A Reduced Overhead Replacement Policy for Chip Multiprocessors having Victim Retention

Shirshendu Das, Dhantu Buragohain and Hemangee K. Kapoor
Dept. of Computer Science and Engineering
Indian Institute of Technology Guwahati,
Guwahtai, Assam, India - 781039
email: (shirshendu, dhantu.buragohain, hemangee)@iitg.ernet.in

Abstract—**Due to the non-uniform distribution of the memory accesses for today's applications some sets of the cache are heavily used while some other sets remain underutilized. CMP-VR is an approach to dynamically increase the associativity of heavily used sets without increasing the cache size. It achieves this by reserving certain number of ways in each set to be shared with other sets and the remaining are private to the set. These shared ways from all sets form common reserve storage, while the private ways form the normal storage. In both the partitions it uses LRU replacement policy. This paper presents an optimization on CMP-VR by removing the LRU policy from the normal storage of the set. A victim from this normal storage can reside in the reserved/shared area and will get evicted from here using the LRU policy. Thus our optimization does not hamper cache performance. At the same time it helps to remove the complexity of implementing true LRU. Storage analysis shows 7-18% reduction in the replacement cost. CPI and miss rate also improve by 4% and 16% respectively for a 4MB 8 way associative LLC.**

Keywords-component; LRU, Pseudo-LRU, Tiled CMP, NUCA, Random-LRU

I. INTRODUCTION

The main component for building today's computer systems is Chip Multiprocessors (CMPs) where multiple CPUs (cores) are placed on the same chip [1]. On-chip caches have multiple levels. In this paper we concentrate on the last level cache (LLC). With increasing associativity of LLCs, implementing effective replacement policy plays a major role in improving the performance of CMP [2]. The best policy for replacement is to always discard the cache blocks which will not be needed for the longest time in the future. This optimal policy is called Belady's optimal algorithm [3]. But since it is generally impossible to predict how far in future the information will be needed, the policy is not implementable in practice. Due to this reason other replacement policies like least recently used (LRU) and pseudo-LRU are mostly used.

In case of highly associative LLC's, true LRU impose additional complexity and area overheads. There are some other interesting proposals for cache replacement policies (discussed in Section II); but most of them have complex mechanism and hardware overhead for maintaining aging bits, counters etc. LRU performs better in many applications having higher temporal locality [1]. Hence, instead of completely removing the concept of LRU, researchers proposed innovative techniques to improve the performance of LRU based policies [4], [5], [6], [7].

In LRU a cache block can remain long time in the cache even after its last use. This is because it takes a long time to make the block as LRU block. Such types of blocks are called dead blocks [8]. Also there can be some blocks in L2 which may never be reused in future; such blocks are immediately dead after loading into L2 cache. These blocks are called never-reused blocks [8]. Dead and never-reused blocks are an unnecessary burden of LRU policy and degrade performance.

Though the size of LLC available in CMPs is increasing, an efficient utilization of it remains an issue. The non-uniform memory access distribution causes some sets to be used heavily while other remains underutilized. To overcome such problem the associativity of each individual set must be allowed to increase/decrease dynamically based on the requirement of the executing application [9]. Dynamic management of associativity allows a heavily used set to increase its associativity by using the so called "idle" ways of the underutilized sets. It improves performance without changing the actual size of the cache [9]. But implementing such dynamic associativity management for a cache with optimized power and area consumption is a major challenge.

In an earlier work we proposed a technique called CMP-VR [10] to dynamically manage the associativity of LLC sets. It divides the ways of each set into two storage groups: normal storage (NT) and reserve storage (RT). NT behaves same as conventional cache while RT takes 25% to 50% ways from each set and the entire RT can be used by any set in the cache. A heavily used set can use the RT section of other underutilized sets to dynamically increase its associativity. The RT behaves as a fully associative cache and a separate fully associative tag-array (TGS) is required to manage it. Instead of removing the conventional one-to-one mapping from the entire cache (as done in [9], [11]) CMP-VR removes it only from RT. Also no forward/backward pointers have been used which results in much lesser storage overhead than V-way. CMP-VR gives 14% performance improvement over conventional cache.

A major challenge in CMP-VR is to reduce the hardware cost of TGS. Though reserving 50% ways for RT gives better improvements, it consumes much higher energy and power than reserving 25%. This is because of the larger size of TGS. In CMP-VR, NT section of each set has its own LRU stack and

all the lines in RT section uses a common global LRU stack. In this paper we propose a mechanism to reduce the hardware overhead of the replacement policy used in CMP-VR. We used the concept of random-LRU [12] in CMP-VR and applies random replacement policy for the NT section of all the sets. The LRU policy is now only maintained in the RT section. This reduces the hardware overhead of the replacement policy used in CMP-VR by 9% and hence the additional cost of larger TGS can be partially compensated. We call the newly proposed CMP-VR as RCMP-VR.

The rest of the paper is organized as follows. The next section presents the related works in CMP cache architectures and replacement policies. Section III describes our proposed cache replacement policy for CMP-VR. Section IV covers performance evaluation using full-system simulation. Finally Section V concludes the paper.

II. RELATED WORK

Recent chip multiprocessors (CMPs) consist of multiple tiles connected with some low-radix (2D-mesh) network of interconnects [1]. Each tile has its own processor, private L1 cache and either a private or a public L2 cache [13], [14],[15].The L2 cache is divided into multiple banks and each tile contains one L2-bank.

The non-uniform distribution of memory access is a major cause for higher conflict misses [9]. A conventional set-associative cache cannot adjust its associativity dynamically because of the static one-to-one mapping between tag entries and data lines. V-way [9] and Z-cache [11] both can manage the associativity of each set dynamically. But to support dynamic associativity management, they completely removed the conventional static one-to-one tag-data mappings. CMP-VR[10] on the other hand has partially removed the traditional one-to-one mapping for implementing such dynamic associativity.

Replacement policies have been well-studied in past [16],[2]. But emergence of larger sized LLCs in CMP, motivated researchers for more innovation in this field. It has been generally believed that some version of LRU based policy performs better than other replacement policies [1]. But multicore and hierarchical cache organizations affect the performance as well as the cost of LRU policies. The pros and cons of both local and global replacement policies are discussed in [17].

Cache replacement policy is a major area of research and there are many more innovative ideas already been proposed. Some recent papers in this area include [6], [18], [19], [20]. Most of the LRU based policies [2], [7], [8], [17], [16] perform better than LRU but most of them are more performance oriented than hardware cost. Random-LRU [12] is an alternative of such replacement policies with much lesser hardware requirements. Most of above proposals implemented their replacement policy for the entire set but in random-LRU the LRU policy is not implemented for the entire set and random replacement policy having no additional hardware requirements, is used for some number of ways (min. 50% from each set).

III. PROPOSED ARCHITECTURE

A. CMP-VR

In an earlier work (as discussed in section I) we proposed a tag-array based technique (CMP-VR [10]) for dynamic associativity management. In CMP-VR a set can access or use the reserve storage (ways) of any other set in the cache which means that any RT location can store blocks from any set and to search a block in RT the entire RT has to be searched. Searching the entire RT directly from the cache is an expensive operation, as it has to search all the sets into the cache (RT sections have ways from each set). An additional tag-array (TGS) for RT has been used to overcome the problem. Each RT location (L) has a corresponding entry in TGS which contains the tag address of the block currently residing in L. An example of CMP-VR together with TGS is given in figure 1.

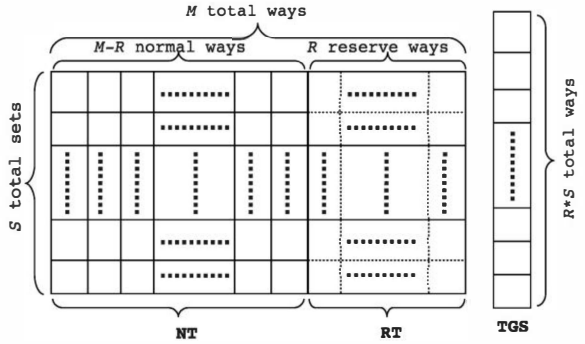


Fig. 1. CMP-VR: way distribution and tag array.

If a block is not present in NT then the block may be available in RT. In CMP-VR, tag matching for a block in NT and TGS (not in RT) is done simultaneously. If the tag is found in NT then it is a direct hit and if the tag matches in TGS then it is an indirect hit. In case of indirect hit the block is in RT and needs to be moved into NT. Moving the block from RT to NT is easy provided NT has a free (invalid) way, otherwise it has to swap the block with the LRU block of NT.

During replacement, instead of removing a victim block (LRU) completely from the cache, CMP-VR moves it to RT. Moving a victim block (V) to RT is easy if RT has a free (invalid) location, otherwise it needs to replace the LRU block of RT with V . Note that, whenever we refer to NT, we only consider the dedicated set on which the block is mapped into. Whereas in the case of RT we consider the entire RT. Also, CMP-VR do not search RT directly, instead it searches the TGS which has a dedicated location for each corresponding RT location. CMP-VR is only proposed for LLC and it does not change the architectures of the upper level caches.

B. RCMP-VR

In CMP-VR, each set uses its own LRU policy for the NT and all the sets share a common global LRU policy for the RT. Each block evicted from NT is moved to RT and returned back to NT whenever it is requested again (indirect hit) in future. Hence to evict a block completely from the cache, first it has to become LRU in NT and then eventually become LRU in RT. This motivated us to remove the LRU policy from NT

and instead select the victim block randomly. Since the randomly selected victim block from NT is not evicted from the cache, but is moved to the MRU position of RT, the applications having higher temporal locality will not be affected by this random selection. If the randomly selected block is a highly requested block, then during its next access it will be found in RT (indirect hit) and moved back to NT. Hence, random replacement policy for NT does not hamper block availability in comparison to LRU and at the same time it can reduce the replacement cost of CMP-VR. Another major challenge in CMP-VR is to reduce the hardware overhead of TGS when the number of ways in RT increases; this being the reason why it has been advised to use only 25% of ways from each set as RT even when reserving 50% gives better performance. Our claim is that the higher hardware cost of TGS in CMP-VR can be compensated (to some extent) by reducing the cost of the replacement policy of the NT.

Similar to CMP-VR, in RCMP-VR all the cache-hits within the NT are treated as in regular CMPs, without any changes. The modifications are only required for the misses in NT. The necessary changes are discussed below:

On NT miss (block not in NT): Search the TGS. Upon a tag match in TGS identify the corresponding block in RT and move it to NT. If tag match fails in TGS, it is a cache-miss.
Move a block B from RT to NT: If free space (way) is available in the desired set of NT, then move B in that free space. Otherwise, swap B with a randomly selected block in the corresponding set of (NT).
On Cache Miss: Bring block from main memory and store it in the desired set of NT.
Store a newly arrived block B into NT: If free space (way) is available in the desired set of NT then place B into the free space. Otherwise make space in NT by evicting a random block. This evicted block moves to RT.
Store block V (evicted from NT) into RT : Search the TGS. If space available in RT then move V into the free space of RT. Otherwise select a victim block (V) from RT (based on LRU replacement policy) and replace it with V . The new victim V is evicted from the cache bank and written back to the main memory (if required). Update TGS entry by overwriting the tag of V by the tag of V.

IV. EXPERIMENTAL EVALUATION

A. Tiled Chip Multiprocessor (TLA)

We used 16 core Tiled CMP architecture (TLA) [1] as the baseline design. Both CMP-VR and RCMP-VR has been implemented on top of it. In TLA, each tile has a processor, a private L1-cache and a shared L2-cache. The tiles (or processor nodes) are connected to each other over a 2D mesh popularly known as network-on-chip (NoC). All the experimental results shown in this section are for the entire LLC (L2), combining the results of all the banks together.

B. Experimental Setup

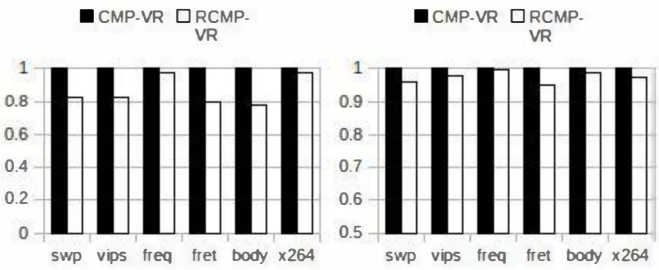
Both RCMP-VR and CMP-VR are implemented separately on each tile of the TLA, i.e., each tile has its own NT, RT and TGS. In order to evaluate, simulations are performed by running benchmarks on a multi-core simulator GEMS [21] with the help of SIMICS [22], a full-system functional simulator. Six multi-threaded benchmarks from PARSEC [23] benchmark suite has been used for simulation. The benchmarks are: swaptions(swap), bodytrack(body), ferret(fert), freqmine(freq), vips and x264(x64).

TABLE I
SYSTEM PARAMETERS

Component	Parameters
No. of tiles	16
Processor	UltraSPARCIII
L1 I/D cache	64KB, 4-way
L2 cache bank	256KB, 4-way/8-way
Memory Bank	1GB, 4KB/page
CMP-VR/RCMP-VR: reserve ways per set	50%

A TLA having 16 tiles and the L2 cache size of 4MB is used for the experiments. The size of each L2-bank is 4MB/16=256KB. Number of reserve ways per set is considered as 50% for all the experiments. Performance comparison of RCMP-VR and CMP-VR has been given for both 4 way and 8 way associative L2-banks. The detailed configurations are given in table I. Random replacement policy in NT helps to remove the dead blocks and never-reused blocks from NT early. This improves the miss rate and CPI of RCMP-VR.

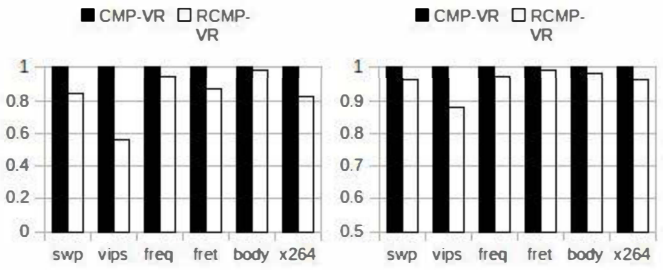
Figure 2 shows the performance comparison of RCMP-VR with CMP-VR for 4-way associative L2-banks. Both graphs in the figure show the results normalized to the corresponding values of the baseline design. Figure 2(a) shows that RCMP-VR gets 2.22% to 22.48% reduction in miss-rate with an average of 16%. Figure 2(b) shows the performance comparison in terms of CPI. It shows that in case of RCMP-VR, CPI improves by 0.05% to 4.8% with an average improvement of 2.47%.



(a) Miss Rate (b) Cycles per instruction.

Fig.2. Normalized performance comparison of RCMP-VR with CMP-VR. Total cache size 4MB, cache bank size=256KB, associativity=4, reserve ways per set=2.

Figure 3 shows the performance comparison of RCMP-VR with CMP-VR for 8-way associative L2-banks. Figure 3(a) shows that RCMP-VR gets 1.93% to 43.57% reduction in miss-rate with an average of 15.84%. Figure 3(b) shows the performance comparison in terms of CPI. It shows that in case of RCMP-VR, CPI improves by 0.52% to 12.20% with an average improvement of 4%. Table II lists the various values.



(a) Miss Rate (b) Cycles per instruction.

Fig.3. Normalized performance comparison of RCMP-VR with CMP-VR. Total cache size 4MB, cache bank size=256KB, associativity=8, reserve ways per set=4.

Comparison with TLA: CMP-VR is built on top of TLA and gives 14% and 45.5% improvements on CPI and miss rate respectively as compared to the baseline. The main focus of RCMP-VR is to reduce the cost of replacement policy used in CMP-VR. So we can conclude that RCMP-VR having 4MB LLC (L2) with 16 4-way set associative L2-banks improves performance by (14+2.47) =16.47% and (45.5+16) =61.5% on CPI and miss rate respectively when compared with same size TLA.

Storage analysis of replacement policy: As explained in [24] each cache block needs to be augmented with some additional bits to specify the position of the block in the LRU stack. Hence the total number of additional bits required for each block depends on the associativity of the cache. If A is the associativity of the cache then the additional bits required for each cache line is $\log_2(A)$. Hence the total additional bits required is $S*A*\log_2(A)$, where S is the number of sets in the cache. For example, in a 4-way set associative cache, 2 bits are required to specify the stack positions (e.g. 11 for MRU, 00 for LRU etc). When a new block arrives, the cache controller chooses a victim by searching the LRU stack having stack position bit value as 00. On each cache access corresponding bits needs to be changed for managing the LRU policy. So true LRU increases design complexity as well as leads to a high area and power overhead of replacement policy. In CMP-VR each set maintains its own LRU stack for the ways belonging to NT and all the sets maintain a common global LRU stack for the ways in RT. In a TLA, the additional storage bits required by each LLC bank to implement LRU policy for its NT section (NTBits) can be calculated as:

$$\mathbf{NTBits = S * (A - R) * \log_2 (A - R).}$$

Where *S* is the number of sets in the bank, A is the set associativity and *R* is the number of ways reserved for RT per set. Since each bank maintains only one LRU stack for the entire RT, the additional bits required for RT (RTBits) can be calculated as:

$$\mathbf{RTBits = S * R * \log_2(S * R)}$$

TABLE II

PERFORMANCE IMPROVEMENT (IN %) CHART OF RCMP-VR OVER CMP-VR (CACHE SIZE=4MB, RESERVE WAYS PER SET=50%). 4W MEANS 4 WAY ASSOCIATIVE L2 BANK AND 8W MEANS 8-Way ASSOCIATIVE L2-BANK

-	CPI		Miss Rate	
	4W	***8W***	***4W***	***8W***
swp	4.21	3.69	***17.40***	***15.30***
vips	2.02	12.2	***17.86***	***43.57***
freq	0.05	2.54	***2.22***	***5.71***
fret	4.82	0.52	***20.38***	***12.68***
body	1.23	1.51	***22.48***	***1.93***
x264	2.88	3.34	***2.37***	***17.5***

Now, the total number of additional bits required for implementing replacement policy in CMP-VR (only for LLC) can be calculated as:

$$\mathbf{Total_{CMP\text{-}VR} = B * (NTBits + RTBits)}$$

Where, *B* is the total number of banks in the TLA. Since RCMP-VR has no LRU stack for NT, the total additional storage cost of RCMP-VR (only for LLC) is:

$$\mathbf{Total_{RCMP\text{-}VR} = B * RTBits}$$

For example, considering a 4MB TLA with 16 banks where each bank is 4-way associative and 2 ways from every set are reserved for RT. There are 1024 sets in one bank. Now the storage bits required can be calculated as:

NTBits = 1024 * 2 * 1 = 2048 bits.
RTBits = 1024 * 2 * 11 = 22528 bits.
$\mathbf{Total_{CMP\text{-}VR}}$ = 16 * (2048 + 22528) = 49152 Bytes.
$\mathbf{Total_{RCMP\text{-}VR}}$ = 16 * 22528 = 45056 Bytes.

Table III shows the improvement of RCMP-VR over CMP-VR in terms of storage consumption for replacement policy for various size caches. It can be observed from the table that, as the associativity increase the percentage savings in terms of replacement cost also increases. In particular for a 16MB cache we obtain 7% reduction for 4-way associativity, whereas the reduction increases to 18.75% for 16-way associativity.

Area and power analysis: Since RCMP-VR uses less number of LRU stacks the replacement policy of RCMP-VR consumes less area and power as compared to CMP-VR.

IV. CONCLUSION

CMP-VR can dynamically increase the associativity of heavily used sets without increasing the cache size. It divides the last level cache (LLC) into two sections: normal storage (NT) and reserve storage (RT). From each set, 25 to 50% ways are reserved for RT and the remaining ways belongs to NT. It allows a heavily used set to use the RT section of other underutilized sets and hence increase the associativity of the set dynamically without increasing the cache size. CMP-VR used LRU replacement policy for each set separately in NT but uses a common global LRU policy for the entire RT.

TABLE III

COMPARISON OF ADDITIONAL STORAGE REQUIRED FOR IMPLEMENTING REPLACEMENT POLICY OF CMP-VR AND RCMP-VR.THE NUMBER OF WAYS RESERVED FROM EACH SET IS TAKEN AS 50%.

Size	Assoc	Banks	CMP-VR (bytes)	RCMP-VR (bytes)	Improvement (%)
4MB	4	16	49152	45056	8.38
4MB	8	16	53248	45056	15.38
8MB	4	16	106496	98304	7.69
8MB	8	16	114688	98304	14.29
16MB	4	16	229376	212992	7.14
16MB	8	16	245688	212992	13.31
16MB	16	16	262144	212992	18.75

As the number of cores and associativity of the last level cache (LLC) on Chip Multi-processor increases, the role of replacement policies becomes more vital. True LRU imposes additional complexity and area overheads when implemented on highly associative LLCs. Hence, we propose a less expensive replacement policy for CMP-VR. The LRU policy from the NT section is replaced with random replacement policy. Such combination helps to reduce the hardware cost as well as for the early removal of dead blocks from the cache, resulting in performance improvement. Replacement cost reduces by 7-18%. Comparing with CMP-VR, CPI and miss rate improves by 4% and 16% respectively for a 4MB 8-way associative LLC. On the other hand, the improvement is 16.47% and 61.5% in terms of CPI and miss rate respectively when compared with shared tile based CMP or TLA.

ACKNOWLEDGMENT

We wish to acknowledge Department of Electronics & Information Technology (Deity), Ministry of Communications & IT, Government of India, for the financial assistance provided for this work.

REFERENCES

[1] Balasubramonian, R., Jouppi, N.P., Muralimanohar, N.: Multi-Core Cache Hierarchies. Morgan & Claypool Publishers (2011)
[2] Wong, W., Baer, J.L.: Modified lru policies for improving second-level cache behavior. In: High-Performance Computer Architecture, 2000. HPCA-6. Proceedings. Sixth International Symposium on. (2000) 49 –60
[3] Belady, L.: A study of replacement algorithms for a virtual-storage computer. IBM Systems Journal 5(2) (1966) 78–101
[4] Qureshi, M.K., Jaleel, A., Patt, Y.N., Steely, S.C., Emer, J.: Adaptive insertion policies for high performance caching. SIGARCH Comput. Archit. News 35(2) (June 2007) 381–391
[5] Wong, W., Baer, J.L.: Modified lru policies for improving second-level cache behavior. In: High-Performance Computer Architecture, 2000. HPCA-6. Proceedings. Sixth International Symposium on. (2000) 49–60
[6] Jain, A., Shrivastava, A., Chakrabarti, C.: La-lru: A latency-aware replacement policy for variation tolerant caches. In: VLSI Design (VLSI Design), 2011 24th International Conference on. (2011) 298–303
[7] Qureshi, M.K., Lynch, D.N., Mutlu, O., Patt, Y.N.: A case for mlp-aware cache replacement. SIGARCH Comput. Archit. News 34(2) (May 2006) 167–178
[8] Kharbutli, M., Solihin, Y.: Counter-based cache replacement and bypassing algorithms. IEEE Trans. Comput. 57(4) (April 2008) 433–447
[9] Qureshi, M.K., Thompson, D., Patt, Y.N.: The v-way cache: Demand based associativity via global replacement. SIGARCH Comput. Archit. News 33(2) (May 2005) 544–555
[10] Das, S., Kapoor, H.K.: Victim retention for reducing cache misses in tiled chip multiprocessors. Elsevier Journal of Microprocessorsand Microsystems DOI: 10.1016/j.micpro.2013.11.005.
[11] Sanchez, D., Kozyrakis, C.: The zcache: Decoupling ways and associativity.In: Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture. MICRO '43 (2010) 187–198
[12] Das, S., Polavarapu, N., Halwe, P.D., Kapoor, H.K.: Random LRU:A replacement policy for chip multiprocessors. In: Inte-rnational Symposium on VLSI Design and Test (VDAT), 2013.
[13] Kim, C., Burger, D., Keckler, S.W.: An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches. SIGOPS Oper. Syst. Rev. 36 (October 2002) 211–222
[14] Beckmann, B.M., Wood, D.A.: Managing wire delay in large chip multiprocessor caches. In: Proceedings of the 37th annual IEEE/ACM International Symposium on Microarchitecture. MICRO 37, IEEE Computer Society (2004) 319–330
[15] Huh, J., Kim, C., Shafi, H., Zhang, L., Burger, D., Keckler, S.W.: A nuca substrate for flexible cmp cache sharing. In: Proceedings of the 19th annual international conference on Supercomputing. ICS '05 (2005) 31–40
[16] Belady, L.: A study of replacement algorithms for a virtual-storage computer. IBM Systems Journal (1966)
[17] Zahran, M.: Cache replacement policy revisited. In: The Annual Workshop on Duplicating, Deconstructing, and Debunking (WDDD) held in conjunction with the International Symposium on Computer Architecture (ISCA). (jun 2007)
[18] Fricker, C., Robert, P., Roberts, J.: A versatile and accurate approximation for lru cache performance. In: Teletraffic Congress (ITC 24), 2012 24th International. (2012) 1–8
[19] Morales, K., Lee, B.K.: Fixed segmented lru cache replacement scheme with selective caching. In: Performance Computing and Communications Conference (IPCCC), 2012 IEEE 31st International. (2012) 199–200
[20] Juan, F., Chengyan, L.: An improved multi-core shared cache replacement algorithm. In: Distributed Computing and Applications to Business, Engineering Science (DCABES), 2012 11th International Symposium on. (2012) 13–17
[21] Martin, M.M.K., Sorin, D.J., Beckmann, B.M., Marty, M.R., Xu, M.Alameldeen, A.R., Moore, K.E., Hill, M.D., Wood, D.A.: Multifacet's general execution-driven multiprocessor simulator

(gems) toolset. SIGARCH Comput. Archit. News 33(4) (November 2005) 92–99 [22] Magnusson, P.S., Christensson, M., Eskilson, J., Forsgren, D.,H°allberg, G., H¨ogberg, J., Larsson, F., Moestedt, A., Werner, B.: Simics: A full system simulation platform. Computer 35(2) (February 2002) 50–58

[23] Bienia, C.: Benchmarking Modern Multiprocessors. PhD thesis, Princeton University (January 2011)

[24] Kedzierski, K., Moreto, M., Cazorla, F., Valero, M.: Adapting cache partitioning algorithms to pseudo-lru replacement policies. In: Parallel Distributed Processing (IPDPS), 2010 IEEE International Symposium on. (2010) 1–12