Deadblock Aware Adaptive Eviction Policy for Shared Last-Level Cache

Zhaohui Wu[1], Wei Chen[1,2], Bin Li[1*], Zequan Liu[2], Shusheng Long[2]

[1] School of Microelectronics, South China University of Technology, Guangzhou 511442, China
[2] Zhuhai Jieli Tech. Co., LTD, Zhuhai 519060, China
* Email: phlibin@scut.edu.cn

2022 IEEE 6th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC) | 978-1-6654-5864-1/22/$31.00 ©2022 IEEE | DOI: 10.1109/IAEAC54830.2022.9930070

Abstract—**The shared Last-Level Cache (LLC) plays a major role in improving the performance of multicore processor. Multiple applications running concurrently interfere with each other, which results in significant performance degradation for *Cache-friendly* applications. Prior work mainly reduces interference by cache partitioning. This paper proposes a novel cache replacement *Deadblock Aware Adaptive Eviction Policy* (DAAEP) that allocates LLC space reasonably according to the different types of applications. Evaluation on SPEC2006 shows that DAAEP provides a weighted speedup of 7.5% over SRRIP and 0.9% over DAAIP on a 2-core system.**

Keywords—Shared Cache; Eviction Replacement Policy; Deadblock

I. Introduction

Since the 1990s, the importance of multicore processor has been increasing because of the reduced returns of instruction-level parallelism (ILP) and the concern over power factors. Shared Last Level Cache is typically employed by modern multicore processor to mitigate the impact of long memory access latency. However, recent studies[1] [2] have shown that the traditional replacement policy does not perform well when multiple running applications compete among each other in such a shared LLC due to the lack of the effective management for different types of applications. *Cache -friendly* (*Cf*) and *Streaming* (*Str*) are two types of applications that are often used. *Cf* perform better than *Str* in temporal data locality. Prior work[3] has shown that *Cf* are generally sensitive to associativity that is defined as the number of ways in a Cache set, while *Str* are on the contrary. When *Cf* and *Str* run concurrently in a multi-core system, the *Str* takes up a large number of ways with little change in performance. Thus, in shared LLC, allocating the space occupied by the *Str* to the *Cf* will lead to better overall performance.

Hardware cache partitioning[4] provides a method to allocate the LLC space more reasonable. However, excessive hardware overhead has also become a problem. Deadblock is defined as a cache block that fails to get any re-reference from entering LLC to its eviction. *Deadblock Aware Adaptive Insertion Policy* (DAAIP) [5] provides a method to use Deadblock Rate (DR) to identify *Cf* and *Str*. Generally, *Str* have a high DR level, while the DR of *Cf* remains at a low level. Then DAAIP inserts the incoming block with different Re-Reference Interval Prediction (RRPV), which makes the *Str* have a shorter residence time in Cache. But it doesn't improve the eviction policy, resulting in its lack of the ability to allocate the cache space directly. In this paper, the Deadblock Aware Adaptive Eviction Policy (DAAEP) is proposed, which uses the DR to adaptively select the eviction block. In more detail, when the DR of an application is greater than the threshold, the space occupied by itself will be allocated to the application with DR less than threshold.

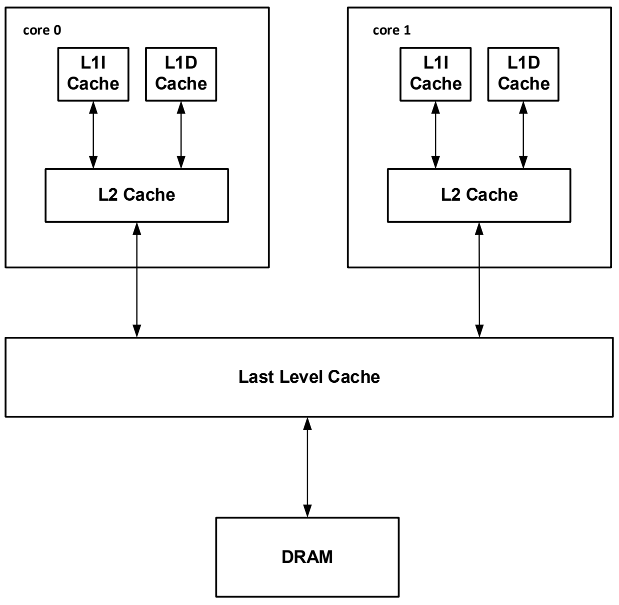
The structure of this paper is as follows. The new ideas are concentrated in Section 2, where the DAAEP is presented. Section 3 illustrates the performance of the proposed policy, after which Section 4 concludes the paper.

II. Design

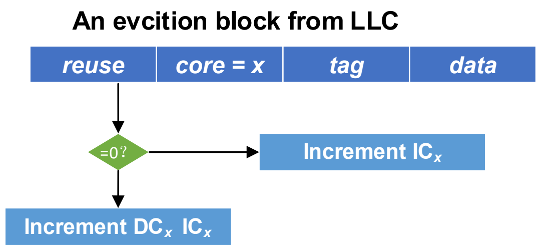
Figure 1 shows the overall structure of DAAEP in a 2-core system. We now describe the design in detail. The detail of the design is described now.

A. Statistics of Deadblocks

Since DAAIP performs well by tracking Deadblocks of per core, we use the same method to collect the statistics of Deadblocks. For each cache block, a *reuse* bit is added to identify if a block was never re-referenced until its eviction, and $\log_2 N$ *core* bits are added to identify the core which brought the block to the cache in a N-core system. The *reuse* bit is set when the block is re-referenced, otherwise it remains at zero. For each core, two 16-bit counters are added to record the total numbers of insertion blocks and Deadblocks. If a core's memory request produces LLC miss, the corresponding Insertedblock Counter (IC) is incremented, and the Deadblock Counter (DC) is also incremented if the eviction block's *reuse* bit is zero. Deadblock Rate is calculated as $DR_x = DC_X/IC_X$ at the end of an application phase, which is defined as 2^{16} cache misses to the LLC (means that when the Insertedblock Counter is



(a) Structure of 2-core system



(b) Statistics of Deadblocks

Figure 1. Structure of DAAEP

saturated). After that the IC and DC are both halved to retain the past behavior and record the recent information.

B. Adaptive Eviction Policy

Figure 2 shows the relationship between two different types of applications on the associativity. The *bzip2*, *Cf*, is sensitive to the number of ways, while the *lbm*, *Str*, is on the contrary. From the perspective of the commonly-used Least-Recently Used (LRU) replacement policy, when *Cf* and *Str* run concurrently in the multicore system, the *Str* may takes up a large number of ways with little gains, which in turn reduces the performance of the *Cf*. Hence, allocating the space occupied by the *Str* to the *Cf* will bring an overall performance improvement. In our work, we change the Eviction Policy on the basis of SRRIP[6].

Eviction Policy: Take the 2-core system as an example, in which application A and B are running concurrently in the system respectively. When the request of application A is LLC miss and the corresponding set is full, the eviction block is selected as follows:

a) If the DR of A is greater than the threshold, select the block whose *core* bits = A and RRPV=3. If not selected, choose the block whose *core* bits = B and RRPV = 3. If still not selected, increment the RRPV of all blocks in the set, and repeat the above procedures.

b) If the DR of A is less than the threshold, select the block with *core* bits = B and RRPV = 3. If not selected, choose the block whose *core* bits = A and RRPV = 3. If still not selected, increment the RRPV of all blocks in the set, and repeat the above procedures. This step enables the *Cf* to use the space occupied by the *Str*.

c) If the DR of A and B both less than the threshold, select the block whose RRPV=3, no matter its *core* bits is A or B.

In this paper the DR threshold is set to 0.9.

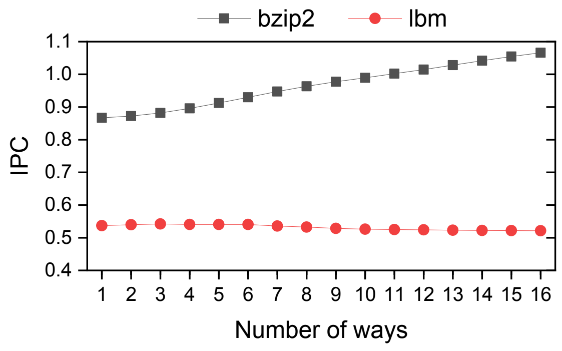


Figure 2. Applications performance change with different LCC associativity

Table 1. Simulation parameters

Core	2 cores, 4GHz, out-of-order, 6-wide
L1I	32KB, 8-way, 4-cycle latency, LRU
L1D	48KB, 12-way, 5-cycle latency, LRU
L2	512KB, 8-way, 10-cycle latency, LRU
LLC	2MB/core, 16-way, 20-cycle latency, DAAEP
DRAM	4GB, 1600 MT/sec

III. Experimental Analysis

A. Experimental Methodology

We evaluate the new policy using ChampSim[7], a trace- based simulator that models out-of-order cores. All simulated systems use a three-level cache hierarchy, which is based on the Intel Core i7 system. The L1 and L2 caches are private to each core and use LRU replacement. The LLC is shared across all cores and uses the proposed policy. The detailed simulation parameters are shown in Table 1. We collect SimPoint[8] traces from 14 memory intensive SPEC CPU2006 applications with behavior of *Cf* or *Str,* as shown in table 2. For each application, we use the trace with highest weight. Here we use 27 different combinations from our applications set which are either *Cf-Str* or *Cf-Cf* to evaluate DAAEP on a 2-core system. For all experiments, each trace is warmed up with 50M instructions and statistics are collected over the next 200M instructions. If the end of trace is reached, our model restarts the application automatically.

For multi-core systems, two metrics are used to evaluate the shared LLC performance: (1) Individual Instruction Per Cycle (IPC), (2) Weighted Speedup (WS). All the metrics are normalized to SRRIP[6] for each combination. The Weighted Speedup is calculated as follows: $WS = \sum_0^{N-1} \frac{IPC_i^{shared}}{IPC_i^{single}}$; IPC_i^{shared} is the IPC of core i when it works with other N-1 cores in a N-core system. IPC_i^{single} is the IPC of core i when no other applications are running in a N-core system.

Table 2. Application's behavior

Application	Type	Application	Type
bzip2	*Cf*	*soplex*	*Cf*
cactusADM	*Cf*	*sphnix3*	*Cf*
Gcc	*Cf*	*h264ref*	*Cf*
gemsFDTD	*Cf*	*zeusmp*	*Cf*
gmmer	*Cf*	*libq*	*Str*
leslie3d	*Cf*	*lbm*	*Str*
perl	*Cf*	*milc*	*Str*

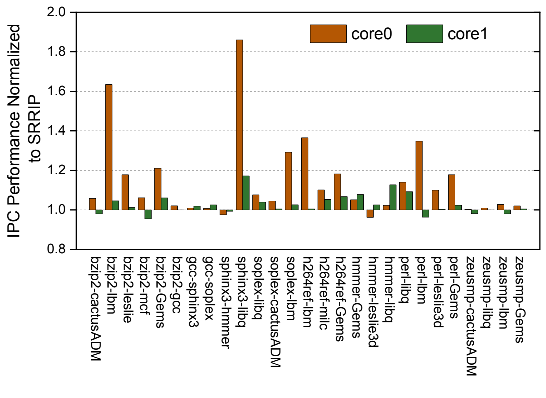


Figure 3. Individual benchmarks performance normalized to SRRIP

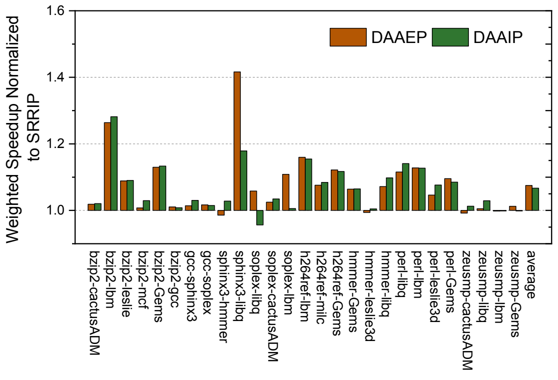


Figure 4. Overall performance normalized to SRRIP

B. Performance

Figure 3 illustrates the individual IPC speedup. All results are normalized to SRRIP. As shown in the figure, for all *Cf-Str* mixes, like *bzip2-lbm, bzip2-milc, sphinx3-lbm*, both *Cf* and *Str* improve the performance. Especially *sphinx3-libq,* outperforms SRRIP in achieving a performance improvement of 84% and 17% respectively. This benefit due to the reasonable allocation of space occupied by *Str* in LLC. For *Cf-Cf* mixes, some of them show the increase of performance, while some show the opposite. Combinations like *bzip2-leslie*, we see the both performance increase. This is because the behavior of some *Cf* varies in different phase, and DAAEP adapts to the change of phase, which makes the performance of both applications improve. Figure 4 shows that the performance improvement of the selected 27 workloads is 7.5% under DAAEP and 6.6% under DAAIP. Obviously, the method proposed in this paper effectively improves the overall system performance.

IV. Summary

This paper proposes DAAEP to address the weakness of shared LLC management in multicore system, which utilizes the information of Deadblock Rate to adaptively allocate the space occupied by the *non-Cf* to the *Cf.* DAAEP performs well for the combination of *Cf-Str* and stable for most combination of *Cf-Cf*, which improves the WS by 7.5% and 0.9% compared with SRRIP and DAAIP respectively.

Acknowledgments

This work was supported by the Zhuhai Industry University Research Cooperation Project (ZH22017001200154PWC)

REFERENCES

[1] Wanli Liu and D. Yeung, "Using Aggressor Thread Information to Improve Shared Cache Management for CMPs," in *2009 18th International Conference*

on Parallel Architectures and Compilation Techniques, Raleigh, NC, Sep. 2009, pp. 372–383. doi: 10.1109/PACT.2009.13.

[2] K. K. Dutta, P. N. Tanksale, and S. Das, "A Fairness Conscious Cache Replacement Policy for Last Level Cache," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Grenoble, France, Feb. 2021, pp. 695–700. doi: 10.23919/DATE51398.2021.9474096.

[3] P. Lathigara, S. Balachandran, and V. Singh, "Application behavior aware re-reference interval prediction for shared LLC," in *2015 33rd IEEE International Conference on Computer Design (ICCD)*, New York City, NY, USA, Oct. 2015, pp. 172–179. doi: 10.1109/ICCD.2015.7357099.

[4] V. Selfa, J. Sahuquillo, S. Petit, and M. E. Gomez, "A Hardware Approach to Fairly Balance the Inter-Thread Interference in Shared Caches," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 11, pp. 3021–3032, Nov. 2017, doi: 10.1109/TPDS.2017.2713778.

[5] Newton, S. K. Mahto, S. Pai, and V. Singh, "DAAIP: Deadblock Aware Adaptive Insertion Policy for High Performance Caching," in *2017 IEEE International Conference on Computer Design (ICCD)*, Boston, MA, Nov. 2017, pp. 345–352. doi: 10.1109/ICCD.2017.60.

[6] A. Jaleel, K. B. Theobald, S. C. Steely, and J. Emer, "High performance cache replacement using re-reference interval prediction (RRIP)," in ACM SIGARCH Computer Architecture News, vol. 38, no. 3. ACM, 2010, pp. 60–71.

[7] "'ChampSim repository.' in https://github.com/ChampSim/ChampSim."

[8] E. Perelman, G. Hamerly, M. V. Biesbrouck, T. Sherwood, and B. Calder, "Using SimPoint for Accurate and Efficient Simulation," SIGMETRICS Perform. Eval. Rev., vol. 31, no. 1, pp. 318–319, 2003.