A Space-Efficient On-Chip Compressed Cache Organization for High Performance Computing[1]

Keun Soo Yim[a], Jang-Soo Lee[b], Jihong Kim[a], Shin-Dug Kim[c], and Kern Koh[a]
[a] School of Computer Science and Engineering, Seoul National University, Seoul, Korea
[b] IBM Poughkeepsie, NY, USA
[c] Department of Computer Science, Yonsei University, Seoul, Korea
{ksyim, kernkoh}@oslab.snu.ac.kr, jangsoo@us.ibm.com, jihong@davinci.snu.ac.kr, sdkim@cs.yonsei.ac.kr

Abstract. In order to alleviate the ever-increasing processor-memory performance gap of high-end parallel computers, on-chip compressed caches have been developed that can reduce the cache miss count and off-chip memory traffic by storing and transferring cache lines in a compressed form. However, we observed that their performance gain is often limited due to their use of the coarse-grained compressed cache line management which incurs internally fragmented space. In this paper, we present the fine-grained compressed cache line management which addresses the fragmentation problem, while avoiding an increase in the metadata size such as tag field and VM page table. Based on the SimpleScalar simulator with the SPEC benchmark suite, we show that over an existing compressed cache system the proposed cache organization can reduce the memory traffic by 15%, as it delivers compressed cache lines in a fine-grained way, and the cache miss count by 23%, as it stores up to three compressed cache lines in a physical cache line.

Keywords. Parallel processing, processor-memory performance gap, on-chip compressed cache, fine-grained management, internal fragmentation problem.

1 Introduction

As the performance gap between processor and memory has increased by 28-48% every year, the memory system performance typically dominates the whole computer system performance [1]. In order to improve the memory performance, high-end computers are based on large size on-chip caches with a high off-chip memory bandwidth. Although these are effective in improving the memory performance, they are restricted by physical device limits such as the on-chip area and off-chip pin count.

On-chip compressed cache is an alternative approach of improving the memory performance. Compressed caches, e.g. SCMS [2] and CC [3], store and transfer some cache lines in a compressed form, thereby, reducing both the on-chip cache miss count and off-chip memory traffic without having to face the physical limits. The existing compressed caches manage variable-size compressed cache lines in a coarse-grained manner. Specifically, as exemplified in Figure 1, if a cache line can be compressed to less than half of the original size, they treat the cache line size as half of the original size, thereby, incurring internally unused space, namely internal fragmentation. Otherwise, they do not handle the cache line in a compressed form. Thus, at

[1] This work was supported by University IT Research center project in Korea.

most two compressed cache lines can be stored in a physical cache line, and only 1 bit is required to specify the status of a physical cache line whether it embeds two compressed lines or one uncompressed line.

However, their performance gain is often limited by this coarse-grained management. Figure 2 shows a cumulative compression rate distribution of L2 data cache lines on an Alpha machine simulator using the SPEC CPU2000 benchmark suite [7] where the compression rate is defined as the ratio of the compressed data size and the original data size. It shows that over 60% of cache lines are compressed to less than 25% of the original cache line size. This high compression efficiency, mainly due to the frequent value locality [4], strongly suggests that the coarse-grained management is overly conservative.

In order to fully exploit this high compression efficiency, in this paper we present the *Fine-grained Compressed Memory System* (FCMS) based on the four key techniques. First, the FCMS manages compressed cache lines in a fine-grained manner so that it reduces the fragmented space. This implies that the FCMS is more effective in reducing the off-chip memory traffic than the existing compressed caches. Second, based on this fine-grained management, the FCMS stores up to 3 compressed cache lines in a physical cache line in order to further reduce the cache miss count. Unfortunately, this fine-grained management can bring out a large size of metadata. Third, we thus present two additional techniques that limit an increase in the size of both on-chip cache tag address and VM page table without diminishing the obtained performance gain. Fourth, we firstly apply the cooperative parallel decompression technique [5] to on-chip compressed caches in order to reduce the decompression time without harming the compression efficiency significantly.

In order to evaluate the effectiveness of the FCMS, we modified the SimpleScalar [6] simulator and used the SPEC benchmarks. The experimental results show that the FCMS reduces the average execution time by 5% and 12% over the SCMS and a conventional cache system, respectively. In particular, the FCMS reduces the on-chip L2 cache miss count by 23% and 25% and the off-chip memory traffic by 15% and 46% over the SCMS and the conventional system, respectively, in an average case.

The rest of this paper is organized as follows. In Section 2, we provide the organization of the FCMS. The experimental methodology is described in Section 3, while the evaluation results are given in Section 4. We review the related work in Section 5 and conclude this paper with a summary and a future work in Section 6.

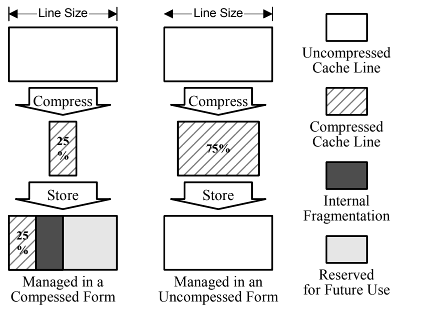


Fig. 1. The existing coarse-grained compressed cache line management.

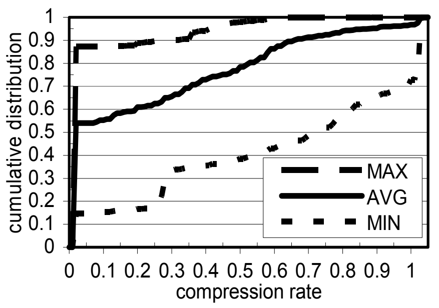


Fig. 2. A cumulative compression rate distribution of on-chip L2 data cache lines.

2 Fine-Grained Compressed Memory System (FCMS)

Figure 3 illustrates the overall cache and memory organization of the FCMS, in which both the on-chip unified L2 cache and the main memory are managed in a compressed form. When a data memory page is firstly loaded into the main memory, all L2 cache lines in the page are individually compressed using a hardware compressor. We use the X-RL compression algorithm [8] because of some of its desirable properties, such as the high compression efficiency with small size data and fast (de)compression speed of at least four bytes per cycle. If a compressed cache line stored in the memory is accessed, the off-chip memory bandwidth can be expanded as the line is transferred to the on-chip cache in a compressed form. Moreover, as the line is stored in the on-chip L2 cache in a compressed form, the effective L2 cache capacity is also expanded.

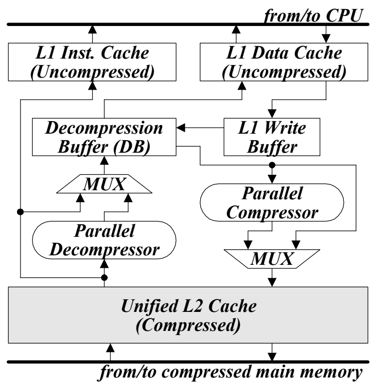
While the line is stored in the L2 cache, the line is concurrently decompressed on the fly using a hardware decompressor in order to deliver the required L1 cache line to the L1 cache. As the remaining L1 cache lines in the decompressed L2 cache line have a high probability of accessing in the near future due to spatial locality, the decompressed L2 line is stored in a decompression buffer which can be accessed in one cycle. The decompression buffer consists of a small fully-associative cache (8 entries [2]) and is managed in the same way as the victim cache does [9]. In this paper, we assume that the decompression buffer can be concurrently accessed with the L1 caches for the fair performance evaluation with the conventional cache systems.

In the FCMS, only data cache lines are managed in a compressed form. As instruction cache lines result in the lower compression efficiency while incurring the larger decompression overhead, the overall cache and memory system performance can be degraded if they are managed in a compressed form. Fortunately, as instruction cache lines are not generally modified at runtime, several off-line code compaction techniques [10] can be used for instruction cache lines for better performance.

2.1. Fine-grained compressed cache line management

The size of compressed cache lines is various. In order to efficiently manage the variable size data, on-chip compressed caches are typically based on a fixed-size allocation method [11]. In this paper, the fixed unit of managing the compressed cache line is called as *cache bucket*, and the *cache bucket unit* (*CBU*) is defined as the ratio of the cache bucket size and the original cache line size. The fixed-size allocation uses several consequent cache buckets for a variable size data. Thus, the last cache bucket can incur internal fragmentation, and the average fragmentation size is equal to half of the cache bucket size.

We can notate 1/2 as *CBU* of the coarse-grained management of the SCMS and the CC. As the CC is targeting for embedded processors, we only use the SCMS as a compressed cache which uses the coarse-grained management scheme. This implies that due to the internal fragmentation the coarse-grained management wastes about 25% of compressed cache space in an average case. In order to reduce the fragmented space, we use the fine-grained compressed cache line management in the FCMS. Thus, the variable-size compressed cache lines can be stored in a more fitting cache bucket while reducing the internal fragmentation. Figure 4 visualizes the fraction of



from/to CPU
L1 Inst. Cache (Uncompressed)
L1 Data Cache (Uncompressed)
Decompression Buffer (DB)
L1 Write Buffer
MUX
Parallel Compressor
Parallel Decompressor
MUX
Unified L2 Cache (Compressed)
from/to compressed main memory

Fig. 3. Overall cache and memory hierarchy organization of the FCMS.

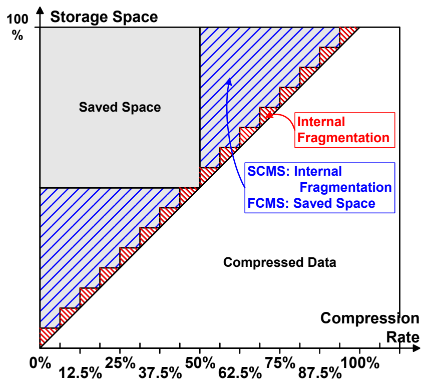


Fig. 4. Space efficiency of the fine-grained management of the FCMS (*CBU*=1/16).

internal fragmentation as a function of the compression rate where *CBU* of the FCMS is assumed to be 1/16. In the figure, white area means the compressed data size, red areas mean the internally fragmented space for both the FCMS and the SCMS, blue areas mean the saved space for the FCMS and the fragmented space for the SCMS, and gray area means the saved space for both the FCMS and the SCMS. Thus, the amount of internal fragmentation space can be greatly reduced in the FCMS.

In the SCMS, cache lines whose compression rate is higher than or equal to 50% do not provide any space-efficiency although they are managed in a compressed form because of the internal fragmentation. On the other hand, in the FCMS, cache lines that have high compression rate of up to $(CBU^{-1}-1)/CBU^{-1}$ % (93.75% if *CBU* is 1/16) can provide performance benefits. However, the provided benefits of the inefficiently compressed lines are quite small, while the compressed lines bring out the decompression overhead. Thus, we use the selective compression technique [2] where a cache line is managed in a compressed form only if its compression rate is less than a specified threshold value (*THRD*). If the threshold value is set to be a lower value, both the performance benefit and the decompression overhead are reduced because only small fraction of cache lines are compressed. Otherwise if the value is set to be a higher one, conversely both the benefit and the overhead are increased.

In the FCMS, maximum CBU^{-1} (16 if *CBU* = 1/16) numbers of compressed cache lines can be stored in a physical cache line when all the compressed cache lines can be stored in a bucket whose size is *CBU*. Actually, in order to do this, the FCMS has to use CBU^{-1} numbers of tag addresses, valid bits, and dirty bits per every physical cache line. However, this requires a large overhead in terms of on-chip area and energy consumption, while typically only a part of the CBU^{-1} tag addresses is used. Thus, the number of tag address (*NTAG*) is another design parameter of the FCMS. In addition, we use LRU as the replacement algorithm of compressed cache lines stored in a physical cache line.

2.2. Metadata size reduction techniques

The fine-grained management at the same time requires a large amount of metadata, lgCBU^{-1} bits (e.g. 4 bits if *CBU* is 1/16), to specify the number of cache buckets used for a compressed cache line, while the coarse-grained management where *CBU* is 1/2 requires only 1 bit for a cache line. When the L2 cache line size is 128 bytes and the memory page size is 4 kilobytes, the fine-grained management requires *4096*/*128** lgCBU^{-1} bits (e.g. 128 bits if *CBU* is 1/16) of metadata per every memory page. This metadata size required by the FCMS is relatively large as compared with the VM page table entry size of about 32 bits.

In order to reduce this metadata size without lowering the granularity of managing compressed cache lines, we use a metadata grouping technique. Specifically, we observed that the number of cache buckets used for all compressed cache lines in an identical memory page is quite similar to each other because the cache lines have relatively similar compression rate mainly due to the spatial locality of data. Therefore, in the FCMS, we only store the maximum number of all cache buckets used for all cache lines in a same memory page as the metadata. Then, the metadata size of the FCMS is only lgCBU^{-1} bits (4 bits if *CBU* is 1/16) per memory page. In this paper, we assume that this small size of metadata can be embedded in the VM page table entry, which typically has some unused bits.

Moreover, if the physical memory capacity is 256 megabytes and cache line size is 128 bytes, the size of tag addresses for a cache line in the compressed cache is *NTAG**lg(256M/128) bits (63 bits if *NTAG* is 3). In order to reduce this tag field size, we use a segmented addressing technique. Specifically, a tag address is divided into a tag segment and a tag offset, and all compressed cache lines stored a physical cache line should have an identical tag segment. With this technique, if the tag offset size is *TOFF* bits, the tag field size per a cache line is only lg(256M/128) – *TOFF* + *NTAG* * *TOFF* bits (36 bits if *TOFF* is 7 bits and *NTAG* is 3), while that for a conventional cache is lg(256M/128) = 21 bits. As the FCMS aims to an L2 unified cache where the cache line size is between 512 bits and 2048 bits, we believe that the additional bits used for tag fields in the FCMS are acceptable design overhead.

2.3. FCMS-based direct-mapped cache organization

Figure 5 shows the compressed L2 cache organization of the FCMS in a direct-mapped scheme. The same organization technique applies for set-associative caches. In this organization, the tag address count (*NTAG*) is 3, the tag offset size (*TOFF*) is 4 bits, and the cache bucket unit (*CBU*) is 1/16. Thus, the tag RAM has three valid bits, three dirty bits, a tag-segment address, and three tag-offset addresses per every cache line, and the data RAM is divided into 16 cache buckets.

When a request is generated, the tag segment and the 3 tag offsets of the requested cache set are concurrently compared with that of the generated address. The results are analyzed by using three OR-gates and an AND-gate so as to determine the hit or miss. Because only three gates are additionally used in the critical path of the FCMS as indicated by the bold lines in Figure 5, we assume that the FCMS-based compressed caches do not cause any additional delay as regarded in the access time. Moreover, we can use a three-input OR-gate instead of the three two-input OR-gates to decide the hit or miss, and this reduces the gate delay as regarded in the access time.

We use lgCBU^{-1} = 4 bits to encode the location of a compressed cache line stored in the data RAM. Since the location of the first compressed cache line is always fixed as 0, we use only 8 bits for the location field. The location information is

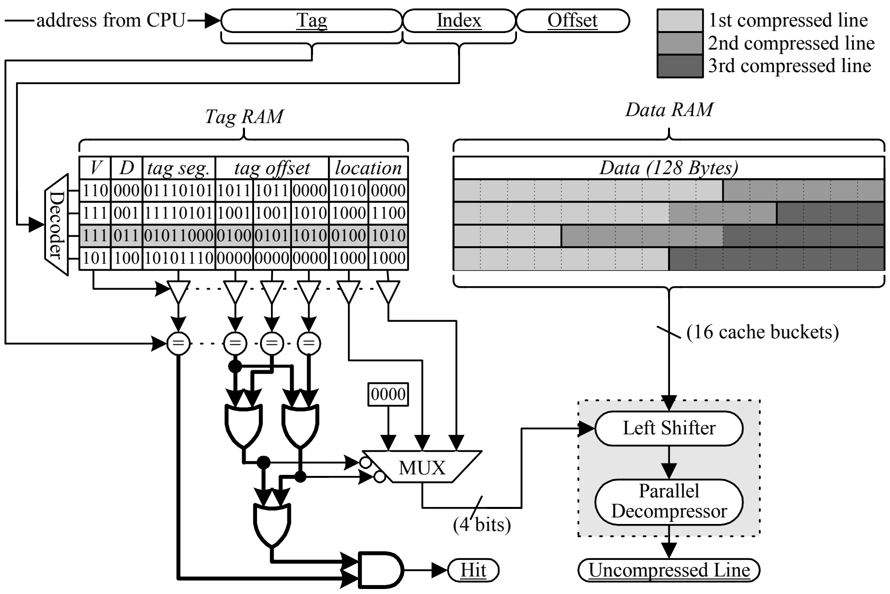


Fig. 5. A fine-grained compressed cache architecture in direct-mapped scheme.

used as an input of the MUX logic, which selects an appropriate one and routes to the left shifter and parallel decompressor logic. The size of the requested compressed cache line is concurrently calculated by subtracting the two adjacent location values.

For example, if the second compressed cache line whose tag area is marked by gray color is accessed, a cache hit occurs and the location data of $0100_{(2)} = 4$ is routed to the left shifter logic, which performs a left shift operation for 4 cache buckets. Also its size is calculated by subtracting its location value from the location value of the third one. The result is $1010_{(2)} - 0100_{(2)} = 6$ cache buckets. Then, the selected cache buckets are routed to the cooperative parallel decompressor logic. Finally, they are delivered to a decompression buffer and an L1 cache in an uncompressed form.

In on-chip compressed caches, the decompression time can seriously degrade the effectiveness of the data compression technique. For example, the X-RL decompressor takes up to 3+*LS*/4 cycles for decompressing a cache line, whose size is *LS* bytes. This means that a smaller compression unit size results in the shorter decompression time. However, it simultaneously incurs the lower compression efficiency. Fortunately, the cooperative parallel decompressor [5] can reduce the decompression time by about 75% while it slightly degrades the compression efficiency. Thus, we use the parallel (de)compressor in the FCMS and evaluate the performance impact.

3 Experimental Methodology

In cache and memory compression systems, the size of both compressed cache lines and memory pages is liable to change after performing a write operation. As a result, the access time of the compressed cache and memory is not fixed but it depends on runtime status. However, it is difficult to reflect this kind of runtime behaviors in trace-driven simulations due to their use of static trace data. Moreover, trace-driven simulations generally do not provide essential operations of superscalar micro-processors such as out-of-order execution, which is used to adaptively cope with this variable memory access time. Thus, we implemented FCMS and SCMS based on an execution-driven simulator of SimpleScalar 3.0 [6]. We mainly modified the on-chip cache, memory bus, and virtual memory modules of the simulator and newly supplied the compression and decompression modules.

Specifically, we used Alpha instruction set architecture, which accurately reflects the high-performance processor architecture. We used *sim-outorder* to quantitatively evaluate the performance of cache and memory systems. Table 1 describes the baseline model used in our experiments. The model follows an aggressive 8-issue out-of-order processor. The cache configuration parameters for the base line model are assumed to be two 32 kilobytes L1 caches and a unified 256 kilobytes L2 cache with four-way associativity and 128 bytes cache line size. We referenced an accurate cache timing model of CACTI for calculating the access time of on-chip caches [12].

We used the SPEC CPU2000 benchmark suite [7] with reference input workload. The benchmark suite is compiled by using the Compaq Alpha compiler with SPEC peak settings. The virtual memory image of this benchmark suite is captured after full execution by using *sim-safe*. For the *sim-outorder* simulations, we used a fast forwarding technique [13] where 1.5 billion instructions are accurately executed after a coarse-grain simulation of 0.5 billion instructions so as to reduce the simulation time without notably compromising the simulation accuracy.

Table 1. Base line model.

Parameter	Value
Processor Core	2.4 GHz (6 x 400 MHz), 0.13 Micron, Alpha ISA, 8 fetch/issue/decode/commit, 128-RUU, 128-LSQ.
Branch Predictor	Bimodal 2K, 512-entry 4-way BTB, 8-entry RAS.
TLB (Inst. / Data)	16 / 32 entry, 4KB page size, 4-way, LRU, 72 cycle latency.
L1 Cache (Inst. / Data)	Each 32KB, 1-way, 32B block, LRU, 1 cycle latency, write-back.
L2 Cache (Unified)	256KB, 4-way, 128B blocks, LRU, 7 cycle latency, write-back.
Main Memory	72 cycle latency, 8 bytes bandwidth, 400 MHz bus clock.

4 Performance Evaluations

In this section, we evaluate the performance of the FCMS over a conventional cache system (CS), a conventional system with a decompression buffer (CSDB), and the SCMS. In CSDB, the decompression buffer is only used for prefetching the L2 cache lines to L1 caches.

First, as shown in Figure 6, we measured the average memory cycles spent to transfer a data cache line where the memory latency is excluded. Because L2 cache line size is 128 bytes and memory bus bandwidth is 8 bytes, both CS and CSDB require 8 memory bus cycles to deliver a cache line. On the other hand, in both SCMS and FCMS, the required bus cycles are significantly reduced as they deliver data cache lines in a compressed form. Although they use the same compression algorithm

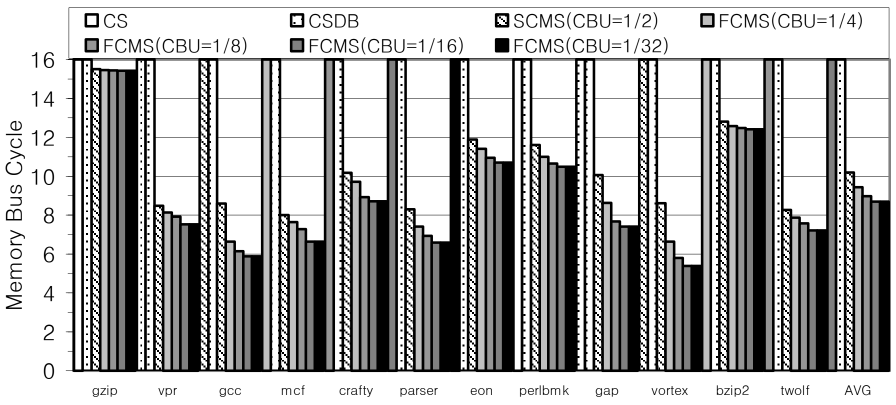


Fig. 6. Average memory bus cycles elapsed to deliver a data cache line.

and the same compression threshold value (50%), the FCMS with a finer-grained cache bucket size uses the smaller memory bus cycles. This is because the fact that the fine-grained management more accurately specifies the actual size of compressed cache lines, thereby, reducing the internal fragmentation. As this tendency is stabilized when *CBU* is larger than 1/16, we set 1/16 as *CBU* of the FCMS. The FCMS where *CBU* is 1/16 requires about 8.7 memory bus cycles, while the SCMS requires about 10.2 cycles in an average case. This means that the FCMS reduce the amount of memory traffic by 15% and 46% as compared with the SCMS and CS, respectively.

Second, we measured the miss count of the on-chip unified L2 cache as shown in Figure 7. The miss count of CSDB is slightly higher than that of CS because its decompression buffer filters several L2 cache accesses and consequently disturbs the reference history of the L2 cache, incurring inefficient cache replacements. The miss count of the SCMS is reduced by about 2% as compared with both CS and CSDB in an average case. This is because in the SCMS two compressed cache lines whose tag addresses are same except for the least significant bit can be stored in a physical cache line. Thus, even cache lines are sufficiently compressed as shown in Figure 6,

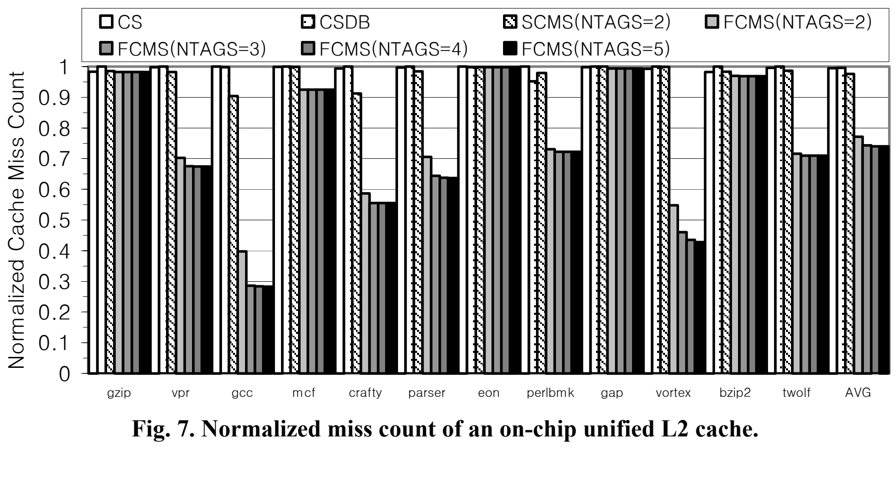


Fig. 7. Normalized miss count of an on-chip unified L2 cache.

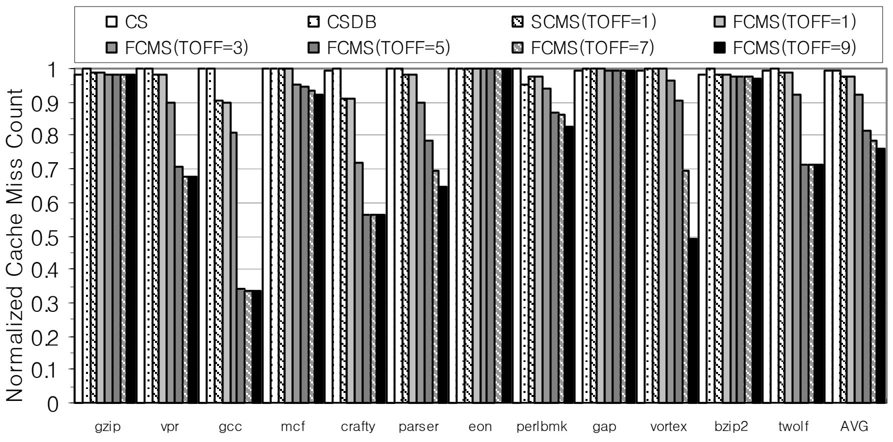


Fig. 8. Normalized miss count of the FCMS as a function of *TOFF* value.

the probability of storing two compressed lines in a physical cache line is quite low. On the other hand, in the FCMS the miss count is reduced by 22%, 25%, 25%, and 26% when the *NTAG* value is 2, 3, 4, and 5, respectively. Thus, we set the *NTAG* to be 3 in the FCMS. For benchmarks *gzip*, *eon*, *gap*, and *bzip2*, the miss count is slightly reduced in both the SCMS and the FCMS because their workload sizes are relatively small, thereby, even if a twice large size cache is used the miss count is seldom reduced.

Third, we measured the cache miss count of the FCMS by changing the *TOFF* value as shown in Figure 8. In the FCMS, the miss count is reduced by 2%, 7%, 18%, 21%, and 23% when the *TOFF* value is 1, 3, 5, 7, and 9, respectively, in an average case as compared with CS and CSDB. The miss count is stabilized when the *TOFF* value is larger than 7 because the accessed cache lines have strong spatial locality, and the address space covered by the tag-offset and cache line size, 2^7x128 = 16 kilo-bytes, is sufficient to cope this locality. Thus, we set the *TOFF* of the FCMS to be 7.

Fourth, we evaluated the average memory access time (AMAT) in order to analyze the decompression overhead of the FCMS and the SCMS. We calculated the code AMAT of these two systems in a similar way of calculating AMAT of CS and CSDB [1]. On the other hand, Eq. 1 is used to calculate the data AMAT of the FCMS and the SCMS. In the formula, *A*, *M*, *C*, and *DO* mean the access time, miss rate, fraction of compressed lines, and decompression cycles respectively, while the small symbols of *L1*, *DB*, *L2*, and *MM* mean the L1 data cache, decompression buffer, unified L2 cache, and main memory, respectively.

$$AMAT_{Data} = A_{L1} + M_{L1}\left[A_{DB} + M_{DB}\left\{ \begin{array}{l} A_{L2} + C_{L2}DO_{L2;avg} + \\ M_{L2}\left(A_{MM} + C_{MM}DO_{MM;avg}\right) \end{array} \right\} \right] \quad (1)$$

Based on this, we measured the data AMAT as shown in Figure 9. It shows that the FCMS reduces the data AMAT by 29%, 14%, 16%, and 8% in an average case as compared with CS, CSDB, SCMS, and the SCMS with the cooperative parallel

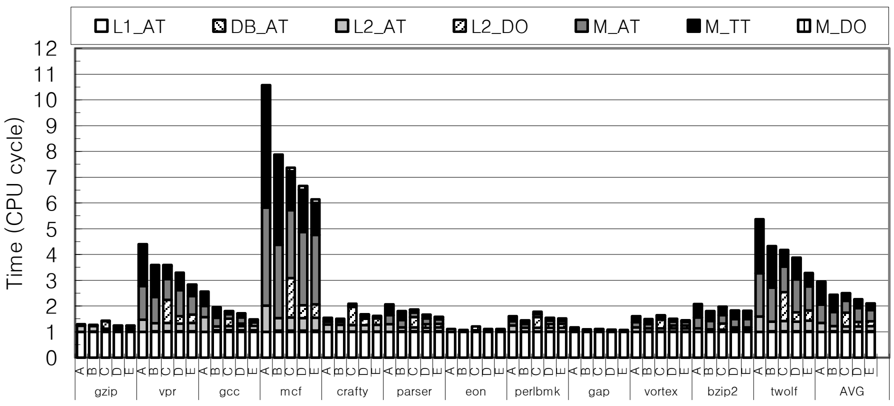


Fig. 9. Average data memory access time (Data AMAT).
*** DB: decompression buffer; MM: main memory; AT: access time; DO: decompression overhead; TT: transfer time. (A: CS; B: CSDB; C: SCMS; D: SCMS-Parallel, E: FCMS)**

decompressor (SCMS-Parallel). Due to the use of parallel decompressor, the decompression overheads (*L2_DO* and *MM_DO*) are significantly reduced in both the SCMS-Parallel and the FCMS. We used the early restarting technique, which provides the ability of accessing the critical word as early as possible without waiting for the complete decompression of the whole cache line, and the decompression time overlapping technique, which overlaps the transfer time of a compressed cache line from the main memory and its decompression time, in order to lessen DO_{L2} and DO_{MM}, respectively. We observed that most of the decompression times are absorbed by the decompression buffer in both the FCMS and the SCMS since the hit ratio of the decompression buffer in the FCMS is over 40% in an average case. We also observed that the FCMS reduces the code AMAT by about 1% as compared with CS, CSDB, the SCMS, and the SCMS-Parallel as it reduces the miss count of the unified L2 cache.

Fifth, we measured the instructions per cycle (IPC) as shown in Figure 10. The average IPC is 1.62, 1.65, 1.72, 1.74, and 1.82 in CS, CSDB, the SCMS, the SCMS-Parallel, and the FCMS, respectively. This implies that the execution time of the FCMS is reduced by 12%, 10%, 6%, and 5% in an average case as compared with CS, CSDB, the SCMS, and the SCMS-Parallel, respectively. In this experiment, the compression threshold of the FCMS is set to be 50%, which is identical configuration to that of the SCMS. If the compression threshold is set to be a higher value, both the performance gain and the decompression overhead are increased. Because of this, we observed that in the FCMS, IPC is slightly influenced by the compression threshold value, and it is maximized when the threshold value is set to be 50%.

Furthermore, when we use 93.75% (=15/16) as the threshold value, IPC of the FCMS is only reduced by less than 1% in an average case. Since a higher threshold value means that a large amount of cache lines is managed in a compressed form and the fine-grained management of the FCMS significantly reduces the fragmented space over the coarse-grained management of the SCMS, the FCMS has a higher potential of expanding the effective main memory capacity than the SCMS. Therefore, we

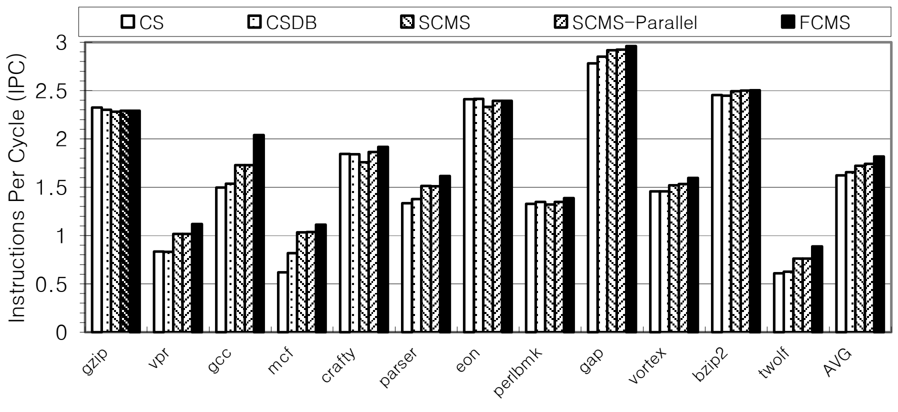


Fig. 10. Instructions per cycle (IPC).

believe that the real improvement in the execution time obtained with the FCMS will be much greater than that we presented in this paper.

5 Related Work

Over the past ten years, several research groups have been studied the on-chip cache and main memory compression systems in order to alleviate the performance gap between processor, memory, and hard disk [14] as well as reduce the energy consumption of memory systems [15] in high-end parallel computers. The existing on-chip compressed caches are typically based on the coarse-grained compressed cache line management because of its simplicity [2, 3]. Although as shown in this paper the coarse-grained management is overly conservative, so far as we know, none has been developed a compressed cache system in this perspective of managing compressed cache lines in a fine-grained manner and storing up to three compressed cache lines in a physical cache line with the appropriate metadata reduction techniques.

6 Conclusion

Recently on-chip compressed caches have been developed to alleviate the processor-memory performance gap in high-end parallel computers. However, we have observed that the performance gain of the existing compressed caches is often limited mainly due to the high compression efficiency of on-chip cache lines. In order to fully exploit the high compression efficiency, in this paper we have presented a novel on-chip compressed cache system based on the four key techniques. First, the proposed system manages the compressed cache lines in a fine-grained manner so that it reduces the fragmented space and consequently reduces the memory traffic over the existing compressed cache systems. Second, based on this, the proposed cache stores up to three compressed cache lines in a physical cache line, thereby, reducing the cache miss count over the existing systems. Third, in order to avoid an increase in the metadata size, the proposed system uses two novel metadata reduction techniques.

Fourth, we firstly have applied a parallel (de)compressor to the on-chip cache systems and have shown the performance impact of using this. The execution-driven simulation results have shown that the FCMS reduces the average execution time by 5% and 12% over the an existing compressed cache system and a conventional cache system, respectively. In particular, the FCMS reduces the on-chip L2 cache miss count by 23% and 25% and the off-chip memory traffic by 15% and 46% over the compressed system and the conventional system, respectively, in an average case.

References

1. J. L. Hennessy, D. A. Patterson, and D. Goldberg, *Computer Architecture – A Quantitative Approach*, 3rd Ed., Morgan Kaufmann Publishers, 2002.
2. J. S. Lee, W. K. Hong, and S. D. Kim, "Design and Evaluation of On-Chip Cache Compression Technology," *In Proceedings of the IEEE International Conference on Computer Design*, pp. 184-191, 1999.
3. J. Yang, Y. Zhang, and R. Gupta, "Frequent Value Compression in Data Caches," *In Proceedings of ACM/IEEE International Symposium on Microarchitecture*, pp. 258-265, 2000.
4. Y. Zhang, J. Yang, and R. Gupta, "Frequent Value Locality and Value-centric Data Cache Design," *In Proceedings of the ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2000.
5. P. A. Franaszek, J. Robinson, and J. Thomas, "Parallel Compression with Cooperative Dictionary Construction," *In Proceedings of the IEEE Data Compression Conference*, pp. 200-209, 1996.
6. T. Austin, E. Larson, and D. Ernst, "SimpleScalar: an Infrastructure for Computer System Modeling," *IEEE Computer*, Vol. 35, Issue 2, pp. 59-67, 2002.
7. J. L. Henning, "SPEC CPU2000: Measuring CPU Performance in the New Millennium," *IEEE Computer*, Vol. 33, Issue 7, pp. 28-35, 2000.
8. M. Kjelso, M. Gooch, and S. Jones, "Design and Performance of a Main Memory Hardware Data Compressor," *In Proceedings of the 22nd EuroMicro Conference*, IEEE Computer Society Press, pp. 422-430, 1996.
9. N. P. Jouppi, "Improving Direct-Mapped Cache Performance by the Addition of a Small Fully Associative Cache and Prefetch Buffers," *In Proceedings of the ACM/IEEE International Symposium on Computer Architecture*, pp. 364-373, 1990.
10. A. Beszedes, R. Ferenc, T. Gyimothy, A. Dolenc, and K. Karsisto, "Survey of Code-Size Reduction Methods," *ACM Computing Surveys*, Vol. 35, No. 3, pp. 223 - 267, 2003.
11. A. Silberschatz, P.B. Galvin, and G. Gagne, *Operating System Concepts*, 6th Ed., pp. 285-287, John Wiley & Sons Inc., 2003.
12. P. Shivakumar and N. P. Jouppi, "CACTI 3.0: An Integrated Cache Timing, Power, and Area Model," *Compaq Computer Corporation Western Research Laboratory*, *Research Report 2001/2*, 2001.
13. I. Gomez, L. Pifiuel, M. Prieto, and F. Tirado, "Analysis of Simulation-adapted Benchmarks SPEC 2000," *ACM Computer Architecture News*, Vol. 30 , No. 4, pp. 4-10, 2002.
14. K. S. Yim, J. Kim, and K. Koh, "Performance Analysis of On-Chip Cache and Main Memory Compression Systems for High-End Parallel Computers," *In Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, pp. 469-475, 2004.
15. L. Benini, D. Bruni, A. Macii, and E. Macii, "Hardware-Assisted Data Compression for Energy Minimization in Systems with Embedded Processors," *In Processing of the IEEE Design, Automation and Test in Europe Conference and Exhibition*, pp. 449-453, 2002.