Rank Idle Time Prediction Driven Last-Level Cache Writeback

Zhe Wang Samira M. Khan Daniel A. Jiménez
The University of Texas at San Antonio
{zhew, skhan, dj}@cs.utsa.edu

ABSTRACT
In modern DDRx memory systems, memory write requests can cause significant performance loss by increasing the memory access latency for subsequent read requests targeting the same device. In this paper, we propose a rank idle time prediction driven last-level cache writeback technique. This technique uses a rank idle time predictor to predict long phases of idle rank cycles. The scheduled dirty cache blocks generated from last-level cache are written back during the predicted long idle rank period. This technique allows servicing write request at the point that minimize the delay it caused to the following read requests. Write-induced interference can be significantly reduced by using our technique.

We evaluate our technique using cycle-accurate full-system simulator and SPEC CPU2006 benchmarks. The results shows the technique improves performance in an eight-core system with memory-intensive workloads on average by 10.5% and 10.1% over conventional writeback using two-rank and four-rank DRAM configurations respectively.

Categories and Subject Descriptors
B.3.2 [**Memory Structures**]: Design Styles-Cache memories; D.3.4 [**Software**]: Processors-Memory management

General Terms
Design, Experimentation, Performance

Keywords
Writeback, Memory Management, LLC

1. INTRODUCTION
In modern DDRx memory systems, memory write requests can delay the service of subsequent read requests targeting the same rank, thus increasing the access latency of the read requests. This *write-induced interference* [9] can cause significant system performance degradation. Intelligent scheduling and adjustment of read/write priority can reduce write-induced interference. Intelligent scheduling allows write requests to be serviced by DRAM efficiency, thus reducing write-imposed penalties to subsequent reads. Write-induced interference can be eliminated by always prioritizing reads over writes. However, writes must have priority over reads at some point due to a limited size of write buffer. Therefore, it is better to prioritize writes at the point that minimize the interference to subsequent reads.

In a conventional writeback policy, dirty cache blocks are sent to the write buffer when they are evicted from the last-level cache (LLC). The write buffer is drained following the buffer management policy. Several proposals [16, 18, 13] improve writeback efficiency using an intelligent scheduling algorithm. However, the write buffer only has a small number of entries due to design complexity and power efficiency, limiting the ability to schedule high locality write requests as well as the possibility to flexible adjust read/write priority.

LLC writeback techniques have been proposed to expand write resources using near least recently used (LRU) position of the LLC. Eager writeback [10] sends dirty cache blocks in the LRU position to DRAM for service when the rank is idle, thus re-distributing write requests. The virtual write queue (VWQ) [20] technique issues scheduled writebacks from near the LRU position in the LLC to improve writeback efficiency. To reducing write-induced interference, both eager writeback and VWQ techniques issue write requests to memory when no read requests target the same rank. Unfortunately, these techniques have no knowledge about when the next read request will come. If a read request comes soon after a write request is issued, the write will still impose large penalty on the read.

Multiple memory controllers and multiple ranks are used to service memory requests in parallel. Due to workload characteristics and load imbalance, some ranks often have idle cycles while the application is running. In this paper, we propose a prediction driven LLC writeback technique. This technique uses a *rank idle time predictor* to predict when a rank will have significant idle time. "Rank idle" means that there will be no read request for this rank that will be delayed by scheduling writeback events. The scheduled write requests can be written back during this idle rank period. We incorporate the rank idle time predictor into the parallelism-aware LLC scheduling technique and propose a prediction driven parallelism-aware LLC writeback technique. The proposed technique applies to the DRAM system that maps the rank and channel into the higher order bits than the column in the physical address. Write-induced interference is significantly reduced by our technique.

This paper makes the following contributions:

- We propose a technique that makes use of the rank idle cycles to isolate the service of memory read and write requests as much as possible.
- We propose a low-overhead *rank idle time predictor* to predict long periods of idle time in memory ranks. This predictor is used to guide the LLC writeback policy.
- We incorporate the rank idle time predictor into the parallelism-aware LLC scheduling technique. Scheduled write requests are written back to the memory guided by the predictor to reduce the write-induced interference.
- We present an evaluation of our techniques with an eight-core processor using the MARSSx86 Simulator [15] together with DRAMSim2 [17] for multi-rank DRAM configurations. The experimental results show a significant performance improvement over previous techniques.

Evaluation with multi-programmed SPEC CPU2006 workloads shows that the technique improves the performance in an eight-core system with memory-intensive workloads on average by 10.5% and 10.1% over conventional writeback using two-rank and four-rank DRAM configurations respectively.

2. BACKGROUND AND MOTIVATION

DRAM systems [2] [3] have multiple channels that can be accessed independently. Each channel is composed of one or multiple memory ranks that share the same address and data bus. A memory rank consists of a set of chips where chips in the same rank can be accessed simultaneously. Each chip is organized as multiple banks that can be operated in parallel. In a DDRx memory module, each rank has a 64-bit data bus. Chips within a rank work in unison to return 64 bits per cycle.

A memory bank has a number of two-dimensional data arrays that arranges as rows and columns. When a memory access request comes, an entire row of bits that contains the required data is brought into the row buffer. Then a column of this row buffer is selected according to the column address. The row-hitting request is the request that goes to a currently open row. Data can be accessed without activating the row buffer again. Therefore, the row-hitting request can be serviced much faster than the request that goes to a row that not currently in the row buffer.

2.1 Address Mapping Scheme

Row ID	Column ID	Rank ID	Bank ID	Channel ID	Cache Line

(a)

Row ID	Rank ID	Bank ID	Channel ID	Column ID	Cache Line

(b)

Figure 1: Address mapping scheme (a) cache line interleaving (b) page interleaving

The memory address mapping scheme [11] maps physical addresses to memory resources. Figure 1 shows an example of two typical address mapping schemes: cache line interleaving and page interleaving mapping schemes. In the cache line interleaving mapping scheme, consecutive cache lines are distributed to different rank/bank/channel combinations to maximize the parallelism of memory access. This mapping scheme will cause read requests to go to different ranks frequently and produce fragmented short idle cycles which might be too short to compensate for large write-induced interference. The page interleaving mapping scheme maps the lower order bits of the physical address into the column address to maximize the number of row buffer hits. The memory access patterns of most applications have spatial locality, so the application tends to access a certain rank for a while before switching to another rank. Therefore, compared with the cache line interleaving mapping scheme, the page interleaving mapping scheme tends to collect small chunks of idle rank cycles into large runs.

Our technique prefers long, consecutive idle cycles in the rank rather than short and fragmented idle cycles. Therefore, our technique works with an address mapping scheme that maps the rank ID bits and channel ID bits higher than the column ID bits. We will introduce our technique in the context of page interleaving mapping scheme. The page interleaving mapping scheme is widely used in DDRx memory systems [2, 7, 6].

2.2 Motivation

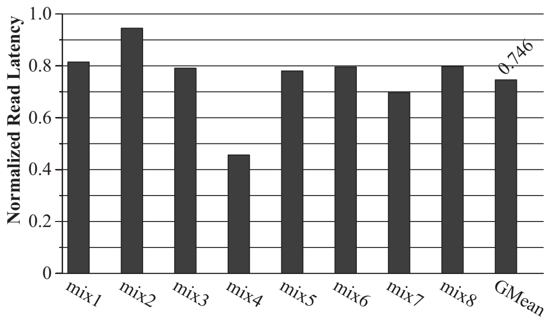

Figure 2: Read latency using conventional writeback and perfect writeback techniques in quard-core processor

We performed experiments to motivate the the use of idle rank time for reducing the write-induced interference.

Figure 2 shows read latency normalized to conventional

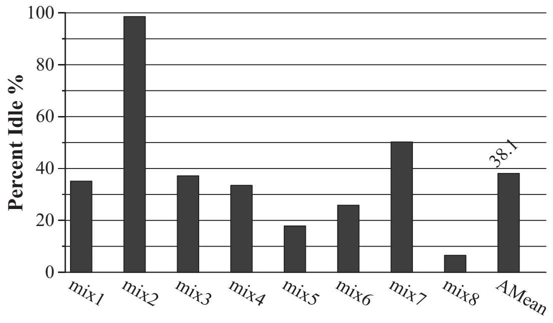

Figure 3: Rank idle percentage

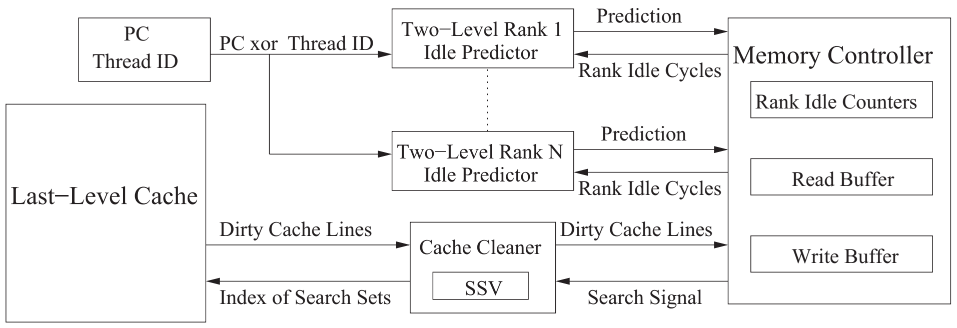

Figure 4: System structure

writeback on a quad-core processor for perfect writeback. Perfect writeback assumes memory write access does not cause any interference to read access, which is the optimal case. Read latency is computed as the sum of the DRAM busy cycles for each core divided by the number of LLC misses.

We randomly choose eight mixes of SPEC CPU2006 benchmarks for the experiment. The system used for evaluation is a 4.8 GHZ quard-core processor with JEDEC [1] based DDR3-1600 memory. The memory system has two channels, one rank per channel, and uses the First Ready-First Come First-Served (FR_FCFS) [16] memory scheduling algorithm.

From Figure 2, we can see the read latency for perfect writeback is 74.6% of conventional writeback. Thus, 25.4% of the read latency suffered by conventional writeback is caused by write-induced interference. Write-induced interference is more obvious in a multi-core system since the performance of a given application is affected by its own writeback accesses as well as the writeback accesses from other applications.

Figure 3 shows the ratio of idle rank cycles to total execution time as a percentage for conventional writeback. Ranks are idle 38.1% of the time on average. The motivation of rank idle prediction is based on these large idle rank percentage and write-induced interference: if this idle time could be used for write access, it could decrease write-induced interference significantly.

3. RELATED WORK

Many proposals [16, 18, 21, 12, 14] focus on improving memory efficiency by scheduling or relocating memory accesses to yield as many row hits as possible or servicing memory accesses in parallel. Some prior work [5, 22] also propose using prediction to optimize memory performance. Hur *et al.* [5] propose a scheduling algorithm that uses a state machine to make the next scheduling decision based on the past behavior. Xu *et al.* [22] use a two-level dynamic predictor to predict which buffer management policy to use for different memory accesses.

Much previous work [21, 12, 14] does not take into account the write interference problem. Eager writeback [10] is the first work that expands write resources by using the LLC to reduce write-induced interference. Eager writeback writes back dirty cache blocks in the least-recently-used (LRU) position of the LLC sets whenever the bus is idle instead of waiting for the block to be evicted to reduce the memory traffic. However, the scheduling window of eager write back is still limited to the size of the write buffer. Thus, the scheduling decision it makes is far from optimal.

Stuecheli *et al.* [20] propose a virtual write queue (VWQ) technique. Their technique takes a fraction of the LRU positions in the LLC as the VWQ. Dirty cache blocks with high locality in the VWQ are written back in a batch, therefore improving writeback efficiency. The drawback of this technique is that it needs to search the dirty cache blocks in VWQ that hit in the same row when mapping to the DRAM. Although it uses the Cache Cleaner technique to help searching, it still consumes significant LLC power and search time. Our technique avoids this overhead.

To reduce write-induced interference, both eager writeback and VWQ techniques issue write requests to DRAM when the rank is idle. Unfortunately, the memory controller does not have knowledge about how long the rank will remain idle. The write-induced penalty might be too high to be hidden by the short rank idle period.

In this paper, we propose a rank idle time prediction driven LLC writeback technique. In contrast to previous work [20, 9] that does not exploit rank idle time, our technique allows the memory to service write requests during the significant idle rank time. The technique can be used with LLC writeback scheduling techniques to improve memory efficiency.

4. RANK IDLE TIME PREDICTION DRIVEN LAST-LEVEL CACHE WRITEBACK

We propose a prediction-driven LLC writeback technique. Our technique fills DRAM idle rank cycles with scheduled writeback requests. It predicts when there will be long stretches of idle rank cycles and issues scheduled writeback requests in those stretches of times such that significant interference with subsequent read requests in the same rank will not occur.

Figure 4 illustrates the structure of our technique. A two-level predictor is used to predict long stretches of idle rank cycles for a given rank. The two-level predictor is composed of two predictors making predictions at different times to predict whether there will be significant idle rank time for a particular rank. Each rank has one two-level predictor. Thus, the number of two-level rank idle predictors for a DRAM system is equal to the number of ranks this system has. A sequence of scheduled dirty cache blocks that are generated from LLC in the direction of Cache Cleaner [20] are written back during a predicted long idle period.

4.1 Two-Level Rank Idle Time Predictor

The two-level rank idle time predictor is used to predict long idle rank periods. The technique works well with applications that have long stretches of idle rank cycles, especially for DRAM system with multiple ranks. For DRAM system with multiple ranks, memory access can conflict in some ranks and leave other ranks idle. From Figure 3, we already see that in multi-rank system, each rank has significant idle time.

The rank idle time predictor is a program counter (PC) based predictor inspired by the PC-based sampler dead block predictor (SDBP) [8]. The SDBP uses PC information to accurately predict whether an LLC block is "dead," i.e. whether it will be accessed again before being evicted. The design of the rank idle time predictor is based on the observation that if there is a long idle rank period after an instruction related to a LLC miss when there are no read requests in that rank, there is a high probability that the same behavior will be observed the next time this instruction causes a miss in the LLC with no read requests in that rank.

4.1.1 *Making Prediction*

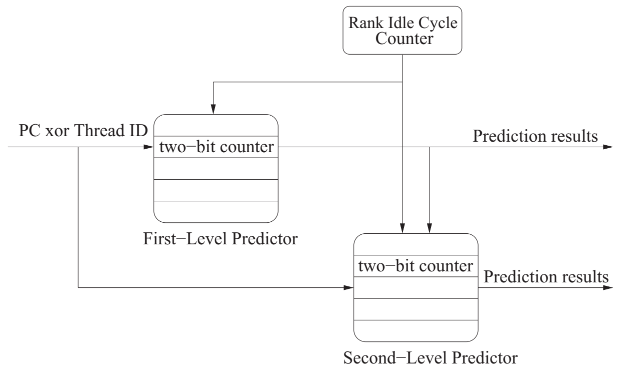

Figure 5: A two-level rank idle time predictor

Each rank has a two-level predictor. The structure is shown in Figure 5. Two levels are used so that if the first predictor mispredicts a long idle period, the second predictor has another chance to predict this long idle period. The two predictors have the same structure, make their predictions at different times, and update at the same time. The prediction state consists of a table of two-bit saturating counters, much like a branch predictor. The predictor table is indexed by the address (PC) of the instruction and the thread number. The PC is that of the last instruction before the rank becomes idle. The predictor makes a prediction according to the high bit of the selected counter: long idle time if the bit is one, short idle time if the bit is zero. The rank idle cycle counter is used to count the number of idle cycles. This number is used to choose the predictor to make a prediction and update the predictor.

4.1.2 *Prediction Driven Writeback Mechanism*

As soon as a rank becomes idle, the first-level predictor makes a prediction about whether there will be read requests coming to that rank in the next m cycles. A sequence of s scheduled dirty cache blocks will be written back to DRAM during the predicted m idle cycles. In the DRAM system with eight-bank per rank, we choose $s = 8$ to maximize

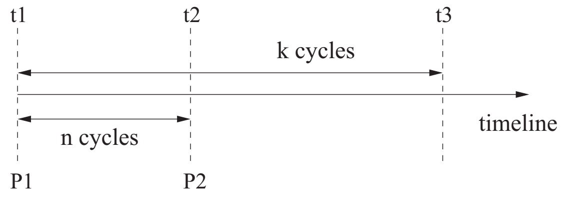

Figure 6: Prediction timeline

the bank-level parallelism when servicing the write requests. The parameter m is related to s; we want to make sure m can cover most of the service time of s scheduled dirty cache blocks.

Figure 6 shows the time to make a prediction during the idle rank cycles. Assuming the rank is idle from time t_1, the rank idle cycle counter starts to count the rank idle cycles and the first predictor makes a prediction at time t_1. If the prediction result from the first level predictor $P1$ is false (i.e., no long idle time predicted) and there are no read requests coming after n idle cycles, the second level predictor $P2$ is used to make a prediction. If the prediction result is true, s scheduled dirty cache blocks will be send to DRAM for service.

If both of the prediction results are false, but the idle rank time is longer than a threshold k, s scheduled dirty cache blocks are written back. This optimization comes from the observation that if there are rank idle cycles longer than k, there is a high probability that the idle cycles are also longer than $k + m$.

If either of the predictor results is true or the idle rank period is longer than the threshold k, the system will monitor the service of the write requests. If all of the previous s write requests have been finished service. and there are still no read requests coming in, another group of s scheduled dirty cache blocks will be sent to the DRAM system for service.

4.1.3 *Predictor Update*

The predictor will be updated when a read request comes and the rank is idle. If the idle rank cycles counted by the rank idle cycle counter are larger than m, the two-bit counter in the first-level predictor indexed by the the last PC and thread ID encountered before the rank was idle will be incremented; otherwise it will be decremented. If there are more than $m+n$ idle rank cycles, the corresponding two-bit counter in the second-level predictor will be incremented, otherwise it will be decremented.

Why does the rank idle time predictor work in multi-core systems.

The memory access patterns of most applications have spatial locality. Our technique is applied to the address mapping scheme that maps the rank and channel bits higher than the column bits, so the application tends to access a certain rank for a while before switching to another rank. In the modern DDRx memory systems, multiple controllers and multiple ranks are used to service the memory requests in parallel, thus in a lengthy stretch, only a small number of applications access a certain rank. Therefore, the memory access pattern for a certain rank is repeatable and predictable. Additionally, the rank idle predictor only makes the prediction when the rank starts to become idle, i.e., when all of the programs leave a rank idle. From our observation, the memory read accesses tend to come in bursts. The same program behavior that leads to one burst tends to lead to other bursts, as well as those bursts ending.

4.2 LLC Writeback Policy

The LLC writeback policy searches for dirty cache blocks near the LRU position in the LLC and sends a sequence of scheduled dirty cache blocks to the write buffer. Scheduled writebacks are used because scheduled write requests map to memory resources in a way that can be serviced more efficiently.

In our implementation, a cache block is considered "near the LRU position" if it resides in the bottom eighth of the LRU recency stack [20]. We incorporate the rank idle time predictor into the LLC parallelism-aware writeback policy.

The LLC parallelism-aware writeback policy searches the dirty cache blocks in the LLC that target to the same rank but different banks. Compared with LLC writeback policy of VWQ, which exhaustively searches the row-hitting cache blocks in the related cache sets in the direction of Cache Cleaner [20], our scheme does not need to search a large amount of cache sets and perform tag matching, thus consuming less power and searching time.

The Cache Cleaner [20] uses a Search Set Vector (SSV) to help searching dirty cache blocks in the LLC that could be serviced more efficiently when mapping to the DRAM resources. In the parallelism-aware scheduling scheme, when the predictor predicts a long idle period, a group of dirty cache blocks composed of the writeback requests to this idle rank but different banks are sent out to the DRAM system. Most modern DDRx systems use an eight-bank per rank memory configuration. Therefore, when more than one group of scheduled write requests are issued during the idle rank period, the rank resource access latency can be overlapped by bus burst cycles, reducing the average write request service time.

If the number of write requests in the write buffer is larger than a threshold, and there are no predicted long idle periods, the write requests will be sent to the DRAM for service whenever the rank is idle or the write buffer is full.

4.3 Storage Overhead

For each rank, we use a two-level rank idle predictor. Both levels are the same size. Each predictor has 8K entries and each entry has a two-bit counter. Thus, the total storage for the two-level rank idle predictor is 4K bytes. For an eight-core, 16M and 16-way LLC, the storage for SSV table is 2K bytes. Therefore, the total storage for the memory system with two memory controllers, two-rank per channel and four-rank per channel are 18K bytes, 34K bytes, respectively. Both of them are less than 0.3% of the capacity of the 16M LLC.

5. METHODOLOGY

This section outlines the experimental methodology used in this study.

5.1 System

We use the MARSSx86 [15], a cycle-accurate simulator for X86-64 architecture. The experiment models an out-of-order eight-core processor with 16M shared LLC. The system configuration is shown in Table 1. The DRAMsim2 [17] is incorporated into MARSSx86 to simulate a detailed cycle-accurate DRAM system. We configure DRAMsim2 to model a DDR3-1600 11-11-11 DRAM system with two channels. Both two-rank per channel and four-rank per channel configurations are evaluated in our experiment.

5.2 Benchmarks

We use the SPEC CPU2006 [4] benchmarks for this study. Of the 29 SPEC CPU2006 benchmarks, 24 could be compiled and run without errors on MARSSx86. Table 2 shows six mixes of these 24 SPEC CPU2006 benchmarks randomly chosen eight at a time. We use these mixes for eight-core simulation. Each benchmark runs simultaneously with the others. For each mix, we made a checkpoint by running the one of the memory intensive benchmarks to a typical phase identified by SimPoint [19]. Then we run the experiment for 2 billion instructions total for all eight cores starting from the checkpoint. Each benchmark is run with the first *ref* input provided by the *runspec* command.

5.3 Techniques

We evaluate six techniques for this study. Table 3 gives these techniques and a legend for their name. For traditional writeback, we simulated the following write buffer management policies: 1) writes in the write buffer are sent to the DRAM for service when the corresponding rank is idle or the occupancy of the write buffer reaches a threshold, 2) writes in the write buffer are sent to the DRAM only when the write buffer is full, 3) writes in the write buffer are sent to DRAM when the corresponding bank is idle or the occupancy of the write buffer reaches a threshold. Our evaluation shows the policy 1) yields the best performance. To ensure fairness we choose to use the policy 1) for conventional writeback evaluation.

The memory scheduling technique we use for evaluation is FR_FCFS [16]. The other memory read scheduling techniques could also work with our write scheduling optimization, we choose FR_FCFS for simplicity.

6. RESULTS

In this section, we give the results of our experiments studies.

6.1 Performance Analysis

The baseline technique in our evaluation is PI-CWB. Figure 7 shows the IPC speedups normalized to baseline in a simulated eight-core processor with a two-rank DRAM system; that is, each channel has two ranks. For each benchmark, we show the speedup of the first run in the random combination. Benchmarks showing in Figure 7 are those the performance of perfect writeback could be improved more than 10% over the baseline. Perfect writeback means all write-induced interference is eliminated. If perfect writeback gives a significant improvement over the baseline for a particular benchmark, that means the performance of this benchmark has a potential to be improved when using writeback optimization. In this experiment, for 16 of 24 benchmarks, the performance of perfect writeback could be improved more than 10% over the baseline. Thus, most of the benchmarks can benefit from writeback optimization in a multi-core system.

Execution core	4.8GHZ, eight-core CMP, out of order, 256 entry buffer, 48 entry load queue 44 entry store queue, 4 width issue/decode, 15 stages, 256 physical registers
Caches	L1 I-cache: 64KB/2 way, private, 64 bytes block size, LRU, 2-cycle L1 D-cache: 64KB/2 way, private, 64 bytes block size, LRU, 2-cycle L2 Cache: 16MB/16 way, shared, 64 bytes block size, LRU, 14-cycle
Main Memory	2 memory controllers, 2/4 ranks per channel, 32-entry write buffer per channel 8 banks per rank, 8K bytes row buffer per-bank, DDR3-1600 11-11-11

Table 1: System configuration

Name	Benchmarks
mix1	hmmer sphinx3 libquantum GemsFDTD gobmk perlbench lbm astar
mix2	perlbench gobmk namd lbm gamess GemsFDTD xalancbmk cactusADM
mix3	omnetpp hmmer cactusADM xalancbmk GemsFDTD gcc soplex astar
mix4	gromacs astar h264ref lbm omnetpp gcc libquantum calculix
mix5	gobmk tonto zeusmp milc bzip2 mcf hmmer astar
mix6	omnetpp libquantum hmmer sphinx3 bwaves milc xalancbmk calculix

Table 2: Multi-core workload mixes

Name	Technique
CI-CWB	Conventional writeback with cache line interleaving mapping scheme
PI-CWB	Conventional writeback with page interleaving mapping scheme
PA-WB	Parallelism-aware writeback
Eager-WB	Eager writeback [10]
VWQ	Virtual Write Queue [20]
RITPD-WB	Rank Idle Time Prediction Driven LLC Writeback in Section 4

Table 3: Legend for various writeback techniques.

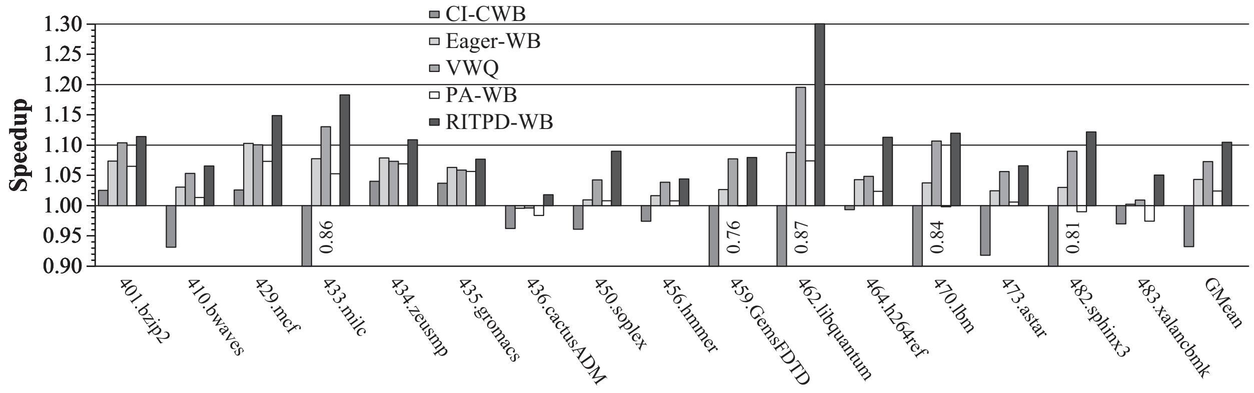

Figure 7: Performance evaluated on eight-core two-rank system

In Figure 7, conventional writeback with page interleaving mapping scheme yields much better performance than conventional writeback with cache line interleaving mapping scheme. Therefore, we implement page interleaving mapping scheme in all the other techniques. From Figure 7, we can see RITPD-WB technique outperforms all the other techniques tested across all the benchmarks. Benchmark *libquantum* has a performance improvement as large as 30.0% when using the RITPD-WB technique due to its high memory access spatial locality. That is, the memory read requests access a particular rank consecutively for a long stretch. So if write requests access the busy rank that services the read requests, there will be significant interference with the read requests. Therefore, *libquantum* benefits significantly by using the prediction driven technique to service memory write requests when the rank is idle.

In Figure 7, eager writeback improves performance by a geometric mean speedup of 4.3%. The performance improvement for the VWQ is 7.3%. Notice that the VWQ technique we implemented is in an optimal assumption that all the row-hitting write requests can be transfered back-to-back [20]; that is all the row-hitting dirty cache blocks in the near LRU position in LLC can be searched and provided during transferring the previous data from write buffer to DRAM. However, it is possible that the optimal assumption is not always satisfied in real systems, because searching a large number of cache blocks for tag matching is time consuming. The row-hitting ratio for write requests will be decreased when the optimal assumption is not satisfied, thus the system performance will be degraded compared with the optimal VWQ. Additionally, searching a large number of cache blocks for tag matching consumes significant

LLC power. PA-WB yields an average speedup of 2.4%. The RITPD-WB technique yields better performance over all other techniques. It improves performance by at least 10% of eight benchmarks and delivers an geometric mean speedup of 10.5%.

Figure 8 shows the average IPC improvement for two-rank and four-rank memory system configurations. For the four-rank configuration, eager writeback yields 3.5% speedup. The VWQ and PA-WB techniques improve performance by 8.9% and 2.7% respectively, The RITPD-WB technique also delivers the best performance among all the tested techniques. It yields a 10.1% speedup.

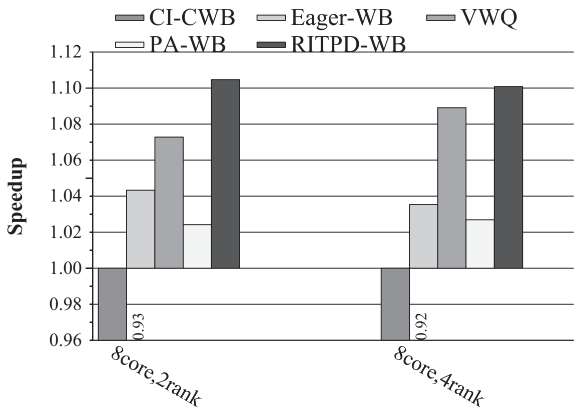

Figure 8: Average performance evaluated on two-rank and four-rank systems

6.2 Prediction Analysis

6.2.1 Predictor Accuracy

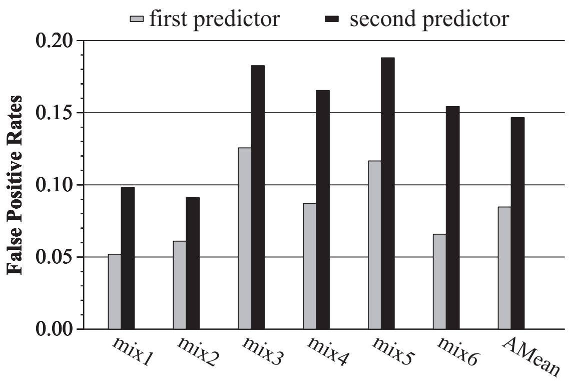

Figure 9: False positive rates for two-level predictor evaluated on eight-core two-rank system

Mispredictions comes in two varieties: false positives and false negatives. False positives are more harmful because they wrongly allow the short rank idle periods to service the LLC writebacks. Those short idle periods can not cover the majority of the service time of writebacks, thus still causing significant write-induced interference. The false positive rate is calculated by the number of mispredicted positive predictions divided by the total number of predictions. Figure 9 shows the false positive rates of the two-level predictor for a two-rank system. False positive rates for the first-level and second-level predictors are 8.5% and 14.7% on average, respectively. These low false positive rates allow our predictor to effectively predict the large rank idle period while minimizing the damage caused by mispredictions.

6.2.2 Choosing Parameter

The threshold m is the minimum number of idle cycles the predictor predicts it will occur. We want m to cover most of the service time of the s (s=8 in our experiment) scheduled writebacks. In the DDR 1600 11-11-11 memory system, servicing a write request requires $\approx 29ns$, and the write-to-precharge latency is $\approx 14ns$. The write-to-read delay is $\approx 8ns$. So if the idle rank cycles $\geq 29 + 14 + 8 = 51ns$, most of the write-induced interference to the subsequent read will be eliminated. With a 4.8GHZ clock frequency, $51ns$ is 245 cycles, so we set $m = 300$ cycles for two-rank configuration. With the number of ranks in the same channel increasing, when a particular rank is idle, the data bus might be busy with transferring the data requested by other ranks. It might take a while for the bus to transfer the write request data for that idle rank. We set $m = 400$ cycles in the four-rank configuration.

The first predictor makes a prediction immediately after the rank becomes idle. The second predictor will make a prediction after the rank has been idle for n cycles. Parameter k is the threshold that if the number of idle cycles more than k, the scheduled writebacks will also be send to DRAM. In our experiment, we found $n = 200$ cycles and $k = 600$ cycles yield the best result.

6.2.3 Eliminated Write Interference

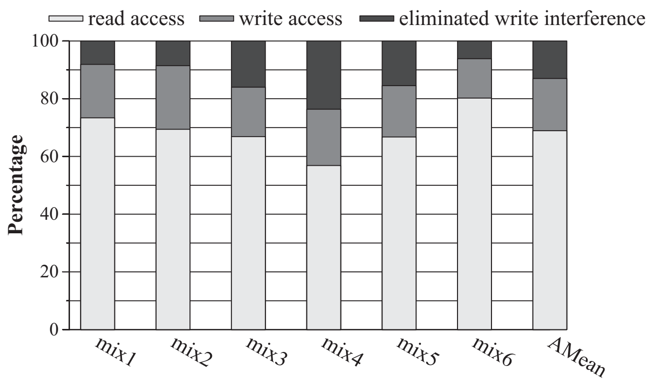

Figure 10: The percentage of write access, read access and completely eliminated write interference

Figure 10 shows the percentage of read accesses, write accesses and the completely eliminated write interference using the rank idle time predictor. Eliminated write interference means write accesses that could be serviced during the predicted rank idle time. Write accesses account for 31.1% memory accesses on average. By using the prediction driven technique, 41.8% write accesses can be serviced during the predicted rank idle time. Our technique significantly reduces the write-induced interference.

6.3 Read Latency Analysis

Figure 11 shows the read latency normalized to eager writeback for the two-rank configuration. The RITPD-WB technique reduces the write-induced interference to read ac-

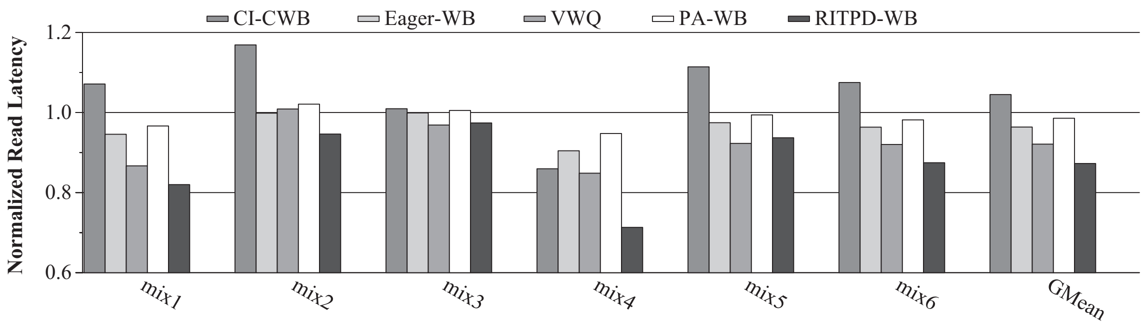

Figure 11: Read latency evaluation on eight-core two-rank system

cesses, thus reducing the average read latency. From Figure 11, we can see the RITPD-WB technique reduces the read latency significantly across all the workloads. The VWQ technique even increases the read latency for mix2; in order to schedule more memory row-hitting write requests, the dirty cache blocks that reside in the bottom fourth [20] of the LRU recency stack are considered eligible for early writebacks in the VWQ technique. Although they use the cleaned bit technique to eliminate the extra writebacks, this technique can not eliminate the extra writebacks caused by early writing back the dirty cache blocks for the first time. Compared with RITPD-WB technique, the VWQ technique has a larger rewrite ratio for mix2. These extra writebacks interfere with the read accesses, thus hurting the performance and increase the average read latency for mix2.

In our experiments, RITPD-WB reduces the read latency on average by 12.7% with two-rank configuration and 14.8% with four-rank configuration.

7. CONCLUSION AND FUTURE WORK

In this paper, we propose a rank idle time prediction driven LLC writeback technique. This technique uses a rank idle time predictor to predict when there will be a long stretch of idle cycles in a memory rank. We incorporate the rank idle time predictor into the parallelism-aware LLC writeback technique. The scheduled write requests that preserve bank-level parallelism to the DRAM system are written back during long rank idle cycles. The technique makes use of the rank idle cycles to isolate the service of read requests and write requests as much as possible, thus significantly reducing write-induced interference. Our evaluation shows a significant performance improvement over previous work. In future work, we plan to investigate the use of the rank idle predictor for other optimizations to improve the memory efficiency.

8. REFERENCES

[1] DDR3 SDRAM standard, JEDEC JESD79-3. *http://www. jedec. org.*

[2] S. B.Jacob and D.T.Wang. *The Memory Systems - Cache, Dram, Disk.* Elseiver, 2008.

[3] V. Cuppu, B. Jacob, B. Davis, and T. Mudge. High-performance DRAMs in workstation environments. *IEEE Trans. Comput.*, 50:1133–1153, November 2001.

[4] J. L. Henning. SPEC CPU2006 benchmark descriptions. *SIGARCH Comput. Archit. News*, 34:1–17, September 2006.

[5] I. Hur and C. Lin. Adaptive history-based memory schedulers. In *Proceedings of the 37th annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 37, pages 343–354, Washington, DC, USA, 2004. IEEE Computer Society.

[6] Intel Corporation. Intel 875P chipset datasheet: Intel 82875P memory controller hub (MCH). 2004.

[7] Intel Corporation. Intel 945G/ 945GZ /945GC /945P/945PL express chipset family datasheet: Intel 82945G/ 82945GZ /82945GC graphics and memory controller hub (GMCH) and Intel 82945P/82945PL memory controller hub (MCH). 2008.

[8] S. M. Khan, Y. Tian, and D. A. Jimenez. Sampling dead block prediction for last-level caches. In *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '43, pages 175–186, Washington, DC, USA, 2010. IEEE Computer Society.

[9] C. J. Lee, V. Narasiman, E. Ebrahimi, O. Mutlu, and Y. N. Patt. DRAM-aware last level cache writeback: Reducing write-caused interference in memory system. In *HPS Technical Report*, TR-HPS-2010-002.

[10] H.-H. S. Lee, G. S. Tyson, and M. K. Farrens. Eager writeback - a technique for improving bandwidth utilization. In *Proceedings of the 33rd annual ACM/IEEE international symposium on Microarchitecture*, MICRO '33, pages 11–21, New York, NY, USA, 2000. ACM.

[11] W.-F. Lin, S. K. Reinhardt, and D. Burger. Designing a modern memory hierarchy with hardware prefetching. *IEEE Trans. Comput.*, 50:1202–1218, November 2001.

[12] O. Mutlu and T. Moscibroda. Parallelism-aware batch scheduling: Enhancing both performance and fairness of shared DRAM systems. In *Proceedings of the 35th Annual International Symposium on Computer Architecture*, ISCA '08, pages 63–74, Washington, DC, USA, 2008. IEEE Computer Society.

[13] C. Natarajan, B. Christenson, and F. Briggs. A study of performance impact of memory controller features in multi-processor server environment. In *Proceedings of the 3rd workshop on Memory performance issues: in conjunction with the 31st international symposium on computer architecture*, WMPI '04, pages 80–87, New York, NY, USA, 2004. ACM.

[14] K. J. Nesbit, N. Aggarwal, J. Laudon, and J. E. Smith. Fair queuing memory systems. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 39, pages 208–222,

Washington, DC, USA, 2006. IEEE Computer Society.

[15] A. Patel, F. Afram, S. Chen, and K. Ghose. MARSSx86: A full system simulator for x86 CPUs. In *Proceedings of the 2011 Design Automation Conference*, June 2011.

[16] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens. Memory access scheduling. In *Proceedings of the 27th annual international symposium on Computer architecture*, ISCA '00, pages 128–138, New York, NY, USA, 2000. ACM.

[17] P. Rosenfeld, E. Cooper-Balis, and B. Jacob. DRAMSim2: A cycle accurate memory system simulator. *Computer Architecture Letters*, PP(99):1, 2011.

[18] J. Shao and B. T. Davis. A burst scheduling access reordering mechanism. In *Proceedings of the 2007 IEEE 13th International Symposium on High Performance Computer Architecture*, pages 285–294, Washington, DC, USA, 2007. IEEE Computer Society.

[19] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, October 2002.

[20] J. Stuecheli, D. Kaseridis, D. Daly, H. C. Hunter, and L. K. John. The virtual write queue: coordinating DRAM and last-level cache policies. In *Proceedings of the 37th annual international symposium on Computer architecture*, ISCA '10, pages 72–82, New York, NY, USA, 2010. ACM.

[21] K. Sudan, N. Chatterjee, D. Nellans, M. Awasthi, R. Balasubramonian, and A. Davis. Micro-pages: increasing DRAM efficiency with locality-aware data placement. In *Proceedings of the fifteenth edition of ASPLOS on Architectural support for programming languages and operating systems*, ASPLOS '10, pages 219–230, New York, NY, USA, 2010. ACM.

[22] Y. Xu, A. S. Agarwal, and B. T. Davis. Prediction in dynamic SDRAM controller policies. In *Proceedings of the 9th International Workshop on Embedded Computer Systems: Architectures, Modeling, and Simulation*, SAMOS '09, pages 128–138, Berlin, Heidelberg, 2009. Springer-Verlag.