WARP: Workload Nature Adaptive Replacement Policy

Balaji S
Final Year (U.G)
Dept. of CSE, SVCE,
Sriperumbudur, Tamilnadu,
India

Gautham Shankar R
Final Year (U.G)
Dept. of CSE, SVCE,
Sriperumbudur, Tamilnadu,
India

Arvind Krishna P
Final Year (U.G)
Dept. of CSE, SVCE,
Sriperumbudur, Tamilnadu,
India

ABSTRACT

In the present universal scenario where the dependence on heterogeneous multi-core processors is tremendous, dealing with algorithms focusing on coherence between shared caches is imperative.

WARP redesigns the replacement policy in the last level cache. In this policy, the shared (clean) lines and the private exclusive line are given the first two priorities dynamically followed by private modified and shared lines. By this method a set of victims are presented over which any replacement policy can be chosen to select a viable victim.

Implementing the same, a significant performance improvement is observed with the last level shared cache. This performance improvement was solely from the policy implemented on the last level cache and not from any other parameters.

General Terms

Computer Architecture, High Performance Computing.

Keywords

Last level, Shared cache, Replacement algorithm, Set dueling.

1. INTRODUCTION

In the current world where real time applications dominate the world of computing, these applications are predominantly multithreaded in nature. A multi-core architecture plays a major role in the performance of these applications. A good replacement policy focuses on reducing the miss-rate thereby improving its overall performance.

1.1 Need for sharing aware algorithm

In an environment where a shared level cache is used, contention for memory from more than one core occurs by which frequent eviction of lines takes place. Care must be taken such that a line evicted might not be used immediately by another core in the next clock cycle. This can be achieved by taking the sharing nature into consideration by which lines which are shared by cores can be kept longer than the lines which are private to a particular core.

All the existing protocols focuses on the multi-programmed workloads where sharing of data is at its minimal.

The following changes are to be considered,

- Take sharing between the threads of the cores into consideration before choosing a line for eviction and
- For certain workloads, private lines may be used predominantly over shared lines, where the concept of set dueling is used.

2. BACKGROUND

2.1 Set Dueling

Set dueling (Moinuddin K. Qureshi et al, 2008) is a policy in which, two contradicting policies are made to compete against each other, by exclusively assigning each policy to a certain number of sets and depending upon the number of misses encountered in these sets, a winner is selected at run time and it is followed in the rest of the sets.

2.2 MESI Protocol:

The MESI protocol is a cache coherency protocol which basically supports the write-back cache. M stands for Modified, E for Exclusive, and S for Shared and I for Invalid.
Therefore, there are three types of lines namely Invalid, Shared and Private. The Shared line can be further classified into Clean and Dirty. The Private line can be subdivided into Exclusive and Modified.

2.2.1 Modified

The data present in the cache is different from the main memory value. This value is termed dirty and is invalid till the cache writes the data to the main memory.

2.2.2 Exclusive

Writing the valid data onto the main memory replacing the dirty block that was present there changes the line to Exclusive. In this case, the cache line is present in the current cache and it is clean, i.e. it’s the same as the data in the main memory. It may be changed to Modified if a new data is written to it.

2.2.3 Shared

When an Exclusive line gets a read request, it may be changed to Shared. It indicates that this cache line may be present in other caches as well and is clean.

2.2.4 Invalid

A line can be changed to Invalid at any time. It indicates that this line is unused (invalid).

2.3 PARSEC Benchmarks

The Princeton Application Repository for Shared-Memory Computers (C.Bienia et al, 2008) is a benchmark suite composed of multithreaded programs. Figure 1 shows the amount of lines which are reused and out of which what fraction of them are shared and private. It is evident that, in normal replacement policies, if sharing is taken into consideration and eviction of lines are done based on the priority, then improvement in the performance is viable but not at its zenith. Hence the notion of both private and shared are considered so as to attain the maximum performance possible. This may happen as a result of the cache lines with a near reuse being retained by the LLC in contrast to others with a distant reuse, when the replacement policy adapts to the sharing nature of the workload.

3. MOTIVATION

WARP’s motivation factor arises from that of CSHARP (Biswabandan Panda and Shankar Balachandran, 2012), namely the sharing and coherence nature of the cache lines. Although CSHARP has been built on the concepts of sharing, not all workloads perform better always, when evicting a shared line over a private line. For a given set of cache lines the reuse nature of the shared blocks are generally more compared to that of the private lines but at times giving priority the other way round may give an additional improvement in the performance.

Thus instead of giving preference to the shared (clean) lines blindly, the concepts of set dueling can be taken into consideration in dynamically choosing between private and shared clean lines, the winner of which is allotted a higher priority over the other.

4. SUGGESTED CHANGES

4.1 The 4 groups:

At a given instance, the blocks from a set from which victims can be chosen can be split into 4 groups: Shared but clean, Private and Exclusive, Private and Modified and other Shared lines. The priority of the groups are also given as same as that of the above order.

4.2 Norms of a replacement policy:

Any cache replacement policy follows the following steps: eviction, insertion and promotion. On a cache miss, a line is “chosen” for eviction. The new line is “inserted” into the set and is assigned an appropriate reuse-register value. On subsequent hits, the line gets “promoted” by reducing the reuse-register value (higher the reuse-register value means distant reference interval).

4.2.1 Eviction:

Initially, determine the cache lines having the highest reuse register values. In case of a tie between two or more cache blocks having the same reuse register values, consider their sharing nature and order them based on the priority assigned to each group, determined dynamically by set dueling and then select a victim.

4.2.2 Insertion:

Keep track of history of that particular cache line, and based on the previous nature of the same cache line, decide on the value to be given to the block which is inserted, as to whether its value lies near to the lower or the higher end of the reuse-register value.

4.2.3 Promotion:

An access to a block in addition to an initial access gives the block an extra probability as to whether it would be accessed in the near future or not. The reuse-register value is assigned a value ’x’ closer to max value if the block remains private, else its assigned a value close to the min value namely ‘zero’ if the tendency of the block shifts from that of private to shared.

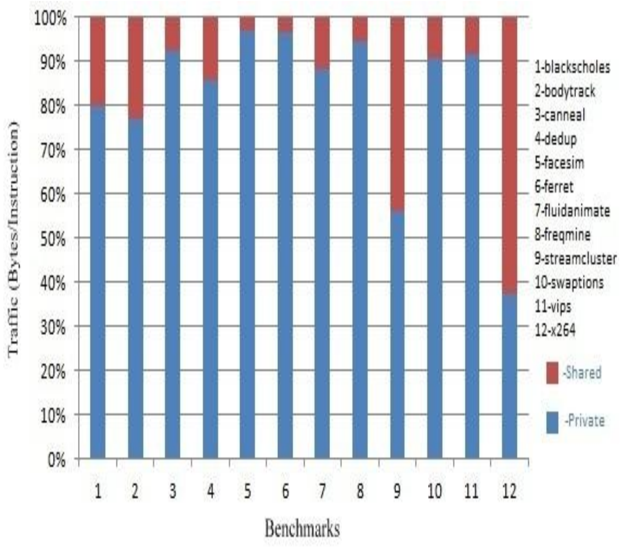

Figure 1: Reuse of a cache line for a 2 core system

5. IMPLEMENTATION

Initially sets are divided so as to perform set dueling. Lowest priority is assigned to shared dirty lines, because their eviction is considered to be costly, owing to the fact that its content should be updated in the main memory, and its copy might exist in higher levels of cache (L1 cache) of more than one core. If that is the case, then the time spent on updating might result in hindering the performance of the processor. The same goes for private modified lines. But these are assigned a higher priority when compared to shared dirty lines mainly because of the fact that their copies can only be present in private cache of a single core, hence the cost incurred on evicting them is much less when compared to shared dirty lines.

Following the concept of set dueling, the highest priority is assigned to shared clean lines in certain sets. Shared clean lines will be evicted as victims in-case there is a tie among cache lines having the same reuse-register values. A global counter variable is maintained. In case of a miss in this particular type of set, the counter variable is incremented by one. Similarly, the highest priority is assigned to private exclusive lines in certain sets, with the only exception that in case of a miss, the global counter variable is decremented. When a miss is encountered in the remaining sets, priorities are assigned based on the global counter variable. If the value is >0 then private exclusive lines are assigned the highest priority and vice versa and then eviction is carried out in the normal way.

First time, when a block is accessed, it is in private mode. The next time, when the same block encounters an access, verification is done as to whether it’s accessed by the same thread. If so, then the block remains in the private mode. If it is accessed by a different thread, other than its original owner, the nature of the block is updated to shared. If a cache line is selected as victim, in case of a miss, after insertion, status is updated to private. During insertion, the sharing nature of cache line is referred, which has to be changed to private after insertion. If it is shared, insert the block with a lesser reuse register value.

Table 1: Parameters of Simulated Machine

	L1 Cache (LRU)	L2 Shared Cache (WARP)
Associativity	2	8
Block Size	64kb	64kb
Latency	1ns	10ns
mshr	10	20
tgts_per_mshr	20	12

During eviction, after selecting a victim, WARP checks if that particular cache line has the maximum reuse register value. If not, the reuse register value of all the blocks present in that set is incremented by one. LRU is used as the replacement policy for levels of cache closer to the processor and WARP is restricted only to the last level of cache, so as to improve instructions per clock cycle.

6. WORKING

The working of WARP can be explained by considering the sequence of events after a cache miss at the last level cache. For instance, when a miss is encountered, the target set is reached and the counter is checked. If the counter is a positive integer, higher priority is given to Private-Exclusive. If it is a negative integer the priority is given to Shared-Clean.

Eviction of a line is done based on a value which is calculated from a mathematical expression involving arguments from the previous stage namely, the reuse-register value and the priority. The block with the highest value is chosen for eviction. In case of a tie, the first-best block is chosen for eviction. Finally, if the evicted block's value is less than the maximum reuse-register value, the reuse-register value of all the blocks' from the target set are incremented by 1.

After eviction, the insertion by the requester thread follows suite, where, in case a block is accessed by more than one thread, it is inserted with the reuse-register value closer to the lowest possible register value. Else if the block is private, insertion is done with a value closer to the maximum allowed register value. The process can now continue as required. The promotion policy is brought to action whenever there are hits.

7. SIMULATOR

Gem5 (N.Binkert et al, 2011), a full system, open source simulator provides the option to run a benchmark either in full system mode or in syscall emulation (SE) mode. Full system mode is used for booting an entire operating system and syscall emulation mode is used for running one or more applications by emulating syscalls. Gem5 also offers the provision to run the benchmarks in different Instruction set architectures like ALPHA, ARM, SPARC, MIPS and x86. As of now, gem5 can be used to simulate SPEC, SPLASH and PARSEC benchmarks. Furthermore, gem5 supports various cache coherence protocols like MESI, MOESI, etc. In spite of having all these features, gem5 is still evolving every day. Gem5, being purely open source, has already attracted millions of users in a short span of three years. Gem5 has an active mailing facility, by which a newcomer can post his queries and the gem5 community will respond to his queries. Further, Gem5 is also used by programmers, to determine the execution time of their program accurately.

8. PERFORMANCE

PARSEC benchmarks are used for characterizing the performance of the algorithm. Metrics such as hit and miss rate are calculated for measuring the performance improvement. For comparison purposes the age old replacement algorithm, namely LRU (Least Recently Used), which has been trusted to be the best replacement algorithm for the various cache levels was used.

At an average LRU showed a miss rate of 0.217 whereas WARP performed even better with a miss rate of 0.216 in a scale of 0 to 1, with 0 meaning no misses and 1 meaning all the accesses leads to a miss.

The results observed for the percentage of hits were even better. In certain benchmarks WARP showed a 15% more increase in the hits compared to LRU which is a significant improvement.

9. FUTURE WORK

Any improvement in performance observed in the field of architecture is always a milestone attained. Even a small improvement in the metrics, however minimal it may (even as small as 0.1%), may lead to a greater overall performance of the system.

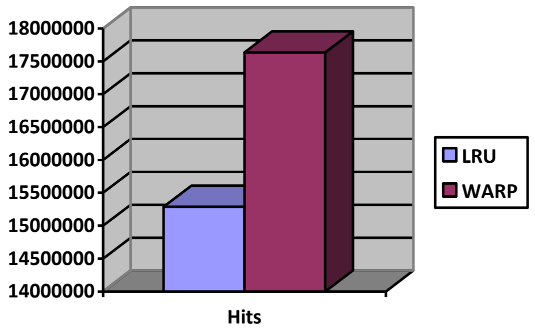

Figure 2: Improvement in Hits when compared to LRU

This performance could be further enhanced by the implementation of Bloom filters along with the policy indicated. Bloom filters are probabilistic data structures used to track the evicted cache lines and it also stores the historical details of the lines such as their owners.
It is believed that implementing Bloom Filters can possibly show an improvement of performance by 10%, in addition to the existing performance.

The performance obtained is the result of implementing WARP over 2 cores, because of which thread awareness was not used. WARP could be implemented in a way such that threads are aware of each other's priorities dynamically in the run time environment. Such implementation could significantly improve the performance metrics in a system of more than 2 cores.

In addition to the above mentioned implementation, a higher Associativity and Block size could significantly show a higher number of hits at the cache and a much lower miss rate, as the notion of Capacity and Conflict misses are substantially reduced.

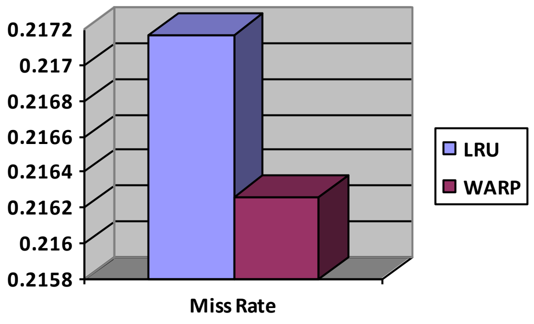

Figure 3: Reduction in Miss-rate observed compared to LRU

10. RELATED WORK

Biswabandan et al. presented CSHARP ,a framework to add Coherence and Sharing awareness to replacement policies that targeted parallel applications at the LLCs. CSHARP assigned priorities to cache lines based on sharing and coherence information and tried to retain high priority lines at LLC as much as possible. The algorithm serves as a framework that works over existing replacement policies.

11. CONCLUSION

WARP serves as a new policy for the LLC in which a line for eviction is chosen dynamically depending on the workload's nature. WARP can be used over other replacement policies which have been proposed already such as the one used: RRIP (A. Jaleel et al, 2010). This not only uses the sharing nature but also takes the cases where private lines are reused more compared to the shared blocks, by which additional performance gain is achieved.

WARP shows significant improvements in the performance namely miss rate and number of hits, a 15% improvement in its hit rate compared to the age old replacement policy LRU; it also showed a significant drop in the miss-rate which is an added advantage.

12. ACKNOWLEDGEMENT

We would like to thank Mr. S. Muthukumar Associate Professor, Computer Science & Engineering Department, SVCE, for his everlasting support and motivation which helped us throughout the paper.

13. REFERENCES

[1] A. Jaleel et al, *"High Performance Cache Replacement using Re-reference Interval Prediction (RRIP)"*, in ISCA 2010, pp.60-71.

[2] Moinuddin K. Qureshi et al, "*Set-Dueling-Controlled Adaptive Insertion For High-Performance Caching*" in MICRO IEEE 2008.

[3] Biswabandan Panda, Shankar Balachandran, "*CSHARP - Coherence and SHaring Awareness Replacement Policies for Parallel Applications*", in *Proceedings of* 24th IEEE International Symposium on Computer Architecture and High Performance Computing, New York, 2012.

[4] N. Binkert et al, *"The gem5 simulator"*, SIGARCH Comput. Archit. News, Aug 2011.

[5] M. Gebhart et al, *"Running PARSEC 2.1 on M5"*, The University of Texas at Austin, Department of Computer Science, Technical Report #TR-09-32, October 2009.

[6] C.Bienia et al, "*The PARSEC Benchmark Suite: Characterization and Architectural Implications*" in PACT 2008.

[7] Major Bhadauria et al, "*Understanding PARSEC Performance on Contemporary CMPs*".

[8] Y. Chen, et al, "*Efficient Shared Cache Management through Sharing-Aware Replacement and Streaming-Aware Insertion Policy*", in IPDPS 2009, pp. 1-8.

[9] A. Jaleel et al, "*Adaptive Insertion Policies for Managing Shared Caches*", in PACT 2008.

[10] L. A. Belady, *"A study of replacement algorithms for virtual storage computers"*, in IBM Systems Journal 1966, pp. 78-101.