MEMORY ACCESS SCHEDULING SCHEMES

Group 1: Huihan Hu, Peter Solomon, Matt Ziminski, Xiao Yang

INTRODUCTION:

Memory access scheduling is an important strategy that can help improve memory system performance and energy efficiency by reordering concurrent memory requests. Many strategies have been proposed for memory access scheduling in recent years. The USIMM simulator application is known as a platform which allows for the benchmarking of memory access scheduling policies. This application allows for the evaluation of performance and power consumption of devices while testing newly employed schemes. For the purposes of this paper, the performance of three different memory access scheduling schemes will be tested using the USIMM application.

SCHEMES:

1. FR-FCFS SCHEDULER

In First-Ready – First Come, First Serve (FR-FCFS) scheduling, the scheduler selects the process that has been waiting the longest in the ready queue and assigns the CPU to it. If a new process arrives while another process is running, the new process is added to the end of the ready queue. When the current process completes or enters a waiting state, the scheduler selects the next process in the ready queue. The FR-FCFS algorithm is a simple algorithm to implement and has low overhead, making it a good choice for systems with a small number of processes. However, it can lead to poor response times if long-running processes are scheduled before short-running ones. FR-FCFS is not often used in modern operating systems since there are other scheduling algorithms that offer much better performance. However, it is often used to compare with other scheduling schemes such as PAR-BS and SMS.

2. PARALLELISM-AWARE BATCH SCHEDULING (PAR-BS):

The PAR-BS system is comprised of two main components. The first one is called the request batching or simply batching component. Its role is to group several DRAM requests into a batch and make sure that all requests within the current batch are serviced before moving on to the next batch. The benefits of batching include providing a convenient granularity for optimizing DRAM command scheduling, as well as ensuring fairness. The second component, called parallelism-aware within-batch scheduling (PAR), aims to reduce thread stall time and increase Chip Multiprocessor (CMP) throughput by attempting to service each thread's requests in parallel in DRAM banks. PAR-BS was implemented by first initializing variables and defining two important parameters, HI_WM and LO_WM, to control write queue draining. It also introduces a BATCH_INTERVAL macro to determine the batch duration. After that, it calculates the batch number for a given request based on its arrival time. The scheduler calculates the current read and write batch numbers using the current cycle value and the write queue length. It then iterates through the read and write queues, checking if the command is issuable and

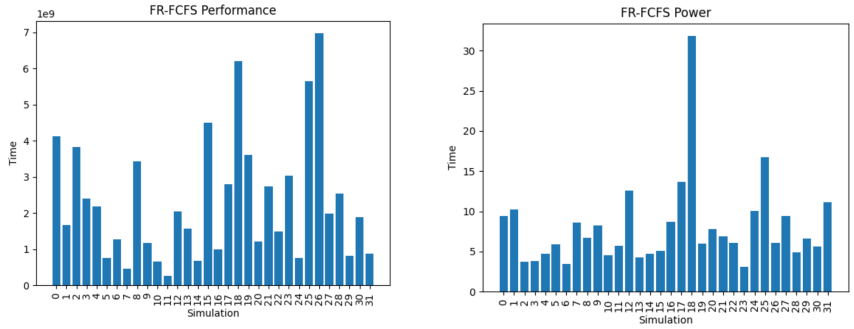
comparing the batch difference between the current batch and the request's batch. The scheduler prioritizes requests with higher batch differences.

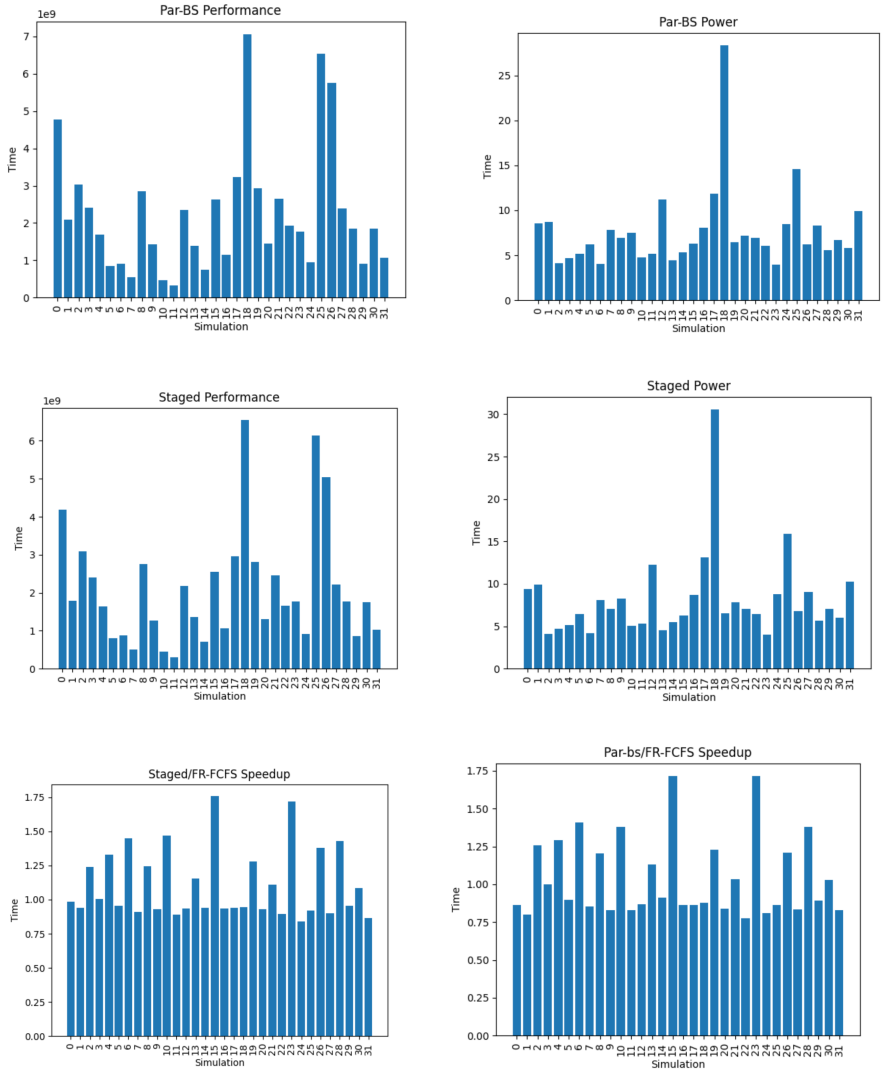
3.STAGED MEMORY SCHEDULING (SMS):

Staged Memory Scheduling (SMS) is a general approach to memory scheduling that aims to improve system performance, fairness, and efficiency in systems where multiple processing units, such as CPUs and GPUs, share the same off-chip main memory. SMS is particularly useful for addressing the challenges posed by the large amount of GPU traffic and the interference between CPU and GPU requests in integrated CPU-GPU systems. The main idea behind staged memory scheduling is to decouple the memory controller's primary tasks into multiple simpler stages, each focusing on a specific aspect of the memory scheduling process. This decoupling simplifies the overall memory scheduling process, reduces complexity, and allows for more efficient memory access and scheduling.
In the reference paper, the authors propose a three stage memory scheduling algorithm that first groups the requests based on the row buffer locality, then schedules the inter-application request and then just uses First In First Out (FIFO) to deal with the low-level DRAM. Due to the time constraint, the full staged memory scheduling was not able to be implemented. Instead, the three stages were simplified into two stages. The scheme first checks if the system is in write-drain mode, which is when the memory controller prioritizes servicing write requests over read requests. If it is, the scheduler will then process the write request. If not, then read requests are separated into a simple two stage model: the first stage identifies the old request (older than AGE_THRESHOLD) and young requests (younger than AGE_THRESHOLD) from the read queue. The second stage prioritizes issuing commands for old requests. If there are no old requests, it issues commands for the oldest young request.

RESULTS:

The benchmark results of the FR-FCFS scheme was used as the reference benchmark and the PAR-BS and SMS schemes were evaluated based on this reference. Here are the visualization results of the USIMM benchmarks:





After the benchmarking, the PAR-BS algorithm implementation resulted in an average performance improvement of 3.9% in terms of the overall execution time.
The staged implementation on the other hand, resulted in an average performance improvement of 10.2% in terms of overall execution time.

FUTURE WORK:

A basic implementation of the actual PAR-BS and SMS algorithm was made for the purposes of this project. In the future, the full implementation of these two algorithms is to be further developed in order to ideally maximize overall system performance. Additionally, because of the time constraints present during the course of this project, two schemes had to be skipped; Fair Dynamic Piping Memory Scheduling (FDPM) and Latency-Aware Memory Scheduling (LAMS). For FDPM the algorithm could not be sufficiently implemented in time and so more research will be required in the future on this topic. The group lost confidence in exploring its implementation within the USIMM simulation tool, because of a

lack of familiarity with the applications capabilities. Therefore a different approach was selected (SMS). For the LAMS scheme, the algorithm seemed straightforward to implement initially (due to having the pseudo code provided), but attempting to integrate it into the USIMM codebase proved to be problematic. Because of this, the team agreed to move on from this approach as well.

REFERENCES PAPERS:

1. Parallelism aware batch scheduling: enhancing both performance and fairness of the shared DRAM system", Onur Mutlu, et. al, ISCA 2008. https://people.inf.ethz.ch/omutlu/pub/parbs_isca08.pdf
2. Rachata Ausavarungnirun, Kevin Kai-Wei Chang, Lavanya Subramanian, Gabriel H. Loh, and Onur Mutlu. 2012. Staged memory scheduling: achieving high performance and scalability in heterogeneous systems. SIGARCH Comput. Archit. News 40, 3 (June 2012), 416–427. https://doi.org/10.1145/2366231.2337207
3. Scott Rixner, William J. Dally, Ujval J. Kapasi, Peter Mattson, and John D. Owens. 2000. Memory access scheduling. In Proceedings of the 27th annual international symposium on Computer architecture (ISCA '00). Association for Computing Machinery, New York, NY, USA, 128–138. https://doi.org/10.1145/339647.339668
4. "LAMS: A Latency-Aware Memory Scheduling Policy for Modern DRAM Systems" https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7820660

INDIVIDUAL CONTRIBUTIONS:

Huihan Hu: Implementation of PAR-BS and SMS with corresponding section report.
Xiao Yang: Understanding of the Par-BS algorithms and possible variables. Run all the 3 simulations. Gather results from all 3 schemes and visualization. Wrote Python script to generate benchmark evaluations.
Peter Solomon: Attempted to implement PAR-BS in USIMM using WSL (would not run due to newer version of linux and missing files). Contributed to the writing and cohesiveness of the group paper.
Matt Ziminski: Attempted to implement the Fair Dynamic Memory Scheduling and Latency Memory Aware Scheduling scheme policies, but was unsuccessful in doing so. Attempted running different schemes for final results, but ran into configuration/unzipping issues that prevented useful contribution to speed up these stages.

SOURCE CODE:

Scheduler-parbs.c, scheduler-parbs.h, scheduler-staged.c, scheduler-staged.h are included in the submission.