FR-FCFS, packet coalescing can further improve row buffer miss rate by 8.7% and 2.87% respectively.

Table 1. Microarchitecture parameters of GPGPU-Sim

Parameter	Value
Number of Shader Cores	28
Warp Size	32
Number of Threads/Core	1024
Number of Registers/Core	16384
NoC Topology / Routing	Mesh / Dimension-Order
NoC Virtual Channel	1
NoC Virtual Channel Buffer	8
NoC Flit Size	32
Memory Controller	8
DRAM Request Queue	32
Memory Controller Scheduling Scheme	FR-FCFS / FIFO / Banked-FIFO
GDDR3 Memory Timing	tCL=9, tRP=13, tRC=34, tRAS=21, tRCD=12, tRRD=8

Table 2. Benchmark programs

Benchmark	Label	Suite
AES Encryption	AES	[5]
Graph Algorithm: Breadth-First Search	BFS	Rodinia[6]
Coulombic Potential	CP	Parboil[2]
3D Laplace Solver	LPS	[5]
LIBOR Monte Carlo	LIB	3rd Party[5]
MUMmerGPU	MUM	3rd Party[5]
Neural Network	NN	[5]
N-Queens Solver	NQU	[17]
Ray Tracing	RAY	3rd Party[5]
Weather Prediction	WP	3rd Party[5]
BlackScholes Simulation	BlackScholes	Nvidia [1]

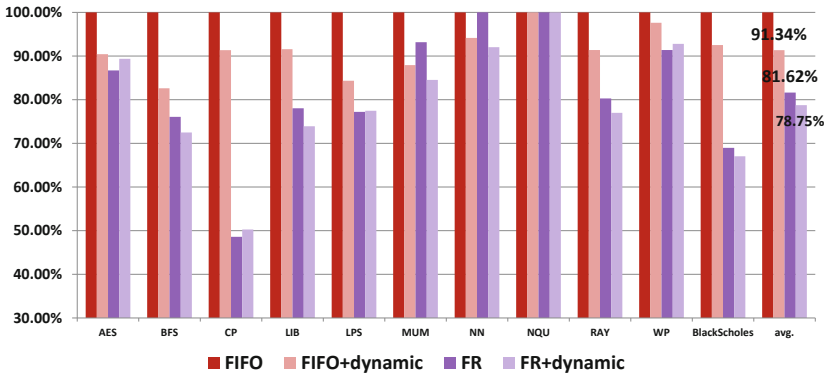


Fig. 3. Row buffer miss rate

The performance of individual benchmarks varies. The figure shows that benchmarks such as LIB, NN, and RAY can achieve a lower miss rate, because the injection patterns of these benchmarks are suitable for coalescing. However, WP does not perform well because the coalescing probability of WP is relatively low compared with other benchmarks. Benchmarks such as AES and NQU have a low packet injection rate to the NoC, and thus are hard to be improved.

The improvement in row buffer miss rate translates into improvement in DRAM accesses, as shown in Fig. 4. The figure shows a more than 25% reduction in the DRAM access time in average. This is because our design will delay packets in the NoC to rearrange their arrival sequence in the memory controllers. Thus, the average memory fetch time is higher than the baseline. However, if we deduct the average NoC traverse time from the average memory fetch time to get the average DRAM access time, our design does reduce the DRAM access time as shown in the figure.

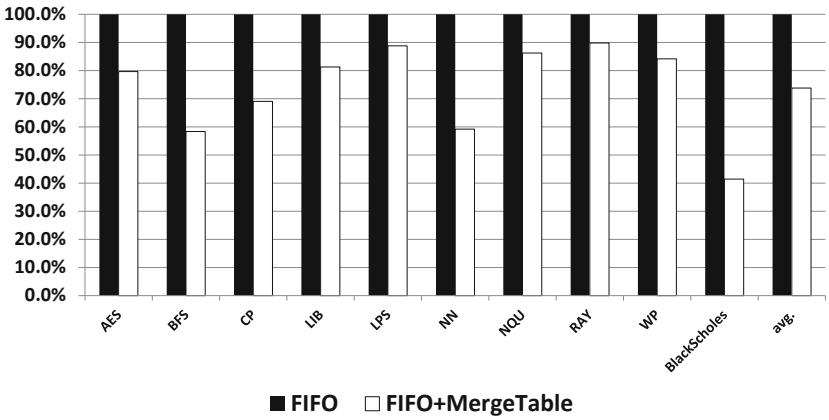


Fig. 4. Average DRAM access time

3.2 Instructions per Cycle

Next, we evaluate the overall performance of the system in terms of *instructions per cycle* (IPC) to see whether the improvement in DRAM accesses can translate into overall performance improvement. The results are shown in Fig. 5. Unfortunately, the IPC of the whole system does not improve much. Apparently, the performance gain in DRAM accesses is not enough to compensate the loss due to the delay in the NoC for packet coalescing. This can be verified in Fig. 5 with the configuration FIFO+extra5cycle, in which no packet coalescing is performed but every packet is delayed 5 cycles on each hop. In other words, this configuration will suffer from the delay in NoC but does not get any benefit from packet coalescing. From the figure, it can be seen that our approach did get some performance gain over FIFO+extra5cycle but not enough to compensate for the cost of delays in NoC.

3.3 Factors Affecting Performance

There are many factors affecting the overall performance of our design. These factors can be classified into two main categories. One category is related to application characteristics and the other is related to microarchitecture. These factors will be discussed below.

Application Characteristics. An important application characteristic is the number of *available warps* in the application. Packet coalescing is based on the assumption that applications have a sufficient amount of warps to hide the longer NoC latency due to coalescing. As shown in Fig. 6, most benchmarks only have 5 or few available warps in average. As the number of available warps is

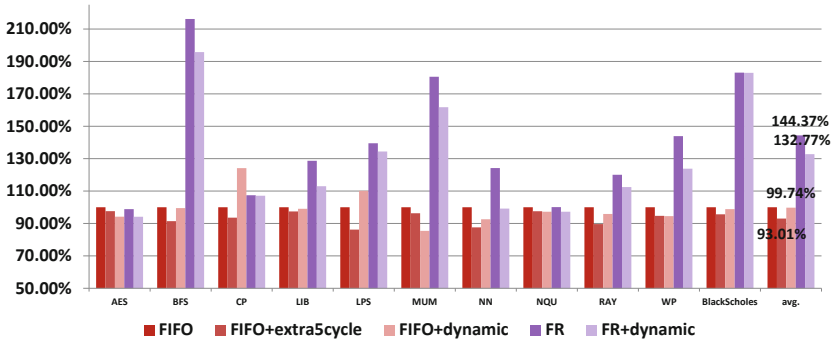


Fig. 5. Overall performance in terms of IPC

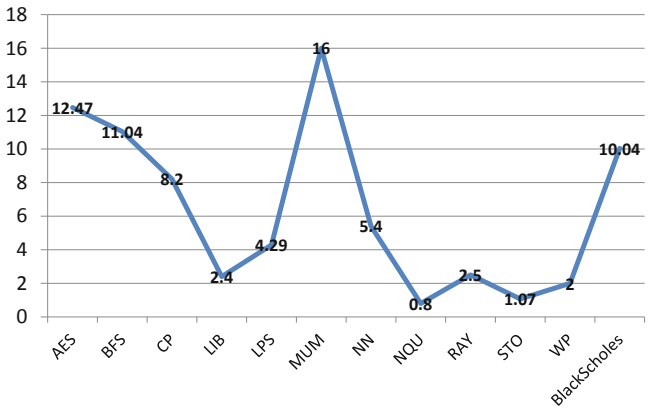


Fig. 6. Available warps of benchmark programs

proportional to the slack time [21], our design suffers from a lack of slack time to hide the overhead in the delay in NoC.

On the other hand, benchmarks such as *AES*, *BFS*, *MUM*, and *BlackScholes* have a relatively higher number of available warps. For these applications, we examine their *coalescing probability*, shown in Fig. 7. We can see that *MUM* only has a coalescing probability of 0.1 for a packet to merge with another. This probability is too low for effective packet coalescing. Furthermore, we found out that the benchmark programs doe not have enough number of memory requests to benefit from our design.

Router Microarchitecture. Modern DRAM chips are usually composed of many banks. *Bank level parallelism* thus plays an important role in DRAM accesses. Traditional memory controllers will queue the memory requests in the memory request queue and, if the requests are to access different memory banks, these requests can be serviced in parallel. As a result, the utilization of the banks

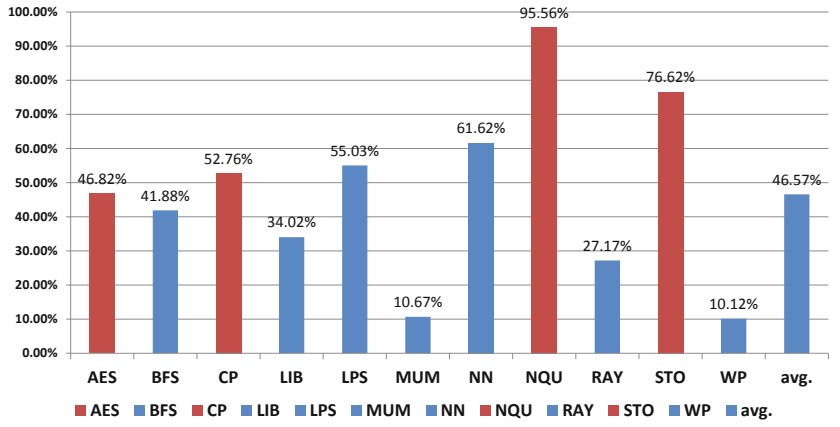


Fig. 7. Coalescing probability of benchmark programs

increases and the DRAM access latency is improved. Now, when we coalesce packets in the NoC, the memory requests going to the same memory bank will be grouped together, effectively serializing the accesses to the memory banks and resulting in the Head-of-Line problem.

A straightforward but effective solution is to distribute the memory request queue to each bank, or *banked FIFO*. Each bank has an independent request queue to buffer the memory requests to that bank. The overall performance of our design coupled with banked FIFO is shown Fig. 8. The figure shows that *BFS* and *BlackScholes* have better performance than the baseline architecture. *BFS* can improve the performance by about 2-3% and *BlackScholes* by 16-17%.

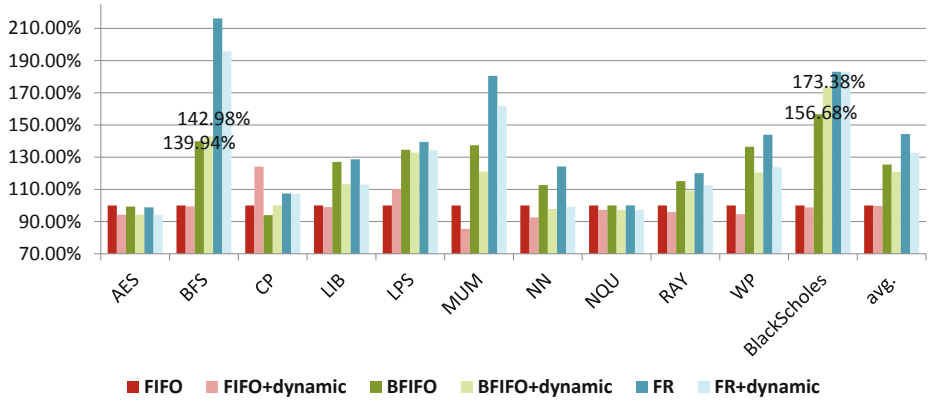
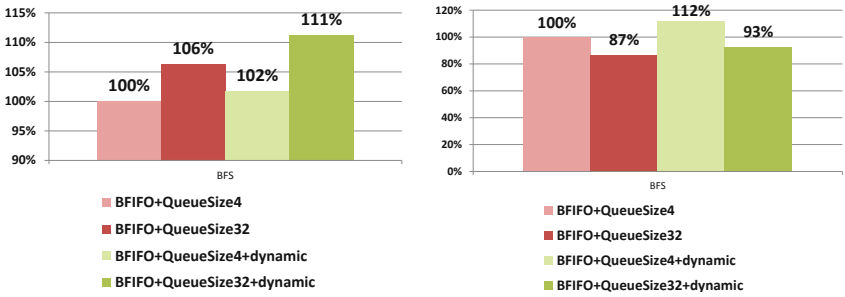


Fig. 8. Overall performance with banked FIFO

Even though banked FIFO can mitigate the Head-of-Line problem, the problem can still occur if the FIFO queue is too small. For example, suppose there are two packets heading to bank 2, followed by four packets heading to bank 0. Assume that the FIFO buffer in bank 0 can only hold two packets. Thus, the other two packets heading to bank 0 will be blocked in the interface queue between NoC and memory controller. These packets will further block other packets no matter which banks they are heading.

A simple solution is to enlarge the FIFO queue in each bank. As mentioned above, BFS is one of the benchmarks that suffer from the Head-of-Line problem. Thus, BFS is studied here and the results are shown in Fig. 9. The performance is evaluated using overall IPC and the metric *bank idle time*. From Fig. 9(a), we can see that the baseline architecture with the enlarged queues can improve the overall IPC by about 6%. Our design with enlarged queues can improve the overall IPC by about 9%. Obviously, there is an additional 3% improvement from eliminating the Head-of-Line problem.



(a) Overall IPC (b) Bank idle time

Fig. 9. Overall IPC and bank idle time with enlarged BFIFO queue

It is also interesting to note from Fig. 9(b) that the bank idle time drops 19% using our design with the banked FIFO, which is better than the baseline architecture with banked FIFO (13%). This further proves that the Head-of-Line problem does exist in configurations with small queues, and enlarging the queues of banked FIFO can eliminate the problem.

4 Related Works

To increase memory access efficiency, out-of-order scheduling such as FR-FCFS [12,13,18] has been studied extensively. Unfortunately, it requires a complex structure [3]. So far, there are very few works investigating memory scheduling in a massively parallel, many-core accelerators. Some works consider moving memory access scheduling out of the memory controller into the on-chip network and GPGPU shader core. For example, Yuan *et al.* [22] observed that memory requests sent from shader cores to DRAM will be disrupted by the NoC. Thus,

they proposed a NoC arbitration scheme called *Hold Grant* to preserve the row buffer access locality of memory request streams. In [10], the idea of *superpackets* is proposed for the shader core to maintain row buffer locality for the memory requests out of the core. While these works focus on maintaining the row buffer locality from a single shader core, our work exploit the coalescing opportunity across the cores inside the NoC. Our design leverages the many-to-few-to-many traffic pattern [4] in GPGPU to merge packets from different shader cores.

In [5], Bakhoda *et al.* showed that non-graphics applications tend to be more sensitive to bisection bandwidth than latency, also known as *bandwidth-sensitive and latency-insensitive*. The slacks of memory accesses are also studied in [9,21]. In [9], memory latency hiding resulting from critical paths is investigated in the traditional SMP systems. The concept of GPGPU packet slack is presented in [21] to detour packets for energy saving. Our design differs in that packet slack time is used to merge packets destining to the same row in DRAM to improve memory access performance.

5 Conclusions

In this paper, we propose a novel NoC design that merges memory requests from different GPGPU cores destining to the same row in the DRAM. This in essence offloads the scheduling of memory requests from the memory controllers to the interconnection network. As a result, our design promotes in-network processing rather than in-memory processing. A rudimentary router with coalescing logic is presented, and the design is evaluated with GPGPU-Sim. The evaluation results show that our preliminary design performs similarly as FR-FCFS on row buffer hit rate. However, it suffers from the delays in the NoC waiting for coalescing. As a result, the overall IPC performance cannot be improved much. Possible sources of inefficiencies are analyzed and discussed. More research is needed to optimize the current the design to achieve the best performance.

Acknowledgements. This work was supported in part by the National Science Council, Taiwan, under Grant 102-2220-E-007-025 and by the Industrial Technology Research Institute, Taiwan.

References

1. Nvidia gpu computing sdk suite, `https://developer.nvidia.com/gpu-computing-sdk`
2. Parboil benchmark suite, `http://impact.crhc.illinois.edu/parboil.php`
3. Ausavarungnirun, R., Chang, K., Subramanian, L., Loh, G., Mutlu, O.: Staged memory scheduling: Achieving high performance and scalability in heterogeneous systems. In: Proceedings of the 39th International Symposium on Computer Architecture, pp. 416–427. IEEE Press (2012)
4. Bakhoda, A., Kim, J., Aamodt, T.: 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)

5. Bakhoda, A., Yuan, G., Fung, W., Wong, H., Aamodt, T.: Analyzing cuda workloads using a detailed gpu simulator. In: Proceedings of IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), pp. 163–174. IEEE (2009)
6. Che, S., Boyer, M., Meng, J., Tarjan, D., Sheaffer, J., Lee, S., Skadron, K.: Rodinia: A benchmark suite for heterogeneous computing. In: Proceedings of IEEE International Symposium on Workload Characterization (IISWC), pp. 44–54. IEEE (2009)
7. Dally, W.: Virtual-channel flow control. IEEE Transactions on Parallel and Distributed Systems 3(2), 194–205 (1992)
8. Dally, W., Towles, B.: Principles and practices of interconnection networks. Morgan Kaufmann (2004)
9. Das, R., Mutlu, O., Moscibroda, T., Das, C.: Aérgia: A network-on-chip exploiting packet latency slack. IEEE Micro 31(1), 29–41 (2011)
10. Kim, Y., Lee, H., Kim, J.: An alternative memory access scheduling in many-core accelerators. In: 2011 International Conference on Parallel Architectures and Compilation Techniques (PACT), pp. 195–196. IEEE (2011)
11. Mutlu, O., Moscibroda, T.: Stall-time fair memory access scheduling for chip multiprocessors. In: Proceedings of the 40th IEEE/ACM International Symposium on Microarchitecture, pp. 146–160. IEEE Computer Society (2007)
12. Mutlu, O., Moscibroda, T.: Parallelism-aware batch scheduling: Enhancing both performance and fairness of shared dram systems. In: ACM SIGARCH Computer Architecture News, vol. 36, pp. 63–74. IEEE Computer Society (2008)
13. Nesbit, K., Aggarwal, N., Laudon, J., Smith, J.: Fair queuing memory systems. In: Proceedings of the 39th IEEE/ACM International Symposium on Microarchitecture, pp. 208–222. IEEE (2006)
14. Nickolls, J., Dally, W.: The gpu computing era. IEEE Micro 30(2), 56–69 (2010)
15. NVIDIA: Nvidia's next generation cuda compute architecture: Fermi (2009)
16. Owens, J., Houston, M., Luebke, D., Green, S., Stone, J., Phillips, J.: Gpu computing. Proceedings of the IEEE 96(5), 879–899 (2008)
17. Pcchen: N-queens solver, `http://forums.nvidia.com/index.php?showtopic=76893`
18. Rixner, S., Dally, W., Kapasi, U., Mattson, P., Owens, J.: Memory access scheduling. In: Proceedings of the 27th International Symposium on Computer Architecture, pp. 128–138. IEEE (2000)
19. Sanders, J., Kandrot, E.: CUDA by example: An introduction to general-purpose GPU programming. Addison-Wesley Professional (2010)
20. Stone, J., Gohara, D., Shi, G.: Opencl: A parallel programming standard for heterogeneous computing systems. Computing in Science and Engineering 12(3), 66 (2010)
21. Yin, J., Zhou, P., Holey, A., Sapatnekar, S., Zhai, A.: Energy-efficient non-minimal path on-chip interconnection network for heterogeneous systems. In: Proceedings of the 2012 ACM/IEEE International Symposium on Low Power Electronics and Design, pp. 57–62. ACM (2012)
22. Yuan, G., Bakhoda, A., Aamodt, T.: Complexity effective memory access scheduling for many-core accelerator architectures. In: Proceedings of the 42nd IEEE/ACM International Symposium on Microarchitecture, pp. 34–44. IEEE (2009)