Performance Analysis of On-Chip Cache and Main Memory Compression Systems for High-End Parallel Computers

Keun Soo Yim, Jihong Kim, and Kern Koh
School of Computer Science and Engineering
Seoul National University
Seoul 151-742, Korea
{ksyim, kernkoh}@oslab.snu.ac.kr, jihong@davinci.snu.ac.kr

ABSTRACT

Cache and memory compression systems have been developed for improving memory system performance of high-performance parallel computers. Cache compression systems can reduce on-chip cache miss rate and off-chip memory traffic by storing and transferring cache lines in compressed form, while memory compression systems can expand main memory capacity by storing memory pages in compressed form. However, these systems have not been quantitatively evaluated on an identical condition, making it difficult to understand the performance of a new system relative to the existing systems. In this paper, we provide an identical execution-driven simulation environment for these systems. To the best of our knowledge, none has been evaluated the performance of cache and memory compression systems by using an execution-driven simulator. Experimental results show that cache compression systems reduce cache miss rate by 16% and memory traffic by 30%, while it expands memory capacity by less than 160%. The results also show that memory compression systems significantly expand memory capacity by over 270%. Based on these experimental analyses, we finally provide future research directions on the compression systems.

Keywords

Performance evaluation, memory hierarchy, cache and memory compression, processor-memory performance gap, I/O bottleneck

1. INTRODUCTION

The performance of a computer system depends mostly on the well-organized memory architectures because the memory system is frequently accessed by the processors. Computer architects usually construct a memory hierarchy where a higher layer has the smaller capacity than the lower layers but it is more quickly accessible. As this memory hierarchy exploits temporal and spatial localities [1], the hierarchical memory systems can afford relatively good performance in a cost-efficient manner.

Unfortunately, throughout over past two decades, the performance gaps between adjacent memory layers have notably increased by every year. For instance, the processor-memory performance gap has increased by 28-48% every year, and the hard disk speed is still five to six orders of magnitude slower than the memory speed [1]. We typically call these gaps as the memory wall and the I/O bottleneck, respectively. These performance gaps can seriously degrade the memory system performance, especially in high-end computers which use a large size of memory workload.

In order to alleviate these performance gaps, modern computer systems are typically based on a large on-chip cache capacity, a wide memory bandwidth, and a large main memory capacity. However, these expansion approaches have physical limitations such as limited on-chip area and narrow off-chip pin bandwidth.

Alternatively, cache and memory compression technologies can logically expand cache capacity, memory bandwidth, and memory capacity by managing data in compressed form. As these are kind of quantitative approaches, they are orthogonal to the existing intelligent cache and memory organizations. Specifically, cache compression systems can alleviate the ever-increasing processor-memory performance gap by expanding effective cache capacity and memory bandwidth [11-15], while memory compression systems can address the I/O bottleneck by expanding main memory capacity [16-23]. However, these systems have not been quantitatively evaluated on an identical condition which is making it difficult to understand the performance of a new system objectively relative to the existing systems.

In this paper, we briefly summarize the overall organization and main contribution of the existing cache and memory compression systems. We then provide an accurate execution-driven simulation environment for the uniform evaluation of these systems. To the best of our knowledge, none has been evaluated and published the performance of cache and memory compression systems by using an execution-driven simulation method. The experimental results show that cache compression systems reduce cache miss rate by 16% and memory traffic by 30%, while it expand memory capacity by less than 160%. The results also show that memory compression systems significantly expand memory capacity by over 270%. Based on these experimental analyses, we finally provide future research directions on the compression systems.

The following is a synopsis of this paper. Section 2 describes the existing memory compression algorithms with their hardware organizations. In Section 3 and 4, we briefly review the existing cache and memory compression systems, respectively. We then describe our execution-driven simulation environment in Section 5, while the evaluation results are given in Section 6. In Section 7, we finally conclude this paper with summary and future research directions.

2. COMPRESSION ALGORITHMS

Essentially memory compression algorithms have to satisfy the following four conditions. First, they must be a lossless algorithm. Second, they have to provide high compression efficiency even though the source data size is small, i.e. less than 4 kilobytes.

Third, they have to (de)compress memory data as fast as possible so as to logically hide the (de)compression overhead in terms of memory access time. Fourth, their hardware organization has to be simple so that they can be used in a practical memory hierarchy.

Dynamic compression algorithms, e.g. Ziv-Lempel (LZ) [2], can satisfy these conditions. These algorithms adaptively organize a mapping dictionary in which the preceding source data is stored based on LRU stack model [10]. Specifically, if the current byte is found in the dictionary, it is a full hit and is encoded with '0' bit and the match location in the dictionary. Then the matched byte in the dictionary is moved to the top of the dictionary. Otherwise when it is missed in the dictionary, the current byte is encoded with a prefix '1' bit. Behind this, the current byte is inserted into the dictionary as a top entry.

In order to further improve the performance of the dynamic algorithms for memory data, several memory compression algorithms, e.g. X-Match [3] and WK [5], have been developed. Both X-Match and WK process memory data in a unit of four bytes, namely word, because the I/O unit size of memory data is either two or four bytes. As the word size is relatively large, these algorithms additionally use a partial hit where the current word partly matches to a word in dictionary. For example, in X-RL, when more than or equal to two bytes of the current word matches a word in the dictionary, it is a partial hit and is encoded with '0' bit, the match location in the dictionary, the match type (means the pattern of matched bytes), and the unmatched literal characters. Similarly, WK algorithm generates a partial hit when low order bits include the least significant bit of the current word match to that of a word in the dictionary. In reverse manner, both X-Match and WK decompress the compressed data.

Rizzo [4] observed that a large fraction of memory data consists of consequent null bytes where the values are zero. To effectively handle the consequent null bytes, X-Match is further mixed with the run-length encoding for null bytes. We call the extended algorithm X-RL (X-Match and Run-Length).

Moreover, it has been reported that values produced by executing instructions exhibit a high degree of value locality [8] and frequent value locality [9]. That is multiple executions of the same instruction often produce the same value. Based on these observations, the frequent value compression technique has been developed which encodes a small number of values that appear frequently during memory accesses while providing the ability to randomly access individual data words in compressed data.

Figure 1 shows the compression rate of these algorithms with code and data memory images of SPEC CPU2000 benchmark [27]. The compression rate means the ratio of the compressed data size and the source data size. It shows that a larger source data size results in the better compression rate, and the rate is generally stabilized when the source data size is larger than or equal to two kilobytes. It also shows that these algorithms provide good compression rate of about 30% for data memory images although the source data size is relatively small of about 256 bytes.

In order to use X-RL in practical memory hierarchy, hardware prototype of X-RL has been developed using FPGA. The X-RL hardware is organized by four pipeline stages with four bytes data bandwidth. It means that after consuming initial three clock cycles, it (de)compresses four bytes in every cycle and decompresses any sequential null bytes in one cycle. Thus, the speed of X-RL hardware is typically as fast as the speed of memory and I/O buses.

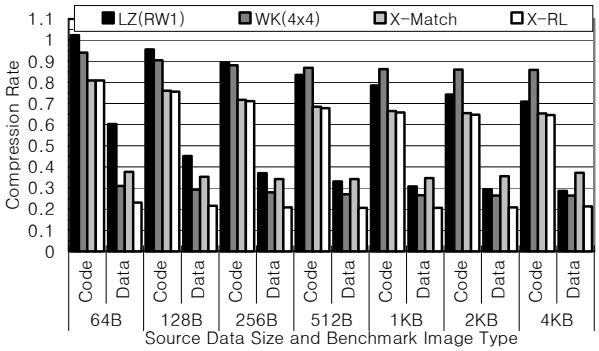


Figure 1. Compression rate of memory compression algorithms.

Unfortunately, if X-RL hardware is used for compressing on-chip cache lines, its decompression time can seriously delay the access time of on-chip caches. Because the (de)compression time of X-RL hardware is directly proportional with the source data size, we can reduce the decompression time by using a small source data size. However, a smaller source data size at the same time results in the lower compression efficiency.

In order to overcome this technical obstacle, Franaszek *et al.* [7] have developed an associative parallel (de)compressor that slightly lowers the compression efficiency, while it significantly reduces the (de)compression time depending on the parallelism degree of four. These parallel (de)compressors are practically used for main memory compression systems.

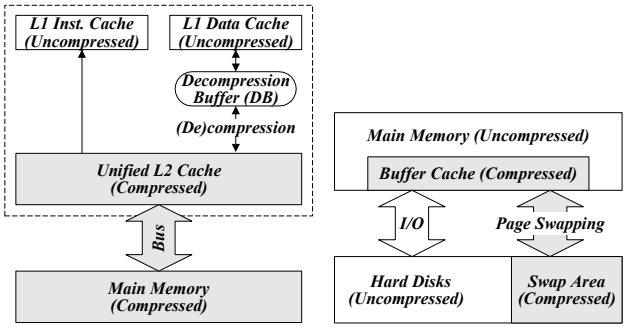
3. CACHE COMPRESSION SYSTEMS

On-chip cache compression systems have been developed so as to alleviate the ever-increasing processor-memory performance gap. These systems are able to reduce the on-chip cache miss rate and off-chip memory traffic by storing and transferring cache lines in compressed form. As compared with the conventional expansion method, which uses a large on-chip cache size and a high off-chip bus bandwidth, these on-chip cache compression systems are an alternative solution that does not have to face physical limits, such as bounded on-chip area and limited off-chip pin bandwidth.

The existing on-chip cache compression systems have been developed on different on-chip cache hierarchies of first-level and second-level caches as described follows.

Lee *et al.* [11] have presented the *Selectively Compressed Memory System* (SCMS) where both on-chip second-level cache and main memory are managed in compressed form. Figure 2(a) shows the overall organization of the SCMS. It shows that the SCMS has two main performance benefits of expanding both the second-level cache capacity and the memory bus bandwidth. In the SCMS, if a compressed cache line is accessed, the cache line has to be all decompressed on the fly from the first word before transferring the requested word to processors. Thus, the decompression time can diminish the benefits obtained from data compression technology although a fast hardware decompressor of X-RL is used.

In order to lessen the decompression overhead, Lee *et al.* used the selective compression technique which means cache lines are managed in compressed form only if the lines are sufficiently compressed. They also used a decompression buffer as a small intermediate cache between first-level and second-level caches.



(a) SCMS (b) CBC

Figure 2. Overall organization of the SCMS and CBC.

The decompression buffer can be concurrently accessed with the first-level data cache in the same way as the victim cache does[13]. As the line size of decompression buffer is identical to the second-level cache line size, it has a prefectching effect to first-level data cache. In addition, a fixed-size allocation method [10] is used for efficient management of variable-sized compressed cache lines.

It has been shown that the SCMS can reduce the average memory access time of SPEC benchmark suite by up to 20% as compared with conventional memory systems. Since the SCMS has mainly focused on alleviating the processor-memory performance gap, it is not effective for expanding the main memory capacity although the main memory is managed in compressed form.

Later, an on-chip first-level cache compression system, namely the *Compression Cache* (CC) [12], has been proposed aiming for embedded micro-processors. The CC uses the frequent value compression method [9] that provides the ability to randomly access individual data words in a compressed cache line. Because of this, the CC does not suffer from the decompression overhead even though it is used for on-chip first-level cache.

However, since various programs have different frequent value localities, and even in a same program the frequent value locality can be changed as a function of the execution path which depends on the runtime environment such as input data. Thus, we need to efficiently handle these dynamics so as to practically use the CC.

Moreover, we believe that the performance impact of the CC would not be great if it is employed in out-of-order processors with a multi-level cache hierarchy because this kind of high-performance processors can hide the short latencies caused by first-level cache misses that hit in the second-level cache.

It has been recently studied that hardware-based data compression technology is able to be effectively used for reducing the energy consumption of on-chip caches [14, 15]. By storing less frequently used cache lines in compressed form and turning off significantly part of on-chip cache area, the compressed caches can reduce the energy consumption while incurring only a small degradation in the cache access time.

4. MEMORY COMPRESSION SYSTEMS

Main memory compression systems can reduce the I/O bottleneck, i.e. page swapping operations [10], by logically expanding the main memory capacity. This section describes the existing main memory compression systems which are based on either software- and hardware-based data compression methods.

4.1 Software-based compression approach

Wilson [16] firstly proposed a memory compression system as a buffer cache of operating systems so as to reduce page swapping operations. Figure 2(b) shows the overall organization of the compressed buffer cache (CBC). When a page swapping operation is required due to the leak of available main memory capacity, the CBC compresses least recently used pages and keeps them in the main memory instead of swapping them out to hard disks. Later, if a page stored in the CBC is accessed, the page is decompressed before being transferred to processors. As the decompression time is usually shorter than the time for page swap-in operation, the CBC can alleviate long I/O latency caused by accessing hard disks.

Douglis [17] has practically implemented the CBC in Sprite OS. He observed speedups for some benchmark programs and slowdowns for others. Then, he concluded that the effectiveness of the CBC depends mostly on program behavior and relative costs of (de)compression and hard disk I/O.

Kaplan *et al.* [5, 18] have presented an adaptive method of controlling the CBC size by reflecting program behavior so as to accomplish consistent performance improvements with various programs. Specifically, the adaptive method retains some recency information for recently evicted pages to perform an on-line cost-benefit analysis. Figure 3 shows an example where the memory size can be larger to 150% of its original size by compressing some memory pages. Here, the cost means the time for performing (de)compression for memory pages that will be retained in the CBC when the cache size is enlarged or decreased to a specific value, while the benefit means the reduced time for page swapping operations due to the reduced buffer cache miss rate by enlarging or decreasing the compressed cache size. If this analysis predicts performance improvement, the adaptive method reconfigures the CBC size dynamically.

Both the static and adaptive CBCs have been implemented on Linux [19, 20]. These studies have shown that the CBCs are a practical method of alleviating the long I/O latencies by reducing page swapping operations. However, there still exist several research topics. One is supporting multi-processor systems as the CBS is more useful for high-performance multi-processor systems. The other is improving the cost-benefit analysis because it may degrade the memory system performance especially when several memory consuming applications are running concurrently.

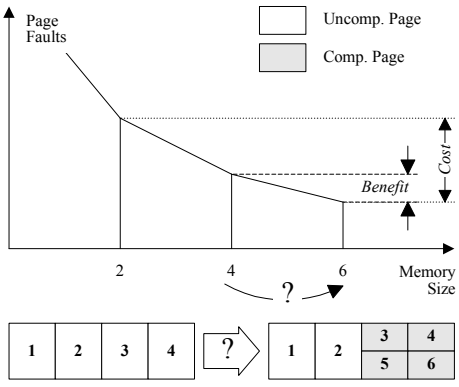


Figure 3. An example of cost-benefit analysis.

4.2 Hardware-based compression approach

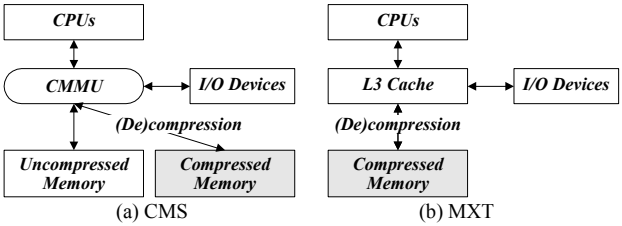
In order to reduce (de)compression cost, hardware-based memory compression systems have been developed. Kjelso *et al.* [21]have proposed the *Compressed Memory System* (CMS), which divides main memory into two exclusive parts, uncompressed and compressed. As shown in Figure 4(a), a compressed memory management unit (CMMU) adaptively manages the compressed memory size in order to provide good performance. If there is a competition for memory resources, the CMMU enlarges the compressed memory size and store less frequently accessed pages in compressed form. If physical memory size is larger than system workload size, the CMMU does not manage memory pages in compressed form as it can delay the memory access time due to decompression. Although a hardware decompressor of X-Match is used for the CMS, decompression overhead is not completely hided as the compressed memory page has to be decompressed from the first word to access the requested cache line.

In general, most applications run 50-20 times slower if half of the required memory size is available [24]. As the CMS can expand main memory capacity significantly, it reduces the execution time of applications by an order of magnitude under heavy memory pressure conditions.

IBM researchers have also developed a hardware-based memory compression system, namely the *Memory Expansion Technology* (MXT) [22, 23]. Figure 4(b) is the overall memory hierarchy organization of the MXT. The MXT uses an off-chip third-level cache as a decompression buffer of the compressed main memory. The off-chip cache and the compressed main memory satisfy the inclusion property in which all data stored in the off-chip cache is a subset of the data stored in the compressed main memory. In practice, the off-chip cache is made of double data rate SDRAM because the MXT is mainly used for multi-processor systems that require a high memory bandwidth. Also the MXT uses a cooperative parallel (de)compressor [7] to reduce decompression time, while slightly lowering the compression efficiency.

The MXT typically expands the memory size by more than twice of its original physical memory size. Because of this, in the MXT, operating systems use a real address space, which is twice larger than physical address space, and a memory controller performs address translation from real to physical address space. The real to physical address translation table is located in a certain part of physical memory. If memory pages are not sufficiently compressed to half of their original size, the memory controller generates an interrupt to the operating system so that it can handle this situation by performing page swap-out operations.

As the MXT expands the main memory capacity notably, it can improve the execution time of applications by a factor of two as compared with conventional memory systems.



(a) CMS (b) MXT

Figure 4. Overview of the CMS and the MXT.

In addition, we believe that in spite of the cost of memory devices has decreased by every year, memory compression systems are still interesting research topic because a system with both larger memory and a memory compression technology is obviously provide better performance than one with only larger memory.

5. EVALUATION METHODOLOGIES

In on-chip cache and main 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 usually do not provide essential operations of superscalar microprocessors such as out-of-order execution, which is used to adaptively cope with this variable memory access time.

Unfortunately, all of the existing cache and memory compression systems have been evaluated by using trace-driven simulations. Thus, we implemented an execution-driven simulator based on SimpleScalar 3.0 [25]. We mainly modified the 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-safe* to capture virtual memory images and *sim-outorder* to quantitatively evaluate the performance of cache and memory systems. The base line 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 first-level caches and a unified 256 kilobytes second-level cache with various associativity degrees and line sizes. We referenced an accurate cache timing model of CACTI for calculating the access time of on-chip caches [26].

We used the SPEC CINT2000 benchmark suite [27] 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. For the *sim-outorder* simulations, we used a fast forwarding technique [28] 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.

6. PERFORMANCE EVALUTIONS

In this section, we provide both the average memory access time of the cache compression systems and the main memory capacity expansion rate of the cache and memory compression systems.

6.1 Processor-memory performance gap reduction

First, we evaluated the average memory access time of the existing cache compression systems. The compressed on-chip cache systems, such as the SCMS, are able to reduce the on-chip cache miss rate by increasing the effective cache capacity, and also expand the memory bandwidth by transferring data in compressed form. Because these features mean that the processor-memory performance gap can be alleviated, the compressed on-chip cache systems can consequently reduce the average memory access time.

Conversely, since most the efficient compression algorithms do not have the ability of randomly accessing a particular byte in compressed data, the decompression time can cause the side-effect not only in the compressed on-chip cache system but also in the main memory compression systems.

We quantitatively analyzed these performance factors of the SCMS using a cycle simulator with popular benchmark suite in an accurate manner. We also compared the performance of the SCMS to that of a conventional memory system (CS), and a conventional memory system with a decompression buffer (CSDB).

Figure 5 illustrates the amount of memory traffics. It shows that the SCMS reduce the code and data memory traffics by 30% in an average case as compared with conventional memory systems. The amount of memory traffic reduction depends heavily on the compression efficiency of the benchmark program workloads. For example, the SCMS reduces the traffic by 42%, 48%, 33%, 44%, 41%, 48%, and 44% for *vpr*, *gcc*, *mcf*, *crafty, parser*, *vortex*, and *twolf*, as their average compression rate of data memory workload is 23%, 28%, 17%, 5%, 28%, 24%, and 28%, respectively.

Then, we analyzed the second-level cache miss rate. Figure 6 shows that normalized cache miss count of a unified second-level cache depending on the benchmark programs and system types. It shows that in CSDB the second-level cache miss rate is slightly increased as compared with the convention system because its decompression buffer filters some second-level cache accesses, so that its second-level cache can not update the recency information of the corresponding cache lines. This can incur some inefficient replacements regardless of the replacement policy.

We also observed that the second-level cache miss rate of the SCMS is reduced by about 16% in an average case as compared with the conventional memory systems. Specifically, the SCMS reduces the cache miss rate by up to 37% as compared with conventional system. The normalized second-level cache miss rate of the SCMS is slightly increased for benchmark *gzip* as compared with the conventional systems. However, as its first-level data cache miss rate is only 1-3%, and its second-level cache misses are increased by less than 1%, this does not seriously degrade the memory access time of the SCMS. On the other, as the SCMS reduces the memory traffic of this benchmark program, the SCMS can even improve the memory access time for this program.

Then, we analyzed the average memory access time of CS, CSDB, and the SCMS for code and data memories as shown in Figure 7 and 8, respectively. The access time for code memory is measured by using the conventional memory access time equation [1], and that for data memory is calculated by using equation 1. Here, *A*, *M*, *C*, and *DO* means the access time, the miss rate, the fraction of compressed lines, and the decompression time, respectively, while the small symbols of *L1*, *DB*, *L2*, and *M* mean the first-level cache, decompression buffer, second-level cache, and main memory, respectively. The results show that the SCMS reduces the average data memory access time by 25% and 8% as compared with CS and CSDB, respectively, in an average case. Likewise, the SCMS slightly reduces the average code memory access time by less than 1% in an average case, as compared with CS and CSDB.

Finally, Figure 9 provides the instructions per cycle (IPC) of CS, CSDB, and the SCMS. This figure shows that the execution time of the SCMS is reduced by up to 67% for benchmark *mcf*. In an average case, the SCMS reduces the execution time by about 14% and 9% as compared with CS and CSDB. In addition, as the SCMS significantly expands the effective memory capacity, we presume that the real reduction in the execution cycles obtained with the proposed memory hierarchy will be much greater that we presented in Figure 9.

$$AMAT = \begin{cases} A_{L1} + \\ M_{L1}A_{DB} + \\ M_{L1}M_{DB}A_{L2} + \\ M_{L1}M_{DB}C_{L2}DO_{L2;avg} + \\ M_{L1}M_{DB}M_{L2}A_{M} + \\ M_{L1}M_{DB}M_{L2}C_{M}DO_{M;avg} \end{cases} \quad (1)$$

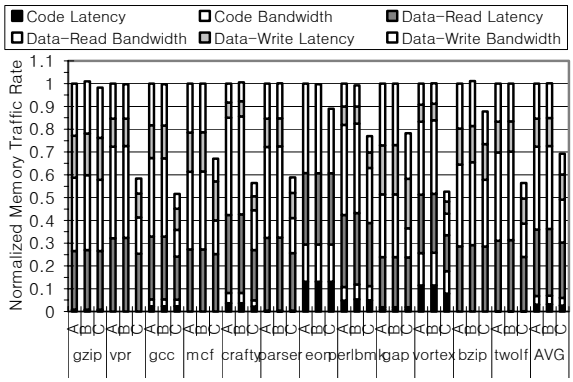


Figure 5. Memory traffics. (A: CS; B: CSDB; C: SCMS)

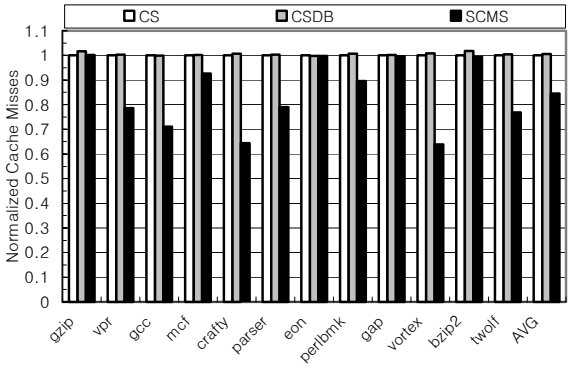


Figure 6. Normalized second-level cache misses.

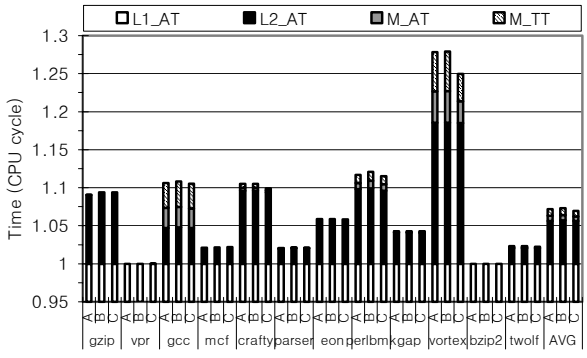


Figure 7. Average code memory access time.

* L1: first-level cache; L2: second-level cache; M: main memory; AT: access time; TT: transfer time. * A: CS; B: CSDB; C: SCMS.

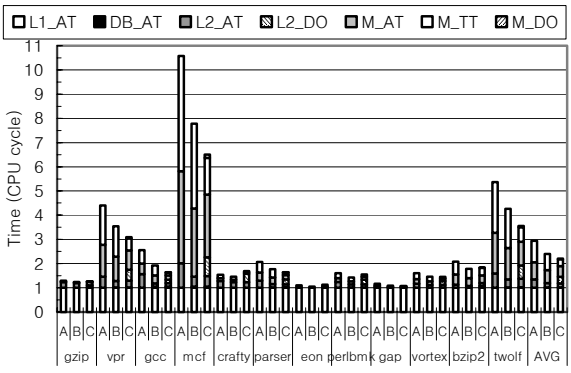


Figure 8. Average data memory access time.

* L1: first-level cache; DB: decompression buffer; L2: second-level cache; M: main memory; AT: access time; DO: decompression overhead; TT: transfer time. * A: CS; B: CSDB; C: SCMS.

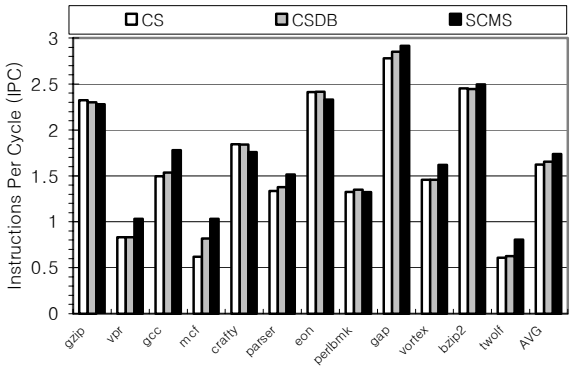


Figure 9. Instructions per cycle (IPC).

6.2 Main memory capacity expansion

In order to measure the expansion rate of main memory capacity, we use the effective compression rate (ECR) as the performance metric, which represents the number of used physical pages over the number of provided logical pages. To precisely compare this metric, we define two additional metrics, namely the compression rate and the internal fragmentation rate. The compression rate (CR) is already defined in Section 2 where a lower compression rate means the better efficiency. The internal fragmentation rate (IFR) is defined as the ratio of the internal fragmentation size to the source block size. Then, the effective compression rate is obtained by adding the average compression rate and the average internal fragmentation rate. We finally define the expansion rate (ER) as the reciprocal of the effective compression rate.

We analyzed the memory expansion rate of the existing cache and memory compression systems of optimal, the CBC, the CMS, the MXT, and the SCMS with the memory images of SPEC CPU2000 benchmark suite. Table 1 summarizes the evaluation results. The optimal system expands the size of data and code memories by 480% and 160%, respectively. The memory compression systems, such as the CBC, the CMS, and the MXT, can expand the code and data memory sizes by 210-300% and 110-140%, respectively, while the compressed on-chip cache system of the SCMS can only expand the data memory capacity by 150-160%.

Table 1. Main memory capacity expansion rate.

System	Comp. Algorithm	Comp. Unit Size	Memory Allocation Unit Size	Data Area			Code Area		
				ER	CR	IFR	ER	CR	IFR
Optimal	X-RL	4KB	-	480	21	0	160	64	0
	LZ			340	29	0	140	71	0
CBC	LZ	4KB	512B	270	29	9	130	71	6
	WK			300	26	6	110	86	6
CMS	X-Match	4KB	512B	230	37	6	140	65	6
			1KB	210	37	11	130	65	10
MXT	LZ	1KB	256B	240	31	11	110	79	12
SCMS	X-RL	64B	2KB	150	23	42	100	80	20
		128B		160	22	41	100	75	25
		256B		160	21	40	100	71	29

* ER: Expansion Rate (%); CR: Compression Rate (%); IFR: Internal Fragmentation Rate (%).

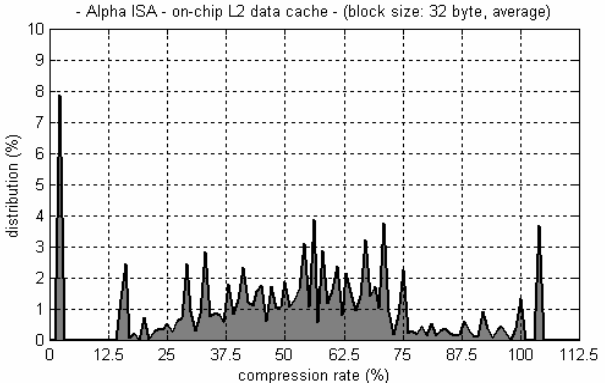


Figure 10. Compression Rate Distribution of Cache Lines.

In the SCMS, the high internal fragmentation rates degrade the memory expansion performance significantly. In Figure 10, we briefly visualize this internal fragmentation problem. The graph means a compression rate distribution of on-chip cache lines on a SimpleScalar/Alpha machine. Although the average compression rate is about 55%, the size of compressed cache lines is varying depending on their compression efficiency. Because of this, the coarse-grained compressed cache line management of the SCMS [11] incurs a large amount of internal fragmentation spaces. For example, when the compression rate is less than 50%, the internal fragmentation space is getting increased as the compression rate is being smaller. Moreover, the internal fragmentation space completely degrades the effectiveness of data compression when the compression rate is higher than 50%. Therefore, the internal fragmentation problem seriously diminishes the benefit obtained from data compression technology in the SCMS.

Table 2 summarizes the performance characteristics of the existing on-chip cache and main memory compression systems. In the table, the decompression layer means the memory layer in which the compressed data are decompressed, and the compressed modules specify the memory modules managed in compressed form. It shows that none of the existing cache and memory compression systems simultaneously accomplishes the two main design goals of alleviating the memory wall and the I/O bottleneck. Therefore, it will be interesting to develop a cache and memory compression system that simultaneously achieves these two design goals.

Table 2. Performance summary.

System	Decomp. Layer	Compressed Modules			Comp. Algorithm
		Cache	Bus	Memory	
CC [9, 12]	CPU	L1	Data	D/C	Hardware (Freq. Value)
SCMS [11]	L1	L2	Data	Aggressive Low	Hardware (X-RL)
MXT [22, 23]	L3	N/A	N/A	Aggressive High	Hardware (Parallel LZ)
CMS [3, 21]	Memory	N/A	N/A	Passive High	Hardware (X-Match)
CBC-Static [17, 19]	Memory	N/A	N/A	Passive High	Software (LZ)
CBC-Adapt. [18, 20]	Memory	N/A	N/A	Passive High	Software (LZ Ext.)

* D/C: Do not consider, N/A: Not available.

7. CONCLUSIONS

As the amount of memory space required by applications has grown by 50-100% every year, modern computer architects have developed the main memory compression technologies for improving the performance but reducing the cost. Recently, on-chip cache compression systems were presented to alleviate the processor-memory performance gap by reducing the cache miss rate and expanding memory bandwidth. In this paper, we have quantitatively evaluated the performance of these systems through accurate execution-driven simulation studies. The experimental results have shown that none of existing cache and memory compression systems sufficiently expands the main memory capacity, while alleviating the processor-memory performance gap. Therefore, we consider that any future works that simultaneously alleviate the processor-memory performance and I/O bottleneck will be interesting especially in the context of high-end computing.

8. REFERENCES

[1] J. L. Hennessy, D. A. Patterson, and D. Goldberg, *Computer Architecture – A Quantitative Approach*, 3rd Ed., Morgan Kaufmann Publishers, 2002.

[2] D. A. Lelewer and D. S. Hirschberg, "Data compression," *ACM Computing Surveys*, Vol. 19, No. 3, pp. 261-296, 1987.

[3] M. Kjelso, M. Gooch, and S. Jones, "Design and Performance of a Main Memory Hardware Data Compressor," *In Proceedings of the 22nd Euromicro Conference*, pp. 422-430, 1996.

[4] L. Rizzo, "A Very Fast Algorithm for RAM Compression," *ACM Operating Systems Review*, Vol. 31, No. 2, pp. 36-45, 1997.

[5] S. F. Kaplan, *Compressed Caching and Modern Virtual Memory Simulation*, Ph.D. Thesis, University of Texas at Austin, 1999

[6] J. L. Nunez and S. Jones, "Loseless Data Compression Programmable Hardware for High-Speed Data Networks," *In Proceedings of IEEE International Conference on Field-Programmable Technology*, pp. 290-293, 2002.

[7] P. A. Franaszek, J. Robinson, and J. Thomas, "Parallel Compression with Cooperative Dictionary Construction," *In Proceedings of the 6th IEEE Data Compression Conference*, pp. 200-209, 1996.

[8] M. H. Lipasti, C. B. Wilkerson, and J. P. Shen, "Value Locality and Load Value Prediction," *In Proceedings of the 5th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 1996.

[9] Y. Zhang, J. Yang, and R. Gupta, "Frequent Value Locality and Value-centric Data Cache Design," *In Proceedings of the 9th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2000.

[10] M. Maekawa, A. E. Oldehoeft, and R. R. Oldehoeft, *Advanced Concepts in Operating Systems*, McGraw-Hill Science/Engineering/Math, 1994.

[11] J.-S. Lee, W.-K. Hong, and S.-D. Kim, "Design and Evaluation of On-Chip Cache Compression Technology," *In Proceedings of the 17th IEEE International Conference on Computer Design*, pp. 184-191, 1999.

[12] J. Yang, Y. Zhang, and R. Gupta, "Frequent Value Compression in Data Caches," *In Proceedings of the 33rd ACM/IEEE International Symposium on Microarchitecture*, pp. 258-265, 2000.

[13] N. P. Jouppi, "Improving Direct-Mapped Cache Performance by the Addition of a Small Fully Associative Cache and Prefetch Buffers," *In Proceedings of the 17th ACM/IEEE Annual International Symposium on Computer Architecture*, pp. 364-373, 1990.

[14] 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.

[15] J. Abella and A. Gonzalez, "Power Efficient Data Cache Designs," *In Processing of the IEEE International Conference on Computer Design*, pp. 8-13, 2003.

[16] P. R. Wilson, "Operating System for Small Objects," *In Proceedings of the International Workshop on Object Orientation in Operating Systems*, pages 80-86, 1991.

[17] F. Douglis, "The Compression Cache: Using On-line Compression to Extend Physical Memory," *In Proceedings of the 1993 USENIX Winter Technical Conference*, pp. 519-529, 1993.

[18] P. R. Wilson, S. F. Kaplan, and Y. Smaragdakis, "The Case for Compressed Caching in Virtual Memory Systems," *In Proceedings of the 1999 USENIX Summer Technical Conference*, pp. 101-117, 1999.

[19] R. Cervera, T. Cortes, and Y. Becerra, Improving Application Performance through Swap Compression," *In USENIX'99 – Freenix Refereed Track*, 1999.

[20] R. S. Castro, A. P. Lago, and D. D. Silva, "Adaptive Compressed Caching: Design and Implementation," *In Proceedings of the 15th Symposium on Computer Architecture and High Performance Computing*, 2003.

[21] M. Kjelso, M. Gooch, and S. Jones, "Performance Evaluation of Computer Architecture with Main Memory Data Compression," *Journal of Systems Architecture*, Vol. 45, pp. 571-590, 1999.

[22] R. B. Tremaine, P. A. Franaszek, J. T. Robinson, C. O. Schulz, T. B. Smith, M. E. Wazlowski, P. M. Bland, "IBM Memory Expansion Technology (MXT)," *IBM Journal of Research and Development*, Vol. 45, No. 2, 2001.

[23] B. Abali, S. Xiaowei, H. Franke, D. E. Poff, and T. B. Smith, "Hardware Compressed Main Memory: Operating System Support and Performance Evaluation," *IEEE Transactions on Computers*, Vol. 50, Issue 11, pp. 1219-1233, 2001.

[24] H. Garcia-Molina, A. Park, and L. R. Rogers, "Performance Though Memory," *In Proceedings of the 1987 ACM SIGMETRICS Conference*, pp. 122-131, 1987.

[25] T. Austin, E. Larson, and D. Ernst, "SimpleScalar: an Infrastructure for Computer System Modeling," *IEEE Computer*, Vol. 35, Issue 2, pp. 59-67, 2002.

[26] 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.

[27] J.L. Henning, "SPEC CPU2000: Measuring CPU Performance in the New Millennium," *IEEE Computer*, Vol. 33, Issue 7, pp. 28-35, 2000.

[28] 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.