The Compact Memory Scheduling Maximizing Row Buffer Locality

Young-Suk Moon, Yongkee Kwon, Hong-Sik Kim,
Dong-gun Kim, Hyungdong Hayden Lee, Kunwoo Park
SK Hynix Inc.
{youngsuk.moon, yongkee.kwon, hongsik.kim,
donggun.kim, hyungdong.lee, kunwoo.park } @ skhynix.com

Abstract*— *Throughout many generations of memory from DDR1 to DDR3, the internal memory architecture and the performance related characteristics of DRAM has experienced little change. Therefore, memory scheduling algorithm is important to improve total memory sub-system performance. In this paper, Row Buffer Locality based Drain Policy (RLDP) is proposed to enhance the row buffer locality of memory request sequences. Based on experimental results, the proposed scheduling algorithm could improve total execution time by 9.99% compared to FCFS on average.

I. INTRODUCTION

DRAM (Dynamic Random-Access Memory), the most commonly used technology for building main memory for modern computer system, has been a major performance bottleneck for decades [1]. Throughout many generations of DRAM, from DDR1 to DDR3, internal memory architecture and performance-related characteristics of DRAM has experienced little change [1,2]. Therefore, memory request scheduling algorithms have been studied to address the performance bottleneck due to DRAM and row buffer locality is the key characteristic which memory request scheduling algorithms exploit [3].

DRAM architecture is segmented into 4 or 8 banks (DDR2, DDR3 respectively), and each DRAM bank consists of rows and columns of DRAM cells. Each bank is accessible through a row buffer (or sense amplifier) with row address and data in a row buffer are to be read with column address. Due to this row buffer which stores data of the most recently accessed row, the data in the row buffer can be accessed faster than the data in different rows in the same bank, which is called **row buffer locality** [3].

Since Rixner et al. proposed FR-FCFS, memory scheduling that takes into account of row buffer locality [4], lots of scheduling ideas have been published to improve the throughput and the fairness [5, 6, 7, 8, 9, 10, 11]. Write latency involved with tWR (Write recovery time) is getting worse as technology scales down and the emerging new memories such as PCRAM, STT-RAM, and ReRAM have much slower write performance than current DRAM technology [15]. Performance related with write operation, therefore, is expected to be a key performance bottleneck in future.

In this paper, we prioritize a request which employs a benefit of row buffer locality during write drain operation, resulting in performance improvement. The proposed write draining policy divides into two parts: Delayed Drain and **RLDP (Row Locality based Drain Policy)**. Traditional write draining is to drain writes when there are no-pending read requests or when the number of write requests in the write queue exceeds a specific number. Our observation reveals traditional write draining policy has more room to be improved by exploiting row buffer locality of both write and read requests. In this paper, RLDP (Row Locality based Drain Policy) is proposed to maximize the benefit of row buffer locality, combined with the modified delayed write drain, and the adaptive delayed close policy.

Our experimental results show that the proposed algorithm can achieve performance improvement of 9.99% over the baseline memory scheduling algorithm, FCFS.

II. PROPOSED ALGORITHM

A. Row Locality based Drain Policy

Write drain algorithm is important for the memory controller performance. Write drain operation without considering row buffer hit status of pending requests in the read queue and write queue can increase the access or queuing delay by interrupting read requests.

Most efficient conventional write drain scheme so far is the delayed write drain algorithm [13]. Delayed write scheme assumes that read request will arrive soon when the read queue is empty, so that write draining is delayed for a short time to wait for potential read request. Write drain is performed if no read request arrived. When the number of pending write requests in the write queue exceeds “high watermark”, the write requests in the queue is compulsorily drained until the number of pending write requests in the queue reaches

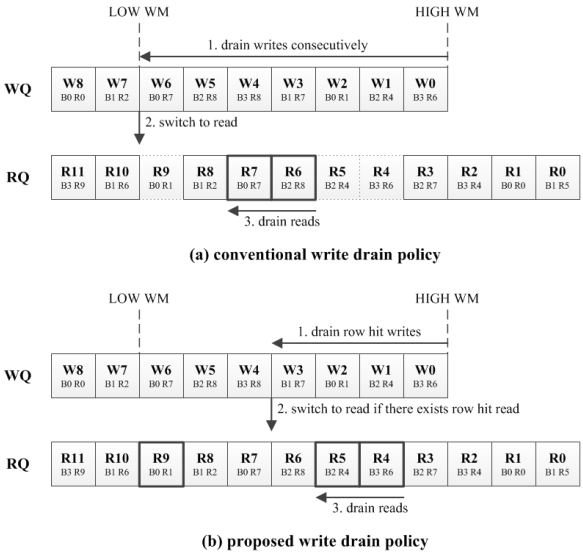

Figure 1 Write Drain Policy

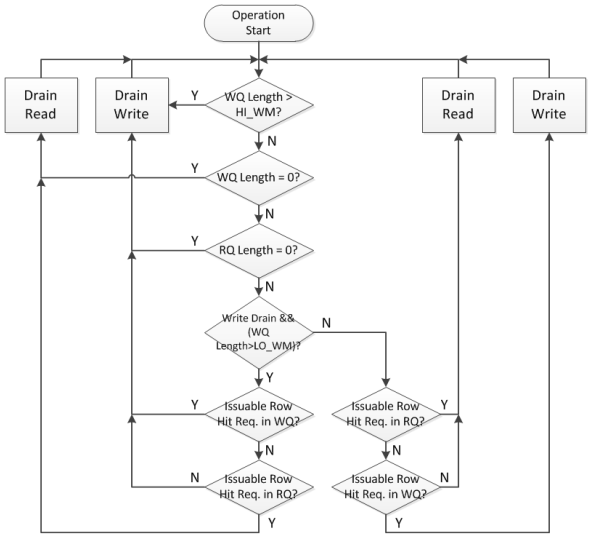

Figure 2 Flow Chart of the RLDP

"low watermark". However, in the previous write drain methods, row buffer locality is not considered, so that the locality of issued request sequence can be broken by write drain operation.

Figure 1(a) illustrates that the conventional write drain mechanism has possibility to degrade performance. Write drain is started as the number of write requests in the write queue reaches high watermark, and it is continued be drained until the number of write requests in write queue reaches low watermark. The write requests are represented by Wx, and the read requests are represented by Rx. While R4, R5 and R9 reference same pages with W0, W1 and W2, respectively, the pages are closed by W4, W5 and W6 with conventional write drain policy. This leads to unnecessary row activation when issuing R4, R5 and R9, thereby degrading performance significantly. In addition, row buffer locality is attacked when draining read requests.

In order to consider row buffer locality for write drain operation, the RLDP (Row Locality based Drain Policy) is proposed. Figure 1(b) shows how the proposed algorithm utilizes the row buffer locality in write-to-read switching. While conventional write drain policy issues write requests until the number of the write requests in the write queue reaches low watermark, the proposed write drain policy issues write requests until the "row hit" write requests in the write queue are completely consumed.

The pseudo code of RLDP is described in Figure 2. If the number of write requests in the write queue reaches high watermark, write drain is started. Once write drain is started, "row hit" write requests are issued consecutively, even if there exists pending read requests in the read queue. Write to read switching is occurred only when there is no "row hit" write requests in the write queue with "row hit" read request in the read queue. Same policy is applied for read to write switching so that row locality interference is to be reduced during request drain.

B. Delayed Write Drain & Delayed Close

The RLDP is combined with delayed write drain [13] and delayed close policy [17] in order to increase performance and utilize row buffer locality.

Delayed write drain is applied adaptively based on historical request density. If the number of the requests is large enough, performing the write request in the write queue without delay is more effective than waiting for the incoming read request. Requests are observed per 10k CPU cycles, and if the number of the requests for a channel exceeds 200, delayed write drain policy is disabled.

Per-bank delayed close scheduling algorithm references row hit rate periodically in order to determine whether postpone a precharge or not. In previous research [17], per-bank adaptive delayed close is performed based on prediction result (success or fail). This scheme is very simple, but historical statistics can't be fully considered. In order to consider historical information, the number of the read commands and the active commands for read is observed for each bank. If a read command is issued to the DRAM, "read history counter" is incremented while if an active command for read request is issued to the DRAM, "read history counter" is decremented. For every 10k CPU cycle, according to the value of "read history counter", delayed close policy is selectively applied. If the value of the "read history counter" exceeds zero, the delayed close scheme is applied to the corresponding bank. Otherwise, pure close policy is applied. The "read history counter" is attenuated by 0.5 for every 1M CPU cycle, and the same scheme is applied to the write case

Workload	Config	Sum of exec times(10M cyc)			Max Slowdown			EDP(J.s)		
		FCFS	Close	Proposed	FCFS	Close	Proposed	FCFS	Close	Proposed
MT-canneal	1 chan	418	404	377	NA	NA	NA	4.23	3.98	3.49
MT-canneal	4 chan	179	167	157	NA	NA	NA	1.78	1.56	1.37
bl-bl-fr-fr	1 chan	149	147	140	1.20	1.18	1.12	0.50	0.48	0.44
bl-bl-fr-fr	4 chan	80	76	74	1.11	1.05	1.02	0.36	0.32	0.30
c1-c1	1 chan	83	83	79	1.12	1.11	1.06	0.41	0.40	0.37
c1-c1	4 chan	51	46	46	1.05	0.95	0.94	0.44	0.36	0.35
c1-c1-c2-c2	1 chan	242	236	217	1.48	1.46	1.35	1.52	1.44	1.22
c1-c1-c2-c2	4 chan	127	118	113	1.18	1.10	1.06	1.00	0.85	0.78
c2	1 chan	44	43	42	NA	NA	NA	0.38	0.37	0.35
c2	4 chan	30	27	26	NA	NA	NA	0.50	0.39	0.37
fa-fa-fe-fe	1 chan	228	224	206	1.52	1.48	1.37	1.19	1.14	0.97
fa-fa-fe-fe	4 chan	106	99	92	1.22	1.15	1.06	0.64	0.56	0.49
fl-fl-sw-sw-c2-c2-fe-fe	4 chan	295	279	261	1.40	1.31	1.21	2.14	1.88	1.65
fl-fl-sw-sw-c2-c2-fe-fe -bl-bl-fr-fr-cl-cl-st-st	4 chan	651	620	581	1.90	1.80	1.68	5.31	4.76	4.17
fl-sw-c2-c2	1 chan	249	244	224	1.48	1.43	1.30	1.52	1.44	1.20
fl-sw-c2-c2	4 chan	130	121	117	1.13	1.06	1.03	0.99	0.83	0.78
st-st-st-st	1 chan	162	159	151	1.28	1.25	1.18	0.58	0.56	0.50
st-st-st-st	4 chan	86	81	79	1.14	1.08	1.05	0.39	0.35	0.33
Overall		3312	3173	2981	1.30 PFP: 3438	1.24 PFP: 3149	1.17 PFP: 2791	23.88	21.70	19.11

Figure 3 Comparison of Key Metrics on Baseline and Proposed Schedulers

as well.

III. Experimental Results

A. Basic Algorithms

The proposed algorithm is combined with conventional close policy, and FR-FCFS. Close scheduling policy is an approximation of a true close-page policy. In every idle cycle, the scheduler issues precharge operations to banks that last serviced a column read/write. Unlike a true close-page policy, the precharge is not issued immediately after the column read/write. FR-FCFS first tries to issue request of any open row hits, and then first come request. In order to increase bank-level parallelism, read queue is scanned sequentially from head in queue until issuable request in the current cycle is found.

B. Execution Time

Figure 3 shows the total execution time of the proposed algorithm. The simulation is performed using USIMM1.3 [16] and PARSEC [18] traces. For the multi core cases, total execution time is calculated by adding each execution time of all cores. Figure 4 shows the simulation results of the base algorithms and the proposed scheduling algorithm. All the total execution time is normalized to the execution time of the FCFS scheduling algorithm.

Among the proposed algorithm, "RLDP+DELAYED-CLOSE" shows best improvement in terms of the execution time. Compare to the FCFS scheduling algorithm, the DELAYED-CLOSE scheduling algorithm enhances the execution time by 6.86%, the RLDP scheduling algorithm enhances the execution time by 8.78% and RLDP + DELAYED-CLOSE scheduling algorithm improves the execution time by 9.99%. DELAYED-CLOSE scheduling algorithm shows an improvement in 1-channel configuration because the algorithm is designed to selectively delay precharge command, expecting potential row hit. RLDP scheduling algorithm shows an improvement for all cases.

C. Hit Rate

Figure 5 and 6 shows the row hit rate of the read and the write requests respectively for each scheduling algorithm. For the case of the RLCP scheduling algorithm, the average row hit rate of the write requests is dramatically enhanced from -22.78% to 7.27%, and the average row hit rate of the read requests is enhanced slightly from 19.11% to 19.46%, compare to the CLOSE+FR-FCFS scheduling algorithm. These results indicate that conventional write drain policy hurts the row buffer locality of the write requests. For the case of the DELAYED-CLOSE scheduling algorithm, the average row hit rate of the write requests increased from -22.78% to -12.14% and the average row hit rate of the read requests is increased from 19.11% to 21.31%. It is shown that DELAYED-

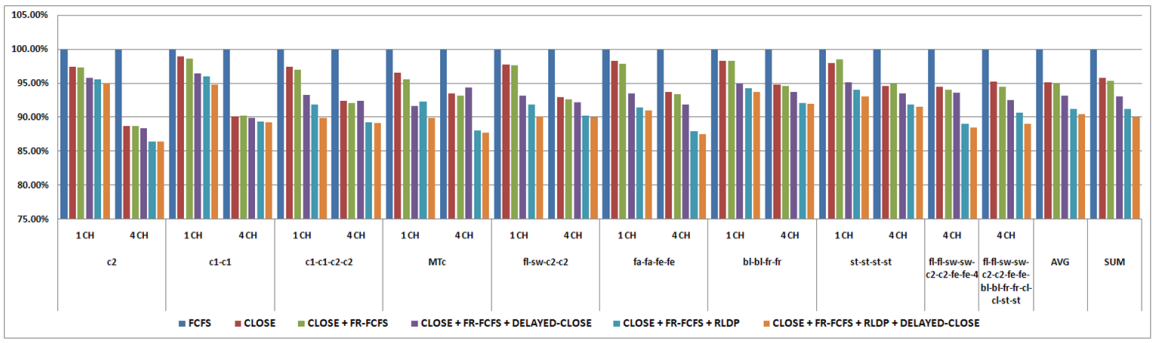

Figure 4 Total Execution Time

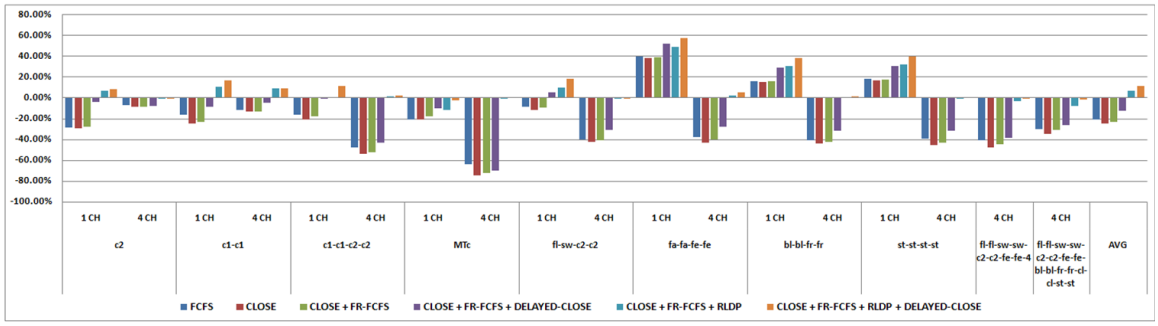

Figure 5 Row Hit Rate of the Write Request

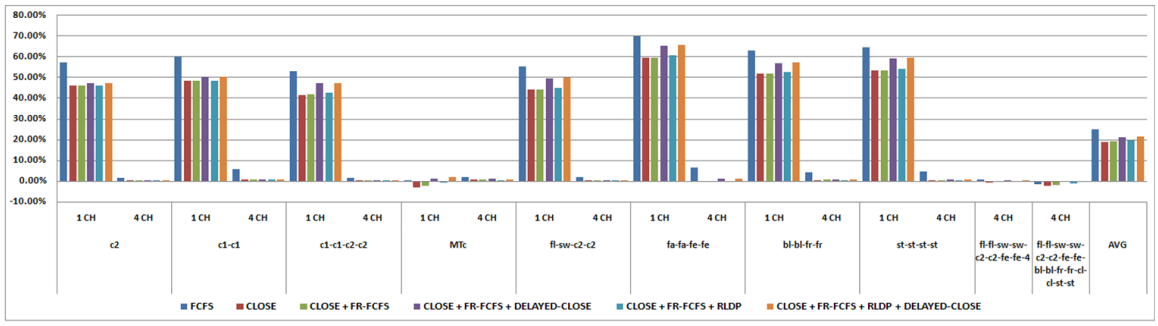

Figure 6 Row Hit Rate of the Read Request

CLOSE scheduling algorithm effectively utilizes the row buffer locality for both read and write request.

The row hit rate of the write request for the whole simulation runtime is captured in Figure 7. Simulation is performed for the 4-channel configuration and four multi-threaded canneal traces. The row hit rate is calculated for every 100k cycles which is represented by the circle and the cross. The circle represents the row hit rate of the write request with the CLOSE+FR-FCFS scheduling algorithm, and it is lower than zero for the considerable duration of the simulation time. The cross represents the row hit rate of the write request with the proposed scheduling algorithm, and it shows better result compare to the circle due to elimination of unnecessary active commands.

Briefly, the RLDP scheduling algorithm improves the row hit rate of the write requests dramatically, and the DELAYED-CLOSE scheduling algorithm improves the row hit rates of both read and write requests. Therefore, the cost of the row conflict is reduced so that the performance improves. Compared to the CLOSE+FR-FCFS scheduling algorithm, the row hit rate of the write request is increased from -22.78% to 11.50%, and the row hit rate of the read requests is increased from 19.11% to 21.32%. The proposed scheduling algorithm improves the row hit rates of both read and write requests, especially for the write request.

Figure 8 High Watermark Residence Time / Read-Write Switching Frequency

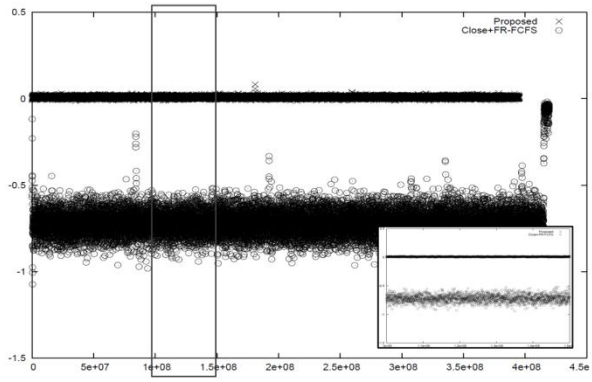

Figure 7 Row Hit Rate of the Write Request

D. Write Caused Interference

Figure 8 shows how the write caused interference is reduced by the proposed algorithm. The left side of the graph indicates the amount of time when the number of the pending requests in the write queue exceeds high watermark. Right side of the graph indicates the frequency of the read/write switching that is calculated by adding the read-to-write switching frequency and the write-to-read switching frequency. For the "HIGH WATERMARK RESIDENCE TIME" of the proposed algorithm is reduced by 69% and 59.57%, compare to the FCFS scheduling algorithm and the CLOSE+FR-FCFS scheduling algorithm, respectively. The "READ-WRITE SWITCHING FREQUENCY" of the proposed algorithm is increased by 33.78% and 24.67%, compare to the FCFS scheduling algorithm and the CLOSE+FR-FCFS scheduling algorithm, separately. "HIGH WATERMARK RESIDENCE TIME" represents the un-wanted read to write switching frequency that hurts row buffer locality. In contrast to the conventional scheduling algorithm, the read/write switching of the proposed algorithm is performed at proper time with low timing overhead, by preventing the potential row conflict in the future. By reducing un-wanted read/write switching frequency, the total number of row conflict can be decreased which leads to performance improvement.

	# Total Register	Information Bit	CH	RANK	BANK
recent_colacc	64	1	4	2	8
refreshes	256	4	4	2	8
dd_counter	8	2	4		
read_history_counter	1216	19	4	2	8
write_history_counter	1216	19	4	2	8
request_density_counter	28	7	4		
delayed_close_valid_r	64	1	4	2	8
delayed_close_valid_w	64	1	4	2	8
delayed_close_counter_r	192	3	4	2	8
delayed_close_counter_w	192	3	4	2	8
DD_MAX	2	counting 2			
HIT_RATE_UDPATE_PERIOD	10	counting 1,000			
COUNTER_ATTENUATION_PERIOD	18	counting 1M/4			
DENSITY_UPDATE_PERIOD	9	counting 500			
REQUEST_DENSITY_THRESHOLD	4	counting 10			
CLOSE_DELAYING_TIME	3	counting 7			
Total	3282				

Figure 9 Hardware Overhead of the Proposed Algorithm

E. Hardware Overhead & Complexity

The hardware overhead of the proposed scheduling algorithm is shown in Figure 9. The *recent_colacc* is the per-bank register that indicates whether a certain bank is a precharge candidate or not. The *refreshes* is the per-bank register that stores the number of refresh commands issued in 8*tREFI refresh window. The *dd_counter* is the per-channel register that is used for delayed write drain.

The *read_history_counter* is the register that stores the difference between the number of the read and the active for read. The *write_history_counter* is the register that stores the difference between the number of the write and the active for write. The contents of both counters are reduced by half for every 1M processor cycle (0.25M memory cycle). The *request_density_counter* is the per channel counter that tracks the read and write commands issued to the DRAM during a certain period. The *delayed_close_valid_r* and the *delayed_close_valid_w* are the per-bank registers that indicate whether the corresponding bank is the candidate for delayed close or not. The *delayed_close_counter_r* and the *delayed_close_counter_w* are the per-bank counter used for delayed close policy. Additional registers are needed to store constant value that is referenced

periodically to update registers. Total number of registers required to implement the proposed scheduling algorithm is about 3.3k.

It is not complex to design the control logic. Because RLDP scheduling algorithms only searches for the write queue and read queue so as to find a row hit request, it is easy to implement with small control logic and comparators. For the case of the delayed close scheduling algorithm, the difference between the read/write command and the active for read/write command is considered. It is easy to implement because the control logic only needs to check if the counter exceeds zero in order to apply delayed close.

IV. CONCLUSION

RLDP scheduling algorithm is proposed to utilize the row buffer locality. The proposed scheduling algorithm improves the row hit rate of both write request and read request. The number of the active command is reduced so that the total execution time is improved by the amount of 5.64% and 9.99% compare to the CLOSE+FR-FCFS scheduling algorithm, and FCFS scheduling algorithm, respectively.

REFERENCES

[1] B. Jacob, S. W. Ng, and D. T. Wang. Memory Systems - Cache, DRAM, Disk. Elsevier, Chapter 7, 2008

[2] JEDEC. *JEDEC standard*: *DDR/DDR2/DDR3 STANDARD* (JESD 79-1,2,3)

[3] Chang Joo Lee. DRAM-Aware Prefetching and Cache Management, HPS Technical Report, TR-HPS-2010-004, University of Texas, Austin, December, 2010.

[4] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens. Memory access scheduling. In *Proceedings of ISCA*, 2000.

[5] M. Awasthi, D. Nellans, K. Sudan, R. Balasubramonian, and A. Davis. Handling the Problems and Opportunities Posed by Multiple On-Chip Memory Controllers. In *Proceedings of PACT*, 2010.

[6] Y. Kim, M. Papamichael, O. Mutlu, and M. Harchol-Balter. Thread Cluster Memory Scheduling: Exploiting Differences in Memory Access Behavior. In *Proceedings of MICRO*, 2010.

[7] O. Mutlu and T. Moscibroda. Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors. In *Proceedings of MICRO*, 2007.

[8] O. Mutlu and T. Moscibroda. Parallelism-Aware Batch Scheduling - Enhancing Both Performance and Fairness of Shared DRAM Systems. In *Proceedings of ISCA*, 2008.

[9] C. Lee, O. Mutlu, V. Nerasiman, and Y. N. Patt, "Prefetch-Aware DRAM Controllers," In *Proceedings of MICRO*, 2008.

[10] I. Hur and C. Lin, Adaptive History-Based Memory Schedulers. In *Proceedings of MICRO*, 2004.

[11] D. Kaseridis, J. Stuecheli, and L. John. Minimalist Open-page: A DRAM Page-mode Scheduling Policy for the Many-core Era In *Proceedings of MICRO*, 2007.

[12] J. Stuecheli, D. Kaseridis, D. Daly, H. Hunter, and L. John. The Virtual Write Queue: Coordinating DRAM and LastLevel Cache Policies. In *Proceedings of ISCA*, 2010.

[13] C. Natarajan et al. A study of performance impact of memory controller features in multi-processor server environment. In *Proceedings of WMPI*, 2004.

[14] N. Chatterjee, N. Muralimanohar, R. Balasubramonian, A. Davis, and N. Jouppi. Staged Reads : Mitigating the Impact of DRAM Writes on DRAM Reads. In *Proceedings of HPCA*, 2012.

[15] B. Lee, E. Ipek, O. Mutlu, and D. Burger. Architecting Phase Change Memory as a Scalable DRAM Alternative. In *Proceedings of ISCA*, 2009.

[16] N. Chatterjee and R. Balasubramonian and M. Shevgoor and S. Pugsley and A. Udipi and A. Shafiee and K. Sudan and M. Awasthi and Z. Chishti. USIMM: the Utah SImulated Memory Module. 2012.

[17] http://www.anandtech.com/show/3851/everything-you-always-wanted-to-know-about-sdram-memory-but-were-afraid-to-ask/6

[18] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The PARSEC Benchmark Suite: Characterization and Architectural Implications. In *Proceedings of PACT*, 2008