Cache Performance, System Performance, and Off-Chip Bandwidth ... Pick any Two

Bushra Ahsan and Mohamed Zahran
Dept. of Electrical Engineering
City University of New York
ahsan_bushra@yahoo.com mzahran@ccny.cuny.edu

Abstract

One of the major challenges NoC and on-board interconnection has to face in current and future multicore chips is the skyrocketing off-chip bandwidth requirement. As the number of cores increases, the demand for off-chip bus, memory ports, and chip pins increases and this can severely hurt performance. It leads to bus congestion, processor stalls and hence, performance loss. Off-chip bandwidth is generated by the on-chip cache hierarchy (cache misses and cache writebacks).

This paper studies the interaction among off-chip bandwidth requirement, cache performance, and overall system performance in multicore systems. The traffic from the chip to the memory is due to the writes which are sent from the last level on-chip cache to the memory, or to the following level external cache, whenever a block is replaced from the cache. We relax some constraints of the well known LRU replacement policy. We call it Modified Least Recently Used (MLRU) policy which essentially reduces the traffic from the chip to the memory system. Simulations using the MLRU policy show considerable writeback decrease by more than 90% with a minimum performance impact. However, it shows also some interesting interaction among cache performance, overall system performance, and off-chip writeback traffic.

1 Introduction

The gap between processor speed and memory speed continues to widen [1]. As the number of cores on chip increases, off-chip bandwidth is becoming a big *wall*. With the increasing number of cores, there is an ever increasing contention for the off-chip bus. The off-chip bandwidth is divided between traffic to memory and traffic from memory. The traffic from memory is mainly due to cache misses. The traffic from chip to memory occurs due to blocks evicted from last level cache to be sent down to the memory.

In this paper we try to decrease traffic from the chip which occurs whenever there is a replacement in the last level on-chip cache(LLC). We try to decrease the bandwidth requirement by modifying the replacement policy of LLC to reduce the number of writes to the memory. The technique we use is the *Modified*-LRU (MLRU). In the new policy, we select the first non-dirty block from LRU to LRU-m as the victim block (0 ¡ m ¡ assoc) to be replaced. If no victim block is found, then the LRU block is used as victim and is written down if dirty. This results in lesser number of write-backs. However we might be overwriting some blocks earlier than in regular replacement scheme which may result in more cache misses. We simulate Splash-2 benchmark to test our replacement policy. The result is a huge reduction in the writeback traffic. We expected some performance loss as we replace non-LRU blocks, but simulations revealed otherwise. Also, the MLRU technique requires very little hardware overhead.

The rest of the paper is organized as follows. The off-chip bandwidth problem is briefly summarized in section 2. We present our technique, the *Modified* LRU in section 3. The experimental setup, methodology and the results are given in section 4. We conclude in section 5.

2 Off-Chip Bandwidth Problem

On-chip speeds have grown consistently over the past several decades creating a corresponding demand for high-bandwidth. The off-chip Bandwidth is shared between traffic from LLC to memory and from memory to LLC. The traffic from memory to LLC occurs whenever there is a cache miss in the LLC and the corresponding block has to be fetched from the memory. These misses if reduced can in turn reduce the number of blocks to be fetched and can thus save traffic coming to the chip. [5] and [6] have shown similar type of work which targets bandwidth optimization by reducing misses. The second type of traffic is that from the LLC to the memory. This is due to the writes sent to the memory whenever a block in LLC is evicted and is dirty. The block chosen to be evicted depends on the replacement policy of LLC. The victim block is then written to the memory.

We aim to decrease the traffic arising from writing

back dirty blocks to the memory when evicted from LLC. We modify the existing technique, LRU, to reduce the number of writebacks to the memory for dirty blocks.

There is a big body of research work that involves modifying the current replacement policies. Although LRU remains the most popular, other schemes like the pseudo-LRU have been proposed to reduce the hardware complexity of the LRU. For example, in Modified Pseudo-LRU (MPLRU) [2] the cache misses are decreased by exploiting the second chance concept of the PLRU scheme.

The more off-chip bandwidth demand increases, the higher the chance of stalls due to contention in buses and memory ports. This reduces the pressure on the NoC due to system stalls, but also affects overall performance. So we are looking for a scheme that reduces the pressure on NoC but also without affecting performance and without increasing the pressure on off-chip interconnect.

3 Modified LRU: Decreasing Writebacks

Whenever a new block is to be placed in the cache, an older blocked is evicted, if the set is full, which is then written down to the memory if dirty. The victim block to be evicted is the least recently used, which is one of the widely used replacement policy together with its many variations. We modified this technique to reduce writebacks to the memory, and observed the effect on performance.

Our technique works as follows. Suppose we have an n way associative cache. The blocks are placed from position 1 to n with the most recently used block at position 1 and the least at position n. In a regular LRU whenever a new block has to be placed in the cache, the block at position n (the LRU block) is evicted. In MLRU, we choose the victim block from LRU to LRU-m. The first non-dirty block from LRU to LRU-n is overwritten with the new block instead of evicting it. This saves bandwidth by retaining the dirty blocks in the cache as long as possible. If however, all the blocks from LRU to LRU-m are dirty, the LRU block is chosen to be evicted like in the regular replacement policy. For example if m=3, the technique would look for the first non dirty block from 3 blocks starting from LRU and overwrite it. We are expecting some performance loss due to victimizing non LRU blocks. The aim of this paper is to study the effect of trading bandwidth for performance.

	Block Size	Associativity	Cache Size
L1	64B	2-way	32KB
L2	64B	8-way	1MB

Table 1: Parameters for the L1 and L2 Cache

Cholesky	24.15%
fft	23.78%
Radix	8.255%
Radiosity	2.234%
Raytrace	-2.25%
fmm	-7.27%
Barnes	-27.1%

Table 2: Percentage of Total Traffic Change for MLRU (Negative numbers show traffic increase)

4 Experimental Evaluation

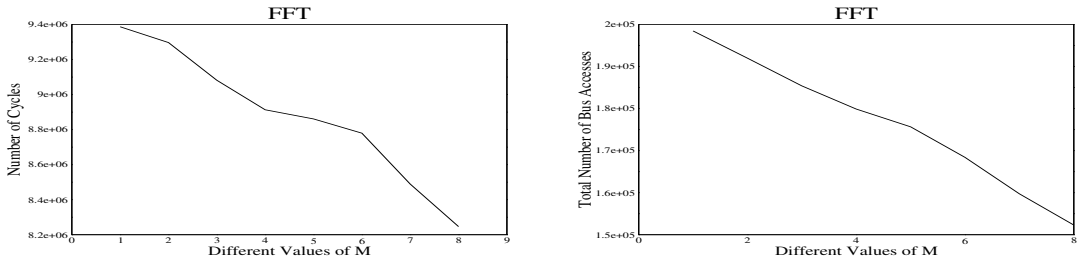
In this section we present the results of our simulations to assess the efficiency of MLRU. First, we present the experimental setup, then we present the results and their implications.

4.1 Experimental Methodology

We use the SESC simulator [4] to model our system and simulate the Splash-2 benchmarks to compare MLRU with the LRU. Out of the Splash-2, four benchmarks have little or no writebacks (`lu, ocean, water-nsquared, water-spatial`) so we do not present their results. We examine the effect of MLRU on the rest of the benchmarks. Each benchmark is simulated for LRU and for MLRU-m where m is varied from 2 till the associativity of the cache (m = 1 is the LRU itself). We are simulating an 8-way set associative cache. We take the number of writebacks, read misses, and the total number of cycles till completion as our bandwidth, cache performance, and system performance measures respectively. It is to be noted that we have used the number of read misses as a measure of cache performance because a write miss is considered a read miss followed by a write hit. Table 1 gives the summary of the cache system. We simulate 4 cores each with a private L1 data cache and instruction cache, and a shared L2. L2 cache is non-inclusive.

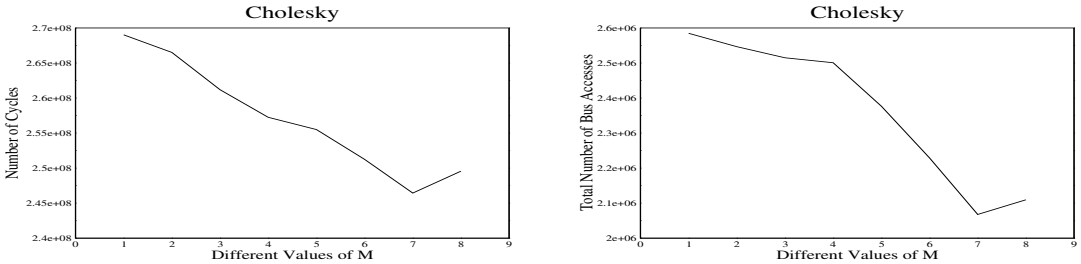
4.2 Experimental Results

In the first set of experiments we analyze the effect of our scheme on the number of writebacks. All of the benchmarks show a decrease in the number of writebacks. This is because we keep dirty blocks in the cache for longer and overwrite them if they are not dirty which reduces the number of writebacks to the mem-



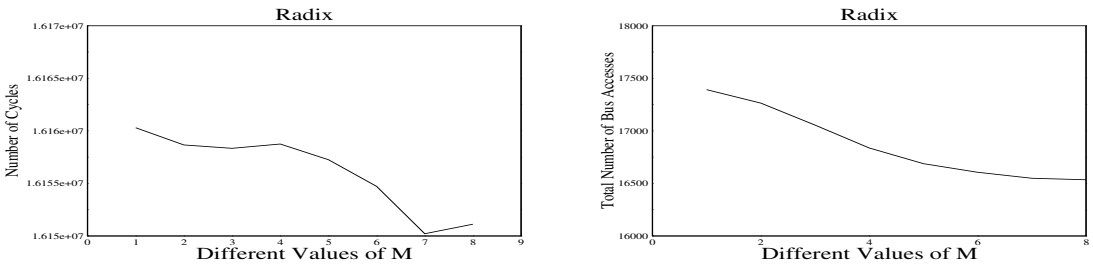
(a) Total Number of cycles for Increasing values of M (b) Total Traffic for Increasing values of M

Figure 1: Comparison of Cycles and Total Traffic with Increasing M for FFT Benchmark



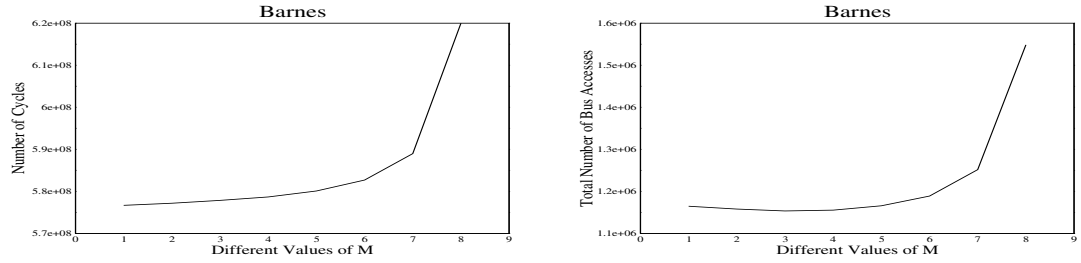
(a) Total Number of cycles for Increasing values of M (b) Total Traffic for Increasing values of M

Figure 2: Comparison of Cycles and Total Traffic with Increasing M for Cholesky Benchmark



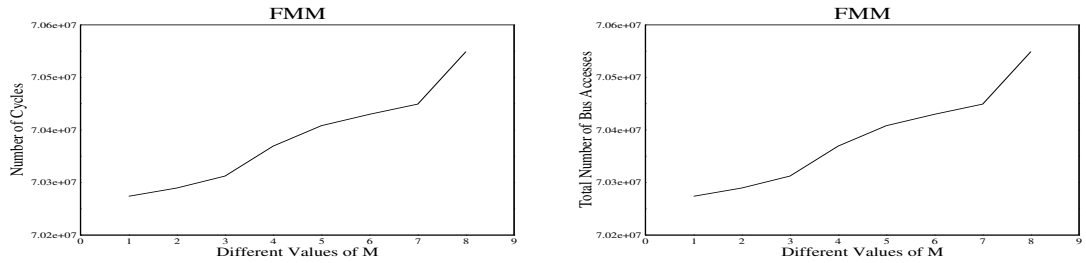
(a) Total Number of cycles for Increasing values of M (b) Total Traffic for Increasing values of M

Figure 3: Comparison of Cycles and Total Traffic with Increasing M for Radix Benchmark



(a) Total Number of cycles for Increasing values of M (b) Total Traffic for Increasing values of M

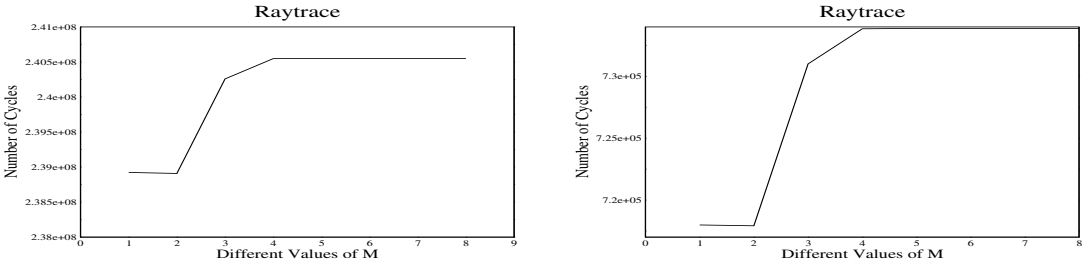
Figure 4: Comparison of Cycles and Total Traffic with Increasing M for Barnes Benchmark



(a) Total Number of cycles for Increasing values of M

(b) Total Traffic for Increasing values of M

Figure 5: Comparison of Cycles and Total Traffic with Increasing M for Fmm Benchmark



(a) Total Number of cycles for Increasing values of M

(b) Total Traffic for Increasing values of M

Figure 6: Comparison of Cycles and Total Traffic with Increasing M for Raytrace Benchmark

Cholesky	36.8%
fft	93.9%
Radix	40.2%
Radiosity	4.73%
Raytrace	100%
fmm	5.9%
Barnes	29.30%

Table 3: Percentage of Writeback Decrease

Cholesky	14%
fft	16%
Radix	2.4%
Radiosity	0%
Raytrace	-2.4%
fmm	-14.8%
Barnes	-56.9%

Table 4: Percentage Decrease in Read Misses (Negative numbers show increase in misses)

Cholesky	7.23%
fft	12.15%
Radix	0%
Radiosity	0.3%
Raytrace	-2.4%
fmm	-0.3%
Barnes	-7.5%

Table 5: Percentage Decrease of Total Number of Cycles (Negative numbers show increase in misses)

ory. The maximum reduction in writebacks is shown by `Raytrace` in which the writebacks are reduced to 0. This means that most of the writebacks for that application were for local variables which are no longer needed or for register spilled. That is, they are dead values. `Cholesky and FFT` show a decrease of 36.8% and 93.9% respectively. Table 3 summarizes the writebacks for all the benchmarks. Results shown in this and the following tables are for M = 8 to show the performance of our technique in its extreme case.

Next we see the effect of our scheme on the number of misses. We expect the misses to increase to some extent because of victimizing non-LRU blocks. This may overwrite some blocks needed by the application, that in the case of non MLRU replacement scheme, would be still in the cache. Hence we expect to see

some increase in misses. However, for most benchmarks the increase in misses is less than 15% except for `barnes` which has an increase in miss of 56.9%. The high increase in misses of `barnes` means that it is LRU friendly, and it has high temporal locality. For `FFT and Cholesky` we even see a decrease in the number of misses by 16% and 14% respectively. This means that LRU is not always the best replacement policy, and it is better sometime to *violate* LRU to gain reduction in off-chip bandwidth. Table 4 summarizes the misses for all the benchmarks.

We measure the total traffic to be the total number of times the bus is accessed. Each time the bus is accessed, a cache block is sent or received. This is the combined sum of the writebacks and the number of misses. The overall effect on the traffic is shown in Table2. The total traffic for `Cholesky and FFT` decreases by 24.15% and 23.78% respectively since both the misses and the writebacks decrease for these benchmarks.

We then see the effect on the overall system performance in terms of total number of cycles for the whole execution. For three benchmarks (`FFT, Cholesky and Radix`) we see an increase in performance since the number of cycles have decreased. This is because we decreased delay caused by bus contention and memory port contention. For the benchmarks that have an increase in misses we expect some performance loss. Only three benchmarks show a decrease in performance out of which two benchmarks have increase in number of cycles less than 1% except for barnes which is 7.5%. **This shows how off-chip bandwidth is a severe limitation**

Figures 1 - 6 show the effect of changing the value on M on the total performance (Number of Cycles) and the total traffic of each benchmark. Although we gain by reducing the number of traffic, we tradeoff some performance loss.

5 Conclusions

Off-chip bandwidth continues to challenge microarchitects. As the number of cores on chip increases, the off-chip traffic will continue to increase. In this paper we show that we can trade some performance loss with bandwidth saving. Our simulations show that the technique offers significant reduction in writebacks of up to 39.3% with a performance loss of no more than 7.5%. In some cases the performance gain of up to 12.15% was observed. **This means that we can trade some cache performance loss with decrease in off-chip bandwidth, which can lead to an overall system performance enhancement due to decrease in off-chip contention on buses and memory.** There are still several open questions that we are currently exploring.

- When is it beneficial to violate LRU policy?
- Can we do this violation dynamically?
- How does it work in a multiprogramming environment?
- How does the trade of bandwidth with performance change as we increase the number cores?

References

[1] Doug Burger, James R. Goodman, and Alain Kägi. Memory bandwidth limitations of future microprocessors. In *ISCA '96: Proceedings of the 23rd annual international symposium on Computer architecture*, pages 78–89, New York, NY, USA, 1996. ACM.

[2] Hassan Ghasemzadeh, Sepideh Sepideh Mazrouee, and Mohammad Reza Kakoee. Modified pseudo lru replacement algorithm. In *ECBS '06: Proceedings of the 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems*, pages 368–376, Washington, DC, USA, 2006. IEEE Computer Society.

[3] Hsien-Hsin S. Lee, Gary S. Tyson, and Matthew K. Farrens. Eager writeback - a technique for improving bandwidth utilization. In *MICRO 33: Proceedings of the 33rd annual ACM/IEEE international symposium on Microarchitecture*, pages 11–21, 2000.

[4] P. M. Ortego and P. Sack. SESC: SuperESCalar Simulator. Dec. 2004.

[5] Thomas F. Wenisch, Stephen Somogyi, Nikolaos Hardavellas, Jangwoo Kim, Anastassia Ailamaki, and Babak Falsafi. Temporal streaming of shared memory. In *ISCA '05: Proceedings of the 32nd annual international symposium on Computer Architecture*, pages 222–233, Washington, DC, USA, 2005. IEEE Computer Society.

[6] Michael Zhang and Krste Asanovic. Victim replication: Maximizing capacity while hiding wire delay in tiled chip multiprocessors. *SIGARCH Comput. Archit. News*, 33(2):336–345, 2005.