A Space-Efficient Virtual Memory Organization for On-Chip Compressed Caches

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

Abstract – *On-chip compressed cache systems have been recently developed that reduce the cache miss count and off-chip memory traffic by storing and transferring cache line in a compressed form. In order to further expand the main memory capacity, in this paper, we present a space-efficient virtual memory organization technique for the on-chip compressed caches. Simulation results show that the proposed organization expands the main memory capacity by 104-150% with the SPEC benchmark suite, which is about twice larger than that achievable with an existing on-chip compressed cache.*[1]

Keywords: Virtual memory, memory capacity expansion, and on-chip compressed cache.

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, modern computer systems are typically based on large size on-chip caches with a high off-chip memory bandwidth. Although these techniques are effective in improving the memory performance, they are restricted by physical 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. By transferring and storing cache lines in a compressed form, the compressed caches such as SCMS [2] and CC [3] reduce both the cache miss count and off-chip memory traffic without having to face the physical limits. Also this implies that the compressed caches can reduce the energy consumption of on-chip caches and memory buses [3, 4].

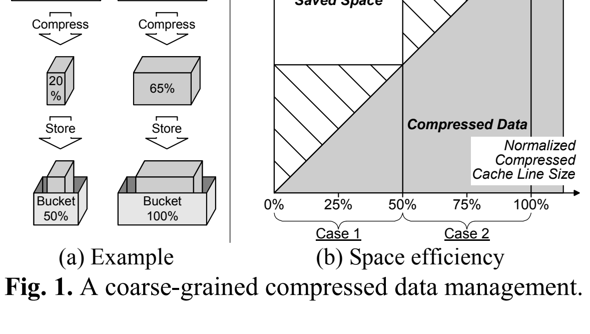
In this paper, we present a space-efficient virtual memory (VM) organization technique in order to further expand the main memory capacity of the compressed caches. If we store the compressed cache lines in main memory, we can expand the effective main memory capacity significantly, and this consequently reduces page swapping operations [5] to storage devices. Since the access time of hard disk is 5 to 6 orders of magnitude slower than main memory devices, if only half of the required memory space is available, application program runs ten to hundred times slower [6]. As the required memory space of applications has grown by 50-100% every year [1], expanding the effective memory capacity is an important factor [7] for improving the memory system performance.

Unfortunately, the existing compressed caches are inefficient at expanding the memory capacity as they manage compressed cache lines in a coarse-grained manner [2, 3]. Specifically, as shown in Figure 1(a), a cache line is managed in a compressed form only if the line can be compressed to less than half of the original size, and then its size is assumed to be half of the original line size. However, this coarse-grained management incurs a large amount of internally fragmented spaces as shown in Figure 1(b). As a memory page consists of many cache lines, the fragmented space can significantly degrade the effectiveness of data compression technique in main memory.

First, in order to address this internal fragmentation problem, we mange the compressed cache lines in a finer-grained manner. However, the fine-grained management requires a large amount of metadata for recording the size of compressed cache lines. Second, we present a metadata grouping technique in order to reduce the metadata size without increasing the fragmented space size significantly.

The SimpleScalar-based [8] simulation results show that the proposed organization expands the main memory capacity by 104-150% with the SPEC benchmark suite [9]. This expansion rate is about twice larger than that achievable with an existing on-chip compressed cache. Nevertheless, the proposed organization only requires 7 bits per memory page as a metadata.

The rest of this paper is organization as follows. The proposed memory organization is described in Section 2, while the evaluation results are given in Section 3. In Section 4, we conclude this paper with a summary.



(a) Example (b) Space efficiency

Fig. 1. A coarse-grained compressed data management.

[1] This research was supported by University IT Research Center Project in Korea.

2 Proposed memory organization

In the proposed memory organization, when a page is loaded into main memory, all cache lines in the page are individually compressed by using a hardware compressor. We use the X-RL (de)compressor because of its good compression rate for small data and fast (de)compression speed of at least four bytes per cycle [7]. If a compressed cache line is accessed, it is delivered to CPU and is stored to an on-chip cache in a compressed form, expanding the effective cache capacity and memory bandwidth in the same way as the on-chip compressed caches do [2, 3, 4].

Our primal design goal is to expand the effective memory capacity significantly, while reducing the extra metadata size. In order to accomplish this goal, the proposed organization is based on the following two techniques.

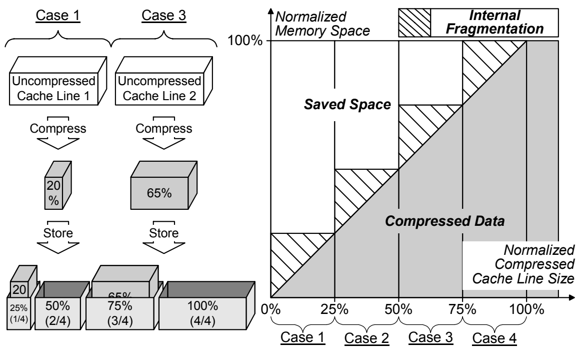
2.1 Enhancement of the memory capacity expansion

We use a fine-grained compressed cache line management. Specifically, if the management granularity is set to be 4, compressed cache lines are handled by the 4 buckets as shown in Figure 2(a). The bucket size is defined as the ratio of the physical bucket size and the cache line size. As shown in Figure 2(b), this fine-grained management reduces the internal fragmentation space by 50% as compared with the coarse-grained management. As this reduction rate is direct proportional to the management granularity, a finer-grained management results in the better memory capacity expansion.

The fine-grained management has additional benefit. That is improving the expansion of memory bandwidth and the reduction of memory bus power consumption as compared with the existing compressed caches. This is because in the fine-grained management a much small amount of internal fragmentation wastes the memory bus bandwidth.

On the other hand, a finer-grained management requires the larger size of metadata for recording the size of compressed cache lines. If the management granularity is MG_{cache}, it uses $\lg MG_{\text{cache}}$ bits per cache line as a metadata.

In main memory, a compressed page size equals to the sum of classified bucket size of all cache lines in the page. Because the compressed page size varies depending on



(a) Example (b) Space efficiency

Fig. 2. Proposed fine-grained compressed cache line management when the management granularity is 4.

the compression rate of cache lines, we also manage the compressed pages by using a fine-grained management in order to reduce the fragmented space. A fine-grained management in main memory requires $2^*\lg MG_{\text{mem}}$ bits per page as a metadata where MG_{mem} is the management granularity. The metadata is used to specify both the size and location of the compressed page. Because a memory page is a superset of cache lines, the parameter MG_{mem} is equal to or smaller than the parameter MG_{cache}.

2.2 Reduction of the metadata size

The total metadata size used for a compressed page is formulated in Eq. 1. The former term means the bits used for the size and location of a compressed page, the latter term means the bits used for the size of all compressed cache lines in the page. In order to reduce this metadata size, we divide a memory page into metadata groups and record only the largest bucket size of compressed cache lines in each metadata group as a metadata. We call this the metadata grouping technique. As formulated in Eq. 2, a larger metadata group size (*MGS*) results in the smaller metadata size.

$$2 \times \lg MG_{mem} + \frac{Page\ Size}{Cache\ Line\ Size} \times \lg MG_{cache} \tag{1}$$

$$2 \times \lg MG_{mem} + \frac{Page\ Size}{Metadata\ Group\ Size} \times \lg MG_{cache} \tag{2}$$

We observed that cache lines stored in close have similar compression rates. For example, when the cache line size is 128 bytes and memory page size is 4KB, the standard deviation of the compression rate of all cache lines in a memory page is 7%. In the experiment, we used the memory image of the SPEC integer benchmark suite.

If *MGS* is 4KB, the compressed page size can be directly calculated from the stored bucket size of the metadata group as formulated in Eq. 3. Thus, the total metadata size can be reduced to $\lg MG_{\text{mem}} + \lg MG_{\text{cache}}$ bits per memory page in this case. This is a significant reduction in the metadata size as compared with the existing memory compression systems which uses about 62 bits per page [7].

$$\lceil Cache\ Bucket\ Size \times MG_{mem} / MG_{cache} \rceil \times Page\ Size \tag{3}$$

2.3 Overall virtual memory organization

Figure 3 shows an example of proposed compressed data management technique where cache line size is 512 bytes, MG_{mem} is 4, MG_{cache} 8, and *MGS* is 4KB. A memory page is divided into 4 sub-pages, and a cache line space is divided into 8 sub-lines (see the dotted lines in Figure 3). Because the largest cache bucket size of the compressed page 2 is 7 over 8, we record 7 as the cache bucket size of the page. Due to this metadata grouping, some cache lines incur internal fragmentation, namely internal cache fragmentation. Then, 4 sub-pages are allocated for the page, incurring internal memory fragmentation. Both the physical page number and the offset of the allocated sub-pages in the physical page (2 bits) are used to specify the location of a compressed page.

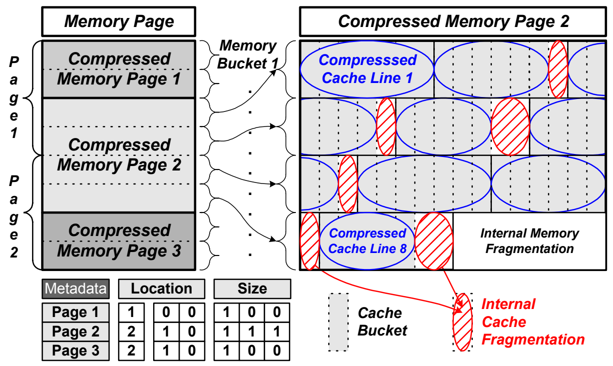


Fig. 3. An example of proposed compressed memory page management when the page size is 4KB, cache line size is 512B, MG_{mem} is 4, MG_{cache} is 8, and MGS is 4KB.

The VM page table entry retains the extra metadata of $\lg MG_{\text{mem}}$ + $\lg MG_{\text{cache}}$ bits per page. Note that we set MG_{mem} and MG_{mem} to be 8 and 16, respectively, in Section 3. Since the entry size is typically 4 bytes, the proposed organization does not cause any additional delay at fetching the entry if the memory bus bandwidth is higher than or equal to 5 bytes. When the entry is fetched, the location bits are stored in TLB and the size bits are managed in the memory controller. Thus, the TLB entry size has to be increased by 3 bits, and this does not delay the access time of TLB typically.

The virtual-to-physical address translation procedure is exemplified in Figure 4. A virtual address is translated into the real address which uses 8 times larger memory space than the physical address space and is only used in on-chip caches. If an L2 cache miss occurs, the memory controller translates the real address into physical address in order to fetch the requested cache line from the main memory as shown in the figure.

Compressed data size can be changed after performing a write operation. When a compressed page size is enlarged and physically adjacent sub-pages are already used, the memory controller has to reallocate new sub-pages for the page. When the memory module is busy, we delay this reallocation operation by storing the enlarged data to a reallocated buffer, retained in the controller. When the module is idle, the controller performs the reallocation operation in partial cooperation with page-level allocator of OS [5], and this is common design complexity

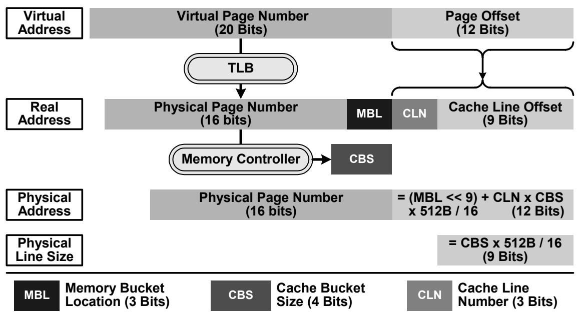

Virtual Address
Virtual Page Number (20 Bits)
Page Offset (12 Bits)
TLB
Real Address
Physical Page Number (16 bits)
MBL
CLN
Cache Line Offset (9 Bits)
Memory Controller
CBS
Physical Address
Physical Page Number (16 bits)
= (MBL << 9) + CLN x CBS x 512B / 16 (12 Bits)
Physical Line Size
= CBS x 512B / 16 (9 Bits)
MBL Memory Bucket Location (3 Bits)
CBS Cache Bucket Size (4 Bits)
CLN Cache Line Number (3 Bits)

Fig. 4. Address translation procedure when L2 cache line size is 512 bytes, MG_{mem} is 8, and MG_{cache} is 16.

in hardware-based memory compression systems [7]. Furthermore, we believe that a greedy algorithm can be used to reduce the frequency of page reallocation operations in such as way of allocating new sub-pages as far as from the previously allocated sub-pages.

3 Performance evaluations

We used the SimpleScalar 3.0 [8], a CPU simulator, with the SPEC CPU2000 integer benchmark suite [9]. We captured the memory image of the benchmark suite after a full execution with the reference input workload. The compression rate of the data and code images is 21-22% and 65-76%, respectively, with the X-RL algorithm. We observed that a larger compression unit size results in the better compression rate, and this tendency is stabilized when the unit size is larger than 512 bytes.

In order to measure the memory expansion rate precisely, we use the effective compression rate (ECR) as a performance metric. ECR represents the size of used physical memory pages over the size of provided logical memory pages. Also the compression (CR) and internal fragmentation rate (IFR) is used as a metric. CR is defined as the ratio of the compressed data size and the original data size, and IFR is defined as the ratio of the internal fragmentation size and the original data size. IFR is used for both internal cache and memory fragmentations. Then, ECR is obtained by adding the average CR and the average IFR. We finally define the memory expansion rate (MER) as the reciprocal of ECR minus 1.

First, we optimized the design parameters of the proposed organization by using the captured data memory images. Figure 5 shows that the internal cache fragmentation rate is similar to half of the reciprocal of MG_{cache}. Based on this result, we set MG_{cache} to be 16. Similarly, we set MG_{mem} to be 8 by considering the result shown in Figure 6. Figure 7 show that a larger MGS results in the higher internal cache fragmentation rate. Based on this result, we set the MGS to be 4KB. Note that when MGS is 1KB, 2KB, and 4KB, the metadata size is 18 bits, 14 bits, and 7 bits, respectively, per memory page.

Second, we measured the memory expansion rate of the proposed organization in comparison with the existing cache and memory compression systems [7]. As summarized in Table 1, the maximum expansion rate is 245-376% and 41-56% for data and code memories,

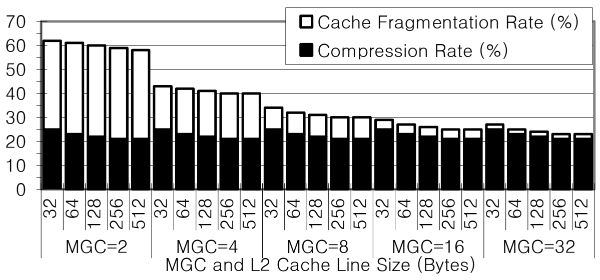


Fig. 5. The measured ECR of data memory images as a function of MG_{cache} (MGC) and L2 cache line size.

respectively, when there is no internal fragmentation. The expansion rate of the proposed organization is 104-150% and 0-12% for data and code memories, respectively, while that of SCMS, an on-chip compressed cache, is only 54-64% and 0%, respectively, as SCMS incurs a larger amount of internal fragmentations. Thus, the memory expansion rate of proposed organization is about twice larger than that of SCMS.

As compared with the hardware-based memory compression systems such as CMS and MXT [7], the proposed organization provides better expansion rate for data memory if its compression unit size is higher than 256 bytes. In contrast, a software-based memory compression system of CBS [7] provides the higher memory expansion rate than the proposed organization as it uses a larger compression unit size, reducing CR significantly. However, the

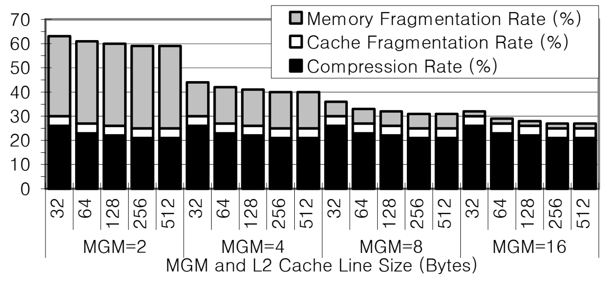


Fig. 6. The measured ECR as a function of MG_{mem} (MGM) and L2 cache line size where MG_{cache} is 16.

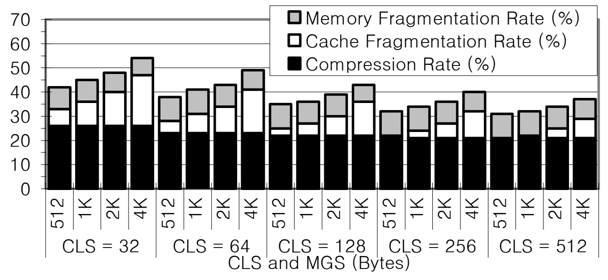


Fig. 7. The measured ECR as a function of L2 cache line size (CLS) and MGS where MG_{cache} is 16 and MG_{mem} is 8.

Table 1. Memory expansion rate.

System	Comp. Algorithm	Comp. Unit Size	Data	Code
			MER (CR,IFR)	MER (CR,IFR)
Optimal	X-RL	4KB	376% (21%, 0%)	56% (64%, 0%)
	LZ(RW)		245% (29%, 0%)	41% (71%, 0%)
Proposed	X-RL	64B	104% (23%,26%)	0% (80%,20%)
		128B	133% (22%,21%)	0% (75%,25%)
		256B	150% (21%,19%)	12% (71%,18%)
SCMS	X-RL	64B	54% (23%,42%)	0% (80%,20%)
		128B	59% (22%,41%)	0% (75%,25%)
		256B	64% (21%,40%)	0% (71%,29%)
CBC	LZ(RW)	4KB	163% (29%, 9%)	30% (71%, 6%)
	WK(4x4)		212% (26%, 6%)	9% (86%, 6%)
CMS	X-Match	4KB	133% (37%, 6%)	41% (65%, 6%)
			108% (37%,11%)	33% (65%,10%)
MXT	LZ(1)	1KB	138% (31%,11%)	10% (79%,12%)

* MER: Memory Expansion Rate, CR: Compression Rate, IFR: Internal Fragmentation Rate.

software-based compression system incurs a long decompression time, and both software- and hardware-based compression systems are not able to alleviate the processor-memory performance gap in the same way as the proposed organization and SCMS do.

4 Conclusion

We have designed and evaluated a space-efficient VM organization for on-chip compressed cache systems. First, the proposed organization manages compressed data in a fine-grained manner, reducing the fragmented space. Second, a metadata grouping technique is used to reduce the size of additional metadata, making the proposed organization applicable to the real memory hierarchy. The simulation results have shown that the proposed organization expands the memory capacity by twice larger than an existing on-chip compressed cache.

For future work, we plan to analyze the energy characteristics of the proposed organization. We believe that the proposed fine-grained management has a high potential of reducing the energy consumption of both on-chip compressed cache and memory buses against the existing on-chip compressed cache systems.

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 (ICCD)*, pp. 184-191, 1999.

[3] J. Yang, Y. Zhang, and R. Gupta, “Frequent Value Compression in Data Caches,” *In Proceedings of the IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 258-265, 2000.

[4] L. Benini, D. Bruni, A. Macii, and E. Macii, “Hardware-Assisted Data Compression for Energy Minimization in Systems with Embedded Processors,” *In Proceedings of the 5th IEEE Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pp. 449-453, 2002.

[5] U. Vahalia, *UNIX Internals*, *The New Frontiers*, Prentice-Hall Inc., 1996.

[6] H. Garcia-Molina, A. Park, L. R. Rogers, “Performance Though Memory,” *In Proceedings of the ACM Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, pp. 122-131, 1987.

[7] 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 (PDPTA)*, pp. 469-475, 2004.

[8] T. Austin, E. Larson, and D. Ernst, “SimpleScalar: an Infrastructure for Computer System Modeling,” *IEEE Computer*, Vol. 35, No. 2, pp. 59-67, 2002.

[9] J. L. Henning, “SPEC CPU2000: Measuring CPU Performance in the New Millennium,” *IEEE Computer*, Vol. 33, No. 7, pp. 28-35, 2000.