Design and Evaluation of a Selective Compressed Memory System

Jang-Soo Lee, Won-Kee Hong, and Shin-Dug Kim
Parallel Processing System Laboratory
Department of Computer Science, Yonsei University
134 Shinchon-Dong, Seodaemun-Ku, Seoul 120-749, Korea
{jslee, wkhong, sdkim}@kuren.e.yonsei.ac.kr

Abstract

This research explores any potential for an on-chip cache compression which can reduce not only cache miss ratio but also miss penalty, if main memory is also managed in compressed form. However, decompression time causes a critical effect on the memory access time and variable-sized compressed blocks tend to increase the design complexity of the compressed cache architecture. This paper suggests several techniques to reduce the decompression overhead and to manage the compressed blocks efficiently, which include selective compression, fixed space allocation for the compressed blocks, parallel decompression, the use of a decompression buffer, and so on. Moreover, a simple compressed cache architecture based on the above techniques and its management method are proposed. The results from trace-driven simulation show that this approach can provide around 35% decrease in the on-chip cache miss ratio as well as a 53% decrease in the data traffic over the conventional memory systems. Also, a large amount of the decompression overhead can be reduced, and thus the average memory access time can also be reduced by maximum 20% against the conventional memory systems.

1 Introduction

As the processor-memory performance gap increases every year, long memory access latencies are becoming the primary obstacle to improve the performance of computer systems. Modern processor designers have attempted to lessen the processor-memory performance gap by using a large space of on-chip cache memory and other aggressive techniques to tolerate the long memory access latencies. However, simply increasing the amount of on-chip cache space results in a large die area and high fabrication cost, and using the latency-tolerance techniques has introduced the limited pin bandwidth as another bottleneck for improving system performance. These two issues, i.e., growing memory access latencies and limited off-chip bandwidth are addressed more seriously under the memory requirement trend that the average amount of memory required by applications has been grown by 50%~100% every year [1].

This research explores any potential of an on-chip cache compression technique which can both reduce cache miss ratio by increasing the effective memory space, and improve the off-chip bandwidth by transferring data in compressed form, if main memory is also managed in compressed form. However, in this approach, when the processor requests a word within a compressed data block stored in the compressed cache or main memory, the compressed block has to be all decompressed on the fly and then the requested word is transferred to the processor. This decompression time causes a critical effect on the memory access time and offsets the compression benefits. Another problem associated with the compressed memory system is that compressed blocks can be generated with different sizes depending on the compression efficiency. Therefore, the length of any compressed block can be rather longer than that of its source block in the worst case. In addition, when a compressed block is decompressed, modified, and re-compressed, its new compressed block cannot be stored in the old position in the compressed memory if its length is longer than that of the old compressed block. Above two problems are called data expansion and fat write problems respectively [2] and they increase the design complexity of the compressed cache architecture.

Several compression techniques have been considered to reduce program size for embedded systems [3, 4, 5], to increase available memory resources by main memory compression [6, 7], and to improve effective off-chip bandwidth by moving compressed addresses, data, and/or code [8, 9]. Unfortunately, there has not been any published paper addressing on-chip cache compression by which compressed data are stored in on-chip cache memory. Only the report in [2] tried to pull up compression technique to the cache

This work was supported by the Korea Research Foundation (1997-001-E00397) made in the program year of 1997.
memory, but it does not provide performance improvement due to the decompression overhead and the cache management problem for fat write.

Some approaches have proposed high performance lossless compression/decompression algorithms [10, 2] and/or their hardware implementations [11, 12, 13]. Among them, X-Match and X-RL [13] compression algorithms offer the most promising combination of throughput, compression ratio, and hardware simplicity. Especially, X-RL algorithm is an extended version of X-Match algorithm to append run-length encoding, so that it can offer much lower compression ratio especially on the memory data with consecutive zeros frequently [14], where compression ratio is defined as the ratio of compressed data length to the source data length. Therefore, our proposed architecture and the simulation results are based on this X-RL algorithm.

This research suggests several techniques to reduce the decompression overhead and to manage the compressed blocks efficiently, which include selective compression, fixed space allocation method, parallel decompression, and the use of decompression buffer. And the compressed cache architecture based on the above techniques and its management method are proposed. The results from trace-driven simulation show that this approach can provide around 7%~90% decrease in the on-chip cache miss ratio as well as a 9%~95% decrease in the data traffic over the conventional memory systems depending on SPEC95 benchmark programs. Furthermore, parallel decompression techniques and decompression buffer can reduce the decompression overhead significantly, and thus the average memory access time can also be reduced by maximum of 20% over the conventional memory systems.

2 Proposed compressed memory system

Proposed compressed memory system is characterized by the selective compression, fixed space allocation for compressed data, and several methods to reduce or hide the decompression overhead. This section presents these characteristics and the overall organization of proposed architecture.

2.1 Characteristics of proposed system

Selective compression technique: Selective compression means that a data block is compressed only if its compression ratio is less than a specific compression threshold value. This technique can reduce the decompression overhead because decompression process is required only for the compressed data blocks. In addition, data expansion problem can be easily solved by compressing only well-compressible blocks. As the threshold value becomes lower, the average compression ratio of whole data blocks increases, so that the compression benefits may be reduced. However, the measurement of tradeoff between decompression time and compression ratio over the variation of threshold value shows that the selective compression technique can provide performance advantage by reducing the impact of decompression overhead without meaningful loss of compression ratio. In this paper, the compression threshold of 0.5 is chosen because this approach can offer a simple and effective way for designing compressed memory system as well as managing compressed data.

Fixed space allocation for compressed data: Our selective compressed memory system uses a fixed space allocation method to manage variable-sized compressed blocks efficiently, where the same size of memory space is allocated for all compressed blocks. This method requires only a small amount of additional information and offers the efficient access mechanism of compressed blocks as in conventional memory system. Also, the fat write problem is also alleviated because the compressed block rewritten in the memory can always be stored in its old position, if its size don’t exceed the allocated space. However, this method may decrease the gains of compression technique because the fixed space is allocated uniformly for compressed blocks even with low compression ratio, causing internal fragmentation. Note that both the fixed space allocation and the selective compression methods are chosen to improve processing time and management efficiency at the cost of the loss of space.

Modification of the compressor and decompressor: The X-RL compressor and decompressor are modified to support parallel decompression and fast decompression in a special case where consecutive zeros are compressed. The key idea to design the parallel decompression is that a source block is split into two sub-blocks, where they are compressed independently and byte-interleaved each other, resulting in one compressed block. However, two compressed sub-blocks cannot be fully interleaved because their sizes may be different depending on the degree of similarity among symbols. Therefore, one byte header is attached to indicate the larger sub-block and the number of interleaved couples in the compressed block. For the decompression, the header information is decoded and each couple within the compressed block is split into two bytes that are fetched by two processing units in parallel. If all couples are fetched, then remaining bytes are fetched by only one processing unit corresponding to the header information. Consequently previous and next uncompressed sub-blocks are generated in parallel and the total decompression time is dependent on the time taken to decompress the larger sub-block. Another modification is to initialize the output buffer within the decompressor with zeroes before processing each compressed block. Decompression mechanism designed in [13] is performed through several pipeline stages, and thus
it takes some pipelining overhead cycles to decompress the first tuple (4-byte symbol) and then each of following tuples is decompressed in a cycle. If the output buffer can be initialized by all zeros in advance, all consecutive zero tuples can be decompressed at one time using simple shift operations by which the output position for the next tuple is shifted by the number of zero tuples. Therefore, the decompression time can be reduced for the compressed blocks containing many zero tuples.

Decompression overhead hiding methods: Decompression overhead can be hidden as follows. First, when the requested data in compressed form is decompressed, the processor is allowed to access it as early as possible without waiting for the completion of decompression for a whole block. Second, the decompression time for any compressed block moved from the off-chip memory can be hidden by overlapping it with its transfer time. Finally, if a compressed block is requested, it is decompressed and its result is sent to the processor at the same time it is stored in the decompression buffer which is a fully set associative cache. After that, if another request occurs on the same compressed block, the decompressed block in the decompression buffer is accessed without re-decompression process. Therefore decompression overhead can be reduced according to the hit ratio of decompression buffer. Moreover, decompression buffer can offer the partial benefits of data prefetching effects.

2.2 Overall organization and data flow model

As the compressed data are stored at the higher level of memory hierarchy, more benefits can be achieved by the compression technique, but at the same time a larger portion of the data access time is wasted to decompress the compressed data. Therefore, our approach is based on the on-chip two level cache system, where L1 cache is managed as the conventional one, but L2 cache, main memory can store the compressed or uncompressed data based on the selective compression technique and fixed space allocation method.

This approach saddles the compiler and operating system with the responsibility for the initial selective compression, and then on-line decompression and re-compression processes are performed by an on-chip compressor and a decompressor at the fast clock cycle of processor. The compiler performs selective compression for an executable file in post-compilation process and generates compression information indicating whether each block is compressed or not. And, if memory allocation is needed during the program execution, the operating system allocates the compressed memory space instead of total requested memory space and thus only a half amount of the requested space is allocated because the allocated memory is initialized by filling them with zeros. In all cases, two block sizes of the L2 cache are regarded as a compression unit.

Figure 1 shows the overall organization of the compressed memory system and data flow model. When CPU performs any data access, the requested data item is searched from the L1 cache, decompression buffer, L2 cache, and main memory one after another. When a miss occurs at L1 cache and the requested data item is found in the decompression buffer, its corresponding L1 block is copied into the L1 cache at the same time it is sent to the CPU. If the requested data item is not in the decompression buffer but in the L2 cache or main memory in compressed form, the compressed block is decompressed by the decompressor first. The decompressed block is stored into the decompression buffer and the requested block with L1 block size is sent to the L1 cache and CPU simultaneously. If the uncompressed block is accessed from L2 cache or main memory, it is managed as conventional memory system. The comparator decodes the compression information of a block transferred from off-chip main memory and determines whether it has to be passed on to the decompressor or sent to the CPU directly.

When a modified L1 block is replaced, it can be modified in the L2 cache as the conventional memory system if the corresponding L2 block is in uncompressed form. However, if it is in compressed form, its uncompressed block is searched and modified in the decompression buffer first. When a miss occurs in the decompression buffer, the compressed L2 block is decompressed and modified in the output buffer of the decompressor. Then, the modified L2 block in the decompression buffer or decompressor is transferred to the compressor, and re-compressed selectively resulting in new compressed block or two uncompressed blocks. In the L2 cache, if a compressed block is transferred, the original compressed block is updated and if not, the uncompressed blocks are stored separately in two locations including the original location.
The main objective of decomposition buffer is to reduce the decomposition overhead guaranteeing the inclusion property with the L2 cache. Therefore, one fundamental rule for managing the decomposition buffer is that it can store only blocks which are compressed in L2 cache. When a compressed L2 block is updated in uncompressed form or replaced with other blocks, the corresponding block in the decomposition buffer is invalidated. For this, there is one invalidation bit for each entry within the decomposition buffer. Consistency between the decomposition buffer and L2 cache is maintained automatically because it is updated first before the L2 cache.

### 2.3 Compressed cache architecture

If the compressed L2 cache and main memory are managed by the selective compression and fixed space allocation method, then one L2 block space can store either an uncompressed L2 block or a compressed block containing two L2 blocks. Therefore, a flag bit indicating whether a compressed block is stored or not, has to be maintained for each block space. This information is stored in tag memory for the compressed L2 cache and page table for the compressed main memory. This paper does not describe management method for compressed main memory due to the space limitation.

Suppose that a specific program consisting of eight L2 blocks (B0~B7) is to be allocated on the conventional main memory as in Figure 2. If they are compressed selectively, five blocks with L2 block size are allocated on the main memory based on the fixed space allocation method. Let’s consider that a simple compressed cache has only two sets with 2-way associativity and the CPU requests L2 blocks as in the order shown in time arrow. If an uncompressed block enters in the compressed cache, it is stored in the set corresponding to its index as the conventional case. However, a compressed block can be stored in any one of two adjacent sets because two L2 blocks in the compressed block show one bit difference between their indices. This results in the effect of doubled associativity for compressed blocks. When a compressed block enters, a free block space is searched first in the set corresponding to the index of requested L2 block (denoted as the first set). If there is no room in the first set, then the other adjacent set (denoted as the second set) is searched for free block space. The number of second set can be obtained by toggling the LSB (least significant bit) of the original index. As shown in the above example, after B4 is requested, the compressed block B4-5 is stored in the second set (set 1) at time T0 because its first set (set 0) is full. If there is not any free block space in two sets, any block space in the first set is replaced with this compressed block using LRU (Least recently used) replacement algorithm. Thus if B6 is requested, the compressed block B6-7 is stored in the first set (set 0) pushing out B0-1 at time T1. The proposed compressed cache architecture relaxes the fat write problem because any modified and re-compressed block with lower compression ratio than 0.5 can be stored in the original location as in the case of block B7. If the compression ratio for any modified block is higher than 0.5, the fat write occurs and the block is de-compressed and split into two uncompressed blocks, one of which is stored in the previous set and the other is stored in the adjacent set. For example, if fat write occurs for B4-5 at T2, the uncompressed block B5 is stored in the previous set and block B2 is replaced with B4 in Set 0 by LRU.

In general, the physical structure of conventional n-way set associative cache consists of n banks. In the case of compressed cache, a pair of sets has to be searched in parallel for any request because a compressed block can be stored in anyone of two different sets. Therefore, each bank of the compressed cache is divided into two sub-banks as shown in Figure 3, like 4-way set associative cache. Sub-bank 0 and sub-bank 1 for each bank are constructed as the even sets and the odd sets respectively. Then, four sub-banks are searched simultaneously for a given address index except for LSB. In the sub-banks containing the first set, only tag comparison is needed for searching the requested block. However in the sub-banks containing the second set, the tag
is compared and also the flag must be checked for confirming the compressed block. So the LSB of a given address index, which is denoted as sub-bank bit, is used to control the flag checking in each sub-bank. If a hit occurs after searching, its corresponding block is transferred to the decompressor or CPU in accordance with the flag bit via router. This approach makes it possible to access compressed data rapidly as in conventional cache.

### 3 Performance evaluation

In this section, simulation methodology is described and the performance of our selectively compressed memory system (SCMS) is compared with that of conventional memory systems (CMS) with architectural variation from the baseline model.

#### 3.1 Methodology

Trace-driven simulation is performed through four phases. First, X-RL compression/decompression algorithms are implemented by S/W. Second, a trace file is generated for the chosen benchmark programs using Shade Tool, which can trace both the address and the data associated with each memory reference instruction. The number of memory accessing instructions contained in the trace is limited by about 268 millions because the data traces collected make the size of a trace file very large. Third, virtual memory space is constructed by placing each of data at its corresponding address and an information table is generated after the compression/decompression processes, where this table contains the length of each compressed block and its decompression time. Especially, two information tables are generated twice before and after processing of store instructions, resulting in a load information table and a store information table respectively for simulating the fat write phenomenon. Finally, the SCMS simulator designed for the on-chip 2 level cache system with selectively compressed L2 cache is executed using the trace file and information tables.

A basic architectural model for simulation is based on the 500 MHz microprocessor. Cache configuration parameters for the baseline model are assumed as 8KB L1 cache with 2-way associativity and 32B block size, and 128KB L2 cache with 2-way associativity and 128B block size. L1 and L2 caches are based on the LRU replacement, write-allocate, and write-back policies. And, the off-chip DRAM main memory in the baseline model is assumed to be connected to the processor through a 64-bit wide bus operated at 166MHz. In addition, it is assumed that the baseline model for SCMS has 8-entry decompression buffer and dual processing units in the decompressor.

The access time for L1 cache access and decompression buffer is assumed as one CPU cycle and The pipelining overhead taken in the decompressor is assumed as 5 cycles. To determine the L2 cache hit time for each model, CACTI model [15] have been used under the consideration of 0.25 µm process rule parameter, 32 bit address width, and 64 output data width. It is assumed that the hit time of the compressed cache is similar to that of conventional cache with double associativity but is rather increased by 10% due to the delay in router determining the path of output data. Main memory access penalty are assumed as 5 CPU cycles control overhead and 100 ns access time. In addition, it is assumed that memory space is so large enough to accommodate the required data.

Eight SPEC95 benchmarks are chosen for the simulation. Table 1 shows the benchmark programs with their inputs, reference behavior, and compression characteristics (X-RL algorithm, compression unit size = 256 Bytes).

<table>
<thead>
<tr>
<th>Spec95 benchmark</th>
<th>Input data (version)</th>
<th>Total mem. ref. count</th>
<th>% of load</th>
<th>% of store</th>
<th>Data set size (KB)</th>
<th>% of Comp. block</th>
<th>Avg. comp. ratio (%)</th>
<th>Avg. decomp. time (cycles)</th>
<th>% of reference to comp. block</th>
</tr>
</thead>
<tbody>
<tr>
<td>go</td>
<td>9stone21.in (ref)</td>
<td>268434432</td>
<td>80.9</td>
<td>99.3</td>
<td>268434432</td>
<td>268434432</td>
<td>24.7</td>
<td>13.9</td>
<td>75.2</td>
</tr>
<tr>
<td>li</td>
<td>boyer.lsp (ref)</td>
<td>120867904</td>
<td>70.4</td>
<td>29.6</td>
<td>381</td>
<td>99.9</td>
<td>87.5</td>
<td>13.9</td>
<td>75.2</td>
</tr>
<tr>
<td>perl</td>
<td>jumble.in (train)</td>
<td>126834432</td>
<td>99.9</td>
<td>78.5</td>
<td>23142</td>
<td>69.5</td>
<td>15.4</td>
<td>15.5</td>
<td>80.9</td>
</tr>
<tr>
<td>vortex</td>
<td>vortex.raw (ref)</td>
<td>268434432</td>
<td>80.9</td>
<td>87.5</td>
<td>11736</td>
<td>96.4</td>
<td>13.9</td>
<td>99.9</td>
<td>70.1</td>
</tr>
<tr>
<td>swim</td>
<td>swim.in (train)</td>
<td>268434432</td>
<td>34.2</td>
<td>24.7</td>
<td>14928</td>
<td>29.1</td>
<td>80.9</td>
<td>91.3</td>
<td>78.5</td>
</tr>
<tr>
<td>applu</td>
<td>applu.in (ref)</td>
<td>268434432</td>
<td>85.4</td>
<td>23142</td>
<td>29123</td>
<td>20.0</td>
<td>87.6</td>
<td>13.9</td>
<td>70.1</td>
</tr>
<tr>
<td>turb3d</td>
<td>turb3d.in (ref)</td>
<td>268434432</td>
<td>70.4</td>
<td>80.9</td>
<td>7062</td>
<td>99.9</td>
<td>5.8</td>
<td>15.5</td>
<td>80.9</td>
</tr>
<tr>
<td>wave5</td>
<td>wave5.in (ref)</td>
<td>268434432</td>
<td>74.0</td>
<td>26.0</td>
<td>25248</td>
<td>0.1</td>
<td>99.9</td>
<td>13.9</td>
<td>70.1</td>
</tr>
</tbody>
</table>

Table 1. Benchmark programs with their inputs, reference behavior, and compression characteristics (X-RL algorithm, compression unit size = 256 Bytes).
pression ratio. Next two columns are the average compression ratio and the average decompression time for all referenced data blocks respectively. It is shown that go, vortex, and turb3d show low compression ratio, but wave5 shows highest compression ratio. Especially turb3d shows highest compression performance in the points of both compression ratio and decompression time. The final column shows the percentage of the number of references to the compressed blocks over the total number of memory references. In the case of wave5, most of total memory reference instructions tend to access the compressed blocks according to the very small portion of total data blocks.

### 3.2 Simulation results

Figure 4 shows the on-chip cache miss ratio and the amount of data traffic for five architectural models including the baseline CMS and SCMS. Other three CMS models have architectural variations on the L2 cache, i.e., doubled associativity, doubled block size, and doubled cache size. Simulation results are normalized by the result of the baseline CMS. As shown in the left graph, the miss ratio of the baseline SCMS is always lower than that of the baseline CMS. Moreover, on go, perl, turb3d, and wave5, the baseline SCMS provides similar or lower miss ratio than the CMS with architectural enhancement. Main reason of low miss ratio by the SCMS is that the compressed L2 cache can provide the effects of increasing space and associativity and blocksize for compressed blocks. In terms of the amount of data traffic, more benefits can be provided. For the most of benchmarks except for li, the data traffic of the baseline SCMS is always lower than that of all CMS models. This result is mainly because the off-chip access frequency is reduced by the compressed L2 cache and the amount of data transferred from the off-chip memory is also reduced when the compressed block is accessed. Especially, in the case of CMS, though doubled block size offers lower cache miss ratio but it rather increases the data traffic. However, the SCMS can reduce the on-chip cache miss ratio without increasing the data traffic.

The degree of benefits achieved from the SCMS is influenced mainly by the memory access pattern of any program, that is, the frequency of references to the compressed blocks and its compression efficiency. By considering the correlation with Table 1, the lower compression ratio and the higher percentage of references to the compressed blocks, the more reductions of on-chip cache miss ratio and data traffic are achieved. In the case of li, the miss ratio and data traffic are decreased significantly on the CMS with doubled L2 cache size. The reason is that the size of data set is so small that most of required data can be stored in the large L2 cache. However, in the case of SCMS, the gains by compression techniques are smaller than other benchmarks due to the lower percentage of references to the compressed blocks. swim and applu show relatively lower impact by the compression due to its poor compression efficiency. On the other hand, though wave5 shows low compression efficiency, but its high percentage of references to the compressed blocks offers the compression benefits sufficiently. Consequently, SCMS approach can provide around 7%~90% (average 35%) decrease in the on-chip cache miss ratio as well as a 9%~95% (average 53%) decrease in the amount of data traffic against the CMS depending on the benchmark programs.

Figure 5 shows the AMATs of five CMS models and four SCMS models for each of benchmarks. Model A is corresponding to the baseline CMS and model B, C, D, and E are the CMS models with doubled L2 cache associativity, doubled L2 cache block size, doubled L2 cache size, and doubled bus width respectively. Model H is the baseline SCMS with 8-entry decompression buffer, and model G and I are corresponding to the SCMSs where the number of entries within the decompression buffer is 4 and 16 respectively. Another model F has the same configuration as the baseline SCMS but its decompressor does not use the parallel decompression mechanism. On all CMS and SCMS models, three CPU cycles for the L2 cache access time are equally obtained from the CACTI simulator. The AMAT of the CMS can be divided into L2 cache access time (L2_AT), main memory access time (M_AT), and the data transfer time (DTT), where L1 cache access time is hidden because the Y axis starts from one cycle. In the SCMS,
Figure 6. Decompression overhead reduction ratio over the number of entries within decompression buffer (Baseline SCMS).

the decompression buffer access time ($B_{AT}$), decompression time for a compressed block accessed from L2 cache ($L2_{DO}$), and decompression time for a compressed block accessed from off-chip memory ($M_{DO}$) are included in the AMAT.

As shown in the results, the SCMS reduces both the main memory access time and data transfer time but these gains are rather decreased by the decompression overhead. However, $M_{DO}$ occupies a small portion of total AMAT because most of decompression time for the compressed blocks moved from the main memory can be hidden by being overlapped with its transfer time. On the other hand, the decompression time for the compressed blocks in L2 cache causes an important effect on the performance of the SCMS especially for perl and vortex. However, this decompression overhead is also reduced as the parallel decompression technique is used and the number of entries within the decompression buffer increases, and thus total AMAT is also reduced. Exceptionally on li, model F with a single decompression unit shows lower AMAT than other SCMS models in spite of the increased decompression overhead. This is because the header attached to each compressed block is not needed for sequential decompression and its removal increases the number of compressed blocks significantly, thus the gains by compression are larger than the increased decompression overhead. If model H is compared with model F, the parallel decompression technique can reduce the decompression overhead by 3%~85% (average 30%) for each of benchmark programs.

To show how much decompression overhead can be reduced by the decompression buffer, its reduction ratio over the number of entries within the buffer was simulated. As shown in Figure 6, the reduction ratio of the decompression overhead increases in proportional to the number of entries, so that in the case of 32 entries, the reduction ratios are shown higher than 80% for the most of benchmarks except for turb3d and vortex. However, if the number of entries is over 16, the degree of increase for the average reduction ratio is less than before. Therefore, if the implementation cost is considered, 8 entries are suitable for the decompression buffer because it can reduces the decompression overhead by 70% on the average. The reason for turb3d and vortex to show low performance of decompression buffer is that most of data blocks are stored in compressed form due to its much lower compression ratio, and also the entries of decompression buffer are replaced frequently.

For most of benchmarks except for li, vortex, the AMATs of the baseline SCMS (model H) are lower than those of not only the baseline CMS (model A) but also all other CMS models with architectural enhancement. In fact, additional cost required for implementing the SCMS is much smaller than that for doubling the L2 cache size or bus width of conventional system. According to the rough analysis of H/W implementation cost, the SCMS needs additional tag memory, comparators, decompression buffer, compressor/decompressor, and other interface logics, but its overall cost corresponds to only about a quarter of the total cost for increasing the conventional cache size from 128KB to 256KB. Therefore, these results imply that SCMS can offer high performance with low cost and small die area.

The SCMS can reduce the L2 access time by prefetching effect because the decompression buffer is accessed in a short cycle time before the L2 cache. The performance improvement of go is mainly achieved by this effect. In the cases of perl, turb3d, and wave5, the reduction of DTT is a major factor for performance improvement. Unfortunately, vortex shows a little performance improvement in spite of significant reduction of on-chip cache misses and data traffic because the decompression buffer does not reduce the decompression overhead sufficiently as shown in Figure 6. turb3d is also corresponding to this case, but good performance improvement is achieved because the compression benefits by the high compression efficiency are much larger than the decompression overhead. Consequently, the SCMS with 8-entry decompression buffer can reduce the average memory access time of the CMS by 1%~20% (average 7%) for each of benchmark programs.

Figure 7 shows the AMATs for the CMS and SCMS over the variation of L1 and L2 cache sizes. Total twelve combinations are simulated varying the L1 cache size from 8KB to 32 KB and the L2 cache size from 128KB to 2MB. It is

<table>
<thead>
<tr>
<th>L2 cache size</th>
<th>CMS model</th>
<th>SCMS model</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Hit time (ns)</td>
<td>Access time (CPU cycles)</td>
</tr>
<tr>
<td>128 KB</td>
<td>4.079</td>
<td>3</td>
</tr>
<tr>
<td>256 KB</td>
<td>4.381</td>
<td>3</td>
</tr>
<tr>
<td>512 KB</td>
<td>5.376</td>
<td>3</td>
</tr>
<tr>
<td>1 MB</td>
<td>6.528</td>
<td>4</td>
</tr>
<tr>
<td>2 MB</td>
<td>9.059</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 2. Hit times and required CPU cycles over the variations of L2 cache size, which are based on 500 MHz processor.
In addition, for buffer does not hide the decompression time sufficiently. They are accessed more frequently, but the decompression compressed blocks are stored in the larger L2 cache and contained in on-chip caches and there is little possibility of the size of L2 cache is over 256KB on occasion buffer, more than 70% of the decompression overhead can be reduced. In terms of the average memory access time, performance improvement by the SCMS is dependent upon the compression efficiency, the ratio of references to the compressed blocks, namely memory access pattern of references, and the ratio of references found in the decompression buffer. The SCMS with 8-entry decompression buffer can reduce the average memory access time of the conventional memory systems by maximum 20% and, for the most of benchmarks, it can provide the lower average memory access times with lower cost than the conventional memory systems with some architectural enhancement for the L2 cache. Especially, the SCMS is a more attractive approach for the future computer systems because it offers high performance in the cases of long DRAM latency and limited bus bandwidth. In addition, because the simulation is performed under no consideration of any advantage by using the compressed main memory, i.e., the reduction of page faults and data loading times, more performance improvement can be expected actually if those factors are considered.

4 Conclusions

This research proposes the selective compressed memory system based on the selective compression technique, fixed space allocation method, and several techniques for reducing the decompression overhead. The proposed system can provide on the average 35% decrease in the on-chip cache miss ratio as well as on the average 53% decrease in the data traffic. And if the SCMS has 8-entry decompression buffer, more than 70% of the decompression overhead occurred in the SCMS without any decompression buffer can be reduced. In terms of the average memory access time, performance improvement by the SCMS is dependent upon the compression efficiency, the ratio of references to the compressed blocks, namely memory access pattern of references, and the ratio of references found in the decompression buffer. The SCMS with 8-entry decompression buffer can reduce the average memory access time of the conventional memory systems by maximum 20% and, for the most of benchmarks, it can provide the lower average memory access times with lower cost even than the conventional memory systems with some architectural enhancement for the L2 cache. Especially, the SCMS is a more attractive approach for the future computer systems because it offers high performance in the cases of long DRAM latency and limited bus bandwidth. In addition, because the simulation is performed under no consideration of any advantage by using the compressed main memory, i.e., the reduction of page faults and data loading times, more performance improvement can be expected actually if those factors are considered.

References