Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Engineering 15 (2011) 3536 – 3540

Procedia Engineering

www.elsevier.com/locate/procedia

Advanced in Control Engineering and Information Science

Two Cooperative Partitions for the Lase Level Cache

Baozhong Yu*, Zening Qu, Jianlang Ma, Tianzhou Chen

College of Computer Science, ZheJiang University, Hanzhou, China
{yubaozhong, majl,tzchen}@zju.edu.cn

Abstract

Currently the most widely used replacement policy in the last cache is the LRU algorithm. Popular as it is, LRU performs poorly when it has a memory-intensive workload whose working set is larger than the available cache capacity. The algorithm evicts a line without taking into account its longer history of usage. The result is that some live lines may be replaced by some newly-referenced blocks, though it is possible that the new blocks will never be used again in the future.

Hence, improving the replacement policy is critical to the improvement of cache efficiency and the overall system performance. In this paper we present a technique to retain the live line in the cache while evicting the dead line early by dividing the last level cache into two cooperative partitions (TCP), both of which use the LRU replacement policy. When a live line is evicted from one partition, it will be placed into the other partition. This new technique named TCP, as observed in our experiment, achieved an L2 cache miss rate reduction of 9.8% on average with fifteen benchmarks from the SPEC CPU2000 suite, and resulted in an IPC improvement of 2.4% on average.

Keywords: cache; replacement; efficient; dead block; re-reference;

* Corresponding author: +8613567128813
E-mail address: yubaozhong@zju.edu.cn

1877-7058 © 2011 Published by Elsevier Ltd. Open access under CC BY-NC-ND license.
doi:10.1016/j.proeng.2011.08.662

1. Introduction

The introduction of the last level cache could reduce the access to the main memory and improve the system performance. The traditional LRU replacement policy and its approximations have remained as the d-e-facto standard for replacement policy for on-chip caches over the last several decades. The LRU family has the advantage of good performance for high-locality workload. However, it can lead to some pathological behaviors for memory-intensive workload that has working sets greater than the available cache capacity [6].

In this paper we propose an approach to divide the last level cache into two cooperative partitions (TCP). The two partitions with the same size and both are managed by the LRU replacement policy. We refer to the first partition of the TCP as TCP-default, into which all new lines will be placed. The other subarea is called TCP-accessed, into which all the re-miss lines are inserted.

When a cache block in the TCP-default is evicted by the LRU replacement, there are two possible outcomes: (a) if it is a dead block, it will be evicted from the cache directly and its tag will be filled into the shadow Tags directory; (b) if it is a live block, then it will be inserted into the TCP-accessed and the reuse counter will be set to zero. Note that we call a LRU line as a LRU live line if it will be re-accessed in the near future.

The TCP-accessed plays the role of a buffer for the TCP-default and TCP-default plays the buffer role for TCP-accessed. If a replacement occurs in the TCP-accessed, then the TCP-default works as a buffer for TCP-accessed. As we will see in the evaluation, with the TCP friendly workload in which LRU live lines will be re-referenced soon. The TCP achieves better performance than the baseline LRU, the baseline LRU with double-way and other similar policies.

The rest of this article is organized as follows. The next section will further explain the motivation of the proposed technique. Section 2 will detail the TCP architecture as well as the cooperation between the TCP-default and TCP-accessed. The TCP strategy will be evaluated in section 3. We analyze the experiment results in section 4. And the last section is devoted to the conclusions.

2. Architecture of the Two Cooperative Partitions

4.1 TCP Architecture

The organization of the TCP cache mentioned above is depicted in Figure 1, and the cache accessed process detain is depicted in Figure 2. The new cache architecture is based on the dual cache paradigm. When an L2 cache access occurs due to a miss in L1 data or instruction cache, the TCP-default and TCP-accessed are searched simultaneously. The same operation as traditional L2 when hit in TCP-default or TCP-accessed with the reuse counter updated. If miss happens in both the TCP-default and TCP-accessed, a request will be sent to the main memory controller. The detailed operations will be represented in the next subsection.

TCP-default and TCP-accessed are of the same size and are under the management of LRU replacement policy. They play the same role as conventional L2 cache. Each entry in both caches contains traditional L2 cache line tag information as well as additional information about reuse frequency. Through our experiment observations, one bit reuse counter is enough for most benchmarks. In our study, zero reused line is a dead block, but elsewhere it is a live line. Previous studies such as [5] introduce the use of a 2-bit reuse counter because they need to filter the lines that are reused less than 2 times, but in our study, one bit is enough since most of the lines are zero reused.

TCP-default performs the same role as the conventional L2 cache: a missed line will be filled into TCP-default if it has not been accessed in the last few cycles. Or if the miss line has been visited (it a re-miss line), it will be placed in TCP-accessed.

As described above, TCP-default and TCP-accessed have similar properties. But in the shadow tag directory, there are some differences in term with the missed line, which could be seen in Figure 2.

The introduction of dead block prediction brings a problem to lines with a long reuse distance. However, workload's temporal locality and line's reused times exploited by the cache is a function of both the replacement policy and the size of working set relative to the cache's available size [6].

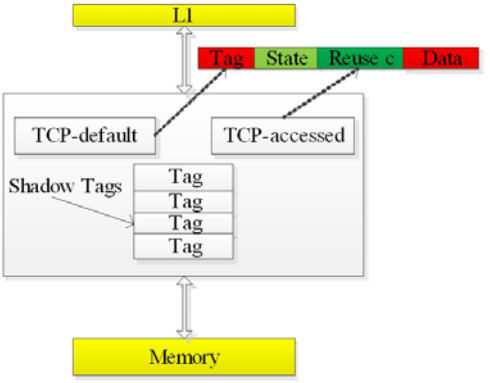

Figure 1: The organization of TCP based on conventional L2 cache

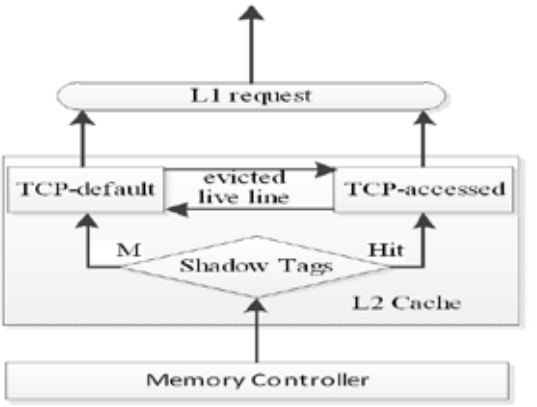

Figure 2: The accessed process of TCP based on conventional L2 cache

4.2 Operations

The implement of TCP is showed in Figure 2. In order to implement a visit request from a higher level cache, a search is performed in TCP-default and TCP-accessed in parallel as the lookup performs in a baseline L2 cache. Since the data in TCP-default and TCP-accessed is mutually exclusive, the lookup will result in either a hit in one of them, or a miss in both of them.

If a higher level cache's request hits in TCP-default or TCP-accessed, we refer to this hit block as H. Then H is handled as in the conventional cache with the reuse counter updated. It will be seem as a live block. When the request misses in both the TCP-default and TCP-accessed, we refer to the line as M. Then it will issue a request to the main memory. Dealing with this missing line, three operations are proposed in our architecture as depicted in Figure 2 and the details are in the following.

Firstly, in order to decide in which partition M should be placed, we should predict whether M is a re-missed block or not. The prediction is obtained by searching the shadow tags. If M is hit in the shadow tag, then it's a re-missed line and should be placed into TCP-accessed, or else it should be inserted into TCP-default. When M is inserted into cache, its reuse counter is assigned to zero.

Secondly, based on the above prediction, an evicted line from TCP-default (V1) or TCP-accessed (V2) should be handled. Since TCP-default and TCP-accessed are under the management of traditional LRU replacement policy, V1 or V2 may be a live block. So we need to examine the evicted line’s reuse info.

If it is a dead block, then the tag will be installed into shadow tag architecture and the data written back to main memory. Otherwise, if it is a live line, it will obtain another chance for re-access from the TCP. We assume the evicted line is V1 (the same to V2), in this case V1 will retain in TCP-accessed and reuse counter will be initialized to zero.

As the prediction shows in shadow tag, TCP-default is filled with missed lines and TCP-accessed is filled with re-miss lines. They even reserve evicted live blocks for each other and then swap out the zero reuse lines early.

In a modern processor, it will cost hundreds of cycles to complete a main memory visiting while a few cycles to finish data access on-chip cache, so the additional latency of the miss can be totally hidden.

3. Simulation Environment

To evaluate our approaches we have used an event-driven simulator SESC [4] with baseline configuration based on a four issue CPU clocked at 4GHz, with two on-chip cache level. The configuration assumes a 45nm technology process. Our experiment method reference the experiment of [3].

We use fifteen representative benchmarks of the SPEC CPU 2000 suites, all of which come from the INT and FP suites. For each benchmark, we execute with the reference input set (ref) during the 5 billion instructions after fast-forward for the first 5 billion instructions.

4. Experimental Results and Analysis

In this section, we will represent and analysis the experiment results. We simulate baseline L2 cache, baseline with double way L2 cache and our TCP. And then present the impact of TCP on L2 missed rate as well as overall processor performance.

4.1 Impact on L2 Miss Rate

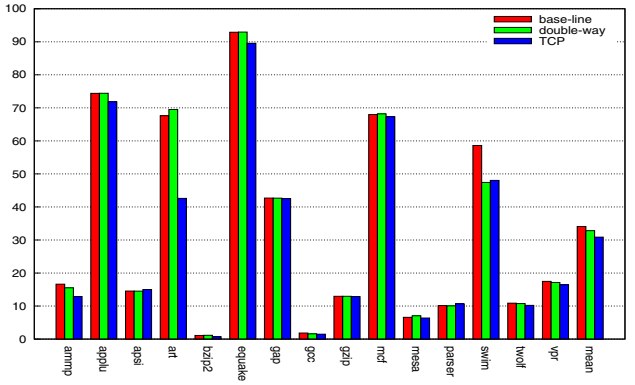

Figure 3: Miss Rate compactions for 1MB 8-way L2 Cache

We compare the miss rate of each benchmark between the baseline, baseline with double-way and TCP in Figure 3. The mean column represents the arithmetic mean of the rate observed in each cache configuration. The bzip2 and gcc benchmark have very low miss rate. In actual, TCP has some reduction for both of them, 33.6% and 19.7% respectively. The one with the most reduction is the art benchmark which reduces about 37%; ammp benchmark reduces about 22.4%; swim about 18%; twolf and vpr both about 6%; applu, mesa, and equake has relatively smaller reduction, about 3.5%, 3.4% and 3.6% respectively; gap, gzip and mcf get negligible reduction; However, for the others, the results are not so good, the miss rate in apsi increases 3.3%; the worst working load is parser which has a increase of 5%. Overall, TCP achieves an average reduction of 9.8% in miss rate in 1 MB baseline L2 cache.

4.2 Impact on System Performance

Figure 4 shows the performance improvement in terms of instruction per cycle (IPC) for each benchmark in L2 cache configuration examined. The figure not only compares the baseline with the TCP, but also with the baseline system where the L2 cache has doubled the associativity. The duplicated way of configuration is tested to depict the difference between associating two sets of K lines in TCP strategy and using sets to 2K lines. The column labelled geomean indicates the geometric mean of the individual IPC improvements seen by each benchmark.

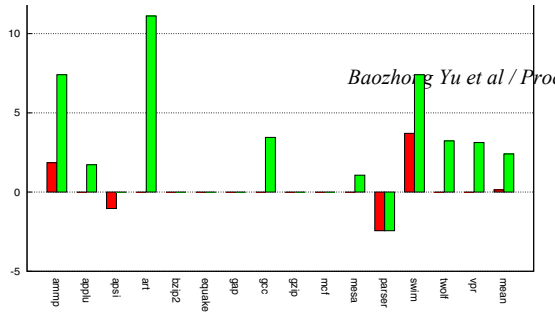

Figure 4: IPC improvement over the baseline

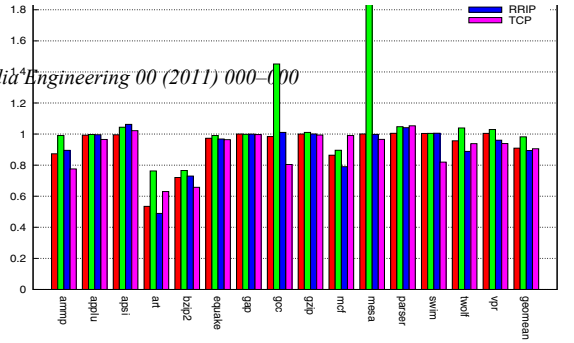

Figure 5: Comparison with recent proposals

Figure 5 compares the miss rates of DIP [6], peLIFO [2], RRIP [1] and TCP in the L2 cache. All of the benchmarks are simulated with 32 dedicated sets and the epsilon is 1/32 for DIP and RRIP and the M=2. The data shown are relative to the miss rate of the baseline configuration. As the figure reveals, TCP works better than other state-of-the-art policies for ammp, applu, bzip2, equake, gcc, mesa, swim and vpr. Only parser has a higher miss rate (about 5.33\%) with TCP than with the baseline, and two benchmarks (art, mcf) have worse performance with TCP than other three policies.

5. Conclusion

In this paper we present the proposal of Dividing Last Level Cache into Two Cooperative Partitions . It is a new design mechanism aiming at improving the efficiency of the Last Level Cache (L2 cache in our study). It splits the cache into two parts. We use re-reference information as the factor of dead block predictor. We applied the reuse counter in our proposal which was represented in prior researches.

Acknowledgements

Our research work is supported by grant from the National Natural Science Foundation of China under Grant No. 61070001 and the Research Foundation of Education Bureau of Zhejiang Province under Grant No. Y200803333 and Y200909683.

References

[1] Aamer Jaleel, Kevin B. Theobald, Simon C. Steely Jr, Joel Emer. High Performance Cache Replacement Using Re-Reference Interval Prediction (RRIP). ISCA‘10.

[2] Chaudhuri, Mainak. Pseudo-LIFO: The Foundation of a New Family of Replacement Policies for Last-level Caches. MICRO’09.

[3] Dyer Rolan, Basilio B.Fraguela, Ramon Doallo. Adaptive Line Placement with the Set Balancing Cache. MICRO’09.

[4] J. Renau et al. SESC simulator. http://sesc.sourceforge.net, 2007.

[5] Lingxiang Xiang, Tianzhou Chen, Qingsong Shi, Wei Hu. Less Reused Filter: Improving L2 Cache Performance via Filtering Less Reused Lines. ICS’09.

[6] Moinuddin K.Qureshi, Aamer Jaleel, Yale N.Patt, Simon C. Steely Jr, Joel Emer. Adaptive Insertion Policies for High Performance Caching. ISCA’07.