ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Website: www.ijircce.com

Vol. 6, Issue 4, April 2018

A Memory Scheduling Algorithm for Multi-core Systems

Hyeon-Ju Yoon[1]

Associate Professor, Department of Computer Engineering, Kumoh National Institute of Technology, Gumi-City, Republic of Korea[1]

ABSTRACT: Multi-core processors share the same DRAM memory, and several threads issues memory access requests simultaneously. The limited bandwidth of shared memory will be a bottleneck of overall system performance as the number of cores increases. Conventional DRAM memory controller designs do not consider the interference among different threads when making scheduling decisions. To maximize the memory bandwidth utilization, First-Ready First-Come-First-Serve(FR-FCFS) policy was proposed. It utilizes the memory access locality up to maximum, but it may not fit to the multi-threaded memory request, because some threads may fall in danger of starvation.

In this paper, we propose an improvement for FR-FCFS algorithm introducing the concept of memory access service rate (MASR). For the practical reason, MASR is implemented with the limited waiting time. From the simulation with well-knows benchmarks, we showed our scheme can improve both the fairness and overall system performance.

KEYWORDS: Multi-core; memory scheduling; fairness; overall system performance

I. INTRODUCTION

Multi-core processors enable multiple threads to run simultaneously on a single chip with a shared main memory. A DRAM memory subsystem is usually used as the main memory to support multiple cores and multiple threads on them. Different threads running on different cores send memory access requests simultaneously and may interfere with each other while accessing the shared resources. The limited bandwidth of shared memory will be a bottleneck of overall system performance as the number of cores increases.

Unfortunately, conventional DRAM memory controller designs do not consider the interference among different threads when making scheduling decisions. To maximize the memory bandwidth utilization, First-Ready First-Come-First-Serve(FR-FCFS) policy was proposed[1]. FR-FCFS prioritizes memory requests that hit in the row-buffers of DRAM banks over other requests. It utilizes the memory access locality up to maximum, but it may not fit to the multi-threaded memory requests. A poor thread might be starved without getting an opportunity to access memory.

In this paper, we propose an improvement for FR-FCFS algorithm introducing the concept of memory access service rate (MASR). For the practical reason, MASR is implemented with the limited waiting time. From the simulation with well-knows benchmarks, we showed our scheme can improve both the fairness and overall system performance.

The remainder of the paper is organized as follows. Section 2introduces DRAM architecture, necessity of memory scheduling and some previous effort to make the memory scheduling feasible for multi-core systems. Our new algorithm is proposed in Section 3, and section 4 shows the simulation result. Finally, some conclusions are given in Section 5.

II. RELATED WORK

A DRAM chip has a 3-dimensional architecture with bank, row and column[1] as you can see in Figure 1. To access data in memory, the address that consists of the bank, row and column fields should be identified. Each bank has a single row-buffer, which can contain at most one row and the data in a bank can be accessed only from the row-buffer. So an access to the memory requires three transactions before the data transfer – bank pre-charge, row activate and column access. A pre-charge charges and prepares the bank. An activate copies an entire row data from the array to the

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Website: www.ijircce.com

Vol. 6, Issue 4, April 2018

row-buffer. Finally, a column data can be accessed from the row data. Depending upon the status of the row-buffer, the amount of time it takes to service a DRAM request is not uniform. It can be categorized into 3 cases.

- Row hit: The request is accessing the row currently in the row buffer. Only a read or write command is needed, so the lowest bank access latency is resulted from the only a column access.
- Row closed: There is no row in the row buffer. An activate command needs to be issued to open the row followed by a read or write command. The bank access latency is a row access plus a column access time.
- Row conflict: The requested row differs from the one currently in the row buffer. The contents of the row buffer first need to be written back into the memory array using the pre-charge command. Then the required row is opened and accessed using activate and read/write commands. This case shows the highest bank access latency.

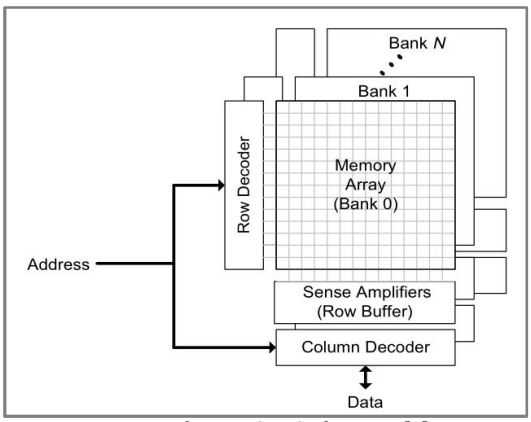

Fig.1. Modern DRAM Architecture [1]

Pre-charge and activate need more time than read or write operation, so something should be done to minimize the number of pre-charge or activate operations. Between the processors and the memory system, a memory controller works to translates memory requests into memory commands. A memory controller consists of a memory scheduler, a request buffer, and write/read buffers. The role of memory scheduler is very important in performance because it reorders requests to meet the scheduling policy of the system for its own goal, such as maximum throughput. Memory schedulers have 2 levels. Each bank owns *per-bank scheduler*, prioritizing requests to this bank and generating a sequence of DRAM commands while respecting the bank timing constraints. The *access-bank channel scheduler* receives the banks' ready commands and issues the one with the highest priority while respecting the time constraints and scheduling conflicts in the DRAM address and data buses. The request buffer contains each memory request's state such as the address, type and identifier of the request. The read/write buffers contain the data being written to/read from the memory.

It is not easy work for the memory controller to acquire maximum performance upon gap between the processors and memory bandwidth. DRAM chips have complex timing constraints including both bank-local and memory-global constraints.

Modern memory controllers use FR-FCFS(First Ready-First Com First Serve) scheduling policy. It gives the top priority to the memory requests that hit in the row-buffer over others. If no request hits the row-buffer, older requests have priority over the younger ones. If the recently processed command is on the row-A, then request for row-A is selected as next among several requests. Consecutive processing on a row can maximize the utilization of memory bandwidth. For single-core systems FR-FCFS policy was shown to provide the best average performance, but this does not take into account the interference among different threads and cores on scheduling decisions. While some threads achieve high throughput, another threads can suffer from memory accessstarvation[2]. In multi-core multi-threaded systems, it is more needed for every thread to have the fair memory access opportunity to get higher overall system performance.

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Website: www.ijircce.com

Vol. 6, Issue 4, April 2018

Several memory scheduling policies for multi-core environment have been proposed. Zhu et al.[3] evaluated thread-aware memory access scheduling schemes and found it may improve the overall system performance by up to 30%. They considered the number of outstanding memory requests, the reorder buffer occupancy, and the issue queue occupancy. They mainly focused on SMP with single core processor, but their work can be adapted to the multi-core systems. Nesbit et al.[4] introduced the fair queueing(FQ) concept which consider the history of memory bandwidth utilization of each thread. Mutlu et al.[5] proposed a Stall-Time Fair Memory(STFM) scheduler to provide quality of service to different threads sharing the DRAM memory system. This scheme equalizes the DRAM-related slowdown experienced by each thread due to interference from other threads without degrading overall system performance. Complicated implementation is a difficult part of their work. Zhu et al.[6] proposed a fair thread-aware memory scheduling algorithm to achieve the fairness and memory system performance. It considers the source thread information, the arriving time and service history of each thread.

III. PROPOSED SCHEDULING ALGORITHM

A. *Priority Measure:*

In FR-FCFS scheduler, it is highly possible that the memory request of a certain thread using the same region of memory is processed with high priority to maximize the memory bandwidth. In our work, another parameter is introduced to consider the overall system performance. In multi-core multi-thread environment, the load balancing has considerable influence on the system performance. But complete fairness is not needed because the working threads have different memory access characteristics. Memory-intensive tasks can utilize the memory access locality, thus maximize the memory bandwidth utilization. It is needed that the tasks with the low memory access should have the opportunities accessing memory in a certain time. So we introduce a measure, memory access service rate (MASR), to decide the priority of the memory requests. This MASR is calculated per thread per unit time. The MASR of thread*i*is:

$$MASR_i = \frac{number\ of\ served\ requests}{number\ of\ requests} \qquad \text{eq.(1)}$$

If the MASR value goes too low, the thread is in danger of starvation and the overall system performance may be degraded. We should find the compromising point between the minimum memory latency and fair processing of all threads.

B. *Description of the Proposed Algorithm:*

To measure the exact MASR is not plausible because it needs the record of all memory access activity per thread and floating point calculation. Instead, we use waiting time TW_i of thread i because the MASR will be converged to 0 for the very long waiting time. Let TS be the service time of recent memory access request, *TH*bethe threshold value of maximum waiting time. The algorithm is as follows:

```
TS = 0
While true
    For newly arrived request, TW_i = 0
    For each memory request in queue, TW_i = TW_i + TS
    if there is a request TW_j> TH
      Select j
    else if there are row-hit access requests
      Select a row-hit access request j in FCFS manner
    else
     Select j with the largest TW
    TW_j = 0
    Complete service for request j
```

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Website: www.ijircce.com

Vol. 6, Issue 4, April 2018

TS = recent memory access service time (calculated by $T_{pre\text{-}charge}$, $T_{activate}$, T_{column})
End of while

When the time to select the next request to be served comes, the memory scheduler finds whether there is any access request to wait too long. The top priority goes to the thread waiting more than the threshold value. If there is no such request, the scheduler selects the request with the same row address as recently served request. Neither case, the next request is selected according to the First-Come-First-Serve policy.

IV. SIMULATION RESULTS

We use a cycle accurate DRAM simulator, DRAMSim2[7] in conjunction with Gem5[8], an event-driven computer system simulator. Parameters for DRAMSim2 come from DDR3 SDRAM datasheet[9]. Table 1 shows the simulation setup parameters.

Table 1 Simulation Setup

Parameters	Value
Processor	4 core, 4.5 GHz, Out-of-order
L1 Caches(per core)	64KB instruction cache, 64KB data cache, LRU
L2 Cache(shared)	1MB Cache, 8-way, LRU
DRAM	1 channel, 1 rank, 8 bank, DDR3-2133

The simulation was performed with SPEC CPU2006 benchmarks[10]. Benchmark programs have different memory access characteristics. We categorized them into 4 memory character groups and performed simulation on the character groups and the combinations of different characteristics on 4-core processors. Table 2 shows the combination of benchmarks and their characteristics.

Table 2. Workgroup and characteristics of benchmarks

Workgroup	Benchmarks	Characteristic
G1	astar,hmmer, bzip2, sjeng	Not memory-intensive, Low RB hit rate
G2	H264ref, gobmk, gcc, povray	Not memory-intensive, High RB hit rate
G3	Soplex,lbm, libquantum, sphinx3	Memory-intensive, High RB hit rate
G4	astar,mcf, hmmer, bzip2	Low RB hit rate
G5	mcf,soplex, lbm, sphinx3	Memory-intensive
G6	hmmer, bzip2, gcc, povray	Not memory-intensive
G7	gobmk,gcc, soplex, sphinx3	High RB hit rate
G8	bzip2, mcf, povray, lbm	Mixed

Before the performance evaluation, we should decide the threshold waiting time, *TH*. If *TH* becomes a big number, our algorithm will be closer to FR-FCFS, and algorithm with very small *TH* will be similar to Round-Robin, thus the memory bandwidth utilization will be the worst.

Referring to the Mutlu et al.'s work[5], we take 2 evaluation metric, *Unfairnessindex* and *Weighted Speedup*. Let MCPI be Memory Cycles per Instruction. Unfairness index is defined as follows.

$$\text{Unfairness} = \frac{max_i MemSlowdown_i}{min_i MemSlowdown_i}, \text{ where MemSlowdown} = \frac{MCPI_i^{shared}}{MCPI_i^{alone}} \qquad \text{eq. (2)}$$

With *Weighted Speedup*, we can estimate the overall system throughput.

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Website: www.ijircce.com

Vol. 6, Issue 4, April 2018

$$\text{Weighted Speedup} = \sum_i \frac{IPC_i^{shared}}{IPC_i^{alone}}, \text{ where IPC is InstructionsPer Cycle} \qquad \text{eq. (3)}$$

We evaluate our algorithm in comparison to baseline FR-FCFS algorithm. Fig. 2 and 3 shows the simulation results of 8 workgroups with *TH* = 50 memory cycles.

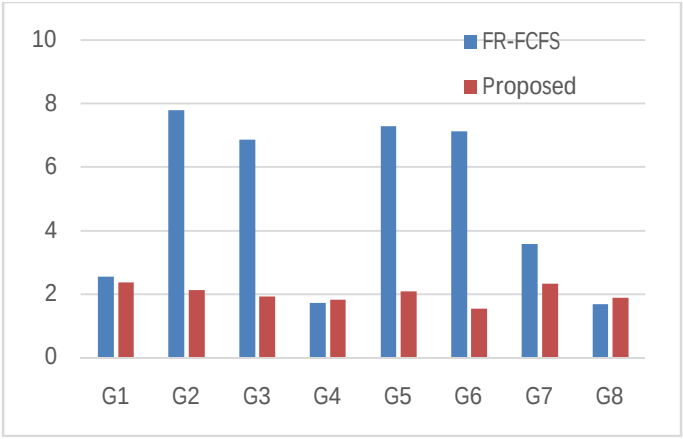

Fig. 2. Unfairness

Figure 2 shows the unfairness oftwo algorithms. Proposed algorithm reduced unfairness average 39.8%, while FR-FCFS shows better fairness in some workgroups. When the memory access patterns of different programs are well mixed, both algorithms worked more fairly.

Our algorithm achieved improvement of weighted speedup with average 10.3%. As we can see in Figure 3, proposed algorithm outperformed FR-FCFS in every cases, although the increase is not much.

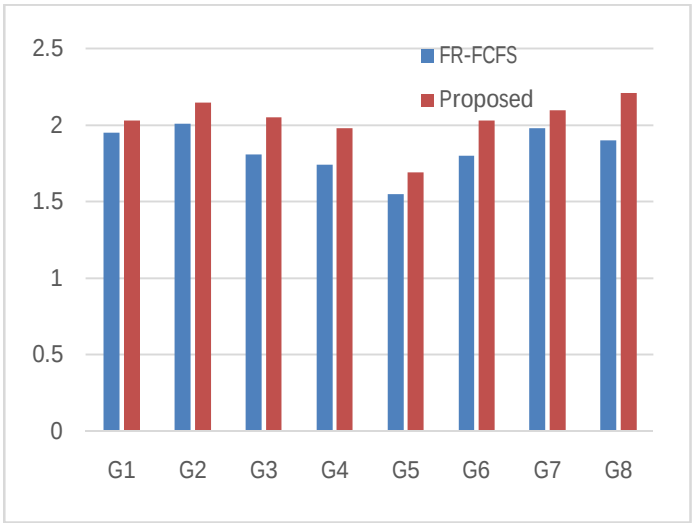

Fig. 3. Weighted Speedup

Both algorithms showed good performance for the mixed workload. The combinations of workload with similar characteristics are not good for both fairness and overall system performance.

Waiting time threshold may be important for the performance. But the difficulty with the value is that it is not easy to find regular or formal method determining the threshold.

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Website: www.ijircce.com

Vol. 6, Issue 4, April 2018

V. CONCLUSION

We proposed a memory scheduling algorithm which reduces the unfairness and improves overall system performance in multi-core multi-threaded computing environment. With the limited waiting time, any thread doesn't fall in danger of memory access starvation. From the simulation, we showed about 39% increase of fairness and 10% improvement of system performance.

We used the SPEC benchmarks for the evaluation, but more multi-threaded applications should be experimented using other kind of benchmark such as PARSEC[11, 12]. Also, for an important factor of our algorithm, waiting time threshold, more systematic approach is needed because the optimal value varies from each combination of applications according to the memory access characteristics. Dynamic estimation of threshold is also considerable method for future research work.

ACKNOWLEDGEMENT

This paper was supported by Research Fund of Kumoh National Institute of Technology.

REFERENCES

1. Rixner, S., Dally, W. J., Kapasi, U. J., Mattson, P., and Owens, J. D., "Memory Access Scheduling", Proceedings of the 27th International Symposium on Computer Architecture, pp.128-138, 2000.
2. Moscibroda, T. and Mutlu, O.,"Memory performance attacks: Denial of memory service in multi-core systems", Proceedings of the 16th USENIX Security Symposium, pp.18:1-18:18, 2007.
3. Zhu, Z. and Zhang, Z.,"A Performance Comparison of DRAM Memory System Optimizations for SMT Processors", Proceedings of the 11th International Symposium on High-Performance Computer Architecture, pp.213-224, 2005.
4. Nesbit, K.J., Aggarwal, N., Laudon, J., Smith, J.E., "Fair Queuing Memory Systems",Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 208-222, 2006.
5. Mutlu, O. andMoscibroda, T.,"Stall-Time Fair Memory Access Scheduling for Chip Multi-processors", Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture, pp.146-160, 2007.
6. Zhu, D., Wang, R., Wang, H., Qian, D., Luan, Z., and Chu, T., "A Fair Thread-Aware Memory Scheduling Algorithm for Chip Multiprocessor", Proceedings of International Conference on Algorithms and Architectures for Parallel Processing, pp.174-185, 2010.
7. Binkert, N.,Beckmann, B.,Black, G.,Reinhardt, S.,Saidi, A.,Basu, A.,Hestness, J.,Hower, D.,Krishna, T.,Sardashti, S.,Sen, R.,Sewell, K.,Shoaib, M.,Vaish, N.,Hill, M. and Wood,D.,"The Gem5 Simulator", ACM SIGARCH Computer Architecture News, vol. 39, no. 2, pp.1-7, 2011.
8. Rosenfeld, P., Cooper-Balis, E. and Jacob, B., "DRAMSim2: A Cycle Accurate Memory System Simulator," Computer Architecture Letters, vol.10, no.1, pp.16-19, Jan.-June 2011.
9. Micron Technology, Inc. Micron DDR3 SDRAM Part MT41J256M16HA, https://www.micron.com/~/media/documents/products/data-sheet/dram/ddr3/4gb_ddr3_sdram.pdf
10. "SPEC CPU 2006", https://www.spec.org/cpu2006.
11. http://parsec.cs.princeton.edu
12. Chen, T., Guo, Q., Temam, O., Wu, Y., Bao, Y., Xu, Z.,and Chen, Y., "Statistical Performance Comparisons of Computers", IEEE Transactions on Computers, vol. 64, no. 5, pp.1442-1455, 2014.