A Reusability-Aware Cache Memory Sharing Technique for High Performance CMPs with Private L2 Caches

Sungjune Youn, Hyunhee Kim and Jihong Kim
School of Computer Science & Engineering, Seoul National University, Seoul, Korea
{spica81, hh0726, jihong}@davinci.snu.ac.kr

ABSTRACT

For high-performance chip multiprocessors (CMPs) to achieve their maximum performance potential, an efficient support for memory hierarchy is important. Since off-chip accesses require a long latency, high-performance CMPs are typically based on multiple levels of on-chip cache memories. For example, most current CMPs support two levels of on-chip caches. While the L1 cache architecture of these CMPs is almost same, the L2 cache organization is quite different. All the processors in the CMP may share the same L2 cache or each processor may have its own private L2 cache. While a private L2 cache has a short access time, a shared L2 cache can better adapt to varying memory requirements of each processor.

In this paper, we propose an on-chip L2 cache organization which takes advantage of both a private L2 cache and a shared L2 cache. In skeleton, our L2 cache organization is based on a private L2 cache organization which has the short access latency. When a cache block in the private L2 cache is selected for an eviction, our proposed organization first evaluates the reusability of the cache block. If the cache block is likely to be reused, we save the evicted cache block in one of peer L2 caches which may have efficiently invalid blocks. By selectively writing evicted cache blocks to peer L2 caches, the proposed L2 cache organization can effectively simulate a shared L2 cache. Experimental results using a CMP simulator showed that the proposed L2 cache organization improved the average memory latency by 14% on average over a pure private L2 cache organization for the SPLASH2 benchmark programs.

1. Introduction

For high-performance chip multiprocessors (CMPs) to achieve their maximum performance potential, an efficient support for memory hierarchy is important. Since the on-chip cache memory space is limited in CMPs and the off-chip memory accesses require a longer latency than the on-chip memory access latency, we need to manage the on-chip cache space carefully to improve the overall system performance [4][5].

Most high-performance CMPs are typically based on multiple levels of on-chip cache memories to manage the on-chip cache space efficiently. For example, most current CMPs support two levels of on-chip caches. In typical CMPs, each processor has a small private L1 cache because the access latency of the L1 cache affects system performance directly. However, the L2 cache organization of CMPs is quite different. All the processors in the CMP may share the same L2 cache or each processor may have its own private L2 cache. A shared L2 cache utilizes cache space more flexibly, reducing the number of the off-chip memory accesses. But, it has the longer access latency and generates more on-chip network traffic than a private L2 cache. On the other hand, a private L2 cache has the short access latency but it is inefficient in utilizing the L2 cache space with many duplicated copies of the same memory block.

In order to use on-chip cache memory space more efficiently in CMPs, several research groups have proposed different on-chip cache organizations such as CMP-SNUCA [8], Victim Replication [3], CMP-NuRAPID [2] and CMP-CC [1]. CMP-SNUCA [8] scheme applies NUCA [7] to the CMPs architecture. They migrate blocks close to the requestor to reduce wire-delay. Victim Replication [3] scheme attempts to keep copies of local primary cache victims within the local L2 cache slice to reduce wire-delay in shared L2 cache. CMP-NuRAPID [2] scheme makes copies close to requestors to allow fast access for read-only sharing, and does not make copies for read-write sharing to avoid coherence misses. They also propose capacity stealing of neighbor's cache when the cache capacity is not enough to store private data. CMP-CC [1] writes back a block to a peer L2 cache[1] when the block is evicted from a private L2 cache *randomly* with a given probability to redistribute private L2 cache space. A similar technique proposed in [11] also selectively writes back L2 victims to a peer L2 cache but it is quite limited in sharing L2 blocks in peer L2 caches.

In this paper, we propose an on-chip L2 cache organization which takes advantage of both a private L2 cache and a shared L2 cache in CMPs. In skeleton, our L2 cache organization is based on a private L2 cache organization which has the short

[1] In this paper, we call a private L2 cache of neighboring processors as a peer L2 cache.

Workshop on Chip Multiprocessor Memory Systems and Interconnects (CMP-MSI) in conjunction with HPCA-13, Feb 2007.

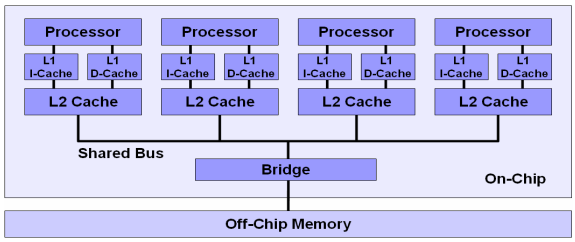

Figure 1. A target CMP architecture.

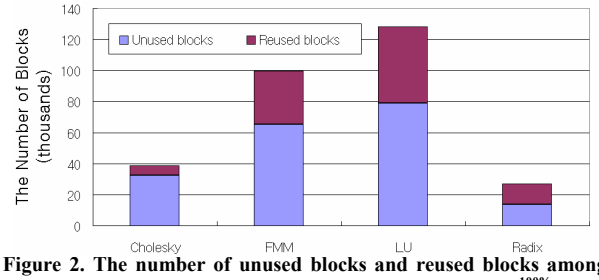

Figure 2. The number of unused blocks and reused blocks among blocks written back to peer L2 cache in CMP-CC$^{100\%}$.

access latency. Figure 1 shows our target CMP architecture with a private L2 cache and a shared bus. When a cache block in the private L2 cache is selected for an eviction, our proposed organization first evaluates the reusability of the cache block. If the cache block is likely to be reused, we save the evicted cache block in the private L2 cache of other processors which may have *efficiently invalid* blocks. Since accessing the private L2 cache of a nearby processor is faster than accessing the off-chip memory, saving cache blocks with high reusability in a peer L2 cache will improve the memory performance. By selectively writing evicted cache blocks to peer L2 caches, the proposed L2 cache organization can simulate a shared L2 cache organization.

Writing back an evicted block to peer L2 caches is not a new idea as proposed in CMP-CC [1]. However CMP-CC does not consider the reusability of evicted L2 blocks in deciding whether the evicted L2 blocks are saved in peer L2 caches. If there are too many blocks which are not reused, the CMP performance may be deteriorated over the CMP architecture with private L2 caches. L2 blocks with low reusability should not be saved to a peer L2 cache. Therefore, it is important to consider the reusability of an evicted block when saving the evicted blocks to peer L2 caches. In this paper, we describe a reusability-aware cache memory sharing technique based on the reusability estimate of the evicted blocks. Our scheme reduces the average memory access latency by 14% over a private L2 scheme on average, thus improving the average IPC by 3%.

The rest of the paper is organized as follows. In Section 2, we explain the proposed reusability-aware cache sharing technique. Performance evaluation is discussed in Section 3.

2. Reusability-Aware Cache Sharing Technique

2.1. Motivation

CMP-CC [1] writes back L2 victims to peer L2 caches randomly with a given probability without any consideration about the property of blocks. [11] also proposes selective L2 cache write back technique, but their technique can not exploit full advantage of writing back to peer L2 caches. Because they write back to peer L2 cache only when there exists a shared line or an invalid line in peer L2 caches even though there is not enough number of a shared line and an invalid line.

Figure 2 shows the number of reused blocks and unused blocks among blocks written back to a peer L2 cache in the CMP-CC$^{100\%}$ scheme which writes back every L2 victims to peer L2 caches. Experimental parameters are shown in Table 1. In every benchmark, the number of unused blocks is larger than the number of reused blocks. The performance of a system can be damaged because of unused blocks. They occupy peer L2 cache space unnecessarily, and they generate additional on-chip shared bus transactions which can cause conflicts on a shared bus. And remote private L2 hits are increased while local private L2 hits are decreased. This is the necessity of the selective write back of L2 victims to peer L2 caches.

So if we reduce the number of unused blocks, system performance can be improved. To get full advantage of the private L2 cache sharing technique, we consider the property of the block and each private L2 cache when we decide which L2 victims to be written back to which peer L2 cache. Our scheme does not write back to a peer L2 cache if the L2 victim block is not likely to be reused. When we write back a block with high-reusability to a peer L2 cache, one of blocks in the peer L2 cache should be evicted. In that case, we only evict a block with low reusability in the peer L2 cache. If there is no peer L2 cache which has a low reusability block, we write back the L2 victim only to the peer L2 cache which has smaller memory demand than the L2 cache's which evicts the block. In Section 2.2 we explain about the block reusability prediction technique, and Section 2.3 describes the memory demand prediction technique. We explain about how the Reusability-Aware Cache Sharing (RACS) technique works in Section 2.4.

2.2. Block Reusability Prediction Technique

In a conventional replacement algorithm of a cache or a buffer, recency or frequency of accesses to a block is used to predict reusability of the block [15]. But in our scheme, we can not consider recency because a block has almost no recency when it is evicted from a private L2 cache. So we consider frequency of accesses to the block and a time interval

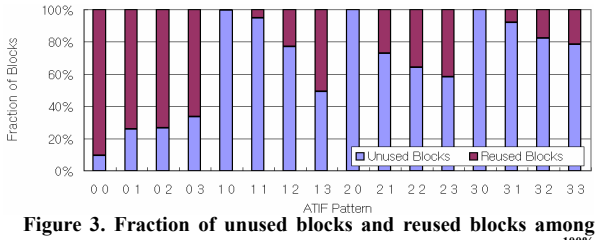

Figure 3. Fraction of unused blocks and reused blocks among blocks written back to peer L2 cache in CMP-CC$^{100\%}$ per each ATIF pattern (Benchmark: Radix).

between two consecutive accesses to the block in the past, instead of recency.

We classify blocks into Access Time Interval and Frequency (ATIF) patterns and monitor how many blocks which are written back to peer L2 cache are reused and unused per each pattern. If the number of unused blocks is much larger than the number of reused blocks in certain pattern, we predict that blocks in that pattern have low reusability and do not write back these blocks to peer L2 caches.

We count the number of accesses with a long time interval and a short time interval per each block while the block is in a private L2 cache to classify blocks into ATIF patterns. We distinguish a short and long time interval with a very simple technique. We estimate the access time interval of the block is long when there is any intervening access to a block that belongs to the same set. If not, we estimate the interval is short. An ATIF pattern of a block is decided by these two counts when the block is evicted from a private L2 cache. We use a 4-bit counter to record the number of accesses to a block with a short access time interval and a 2-bit counter to record the number of accesses with a long access time interval per each block. Blocks are classified into 16 ATIF patterns using an upper 2 bits value of the 4-bit counter and a value of the 2-bit counter.

We add 16 2-bit counters in each private L2 cache to record the reuse rate of blocks per each ATIF pattern. If a block written back to a peer L2 cache is reused, corresponding ATIF pattern counter of the block's original owner L2 cache is increased by one. If the block written back to peer L2 cache is evicted from the peer L2 cache without reuse, the ATIF pattern counter is decreased by one. When the ATIF pattern counter becomes 0, we predict blocks that belong to the ATIF pattern have very low reusability.

Figure 3 shows the fraction of unused blocks and reused blocks among blocks written back to peer L2 caches in CMP-CC$^{100\%}$ per each ATIF pattern. The first number of the ATIF pattern is an upper 2 bits value of a 4-bit counter for a short access time interval and the second number of the ATIF pattern is a value of a 2-bit counter for a long access time interval. Some patterns have very high reusability while some patterns have very low reusability. It shows predicting the reusability of blocks with ATIF patterns is reasonable. The block reusability prediction technique can reflect the property of an application dynamically.

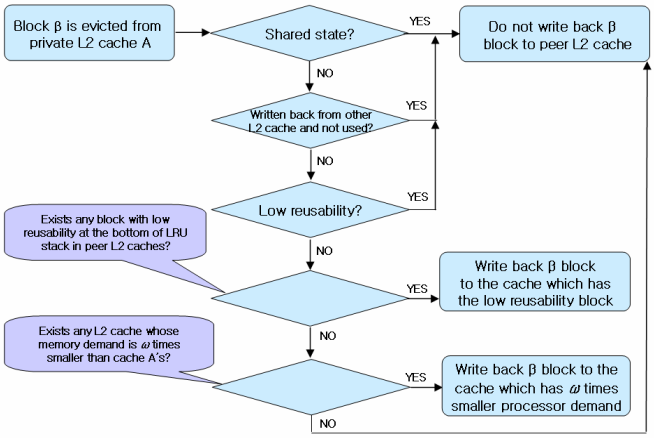

Figure 4. Process of deciding which L2 victims to be written back to which peer L2 cache in the RACS technique.

2.3. Memory Demand Prediction Technique

We predict that a processor requires more memory as more frequently as replacement occurs in a private L2 cache. So we use a replacement time interval history ($Repl_{interval\_history}$) as the prediction value of the processor's memory demand. Each private L2 cache has this value, and it is updated every time replacement occurs in private L2 cache by following formula.

$$Repl_{interval\_history(new)} = \frac{Repl_{interval\_history(prev)} \times 3 + Repl_{interval}}{4}$$

$Repl_{interval\_history(prev)}$ is an old prediction value of a memory demand and $Repl_{interval\_history(new)}$ is a new value. $Repl_{interval}$ is a time interval between last two consecutive replacements. As an L2 cache has smaller $Repl_{interval\_history}$ value, it means the processor requires more memory. This prediction value is used only for comparison of a memory demand between L2 caches, not for an exact memory demand. To calculate the time interval, we use 8-bit counter to record the time from the last replacement. This counter is increased by one every 32 cycle and reset to 0 when the replacement occurs.

2.4. Process of the RACS technique

Figure 4 shows the process of deciding which L2 victims to be written back to which peer L2 cache in the RACS technique. When a block is evicted from a private L2 cache, we do not write back to a peer L2 cache if the state of the block is shared. Because it means that there already exists the same block in other L2 cache. And we also do not write back to a peer L2 cache if the block is written back from other L2 cache but is not reused. Because it means that the block already had a chance. And then we check the reusability of the block. If the reusability is low, we do not write back. If the reusability is high, check if there exists any block with low reusability at the

bottom of LRU stack in peer L2 caches. If there exists, we write back the block to the peer L2 cache. If there does not exist, we check if there exists any peer L2 cache whose memory demand is ω times smaller than the memory demand of the cache which evicts the block. If there exists, we write back the block to the peer L2 cache. If there is no such a peer L2 cache, we do not write back the block to a peer L2 cache even though it has high reusability. ω value varies between 4/3 and 3. And each private L2 cache has its own ω value. ω value is decreased by 1/3 when a block is reused and increased by 1/3 when three blocks are written back to other cache. So it has smaller value as more blocks are reused. It means that if many of blocks are reused we can write back a block to a peer L2 cache even though the difference of memory demand is not so large. Writing back to a peer L2 cache does not cause a subsequent write back to other peer L2 cache to avoid a ripple effect.

We need to communicate with peer L2 caches to decide which peer L2 cache we write back to. So we assume additional peer-to-peer communication lines among L2 caches. If a block has high reusability, the L2 cache sends set number of the block, ω value and $Repl_{interval\_history}$ to peer L2 caches. And then peer L2 caches reply two bits information. One bit indicates the cache has the block with low reusability at the bottom of LRU stack. The other bit indicates $Repl_{interval\_history}$ of the cache is ω times larger than received $Repl_{interval\_history}$. We can decide if we write back a block or not and write back to which peer L2 cache with these information.

The RACS technique has hardware overhead compared to a pure private L2 cache organization because we need additional counters for two prediction schemes and peer-to-peer communication lines between L2 caches. But the maximum data transferred by this line is only up to 21 bits and we need only six lines when the number of private L2 caches is 4. The area overhead of counters is less than 1% of the private L2 cache. So hardware overhead of our technique is not significant.

And write back decision is not on the critical path because it could be made after a block is evicted from the cache and placed in the write back queue.

3. Performance Evaluation

3.1. Simulation Environment

Processor		4 Processors (ARM, inorder)
Cache	L1 D-Cache	16KB, 1-way, 32B block, 1 cycle latency
	L1 I-Cache	16KB, 1-way, 32B block, 1 cycle latency
	L2 Private Cache	256KB, 4-way, 128B block, 6 cycle latency
	L2 Shared Cache	1MB, 16-way, 128B block, 38 cycle latency
Shared Bus		4 bytes bus width, pipelined
Off-Chip Memory		500 cycle access latency

Table 1. Processor and cache/memory configuration.

Private L2 Cache Energy (1 bank)	Read Hit	2.6335 nJ
	Read Miss	3.7536 nJ
	Write Hit	1.1592 nJ
	Write Miss	1.1960 nJ
Shared L2 Cache Energy (4 banks)	Read Hit	31.0816 nJ
	Read Miss	41.8176 nJ
	Write Hit	12.3376 nJ
	Write Miss	12.5664 nJ
Off-Chip Memory Energy	Access	32.5 nJ

Table 2. L2 cache and memory energy per access. (cache and memory configuration in table 1)

We modify CATS [12] multiprocessor simulator to evaluate our technique. We use a MESI protocol for cache coherency. Cache-to-cache transfer of the cache block among private L2 caches is applied in all schemes. Table 1 shows the processor and cache/memory configuration. When a cache is accessed through the shared bus, bus delay cycles and conflict cycles are added on cache access latency. Table 2 shows the L2 cache and memory energy per access. We get cache energy values from eCACTI [14] and memory energy from MICRON MT48V8M32LF SDRAM [16].

Shared L2 cache, private L2 cache, CMP-CC, RACS and oracle$^-$ schemes are implemented and evaluated. And we evaluate our scheme with Cholesky, FMM, LU, Radix in SPLASH2 [10] benchmark.

3.2. Experimental Results

Figure 5 shows the number of unused blocks and reused blocks of each scheme among blocks written back to peer L2 caches. The number of unused blocks and reused blocks are increasing in the CMP-CC scheme with same ratio as the probability increases. The RACS scheme has the almost same number of reused blocks with CMP-CC$^{100\%}$ except Cholesky and reduces the number of unused blocks by 65% on average compared to CMP-CC$^{100\%}$. The reason why Cholesky has the smaller number of reused blocks in the RACS scheme is that it predicts too many blocks not to be reused because Cholesky has too low reusability even though there exist blocks to be reused. The oracle$^-$ scheme writes back the block that will be

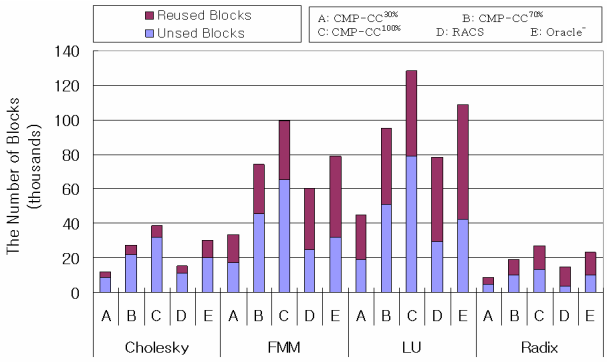

Figure 5. The number of unused blocks and reused blocks of each scheme among blocks written back to peer L2 caches.

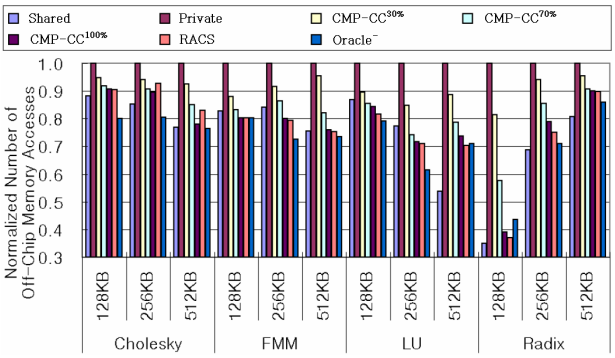

Figure 6. Normalized number of off-chip memory accesses (Normalized to private scheme).

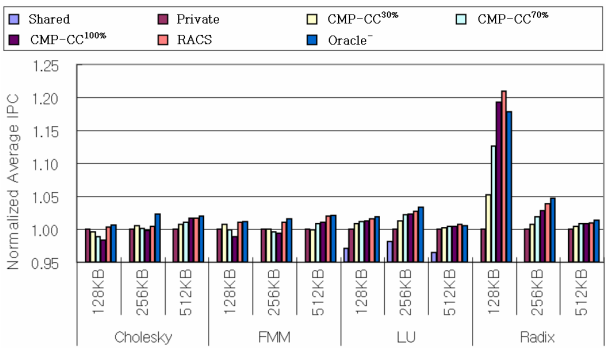

Figure 8. Normalized average IPC (Normalized to private scheme).

reused in the future and evicts the block which will be reused in the farthest future among the blocks at the bottom of LRU stacks of the private L2 caches. The oracle- has larger number of reused blocks than any other schemes and smaller number of unused blocks than CMP-CC100%. But it has larger number of unused blocks than the RACS because the oracle- writes back a block even though block will be reused in too far future, which might not be reused after it is written back to peer L2 cache and evicted without reuse.

Figure 6 shows the normalized number of off-chip memory accesses of each scheme varying the cache size. Each size indicates the size of private L2 cache, CMP-CC, RACS and oracle- scheme, while the shared L2 cache size of shared scheme is four times larger than the size denoted in Figure 6. The number of off-chip accesses is decreasing in CMP-CC as the probability increases. In the RACS scheme, the number of off-chip accesses is decreased up to 4% than CMP-CC100% in LU and Radix. But it has the almost same number of off-chip accesses with CMP-CC100% in FMM and it has larger number of off-chip accesses in Cholesky. The oracle- scheme has the least number of off-chip accesses in most cases, but it has the larger number of off-chip accesses than RACS in LU (512KB) and Radix (128KB). Because oracle- does not write back a block to a peer L2 cache if the block will be reused in the farthest future among blocks at the bottom of the LRU stack in peer L2 caches. In this case, none of blocks could be reused if all of these blocks will be evicted soon. If we would write back the evicted block, at least one block could be reused because it remains at on-chip.

Figure 7 shows the normalized average memory access latency of each scheme varying the cache size. The shared scheme shows the worst performance because of the long access latency. Average memory access latency is decreasing in CMP-CC as the probability increases in most cases, but it is increasing in Cholesky and FMM when the cache size is 128KB and 256KB. Because there are too many blocks written back to peer L2 caches and are not used, they occupy peer L2 cache space and generate shared bus conflicts even though the number of off-chip memory accesses is reduced. Furthermore remote private L2 hits are increased but local private L2 hits are decreased in these cases. The RACS scheme shows the best performance among all schemes except oracle- in most cases. The RACS scheme has the larger number of off-chip memory accesses in some cases, but it has shorter average memory access latency in these cases because it reduces unused blocks. The RACS scheme reduces average memory access latency by 14% and 4% over the private L2 scheme and CMP-CC100% on average, respectively.

Figure 8 shows the normalized average IPC of each scheme varying the cache size. It shows almost same property with

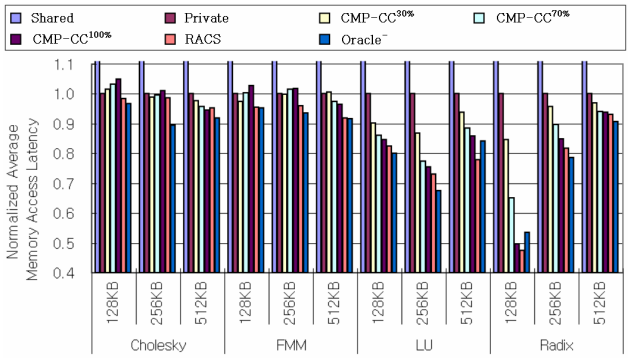

Figure 7. Normalized average memory access latency (Normalized to private scheme).

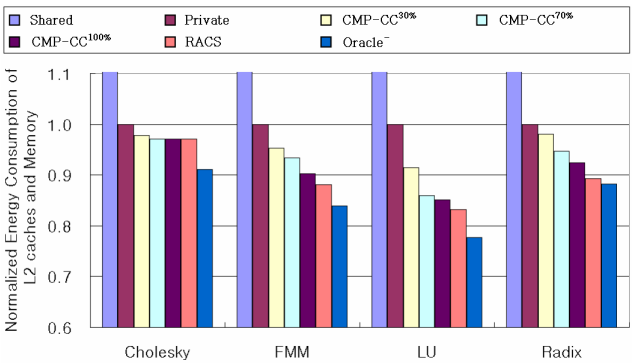

Figure 9. Normalized energy consumption of L2 caches and off-chip memory (Normalized to private scheme).

average memory access latency. But the result of the shared scheme is not appeared in most cases in Figure 8, because it has smaller IPC than 0.95. The RACS scheme improves average IPC by 3% and 1% over the private L2 scheme and CMP-CC$^{100\%}$ on average, respectively.

Figure 9 shows the normalized L2 cache and off-chip memory energy of each scheme when the private cache size is 256KB and the shared cache size is 1MB. Energy consumption of the shared scheme is much larger than any other scheme because it has larger energy consumption per access as shown in table 2. The private scheme has the largest energy consumption except the shared scheme because of the large number of off-chip accesses. RACS consumes less energy than CMP-CC$^{100\%}$ except Cholesky. RACS consumes 10% and 2% less energy on average over the private L2 scheme and CMP-CC$^{100\%}$, respectively

4. Conclusions

We proposed an on-chip L2 cache organization which takes advantage of both a private L2 cache and a shared L2 cache in CMPs. In order to have the short access latency, the proposed L2 cache organization, RACS, is based on a private L2 cache organization. When a cache block in the private L2 cache is selected for an eviction, RACS evaluates the reusability of the cache block. If the cache block is likely to be reused, we save the evicted cache block in one of peer L2 caches which may have efficiently invalid blocks.

We evaluated the RACS scheme using a modified CMP simulator and compared the performance with the private scheme, shared scheme, CMP-CC scheme and oracle scheme. The RACS scheme reduces the number of unused blocks (which were written back to peer L2 cache) by 65% over CMP-CC$^{100\%}$. It also reduces the average memory access latency by 14% and 4% over the private L2 scheme and CMP-CC$^{100\%}$, respectively. The RACS scheme improves the average IPC by 3% and 1% over the private L2 scheme and CMP-CC$^{100\%}$, respectively.

The proposed RACS scheme can be further improved in several directions. First, the prediction heuristic in the RACS scheme can be improved significantly. For example, about 43% blocks are not reused even though the prediction heuristic classifies those blocks as highly-reusable blocks. Second, the RACS scheme has a significant hardware overhead if the number of processors is more than 8 because it needs peer-to-peer communication line among private L2 caches. It is an interesting future work to make the RACS scheme to be more scalable for large-scale CMP processors.

Acknowledgement

This work was supported in part by the Brain Korea 21 Project in 2006 and MIC & IITA through IT Leading R&D Support Project.

References

[1] J. Chang and G.S. Sohi. Cooperative Caching for Chip Multiprocessors. In *Proc. of International Symposium on Computer Architecture (ISCA)*, pages 264-276, 2006.

[2] Z. Chishti, M. D. Powell, and T. N. Vijaykumar. Optimizing Replication, Communication and Capacity Allocation in CMPs. In *Proc. of International Symposium on Computer Architecture (ISCA)*, pages 357-368, 2005.

[3] M. Zhang and K. Asanovic. Victim Replication: Maximizing Capacity while Hiding Wire Delay in Tiled Chip Multiprocessors. In *Proc. of International Symposium on Computer Architecture (ISCA)*, pages 336-345, 2005.

[4] J. Huh, D. Burger and S.W. Keckler. Exploring the Design Space of Future CMPs. In *Proc. of the International Conference on Parallel Architectures and Compilation Techniques*, pages 199-210, 2001.

[5] B. A. Nayfeh, L. Hammond and K. Olukotun. Evaluation of Design Alternatives for a Multiprocessor Microprocessor. In *Proc. of International Symposium on Computer Architecture (ISCA)*, pages 76-77, 1996.

[6] Z. Chishti, M. D. Powell and T. N. Vijaykumar. Distance Associativity for High-Performance Energy-Efficient Non-Uniform Cache Architectures. In *Proc. of International Symposium on Microarchitecture (MICRO 36)*, pages 55-56, 2004.

[7] C. Kim, D. Burger and S. W. Keckler. An Adaptive, Non-Uniform Cache Structure for Wire-Delay Dominated On-Chip Caches. In *Proc. of Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 211-222, 2002.

[8] B. M. Beckmann and D. A. Wood. Managing Wire Delay in Large Chip-Multiprocessor Cache. In *Proc. of the 37th International Symposium on Microarchitecture (MICRO 37)*, pages 319-330, 2004.

[9] N. Jouppi. Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers. In *Proc. of International Symposium on Computer Architecture (ISCA)*, 1990.

[10] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh and A. Gupta. The SPLASH-2 Programs: Characterization and Methodological Considerations. In *Proc. of International Symposium on Computer Architecture (ISCA)*, pages 24-36, 1995.

[11] E. Speight, H. Shafi, L. Zhang and R. Rajamony. Adaptive Mechanisms and Policies for Managing Cache Hierarchies in Chip Multiprocessors. In *Proc. Of International Symposium on Computer Architecture (ISCA)*, pages 346-356, 2005.

[12] D. H. Kim. The CATS Framework for MPSoC System, http://mesl.ucsd.edu/dhkim/CATS/

[13] T. Austin. SimpleScalar. http://www.simplescalar.com

[14] M. Mamidipaka and N. Dutt. eCACTI: An Enhanced Power Estimation Model for On-Chip Caches. http://www.ics.uci.edu/~maheshmm/eCACTI/main.htm

[15] D. Lee, J. Choi, J. Kim, S. H. Noh, S. Min, Y. Cho, C. Kim. LRFU: A Spectrum of Policies that Subsumes the Least Recently Used and Least Frequently Used Policies. In *IEEE Transactions on Computers*, vol. 50, pages 1352-1361, 2001.

[16] Micron Technology Inc., http://www.micron.com/products/dram/mobilesdram/. MICRON Mobile SDRAM MT48V8M32LF Datasheet, 2005.