IEICE TRANS. INF. & SYST., VOL.E95–D, NO.6 JUNE 2012

LETTER

# Throttling Capacity Sharing Using Life Time and Reuse Time Prediction in Private L2 Caches of Chip Multiprocessors*

**Young-Sik EOM**[†], **Jong Wook KWAK**[††a)], **Seong Tae JHANG**[†††], ***Nonmembers***, ***and*** **Chu Shik JHON**[†], ***Member***

**SUMMARY** In Chip Multi-Processors (CMPs), private L2 caches have potential benefits in future CMPs, e.g. small access latency, performance isolation, tile-friendly architecture and simple low bandwidth on-chip interconnect. But the major weakness of private cache is the higher cache miss rate caused by small private cache capacity. To deal with this problem, private caches can share capacity through spilling replaced blocks to other private caches. However, indiscriminate spilling can make capacity problem worse and influence performance negatively. This letter proposes *throttling capacity sharing* (TCS) for effective capacity sharing in private L2 caches. TCS determines whether to spill a replaced block by predicting reuse possibility, based on *life time* and *reuse time*. In our performance evaluation, TCS improves weighted speedup by 48.79%, 6.37% and 5.44% compared to non-spilling, Cooperative Caching with best spill probability (CC) and Dynamic Spill-Receive (DSR), respectively.

***key words:*** *Chip Multi-Processors, private L2 cache, capacity sharing, cooperative caching*

## 1. Introduction

Recently, Chip multiprocessor (CMP) is widely used in server computing, personal computers and even hand held devices. As in single processors, L2 cache of CMP reduces off-chip memory accesses which take hundreds of cycles. There are two kinds of L2 cache design: shared or private cache. The private L2 cache has potential benefits in future CMPs, such as small access latency, performance isolation, tile-friendly architecture and simple low bandwidth on-chip interconnect. However, the major weakness of private cache is the high cache miss rate caused by small private cache capacity.

To deal with this problem, *Cooperative Caching* (CC) [2] conducts capacity sharing by sending (spilling) replaced blocks to other private L2 caches (peer caches). Previously spilled blocks in other caches can be read when the core needs the blocks, through cache-to-cache transfer. In this way, the spilling core utilizes more capacity beyond private cache capacity. In *Dynamic Spill-Receive* (DSR) [7], each cache learns whether it should act as a "spiller cache" or "receiver cache" in order to minimize overall cache misses. Spiller cache can exploit the capacity of other cache without the interference of the spilling of other caches.

However, there are problems in previous proposals. They spill replaced blocks without considering whether they will be reused. Spilled blocks consume the capacity of the cache. That cache replaces blocks more frequently by spilled blocks and experiences higher miss rate. Besides, spilling consumes interconnection bandwidth as well. Therefore, if a spilled block is not reused, it makes no contribution to overall performance and it can even make the performance worse. On the contrary, too passive spilling is also problematic, which means relinquishing potential hit on peer cache.

In this letter, for a spilled block to be reused before eviction from the L2 peer cache which receives the spilled block, the state of the peer cache should be considered. We name the period between insertion and replacement of a spilled block "*life time*" of the cache. A cache with high miss-rate replaces lines frequently and thus has short life time. If a block is spilled to that cache, it moves to LRU position quickly and is replaced without reuse. But if it is spilled to a cache with longer life time, it has great potential to be reused. In addition, the status of a spilled block should be also considered. We call the period between replacement and reuse of a block "*reuse time*". A block with short reuse time can be reused even in a short life time peer cache. A block with long reuse time needs longer life time peer cache to be reused.

Therefore, for a replaced block to be reused in a peer cache, both the life time of the peer cache and the reuse time of the replaced block should be considered together. To predict the reuse of replaced blocks, we propose a Throttling Capacity Sharing (TCS) mechanism. When a block is replaced from local L2 cache, TCS predicts whether its reuse time is smaller than the life time of a peer cache, and only when it satisfies this reuse condition, TCS supposes that, if the block is spilled to the peer cache, it will be reused before replacement.

## 2. Throttling Capacity Sharing

When a block is replaced, to predict whether it is reused in a peer cache, we need to estimate both the life time of the peer cache and the reuse time of the replaced bock, and compare them. TCS spills a replaced block to the peer cache only when the reuse time of the block is smaller than the life time

Manuscript received December 15, 2011.

[†]The authors are with School of EECS, Seoul National University, Korea.

[††]The author is with the Department of Computer Engineering, Yeungnam University, Korea.

[†††]The author is with the Department of Computer Science, The University of Suwon, Korea.

*This research was supported by the Yeungnam University research grants in 2010.

a) E-mail: kwak@ynu.ac.kr

DOI: 10.1587/transinf.E95.D.1676

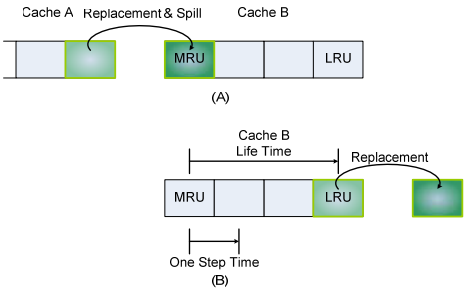



**Fig. 1** Life time prediction.

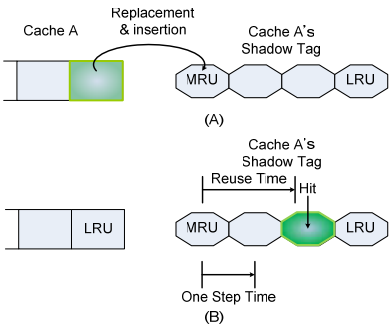



**Fig. 2** Reuse time prediction.

of the peer cache. The following shows the details.

### 2.1 Life Time Prediction

Figure 1 shows how to predict the life time of cache B. There are one sets of Cache A and Cache B having same index number. Life time of cache B is the period between (A) insertion (spilling) and (B) replacement of spilled block. One step time is the period which it takes a spilled block to move one step toward LRU position. Whenever a miss occurs in cache B, a spilled block moves one step. Therefore, one step time can be approximated by the number of interval cycles in collecting statistics over the number of misses in cache B. Then life time can be defined as the number of ways multiplied by one step time as follows.

$$LifeTime_{CacheB} = \frac{Ways \times Cycles_{Interval}}{Misses_{CacheB}} \quad (1)$$

### 2.2 Reuse Time Prediction

Figure 2 shows how to predict reuse time of a replaced block in cache A. To predict reuse time, TCS uses a shadow tag which is the extension of an original tag without data. Each set in cache A has a correspondent shadow tag. When a non-spilled block is replaced in the original tag, it is inserted to MRU position of the shadow tag and other blocks in the shadow tag move one step toward LRU position. Reuse time of a replaced block is time between (A) replacement and (B) reuse. Whenever a miss occurs, the miss address is searched on shadow tag. If it hits, the distance from MRU position to hit position is recorded on distance table. If it misses or an entry is evicted from the shadow tag, infinite value is stored. The value in distance table is as follows (Eq. (2)).

$$newValue = \frac{oldValue + distance}{2} \quad (2)$$

One step time is a period it takes a replaced block to move one step toward LRU position in shadow tag. Since each replacement makes all blocks in shadow tag move one step, one step time is the number of interval cycle in collecting statistics over the number of replacement. Therefore, reuse time can be defined as one step time multiplied by the distance from distance table as follows.

$$ReuseTime_{ReplacedBlock} = \frac{Distance \times Cycles_{Interval}}{Replacement_{CacheB}} \quad (3)$$

Only sample sets have shadow tag, not to overwhelm the cache space with shadow tag. There are 32 sampled sets distributed to all 1024 sets of L2 cache. Each 32 contiguous set has one sample set. We borrow the sample set distribution method from set-dueling [6]. Each shadow tag has 48 entries and each entry has additional 8 bits hashed PC to be used when accessing the distance table. The distance table is indexed by hashed PC and hashed miss address generated by XOR-ing all 8 bit parts. Each tag has 8 bits hashed PC and 1 bit spill bit to represent whether it is spilled block or not. This prediction table style is borrowed from [4].

To determine a replaced block to be reused or not, when it is spilled, TCS does not use life time and reuse time directly but use the threshold distance value. Using predicted life time (Eq. (1)) and reuse time (Eq. (3)), we can get the following *reuse condition* (Eq. (4)) as a threshold value, when a replaced block of cache A is spilled to peer cache B.

$$Distance_{ReplacedBlock} \leq \left( \frac{Ways \times Replacement_{CacheA}}{Misses_{CacheB}} = Threshold \right) \quad (4)$$

### 2.3 TCS Operation

TCS calculates the threshold for every core at the end of interval. We use 100,000 cycle interval. During each interval, TCS records the distances of miss addresses when miss occurs. To calculate the threshold, TCS uses 2 counters in each core. One is *replacement counter* to count the number of non-spilled replacements and the other is *miss counter* to count the number of misses. These two counters are used at the end of interval for calculation of the threshold. Therefore, the threshold value of core $i$ to core $j$ is as follows (Eq. (5))

$$Threshold(i, j) = \frac{Ways \times Replacement_i}{Misses_j} \quad (5)$$

At the end of each interval, TCS calculates all threshold values and all counters are initialized to 0. In four core system, each core has 3 threshold values. When a non-spilled block is replaced, TCS reads the distance of a miss address

from the distance table. Then, after comparing the distance to the thresholds of each cache, TCS spills the block to a cache with bigger threshold than the distance of the block. When there are other caches satisfying the reuse condition, the target cache to spill is randomly chosen among them.

## 3. Performance Evaluation

### 3.1 System Configuration and Workload

We use a cycle accurate, out of order simulator, CMP-SIM, using Alpha 21264 (ev6) binary code [1]. The base configuration is a four core CMP with the given parameters, shown in Table 1. L2 cache is 1 MB per core and it has 10 cycle local hit latency and 40 cycle peer cache hit latency. Block size is 64 Bytes and memory access cycle is 300 cycles.

We use SPEC CPU2000 benchmarks. To evaluate our mechanism, we formed 17 programmed workloads. We classify these workloads into five categories depending on how many "Giver" (G) and "Taker" (T) the workload has. Giver applications are small working set and low utility applications [5] which can lend their cache space without significant performance degrade. Taker applications are high utility applications [5] which benefit from taking more space. Note that all L2 caches in TCS can both spill replaced blocks and receive spilled blocks as opposed to DSR.

**Table 1** Parameters of the simulated architecture.

| Processor Parameters |
|---|
| 4-issue dynamic. INT/FP/INTmulti/FP multi FUs: 1/1/1/1 |
| 16 entry RUU, 8 LSQ BTB(512 entry, 4 way) |
| Branch Predictor(2k entry bimodal) |
| **Memory Parameters** |
| L1 Inst/Data Cache: Each 64KB,4 way, 64B line, 1 cycle hit |
| L2 unified Cache: 1MB per core, 16way, 64B line, 10/40 cycle local/peer hit |
| Memory: 300/4 cycle first/inter chunk latency, 16B bus width |

**Table 2** Multi-programmed workloads.

| Category | No. | Workloads |
|---|---|---|
| G4T0 | 1 | equake, gap, mcf, mesa |
| | 2 | crafty, gcc, mcf, vortex |
| | 3 | eon, gap, lucas, mcf |
| G3T1 | 4 | applu, art, equake, gap |
| | 5 | applu, art, bzip2, mcf |
| | 6 | art, bzip2, gap, vortex |
| G2T2 | 7 | art, galgel, gap, lucas |
| | 8 | art, fma3d, lucas, mcf |
| | 9 | ammp, gzip, mcf, twolf |
| G1T3 | 10 | art, mcf, mgrid, twolf |
| | 11 | ammp, apsi, art, swim |
| | 12 | apsi, art, galgel, mesa |
| | 13 | apsi, art, parser, swim |
| G0T4 | 14 | ammp, mgrid, parser, vpr |
| | 15 | apsi, art, fma3d, parser |
| | 16 | apsi, art, twolf, vpr |
| | 17 | apsi, art, galgel, mgrid |

Table 2 shows total workloads evaluated in this letter.

### 3.2 Result and Analysis

We experiment on CC, DSR and TCS. CC has five configurations, each with spill probability of 0% (non-spilling: base private L2 cache), 25%, 50%, 75% and 100%, respectively. Among them, CC (Best) is the CC which has best performance in each workload.

Figure 3 shows the reuse ratio to spill of CC (Best), DSR and TCS, respectively. X-axis corresponds to 14 workloads. As shown in Fig. 3, DSR and TCS consistently has the better ratio than CC (Best) in all categories since in DSR the only cores which shows the large number of reuse spill blocks and TCS spills the blocks which are expected to be reused. On average, CC (Best), DSR and TCS have the reuse ratios of 65.59%, 85.37% and 93%, respectively.

Figure 4 shows the number of spills of TCS normalized to those of DSR. This result shows that TCS spills 6% more than DSR on average. In case of high reuse ratio, the higher number of spills means the better performance. This is because the hit rate of non-spilled blocks inserted by cache misses is low in L2 cache and the higher number of spills means L2 cache is filled with the more blocks expected to be hit. Therefore, we can infer that DSR spills too conservatively and relinquishes the potential for L2 hits. On the other hand, TCS spills more blocks than DSR with the reuse ratio comparable to DSR and results in the more number of L2 hits than DSR.

Figure 5 shows the number of total spills of CC (100), DSR and TCS, normalized to CC (100) in G4T0 category. In these workloads, the less the number of spills is, the better the system performance is. Therefore, CC (Best) is CC (0%) (non-spilling). DSR and TCS have the reuse ratio of

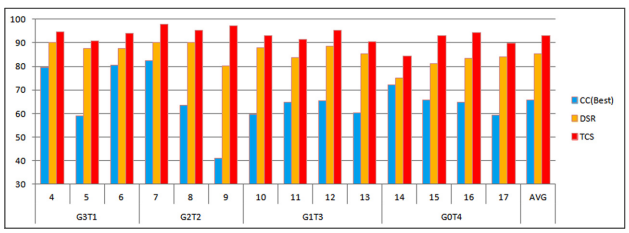



**Fig. 3** The ratio of reuse to spill.

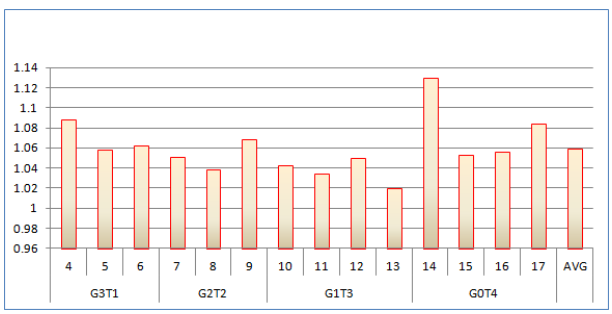



**Fig. 4** The number of spills of TCS normalized to DSR.

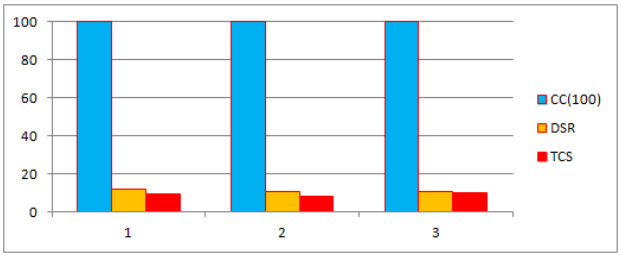



**Fig. 5** The number of spills normalized to CC (100).

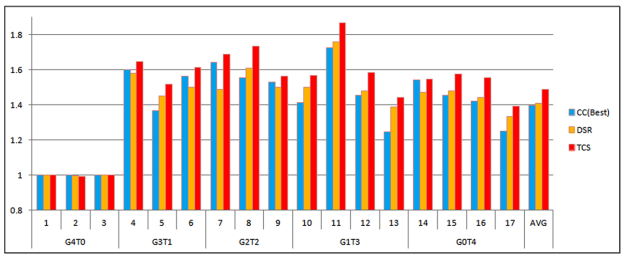



**Fig. 6** Weighted speed up normalized to non-spilling.

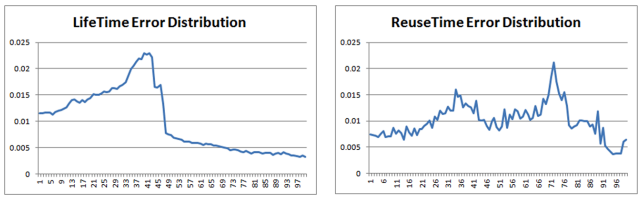
10.67% and 10% of CC (100), respectively. This shows that both TCS and DSR throttle the number of unnecessary spills effectively.

To measure CMP system performance, we use a weighted speed-up. The weighted speed-up corresponds to total system throughput [3]. Figure 6 shows weighted speed up normalized to private L2 cache (CC (0%), non-spilling). In most workloads, TCS outperforms CC (Best) and DSR. This is since TCS reduces the large number of useless spills and satisfies the capacity demand of each core. In category G4T0, all methods show comparable performance. Since all applications in this category are low utility or small working set applications, spilling rarely influences performance. DSR gives the comparable performance to CC (Best) in spite of higher reuse ratio. This is because DSR often makes a wrong choice of spiller and receiver, and it does not spill replaced blocks, which are expected to be reused if spilled. On average, TCS improves weighted speed up by 48.79%, 6.37% and 5.44% over non-spilling, CC (Best) and DSR, respectively.

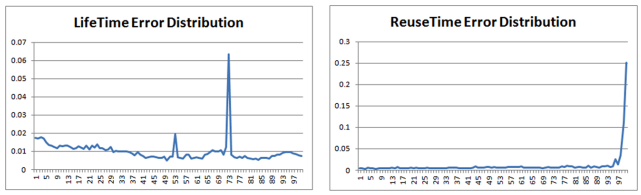
TCS does not estimate life time and reuse time directly. However, we can estimate the relative errors (Eq. (6)) of (1) life time and (2) reuse time as follows, since threshold value is based on them.

$$relativeError = \frac{|Time_{real} - Time_{prediction}|}{MAX(Time_{real}, Time_{prediction})} \times 100 \quad (6)$$

Figure 7 shows the distribution of relative errors of the predictions of life time and reuse time for two representative workloads 10 and 14. The X-axis corresponds to relative error and the Y-axis corresponds to the probability of relative error. Reuse time is more difficult to predict than life time. The error of workload 10 is smaller than that of workload 14. This result is consistent with the result of weighted speed up. In workload 10, TCS has 10.63% speed-up over CC (Best), but 0.34% speed-up in workload 14. If prediction could be more accurate, TCS performance would be improved more.




(a) workload 10




(b) workload 14

**Fig. 7** The distribution of relative errors of life time and reuse time.

## 4. Conclusion

This letter proposes throttling capacity sharing (TCS) for effective capacity sharing of private L2 caches. When a block is replaced from L2 cache, TCS predicts whether its reuse time is smaller than the life time of a peer cache, and only when it satisfies reuse condition, TCS spills the replaced block to the peer cache. We evaluated the performance improvement of TCS and TCS improves weighted speed up by 48.79%, 6.37% and 5.44% on average, compared to non-spilling, CC (Best) and DSR for 17 multi-programmed workloads.

### References

[1] R.S. Baldawa, "CMPSIM: A flexible multiprocessor simulation environment," The University of Texas at Dallas, 1450810, 2007.
[2] J. Chang and G.S. Sohi, "Cooperative caching for chip multiprocessors," ISCA '06: 33rd Int. Symp. on Computer Architecture, pp.264–276, 2006.
[3] S. Eyerman and L. Eeckhout, "System-level performance metrics for multiprogram workloads," IEEE Micro, vol.28, no.3, pp.42–53, 2008.
[4] M. Kharbutli and Y. Solihin, "Counter-based cache replacement and bypassing algorithms," IEEE Trans. Comput., vol.57, no.4, pp.433–447, April 2008.
[5] M. Moreto, F.J. Cazorla, A. Ramirez, and M. Valero, "Explaining dynamic cache partitioning speed ups," Computer Architecture Letters, vol.6, no.1, pp.1–4, 2007.
[6] M.K. Qureshi, A. Jaleel, Y.N. Patt, S.C. Steely, Jr., and J. Emer, "Adaptive insertion policies for high performance caching," ISCA-2007, pp.381–391, 2007.
[7] M.K. Qureshi, "Adaptive spill-receive for robust high-performance caching in CMPs," HPCA, pp.45–54, 2009.