Reducing DRAM Accesses through Pseudo-Channel Mode

Farzaneh Salehiminapour, Jan Lucas, Matthias Goebel, Ben Juurlink[1]

Abstract:

Applications once exclusive to high-performance computing are now common in systems ranging from mobile devices to clusters. They typically require large amounts of memory bandwidth. The graphic DRAM interface standards GDDR5X and GDDR6 are new DRAM technologies that promise to almost doubled data rates compared to GDDR5. However, these higher data rates require a longer burst length of 16 words. This would typically increase the memory access granularity. However, GDDR5X and GDDR6 support a feature called pseudo-channel mode. In pseudo-channel mode, the memory is split into two 16-bit pseudo channels. This split keeps the memory access granularity constant compared to GDDR5. However, the pseudo channels are not fully independent channels. Two accesses can be performed at the same time but access type, bank, and page must match, while column address can be selected separately for each pseudo channel. With this restriction, we argue that GDDR5X can best be seen as a GDDR5 memory that allows performing an additional request to the same page without extra cost. Therefore, we propose a DRAM buffer scheduling algorithm to make effective use of the pseudo-channel mode and the additional memory bandwidth offered by GDDR5X. Compared to the GDDR5X regular mode, our proposed algorithm achieves 12.5% to 18% memory access reduction on average in pseudo-channel mode.

Keywords: Memory Management; Memory Bandwidth; GDDR5X; GDDR6

1 Introduction

Over the last decade, due to the increasing number of memory-intensive applications, DRAM bandwidth is known as one of the major performance bottlenecks in modern multi-core systems. On one hand, new generations of DRAM technologies have been designed to improve memory performance. For example, the Double Data Rate (DDR) standard was introduced to increase memory bandwidth by transferring two data words per clock cycle. Besides, to achieve further bandwidth, new I/O features have been added to modern DRAM standards such as Graphic Double Data Rate 5 (GDDR5) and its modified version GDDR5X. In comparison to GDDR5, GDDR5X employs a new I/O feature which can increase the maximum data rate per pin by up to 14Gb/s/pin [Br18] compared to 7Gb/s/pin [JE16a] for GDDR5. However, as the speed of the internal memory array is not increased the memory burst size is increased to 16 words. With this burst size, 64 bytes are transferred in each

[1] Technische Universität Berlin, Architektur eingebetteter Systeme(AES), Einsteinufer 17, 10587 Berlin, Germany
salehiminapour,j.lucas,m.goebel,b.juurlink@tu-berlin.de

read or write transaction, if using a regular 32-bit wide interface. To reduce the issue of this large memory access granularity, GDDR5X features a pseudo-channel mode which enables the memory controller to fetch two simultaneous memory access with 32-bytes access granularity from the two different pseudo-channels, provided these accesses have the same access type, rank, bank and page [JE16b]. Pseudo-Channel Mode is not limited to GDDR5X, GDDR6 [JE17] is also equipped with the same feature and also uses the pseudo-channel mode to improve the access granularity without the extra signaling lines required for a true dual channel mode.

On the other hand, memory scheduling algorithms play a significant role in modern memory controllers to improve system performance. Prior works [BI12; Ki10; Su14] proposed different memory scheduling algorithms that comply with various applications and characteristics of specific systems. However, most existing memory controllers employ the commonly used *First Ready First Come First Served scheduler (FR-FCFS)* [Ri00] to shorten the overall latency of DRAM accesses. FR-FCFS was developed to prioritize page hits and address the shortcomings of the *First Come First Served (FCFS)* algorithm [Ri00].

We observe that the GDDR5X pseudo-channel mode provides additional bandwidth that can be utilized to design an effective memory scheduler for improving the system performance. Moreover, GPU DRAM, such as the GDDR family, always can be a good predictor for features that will appear in CPU DRAM in the future. This study, therefore, aims to introduce a novel memory scheduling algorithm to maximize the benefit of using GDDR5X pseudo-channel mode. To this end, we modified gem5 [Bi11] to support GDDR5X as a CPU DRAM and evaluated the performance of our proposed scheduling algorithm using trace-driven simulation.

To the best of our knowledge, this is the first work that investigates the GDDR5X pseudo-channel feature to gain further bandwidth improvement.

The contributions of this paper are as follows:

1. We propose a new buffer scheduling policy called *pseudo-channel-aware scheduling* algorithm to exploit further bandwidth of GDDR5X
2. We evaluate the pseudo-channel-aware scheduling algorithm on a wide variety of workloads and compare its performance to a baseline system. On average, the pseudo-channel-aware scheduling algorithm improves the performance by 38% for an eight-core system across different workloads[SP06].

This paper is organized as follows. Related work is discussed in Section 2. In Section 3 the details of our proposed method is presented. Section 4 discusses the experimental setup. Following that, experimental results are given in Sections 5. Finally, Section 6 presents conclusions and future work.

2 Related Work

DRAM buffer scheduling algorithms have been discussed in several prior works ranging from application-aware to QoS-aware memory schedulers. Rixner et al. [Ri00] proposed the First Ready First Come First Service (FR-FCFS) algorithm. This scheduler picks the first ready access, which is an access from an already open page for the case of the open-page policy. Next, if no access is found, the oldest access from the DRAM read/write buffer is picked. Kim et al. [Ki10] proposed a QoS-aware memory scheduler called *ATLAS* to control the number of services allocated for each application at the memory controllers and to prioritize the requests of applications with respect to the minimum received memory service. Subramanian et al. [Su14] developed an application-aware memory scheduler called *Blacklisting Memory Scheduler*. This scheduler categorizes application requests into two vulnerable-to-interface and interface-causing sets, to improve the performance and fairness with a lower cost.

Although a considerable number of works [BI12; Ki10; Su14] have proposed advanced DRAM scheduling algorithms, many memory controllers utilize the FR-FCFS scheduling algorithm since it presents a good performance [Ha14] compared to other more complex algorithms.

Our literature review revealed none of the existing memory schedulers investigated the GDDR5X pseudo-channel mode to design a pseudo-channel-aware DRAM scheduling algorithm with the goal of improving performance. This study presents the design of such a scheduler to exploit further bandwidth from the pseudo-channel mode.

3 Problem Analysis

In this section, first, we present the GDDR5X standard and explain its most important features. Second, we describe our proposed DRAM scheduling algorithm.

3.1 GDDR5X

GDDR5X SGRAM (Synchronous Graphics Random Access Memory), which is an extension of the well-established GDDR5 standard, employs a new I/O feature to achieve a higher bandwidth compared to GDDR5.

In addition to the regular GDDR5 Dual Data Rate (DDR) mode, a Quad Data Rate (QDR) is also supported and transfers four data words per clock cycle and can be enabled via a mode register bit. The DDR mode operates similar to a GDDR5-SGRAM, and provides a burst size of 8 words. In contrast, the QDR mode transfers a longer burst size of 16 words, which increases the memory access granularity from 32 bytes (in DDR) to 64 bytes. Therefore, GDDR5X-QDR doubles the burst size and memory access granularity respectively.

Using QDR GDDR5X is able to provide additional bandwidth compared to GDDR5. The standard enables 10-14Gb/s/pin [JE16b] data rate which is approximately a 2x increase over its predecessor.

The increase in the access granularity from 32 bytes to 64 bytes in GDDR5X may fetch unnecessary data to the main memory, which can reduce effective memory bandwidth. Therefore, GDDR5X supports a feature called pseudo-channel mode in which the memory is split into two 16-bit pseudo channels. Considering the burst length of 16, in pseudo-channel mode, GDDR5X has the same access granularity as GDDR5. However, these two pseudo channels are not fully independent, which means that while this feature allows issuing two simultaneous memory requests with 32-bytes access granularity, access type, bank, and page needs to match. However, different columns can be selected for each of the two pseudo channels. According to [JE16b], GDDR5X utilizes six column address bits per pseudo channel. Therefore, in pseudo-channel mode, 64 different addresses can potentially be paired for each request. Issuing two simultaneous memory requests in Pseudo-channel mode has no performance penalty. We can thus also think of GDDR5X in pseudo channel mode as a GDDR5, which can issue one additional request for free, if the requirements mentioned above, are met.

3.2 Proposed Memory Scheduler

DRAM buffer scheduling algorithms reorder memory requests to improve memory performance. For example, commonly used FR-FCFS prioritizes (1) the page-hit requests and (2) older requests over the younger ones to boost memory throughput. Given the pseudo-channel feature, it is beneficial to make the memory scheduler aware of this feature in order to maximize DRAM performance.

As remarked in Subsection 3.1, in the case of GDDR5X pseudo-channel mode, the memory controller can fetch an additional request simultaneously without any extra cost if two requests have the same type, bank, page, but different pseudo channels; any two requests which meet these criteria will be referred to as pairable requests in this work.

Tab. 1: Request prioritization in our proposed algorithm

1. **Pairable Requests (open page):** Pairable requests from an already open page are prioritized over all other requests
2. **Pairable Requests (non-yet-open page):** Pairable requests from a non-yet-open page are prioritized
3. **Non-pairable Request (open page):** One open-page-hit request is prioritized
4. **FCFS:** Older request is prioritized over younger ones

Hence, utilizing GDDR5X as a CPU DRAM, we propose an extended FR-FCFS algorithm called pseudo-channel-aware scheduling, to exploit additional bandwidth provided by

the GDDR5X pseudo-channel mode. Table 1 summarizes the four steps of the proposed algorithm.

When enabling pseudo-channel-aware scheduling algorithm, first, the memory scheduler proceeds to select all pairable requests either from the read or write buffer queue. Since opening and closing a DRAM page takes a significant amount of time and energy [Ha14] the memory scheduler first picks all available pairable requests from currently open DRAM pages. In the second step, the pseudo-channel-aware scheduler targets all available pairable requests from DRAM non-yet-open pages. Next, if no more pairable requests have been found, the memory scheduler picks the first available unpaired request from an open DRAM page (step 3). Finally, if no page-hit request is available, the pseudo-channel-aware scheduler operates the same as the FCFS algorithm [Ri00] and targets the oldest request in the queue.

4 Experimental Setup

In this study, in order to quickly assess different policies, we choose a trace-driven approach to evaluate our algorithm. Figure 1 illustrates a modern DRAM controller architecture, with split read and write queues, a memory scheduler, and a response queue. The memory scheduler is responsible for executing memory scheduling policies. We generated memory traces using gem5 and implemented the functionality of a DRAM controller in a custom trace-based simulator, written in C++. This simulator was used to evaluated the effectiveness of our proposed memory scheduling algorithm.

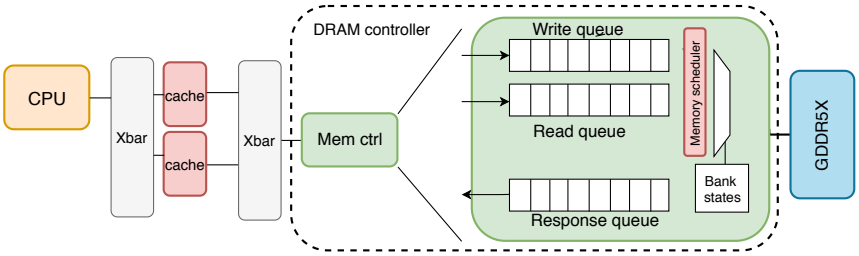

Fig. 1: Overview of DRAM controller architecture

The memory traces were generated with the gem5 simulator using a simulated multi-core system with eight out-of-order cores and a 512 KB shared last level L2 cache. Furthermore, the gem5 simulator was extended in terms of clocking, data rate, and timing parameters [JE16b] to support GDDR5X and its requirements and features. To support the pseudo-channel mode, the address mapping unit in the memory controller was modified to assign a bit to the pseudo channel. For address mapping, we used RoRaBaChPcbCo with Ro, Ra, Ba, Ch, Pcb and Co representing row, rank, bank, channel, pseudo-channel bit and column, respectively, going from MSB to LSB. Table 2 lists the details of all system parameters used in the gem5 simulation.

Tab. 2: Parameters of the gem5 simulation

Processor	Eight-core, 4.0 GHz, Out-of-Order
L1 D/I Cache	32 KB, 2-way set associative
LLC	512 KB, 8-way set associative
Cache Line Size	32 B
Prefetcher at LLC	Stride prefetcher with degree=4
DRAM Controller	On-chip, Open row policy
Address Mapping	RoRaBaChPcbCo
Number of Instructions (sampling)	One billion instructions

In our custom simulator, we utilized the presented pseudo-channel-aware scheduling algorithm to efficiently pick requests from either the read or write queue. As this is a trace-based study, we do not consider read and write dependencies. Therefore, the queues are assumed to be always filled with requests. The simulation was performed with four different queue sizes of 256, 512, 1024, and 2048 entries in both pseudo channel and regular mode. We used workloads from SPEC2006 CPU [SP06]. In particular, we have chosen 14 benchmarks from different categories to evaluate the effectiveness of our proposed memory scheduler. Each benchmark was compiled using GCC and GNU Fortran. All benchmarks are run using the SPEC reference input set.

In order to keep simulation time and trace file size manageable, we used custom gem5 scripts with sampling. As shown in Figure 2, the trace collection function applies sampling to capture all the behavior of the benchmarks and accurately collect DRAM memory accesses.

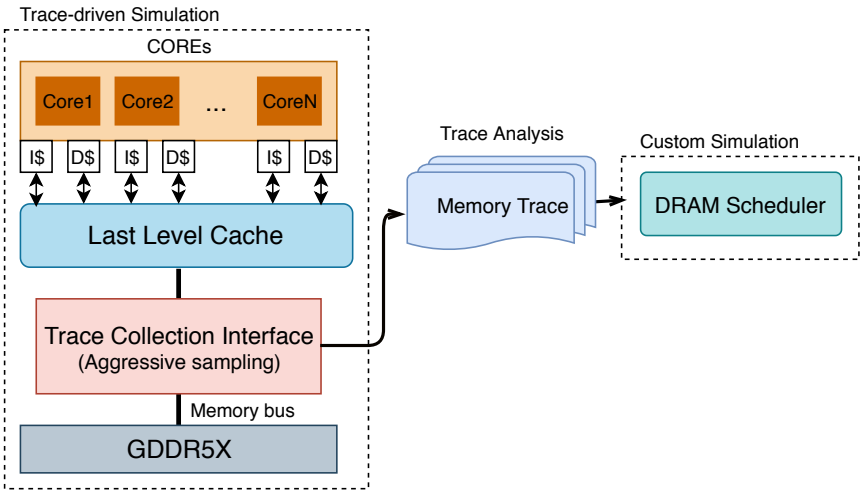

Fig. 2: Overview of our framework

The idea behind sampling is to stay longer in the gem5 functional simulation than in performance simulation and gain significant speedup. Therefore, for each workload, the gem5 CPU was switched back and forth between a O3CPU which is a model of a real out-of-order CPU and a AtomicSimpleCPU [Bi11] which is a model of a purely functional in-order CPU. All simulations were executed using the first one billion instructions.

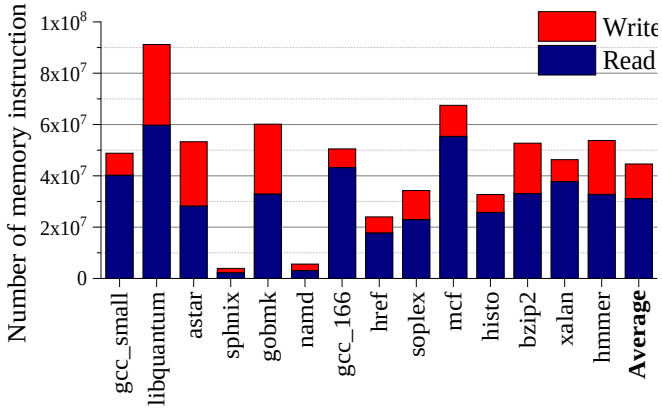

Fig. 3: Memory read/write accesses

5 Experimental Results

As explained in Section 3.1 GDDR5X can operate in two modes: regular and pseudo-channel. Each mode has a different memory access granularity. In the regular mode, GDDR5X uses a large burst size of 64 bytes while in pseudo-channel mode the granularity remains the same as GDDR5 with 32 bytes. Regular mode is the simplest way to use all the bandwidth offered by GDDR5X and was thus chosen as a baseline. However, due to the large access granularity, regular mode will often perform unnecessary reads or writes. As discussed in Section 3.1, pseudo channel mode is more flexible. While both regular and pseudo-channel mode fetch 64 bytes from the same memory page, regular mode basically glues together two 32 byte halves, while in pseudo channel mode, each channel can fetch any 32 byte burst from its half of the memory page. This means each burst on one side of the page can be paired with 64 different bursts from the other half of the page. This additional flexibility of the pseudo-channel mode should increase the effective bandwidth of the DRAM interface and allow us to avoid unnecessary reads or writes.

Figure 3 shows the total number of read/write operations in all benchmarks we studied. The results are extracted from gem5 while running one billion instructions using sampling. As illustrated in Figure 3, the average number of memory accesses in our simulations is approximately five percent of the total number of instructions.

The ratio of total paired requests are presented in Figure 4 and Figure 5 for pseudo-channel and regular modes with limited queue sizes of 256, 512, 1024, and 2048 entries. Moreover, we executed the algorithm with an unlimited queue size for read/write queues to an upper bound for the ratio of paired requests, and thus estimate the maximum achievable improvement in our proposed algorithm. According to Figure 4 and Figure 5, our proposed algorithm yields average upper bounds of 96 percent and 87 percent for the ratio of paired requests in pseudo-channel and regular modes, respectively. Given the limited queue sizes of 256 to 2048, our proposed algorithm on average yields 43 percent (pseudo-channel mode) and 34 percent (regular mode) increase in the number of paired requests. The results show that the proposed algorithm is able to achieve more paired requests in the pseudo-channel mode due its flexibility in selecting paired requests (recall Section 3.1).

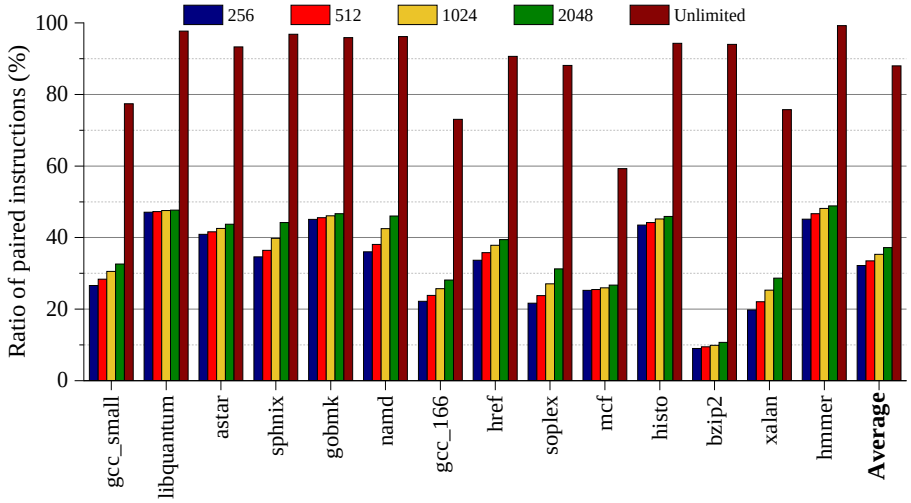

Fig. 4: Number of paired requests over the total number of memory requests in regular mode (baseline)

In order to have a better comparison, the ratio of paired requests is normalized to that of the regular mode of GDDR5X with the smallest queue size of 256 entries. Figure 6 indicates the normalized ratio of total paired requests in pseudo-channel and regular modes across all workloads with different queue sizes. As shown in Figure 6, our algorithm for all four queue sizes in pseudo channel mode can achieve a higher ratio compared to regular mode. In the pseudo-channel mode, the average value of the normalized ratio across different workloads ranges between 1.41 (for the smallest queue size) to 1.64 (for the largest queue size), whereas its average value is limited to less than 1.2 (for the largest queue size) in the regular mode.

Figure 7 shows the percent reduction in total memory accesses obtained in pseudo-channel and regular modes. Similar to Figure 6, the results are normalized to the regular mode of GDDR5X with 256 entries (the smallest queue size). In pseudo-channel mode, Figure 7

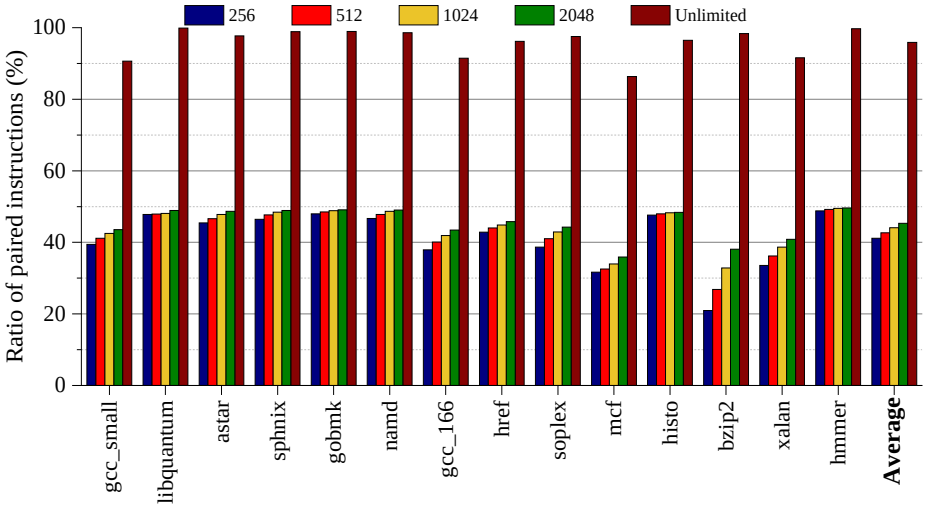

Fig. 5: Number of paired requests over the total number of memory requests in pseudo-channel mode

indicates that pseudo-channel-aware scheduling algorithm on average gives 12.5 percent and 18 percent reductions in number of memory requests for the smallest and largest queue sizes, respectively, for pseudo-channel mode. In contrast, the average reduction in regular mode is bounded to 7.2 percent (in the largest queue size).

As a result, we find that even with the smallest queue size, the proposed algorithm in pseudo-channel mode outperforms the largest queue size in regular mode. This means that our algorithm can achieve higher memory access reduction compared to the baseline across all workloads we studied.

6 Conclusions and Future Work

In this paper, we have presented a novel memory scheduling algorithm called *pseudo-channel-aware scheduling* to exploit the GDDR5X pseudo-channel mode and thus improve memory system performance. The proposed algorithm prioritizes memory requests in read/write queues to select pairable requests with the same accesses type from the same bank and page but different pseudo channels. We have evaluated the proposed scheduling algorithm using 14 various benchmarks from SPEC2006. Furthermore, we performed all simulations with four different read/write queue sizes of 256, 512, 1024, and 2048 entries, with the results been compared to GDDR5X regular mode as a baseline. The results reveal that the proposed algorithm is able to achieve a reduction of 12.5% and 18% on average for the smallest and largest queue sizes of 256 and 2048 entries, respectively. Furthermore,

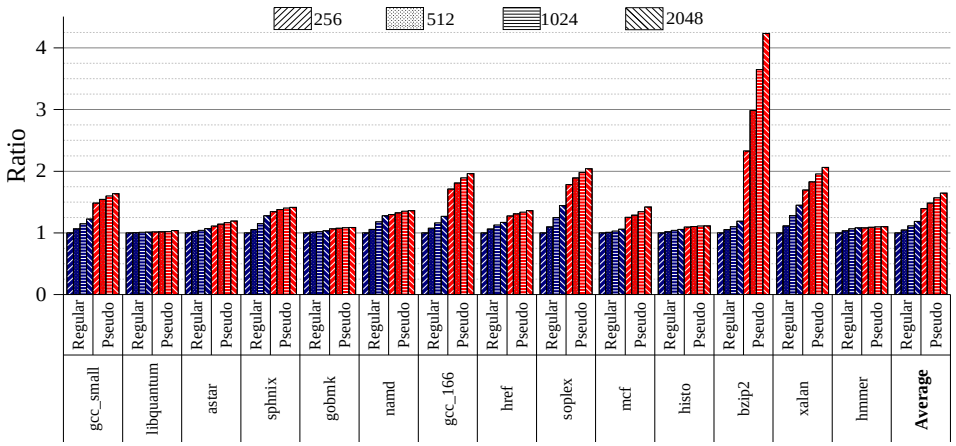

Fig. 6: Ratio of total paired requests normalized to the baseline with a queue size of 256 entries

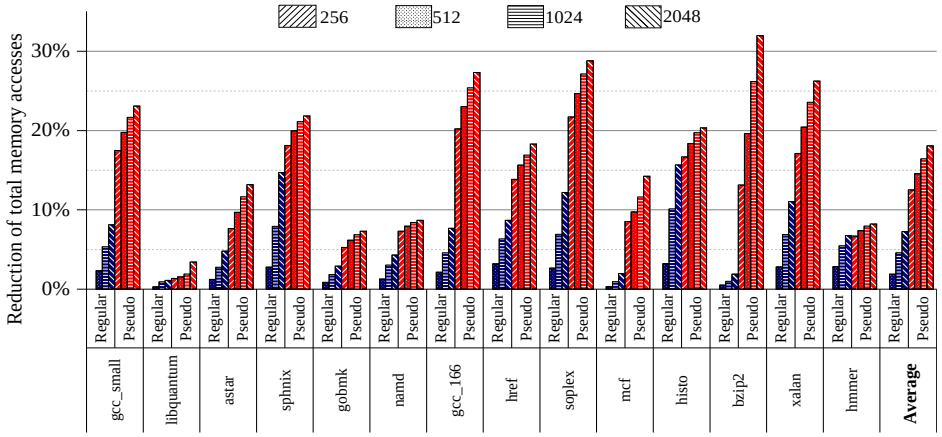

Fig. 7: Percentage of total memory access reduction compared to the baseline with a queue size of 256 entries

our algorithm can achieve an upper bounds of 96% for the ratio of paired requests in pseudo-channel mode, thus demonstrating its potential.

In the future, we aim to continue this study in several directions. As noted in Section 4, this is a trace-based study and read/write dependencies are not reflected in the traces. Therefore, we plan to implement our proposed scheduling policy on gem5 to measure the performance on a real system. In addition, we intend to investigate this study in terms of energy consumption; hence, the effect of the proposed algorithm on energy consumption will be analyzed later. Future work also includes further evaluation with other benchmark suites than SPEC2006 and real-world applications.

References

[Bi11] Binkert, N.; Beckmann, B.; Black, G.; Reinhardt, S. K.; Saidi, A.; Basu, A.; Hestness, J.; Hower, D. R.; Krishna, T.; Sardashti, S., et al.: The Gem5 Simulator. ACM SIGARCH Computer Architecture News 39/02, pp. 1–7, 2011.

[BI12] Bojnordi, M. N.; Ipek, E.: PARDIS: A Programmable Memory Controller for the DDRx Interfacing Standards. ACM SIGARCH Computer Architecture News 40/03, pp. 13–24, 2012.

[Br18] Brox, M.; Balakrishnan, M.; Broschwitz, M.; Chetreanu, C.; Dietrich, S.; Funfrock, F.; Gonzalez, M. A.; Hein, T.; Huber, E.; Lauber, D., et al.: An 8-Gb 12-Gb/s/pin GDDR5X DRAM for Cost-effective High-performance Applications. IEEE Journal of Solid-State Circuits 53/1, pp. 134–143, 2018.

[Ha14] Hansson, A.; Agarwal, N.; Kolli, A.; Wenisch, T.; Udipi, A. N.: Simulating DRAM Controllers for Future System Architecture Exploration. In: IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)., pp. 201–210, 2014.

[JE16a] JEDEC Standard: Graphics Double Data Rate (GDDR5) SGRAM Standard. JESD212C 3/08, 2016.

[JE16b] JEDEC Standard: Graphics Double Data Rate (GDDR5X) SGRAM Standard. JESD232A 5/15, 2016.

[JE17] JEDEC Standard: Graphics Double Data Rate (GDDR6) SGRAM Standard. JESD250 11/17, 2017.

[Ki10] Kim, Y.; Han, D.; Mutlu, O.; Harchol-Balter, M.: ATLAS: A Scalable and High-performance Scheduling Algorithm for Multiple Memory Controllers. In: IEEE 16th International Symposium on High Performance Computer Architecture (HPCA)., pp. 1–12, 2010.

[Ri00] Rixner, S.; Dally, W. J.; Kapasi, U. J.; Mattson, P.; Owens, J. D.: Memory Access Scheduling. In: ACM SIGARCH Computer Architecture News. Vol. 28. 2, pp. 128–138, 2000.

[SP06] SPEC[Online], 2006, URL: https://www.spec.org/.

[Su14] Subramanian, L.; Lee, D.; Seshadri, V.; Rastogi, H.; Mutlu, O.: The Blacklisting Memory Scheduler: Achieving high Performance and Fairness at Low Cost. In: IEEE International Conference on Computer Design (ICCD)., pp. 8–15, 2014.