RC 21662 (Log 97617) 4 February 2000 Computer Science/Mathematics

Research Report

Cache-Memory Interfaces in Compressed Memory Systems

Caroline Benveniste, Peter Franaszek, and John Robinson
IBM Research Division
T.J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598
{*cdb, paf, robnson*}*@us.ibm.com*

Research Division
Almaden · Austin · Beijing · Haifa · T.J. Watson · Tokyo · Zurich

Cache-Memory Interfaces in Compressed Memory Systems

Caroline D. Benveniste, Peter A. Franaszek, and John T. Robinson
IBM Research Division, T. J. Watson Research Center
P. O. Box 218, Yorktown Heights, NY 10598
{*cdb, paf, robnson*}*@us.ibm.com*

February 2, 2000

Abstract

We consider a number of cache/memory hierarchy design issues in systems with compressed random access memories (C-RAMs), in which compression and decompression occur automatically to and from main memory. Using a C-RAM as main memory, segments of main memory are stored in a compressed format, and dynamically decompressed to handle cache misses at the next higher level of memory. Design of main memory directory structures and storage allocation methods in such systems are described elsewhere; here we focus on issues related to cache-memory interfaces. In particular, if the cache line size (of the cache or caches to which main memory data is transferred) is different than the size of the unit of compression in main memory, bandwidth and latency problems can occur. Another issue is that of guaranteed forward progress, that is ensuring that modified lines can be written to the compressed main memory so that the system can continue operation even if overall compression deteriorates. We study several approaches for solving these problems, using trace-driven analysis to evaluate alternatives.

1. Introduction

We consider some design issues in systems with main memories that incorporate automatic compression and decompression to and from main memory. In such systems, the bulk of the main memory contents are maintained in compressed form, with decompression occurring on cache misses, and compression occurring on cache writebacks. Figure 1 shows a block diagram of such a system. We study a system with 3 levels of caching, L1 and L2 caches on each processor, and an L3 shared among the processors. Main memory is denoted by L4.

This approach is significantly different than that of related work in this area (for example, see [1], [2], [3]). As illustrated in Fig. 2, in these alternative approaches, main memory is partitioned into compressed and uncompressed regions, and the OS and applications access only the uncompressed region directly, using a standard memory architecture. Data is moved from the compressed region to the uncompressed region via paging mechanisms in these approaches. In contrast, in the system illustrated by Fig. 1, potentially all of main memory can be compressed, and under normal operation the fact that main memory is compressed is relatively transparent to the OS and applications. This approach is made feasible by very fast hardware compression/decompression of relatively small amounts of data, for example by using parallel compression with shared dictionaries (see [4], [5]).

In the systems we consider here, the real memory space at any point in time is logically divided into a sequence of fixed size segments (of size 512 or 1024 bytes, for example) which are the units of compression. The L4 memory controller uses a compression translation table (shown as the CTT in Fig. 1) as a directory into physical memory to find the location of the compressed contents of each such logical unit of real memory. Issues associated with the design of the L4 directory and the mapping of variable size compressed units of real memory to storage allocated dynamically in physical memory are discussed in [6].

The L3 line size may in some systems, due to bus latency and bandwidth considerations, be smaller than the unit of compression (for example, the L3 line size could be 64 or 128 bytes). As discussed in more detail in Section 2, this can have a performance impact on L3 misses and writebacks. The objective of this study is to design an interface between L3 and L4 which minimizes the penalties due to L3 misses and writebacks of modified L3 lines.

Another issue is *guaranteed forward progress*, that is ensuring that there is enough free (unused) physical memory in the L4 so that, at any point in time, it is possible to write back all modified L3 lines even when these lines result in a lower compression ratio. This problem is made worse when the L3 line size is different than the unit of compression, since in the worst case, for example, all L3 lines could be modified and reside in different units of compression.

The study is based on trace data, in particular on traces of bus transactions in 4-way Pentium machines running TPC-C, SAP, and NotesBench workloads. Each trace consists of approximately 80 million Pentium bus request/response records, and was used to construct a sequence of L2 misses, L2 writebacks, and processor stores. Reconstructed L3 misses and writebacks were used to obtain data on different L3/L4 interfaces and their effects on miss

rates to the L4. L2 misses, writebacks, and stores were used to study issues such as writeback clustering.

Section 2 presents an overview of the system. Section 3 considers methods for reducing performance penalties due to L3 misses. These include caching of directory entries and maintaining some amount of data in uncompressed form. Section 4 addresses issues associated with writebacks. A property of these, distinct to systems with compressed main memory, is that a writeback of part of the contents of a unit of compression may require that the unit be decompressed, then modified and recompressed. For this reason we studied the benefits of "batching", i.e. writing back all modified pieces of any given unit of compression. This is also related to the problem of minimizing the amount of free physical L4 memory required to guarantee forward progress. Section 5 provides some concluding remarks.

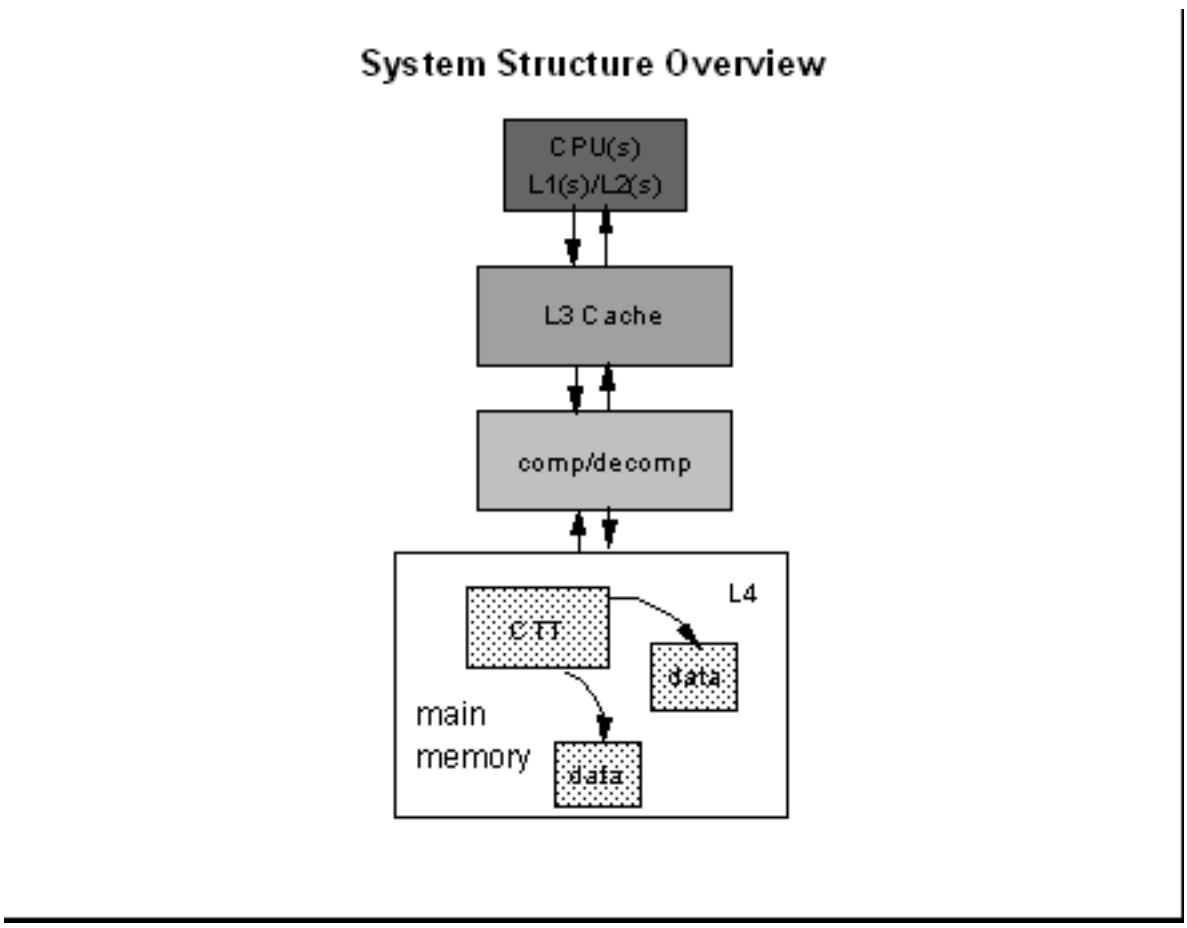
System Structure Overview
CPU(s)
L1(s)/L2(s)
L3 Cache
comp/decomp
L4
CTT
data
main
memory
data

Figure 1.

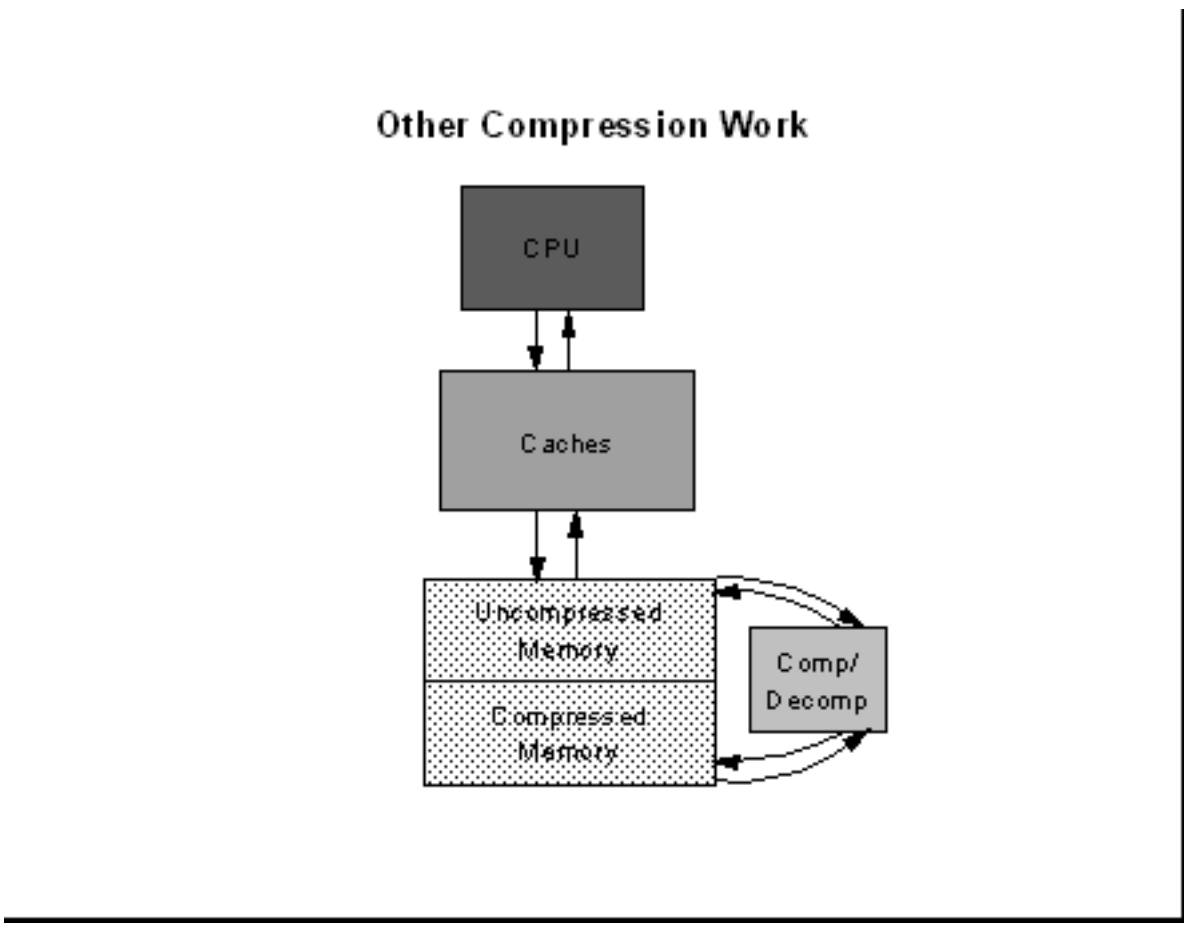
Other Compression Work
CPU
Caches
Uncompressed Memory
Compressed Memory
Comp/ Decomp

Figure 2.

2. System Architecture and Simulation Methodology

To obtain our performance results, we simulated systems as illustrated in Figures 3, 4, and 5. In all cases the unit of compression is logically a 1K segment of the real memory space, aligned on a 1K boundary; thus four consecutive units of compression aligned on a 4K boundary form a logical real page. In each case the L3 line size is 64 bytes. The fact that the L3 line size is significantly smaller than the unit of compression leads to potential performance problems. For example, a sequence of misses to 64 byte lines in the same 1K line could require decompressing the same 1K line repeatedly. Conversely, a writeback of a 64 byte line could require decompression of a 1K line in order to make the update, followed by recompression of the same 1K line. The architectures illustrated by Fig.s 3, 4, and 5 are designed to alleviate these problems.

In Fig. 3, a partition of memory is set aside to hold recently uncompressed data; this paritition is referred to as the compression cache. A directory of the contents of the compression cache is maintained in the memory controller; in order to reduce the size of this directory, sets of 4 1K lines in the same page are stored contiguously. Thus, a directory entry is required in the memory controller only for the beginning of this 4 line set. Note that in this design, each line stored in the compression cache is stored twice: once in uncompressed form, and again outside the compression cache in compressed form.

Fig. 4 illustrates an alternative design in which each logical 1K line of real memory is stored once, either in compressed or uncompressed form. The lines stored in uncompressed form consist of some number of recently accessed lines. The directory entries (from the CTT) for a recently accessed subset are cached in the memory controller. The area shown as the VUC ("virtual uncompressed cache") in Fig. 4 consists of all lines which are stored in uncompressed form. The VUC is managed using a FIFO algorithm. For convenience of illustration, the VUC is shown as a contiguous area, however in practice this set of lines is stored as sets of four 256 byte sectors allocated randomly throughout memory using the usual compressed memory system memory management mechanisms, including pair-wise combining of "fragments", that is, the last partially full sectors used to store each compressed logical 1K line (for a detailed description, see [6]). The VUC (logical) area can be divided into two parts: the VCC ("virtual compression cache"), which consists of those logical lines for which a directory entry is currently cached (in the VCCD, or "VCC directory"), and the UC ("uncompressed area"), consisting of the remaining uncompressed lines.

Fig. 5 illustrates a potential performance enhancement to the system illustrated by Fig. 4, in which in addition to the VCC and UC, some (small) number of recently accessed logical 1K lines are saved in uncompressed form in a number of line buffers in the memory controller.

In the next section these various system designs will be compared, using the results of trace-driven analyses to evaluate alternatives.

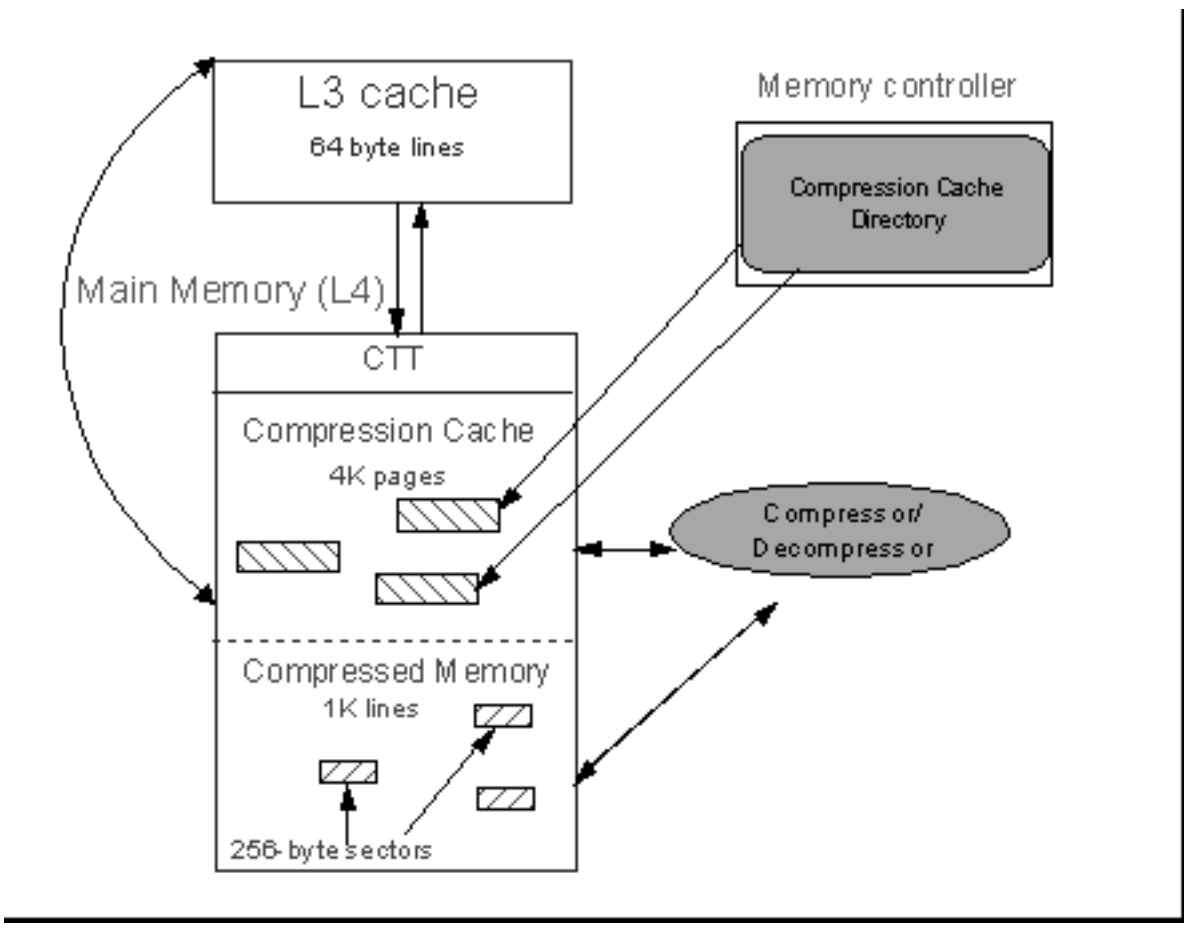

Figure 3. Design using "Compression Cache"

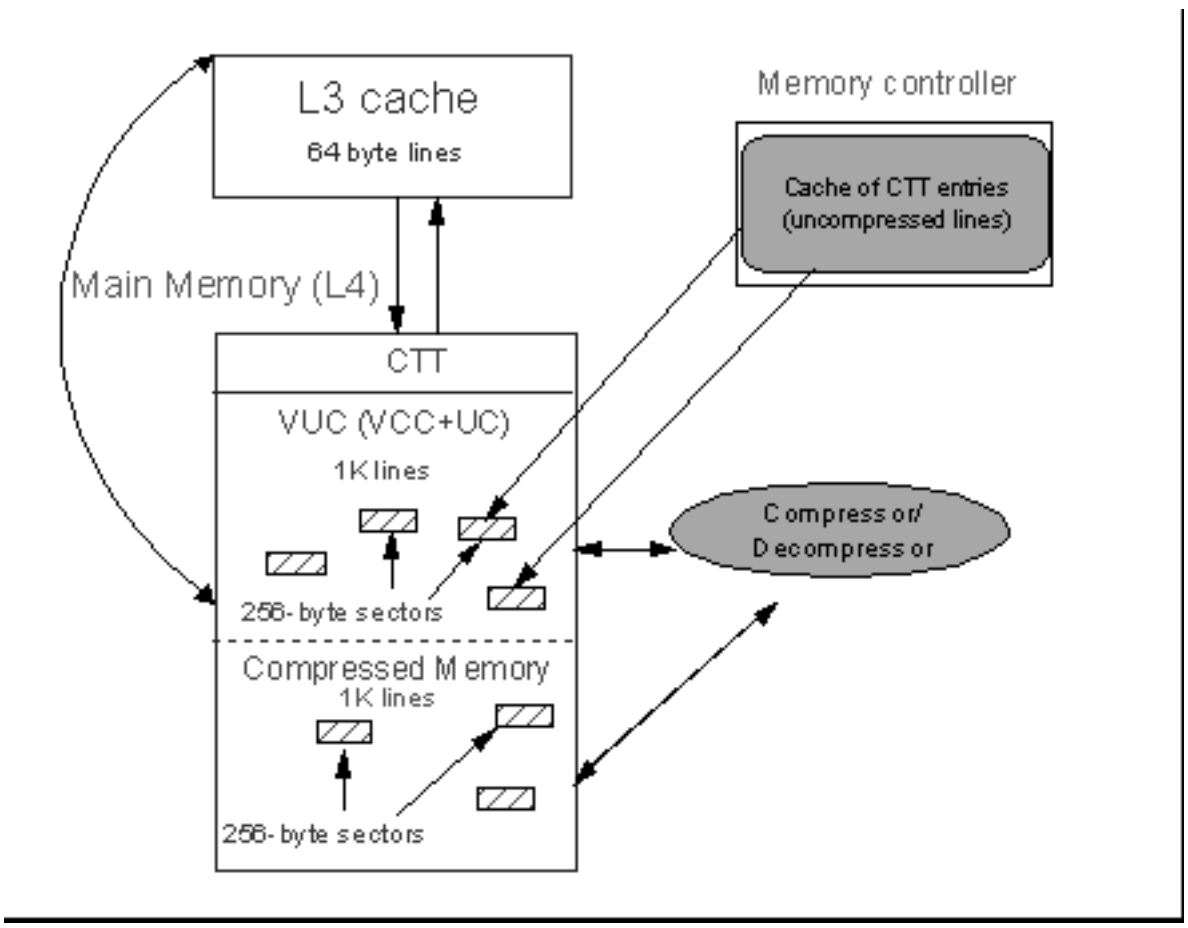

Figure 4. Design using "Virtual Uncompressed Cache" (VUC)

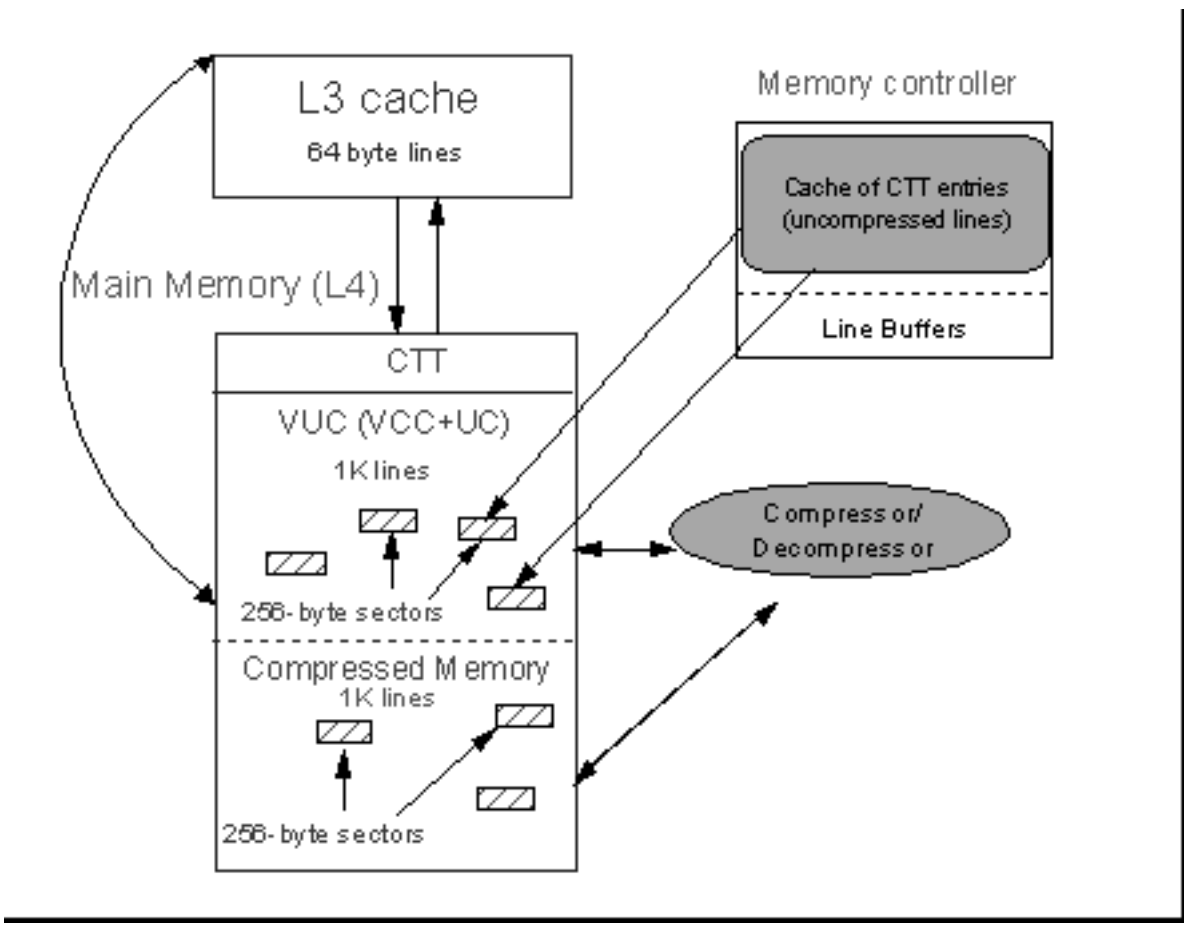

Figure 5. Design using VUC and Line Buffers

3. Reducing L3 Miss Penalties

To compare the performance of different system configurations we designed a trace-driven simulator. Using this simulator we investigated the behavior of a system with a compression cache, as well as a system with a virtual compression cache and and uncompressed area as described in the previous section. In the second system we considered an architecture with and without line buffers.

3.1. System Parameters and Workloads

We make the following assumptions about access times: access to data in the compression cache takes 10 memory bus cycles, access to data whose CTT entry is cached (in the VCCD) also takes 10 cycles, and access to data in the uncompressed area takes 20 cycles (since an additional memory reference is required for CTT lookup). If the data is stored in compressed form then 90 cycles are needed for decompression. If data is found in the line buffers only one cycle is needed for access.

We simulated a random replacement policy in the CTT cache (VCCD), and the uncompressed area was simulated as a FIFO. Cache directories (the L3 directory and VCCD) were simulated as 4-way set-associative. In the case of the VCCD, a CTT entry was cached only for L3 misses but not for L3 writebacks (however both types of accesses caused the uncompressed data to be saved in the compression cache or VUC).

We used traces from three different workloads: TPC-C, NotesBench and SAP. These traces captured bus traffic, and we filtered them so that they contained only L2 misses, L2 writebacks, and processor stores. Each trace was over 1GB in length, and contained over 80 million trace records. The traces were taken from a 4-processor pentium system. For all the experiments we ran we assumed a 2:1 compression ratio. We ran simulations for three different L3 cache sizes: 8MB, 16MB, and 32MB.

3.2. Results

The results for the 32MB L3 simulations are shown in Figures 6-11. The first three graphs compare the performance of a system with a compression cache to one with a virtual uncompressed region and a cache of CTT directory entries. For the TPC-C trace, the miss rate for the system with the compression cache was 67% higher than the miss rate for the VUC system, and the average memory access time was 30% greater. For the NotesBench trace, the miss rate of the system with the compression cache was three times the miss rate of the VUC system, while the access time was 55% greater. For the SAP trace the miss rate in the system with compression cache was almost 4 times greater than the miss rate for the VUC system and the access time was 62% greater. In the VUC system, the hits were almost equally divided between the virtual uncompressed area and the area containing lines whose

CTT directory entries were cached (36% vs. 37% for TPC-C, 43% vs. 47% for NotesBench and 48% vs. 44% for SAP).

A simple calculation, as follows, shows that by using the VUC the same amount of physical memory can be used to store twice as much data in uncompressed form as compared to the compression cache approach (due to avoiding storing lines twice). First, consider the compression cache: suppose that M 1K lines are stored (uncompressed) in the compression cache, that there are N other lines stored outside the compression cache, and that the compression ratio is 2:1. Then, since each line stored in the compression cache is also stored in compressed form, the total physical memory required for the $M + N$ distinct lines is, in 1K byte units, $M + M/2 + N/2 = (3M + N)/2$. Now consider the VUC: using the same amount of physical memory, $2M$ lines can be stored in uncompressed form and $(N - M)$ lines stored in compressed form (for the same total of $M + N$ distinct lines), since the amount of physical memory required in this case is (again in 1K byte units) $2M + (N - M)/2 = (3M + N)/2$, as before.

Since twice as much data can be stored uncompressed using the VUC design as compared to the compression cache, and furthermore at a finer granularity (1K instead of 4K), there is a significant increase in the hit ratio to uncompressed data. The situation regarding uncompressed data for which a CTT entry is cached is slightly more complicated. Directory entries for the compression cache are significantly smaller than for the VCC, since they are pointers to data in a fixed memory partition; based on calculations of required pointer lengths, we assume that four times as many directory entries can be cached for the compression cache as compared to the VCCD (for the same directory entry cache memory size). Furthermore, in the compression cache the amount of uncompressed data covered by each cached directory entry is four times larger (using the compression cache, each cached directory entry is for a logical real page, that is a set of four consecutive uncompressed lines aligned on a 4K real memory address, whereas in the VUC design each cached directory entry is for an uncompressed 1K line); the result is 16 times more addressability for the compression cache directory. On the other hand, the granularity at which uncompressed data is cached is smaller in the VCC. The end result, as seen in Figures 6-8, is that although there is in fact a smaller hit ratio to the VCC than to the compression cache, this is more than offset by the significantly decreased miss rates to uncompressed data: as described above, average memory access times (for the 32MB L3 case) are from 30% to 62% larger using the compression cache approach as compared to those obtained using the VUC design.

In the next set of Figures (9-11) we show the performance of a VUC system with an additional 8K of line buffers. The miss rate is the same as the miss rate in the VUC system without line buffers, since the hits to the line buffer result in an equal decrease in hits to the VCC. However, since line buffer hits take only one cycle (as opposed to 10 cycles for VCC hits), the average memory access times decreases slightly as shown in the Figures.

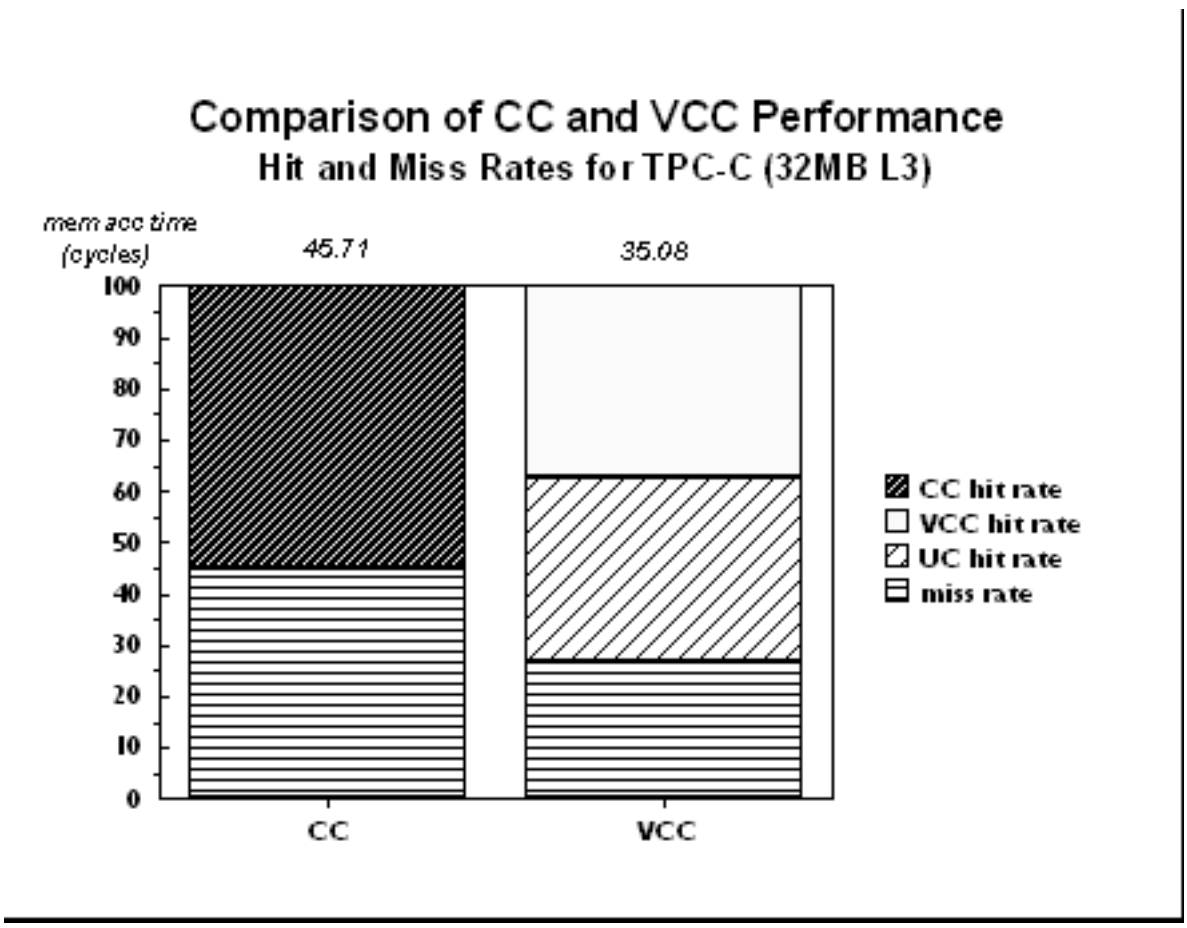
Comparison of CC and VCC Performance
Hit and Miss Rates for TPC-C (32MB L3)
mem acc time
(cycles)
45.71
35.08
100
90
80
70
60
50
40
30
20
10
0
CC
VCC
CC hit rate
VCC hit rate
UC hit rate
miss rate

Figure 6.

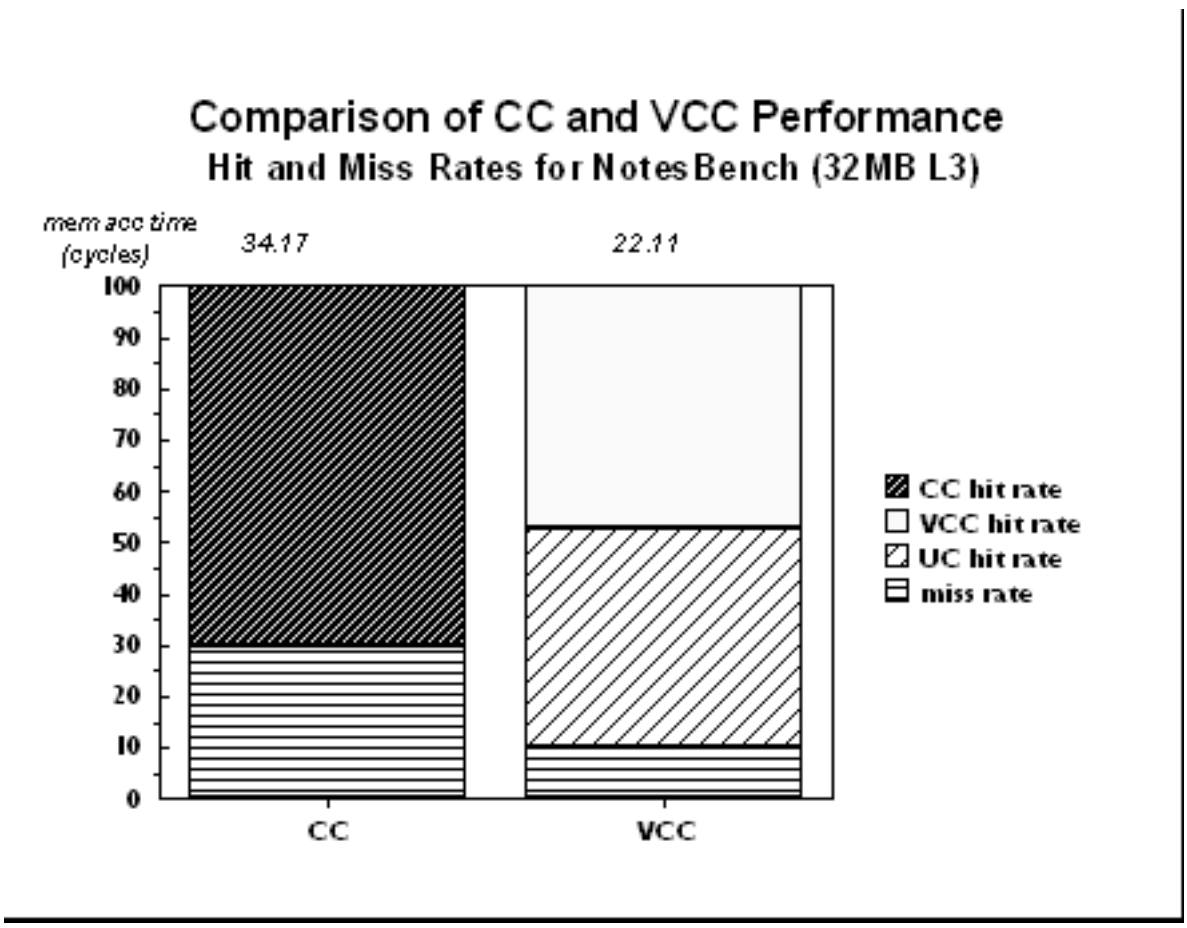
Comparison of CC and VCC Performance
Hit and Miss Rates for NotesBench (32MB L3)
mem acc time (cycles)
34.17
22.11
100
90
80
70
60
50
40
30
20
10
0
CC
VCC
CC hit rate
VCC hit rate
UC hit rate
miss rate

Figure 7.

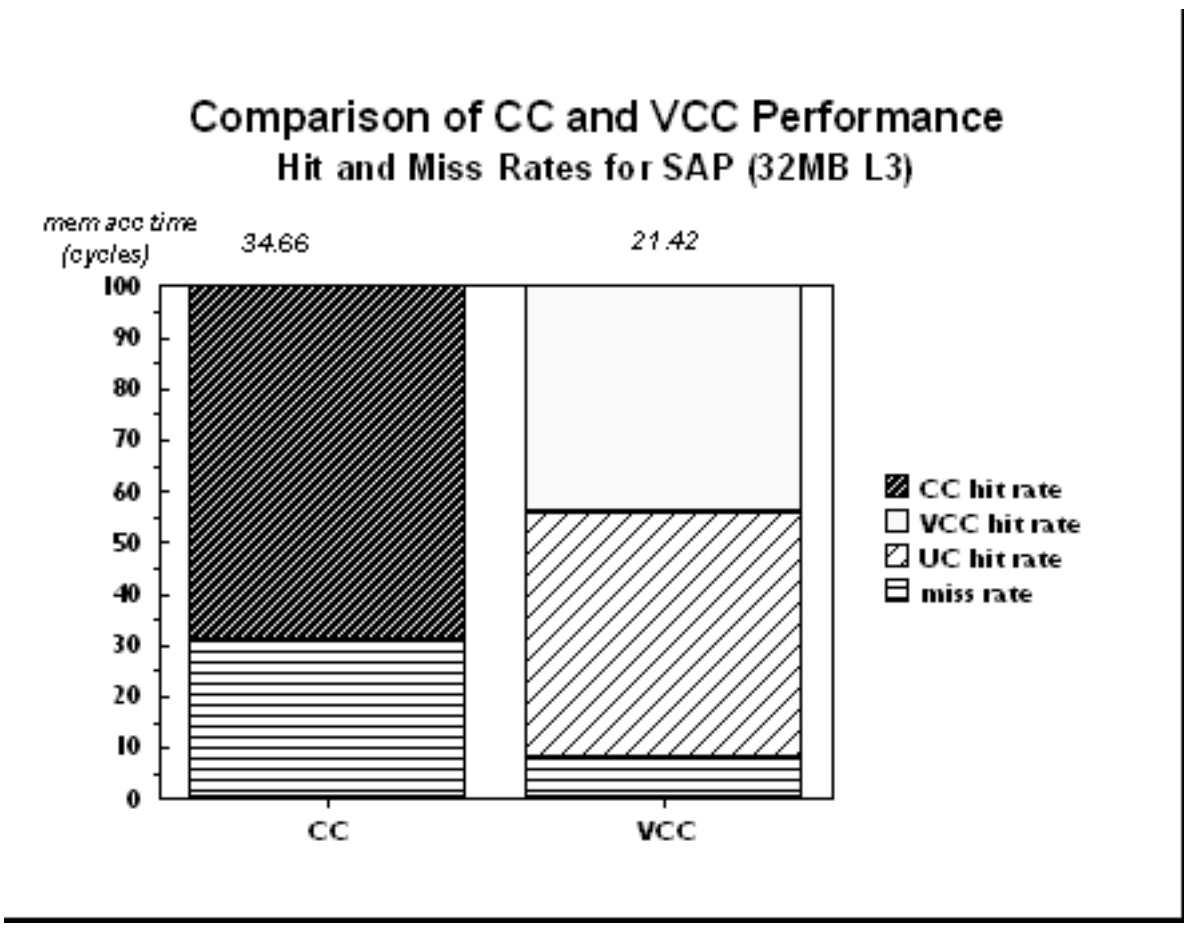
Comparison of CC and VCC Performance
Hit and Miss Rates for SAP (32MB L3)
mem acc time (cycles)
34.66
21.42
100
90
80
70
60
50
40
30
20
10
0
CC
VCC
CC hit rate
VCC hit rate
UC hit rate
miss rate

Figure 8.

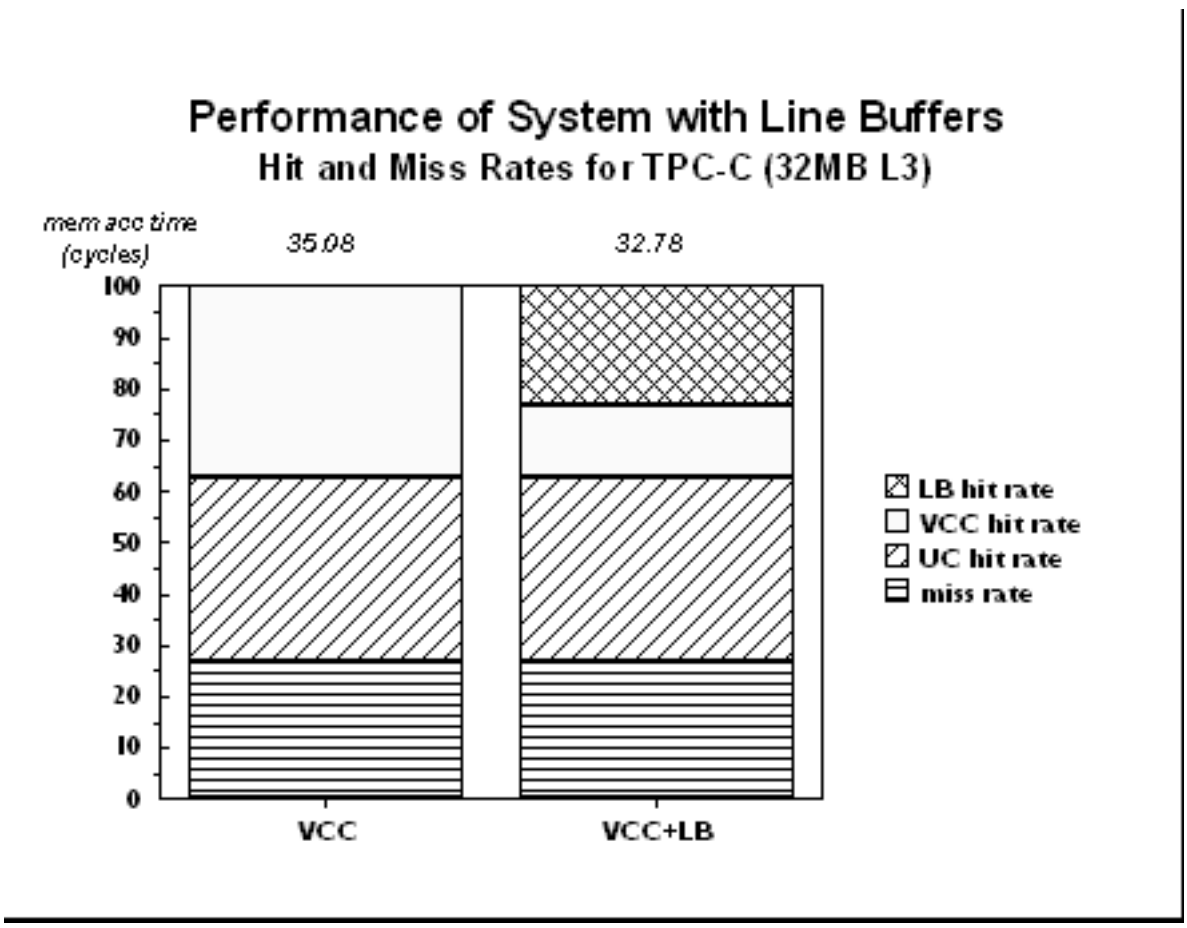
Performance of System with Line Buffers
Hit and Miss Rates for TPC-C (32MB L3)
mem acc time (cycles)
35.08
32.78
100
90
80
70
60
50
40
30
20
10
0
VCC
VCC+LB
LB hit rate
VCC hit rate
UC hit rate
miss rate

Figure 9.

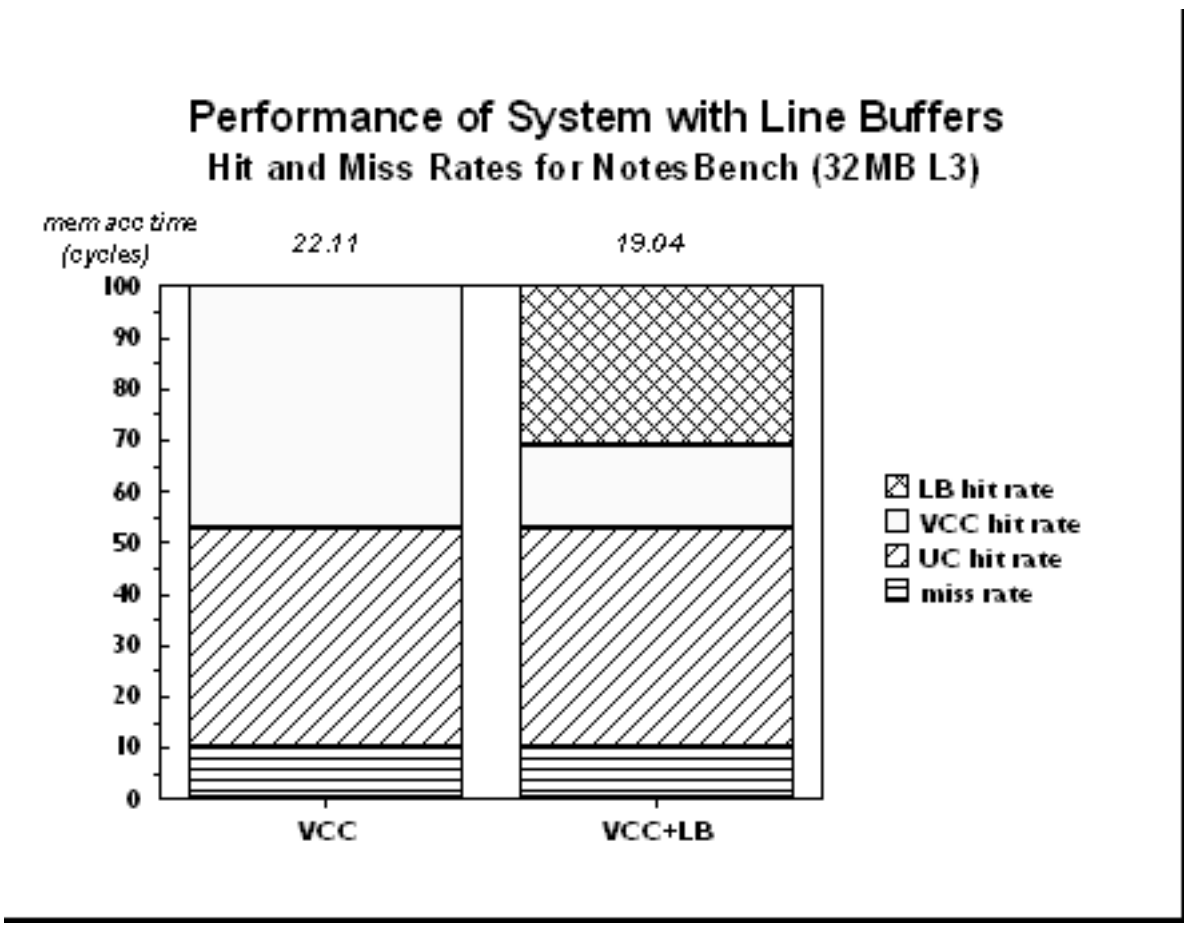
Performance of System with Line Buffers
Hit and Miss Rates for NotesBench (32MB L3)
mem acc time (cycles)
22.11
19.04
100
90
80
70
60
50
40
30
20
10
0
VCC
VCC+LB
LB hit rate
VCC hit rate
UC hit rate
miss rate

Figure 10.

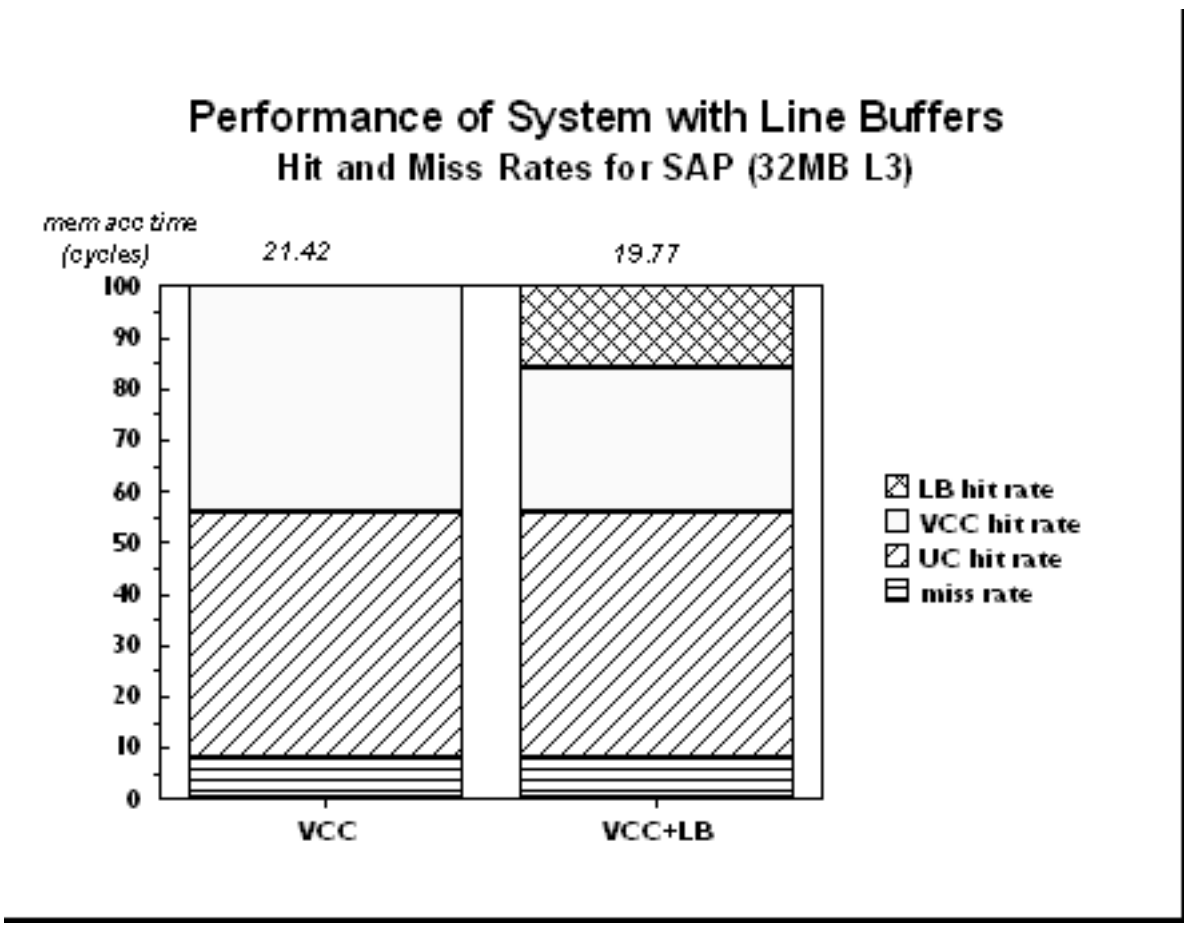
Performance of System with Line Buffers
Hit and Miss Rates for SAP (32MB L3)
mem acc time
(cycles)
21.42
19.77
100
90
80
70
60
50
40
30
20
10
0
VCC
VCC+LB
LB hit rate
VCC hit rate
UC hit rate
miss rate

Figure 11.

4. Reducing L3 Writeback Penalties

In the previous sections we described some of the problems that arise in a system with main memory (L4) compression when the L3 line size is smaller than the unit of compression in L4. In this section we discuss how this can result in additional compression/decompression traffic, and how use of a technique in which writebacks are "batched" can alleviate these problems. In addition, we will show how using batched writebacks decreases the amount of free space that must be reserved to guarantee forward progress, and how the structures needed to support batched writebacks can provide an estimate of the free space required.

4.1. Batched Writebacks

In the systems shown in Figures 4-5, writebacks may have some effects not seen in systems without main memory compression. First, a writeback of a 64-byte line from the L3 may require that the corresponding 1K line be decompressed. This will result in additional compression/decompression traffic in the system. One possible solution is to batch L3 writebacks. This can be done as follows: each time a 64-byte line is written back from the L3 due to a castout, all other 64-byte pieces of the 1K line that are currently present and modified in the L3 are also written back. This can be done either by snooping the L3 to find all the 64-byte pieces of the 1K line, or by maintaining a *dual size directory*. Logically, the dual size directory can be thought of as a combination of the usual L3 cache directory for 64 byte lines, together with an additional directory of 1K lines in which each entry contains 16 bits, one for each 64-byte piece of the 1K line (in an actual design, the logical function of the 1K directory can be realized without implementing a second full directory, but instead, for a 4-way set-associative L3 directory for example, providing a mechanism for reading in parallel 16 sets of 4 tags, and then using a 64-way content addressable memory to compare the appropriate fields of these 64 tags with the address of a 1K line, yielding a bit-vector of length 64 which specifies the "pieces" of the 1K line currently contained in each of the corresponding 16 sets of the 64-byte line L3 directory). If the 64-byte piece is present and modified in the L3, then the corresponding bit is set in the (logical) 1K line size directory. When a writeback from L3 occurs, the system (logically) checks the directory entry associated with the 1K line that contains the L3 line to be written back, and performs a writeback of all modified 64-byte pieces of that line.

We simulated a system with batched writebacks. In Figures 12-14 we show the results for this system and compare the hit and miss ratios to the VCC and UC with those of a system without batched writebacks. Here, a miss to the VUC can occur due to an access associated with an L3 miss or an L3 writeback. For all three workloads, the miss ratio is slightly lower for the system with batched writebacks. This is not surprising since fewer misses occur on writebacks. The reason for this is that in the system without batched writebacks, each time a writeback occurs, if the corresponding 1K line is stored in memory in compressed form, the line must then be decompressed to perform the writeback. In a system with batched

writebacks, if the line is compressed, then it is decompressed once and multiple writebacks occur. The average number of writebacks that occur on an L3 castout is shown in Figure 15 for three different L3 sizes. The VCC hit rate of the system with batched writebacks is also slightly higher than for a system without batched writebacks. The reason for this is that there are fewer total references, but the number of hits to the VCC remains the same for L3 misses while decreasing slightly for L3 writebacks. (As mentioned in Section 3, if a miss to the uncompressed region occurs on an L3 writeback, that line is decompressed, but its directory entry is not cached. Therefore, the number of hits to the VCC caused by writebacks is small compared to the number of hits to the VCC caused by L3 misses). One possible disadvantage of batched writebacks is that a line may be written back more than once if it is modified between the times it is written back (in practice, for the type of compressed memory system architectures we are consdidering here, this will not be a peformance problem, since no extra compression/decompression work is required for batching writebacks, and as we have seen earlier compression/decompression times are the dominant factor for L4 access times). To measure this effect, we calculated the total number of writebacks in a system with batched writebacks, and compared that to the number of writebacks in a system without batched writebacks. For a 32MB L3, the ratios are 1.95 for TPC-C, 2.08 for NotesBench, and 4.89 for SAP.

4.2. Guaranteed Forward Progress

A writeback from L3 may result in an expansion of memory contents due to changes in compressibility. If the writeback goes to a line in the uncompressed area, no immediate expansion occurs; however, since there is a difference in line size between L3 and the unit of compression, writebacks from the L3 can in the worst case each occur to a different L4 1K line and potentially cause a loss of compression for each such 1K line. The latter is significant for purposes of control since the system needs to maintain sufficient free space in L4 physical memory to guarantee forward progress in the worst case in which all modified L3 lines are written back and cause a loss of compression (i.e. avoid crashes due to insufficient space for cache writebacks).

Using batched writebacks can reduce the amount of free space that must be reserved in the system to handle this situation, since the number of modified lines in the L3 will be smaller at any given time for a system with batched writebacks. The results of our simulation show that in a 32MB L3, on average, the number of modified 1K lines in a system without batched writebacks is about 52,000 for NotesBench, 62,000-64,000 for SAP, and up to 100,000 for TPC-C. In contrast, with batched writebacks, these numbers are reduced by a factor of two or more (26,000 modified 1K lines in the L3 for NotesBench, 24,000 for SAP, and 46,000 for TPC-C). In addition, the (logical) dual size directory (as described above), which "keeps track" of modified 1K lines in the L3, can also be used to calculate the amount of free space that should be maintained in L4 to guarantee forward progress.

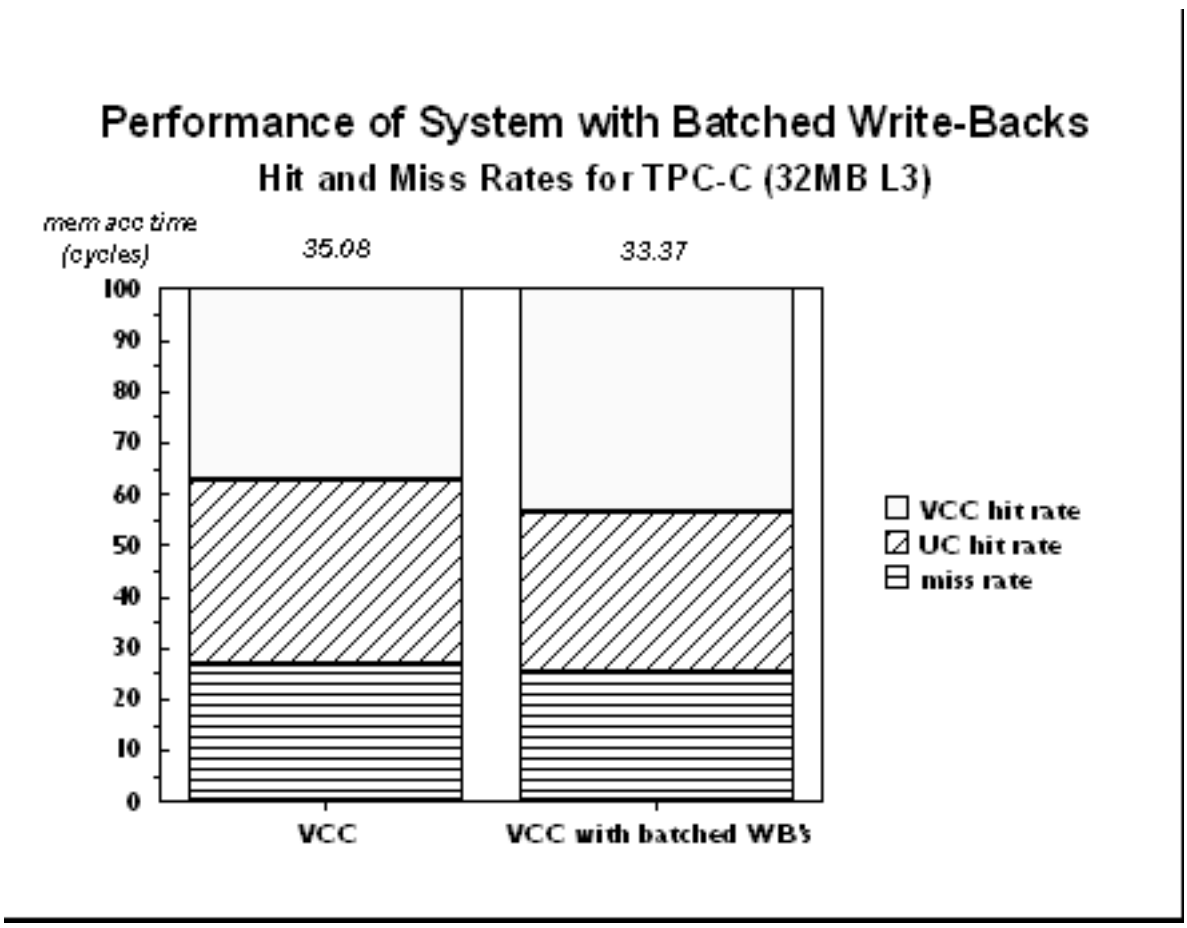
Performance of System with Batched Write-Backs
Hit and Miss Rates for TPC-C (32MB L3)
mem acc time (cycles)
35.08
33.37
100
90
80
70
60
50
40
30
20
10
0
VCC
VCC with batched WB's
VCC hit rate
UC hit rate
miss rate

Figure 12.

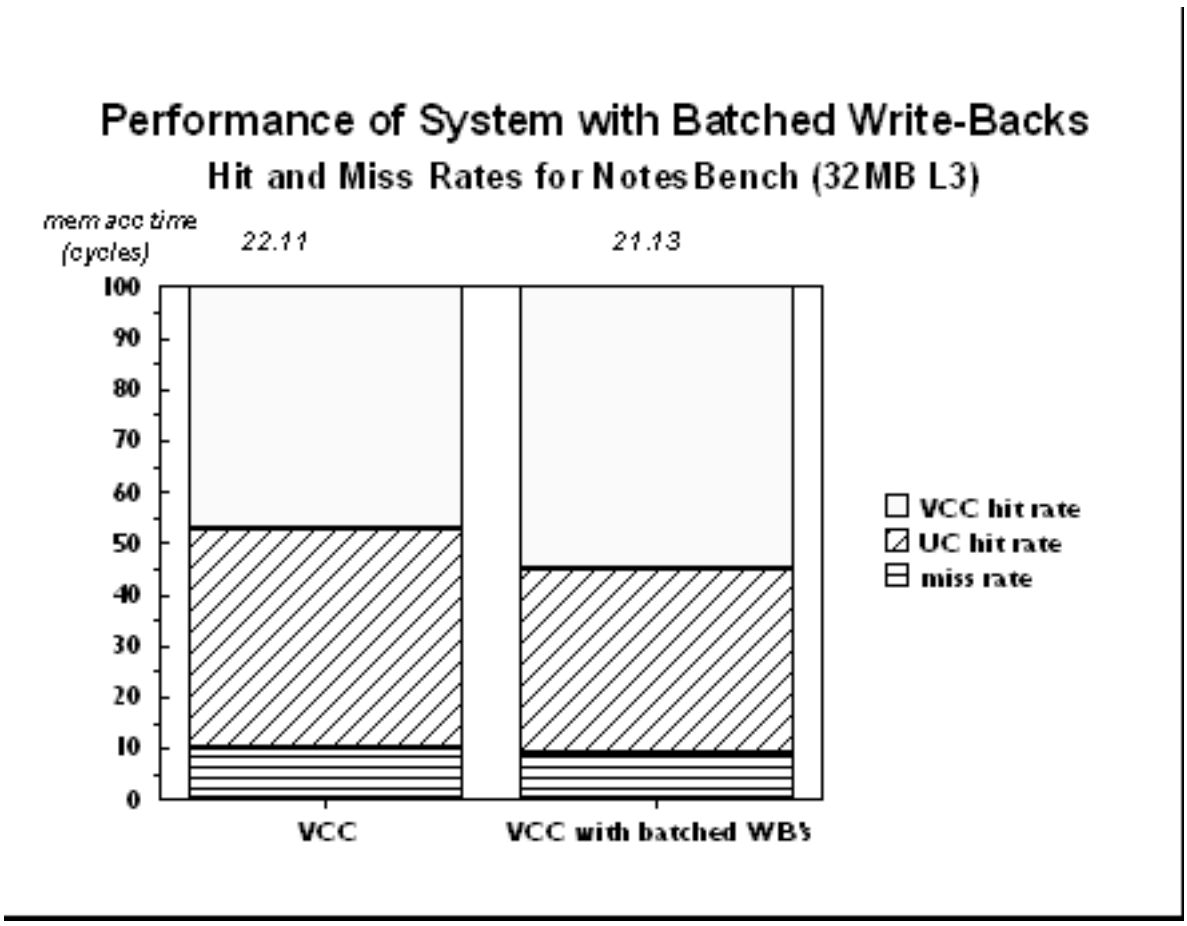
Performance of System with Batched Write-Backs
Hit and Miss Rates for NotesBench (32MB L3)
mem acc time (cycles)
22.11
21.13
100
90
80
70
60
50
40
30
20
10
0
VCC
VCC with batched WB's
VCC hit rate
UC hit rate
miss rate

Figure 13.

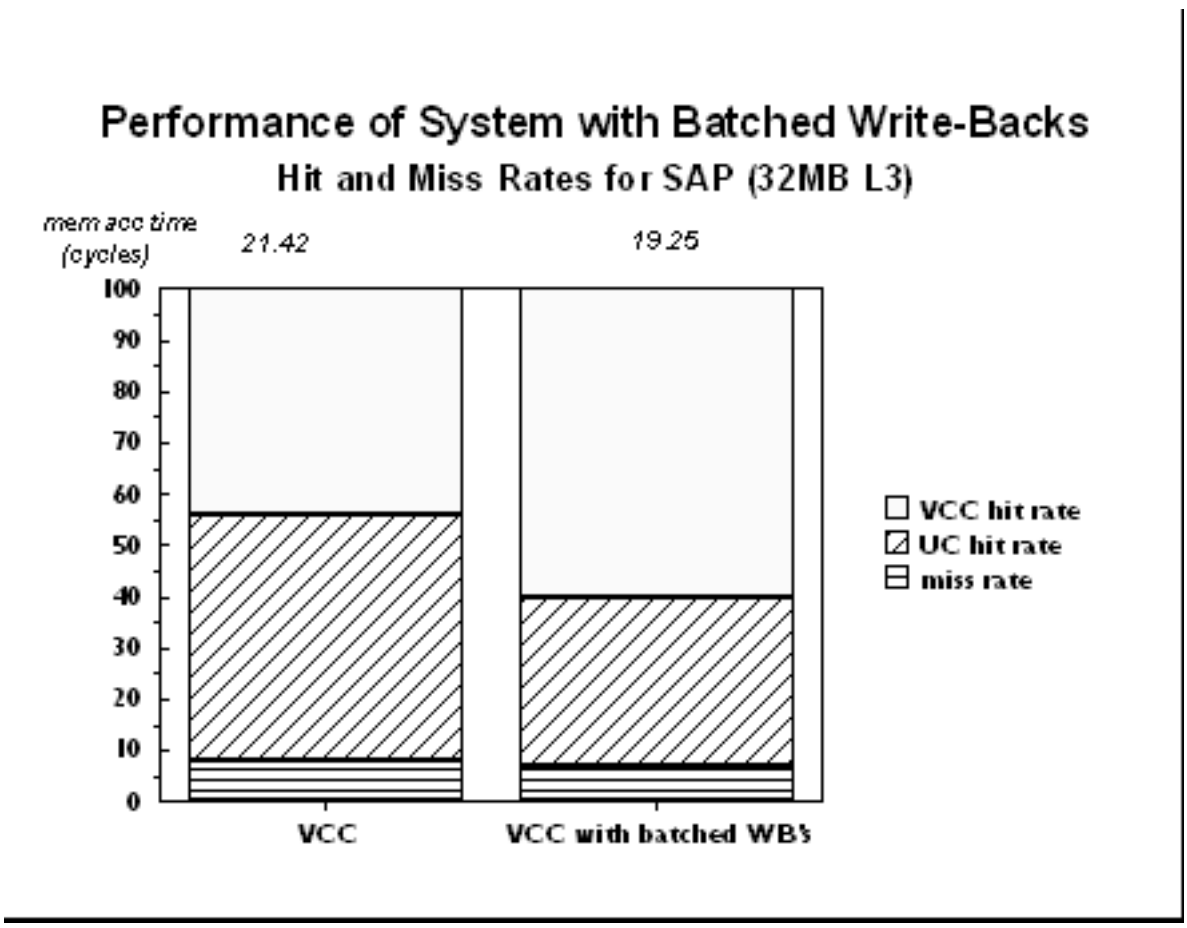
Performance of System with Batched Write-Backs
Hit and Miss Rates for SAP (32MB L3)
mem acc time (cycles)
21.42
19.25
100
90
80
70
60
50
40
30
20
10
0
VCC
VCC with batched WB's
VCC hit rate
UC hit rate
miss rate

Figure 14.

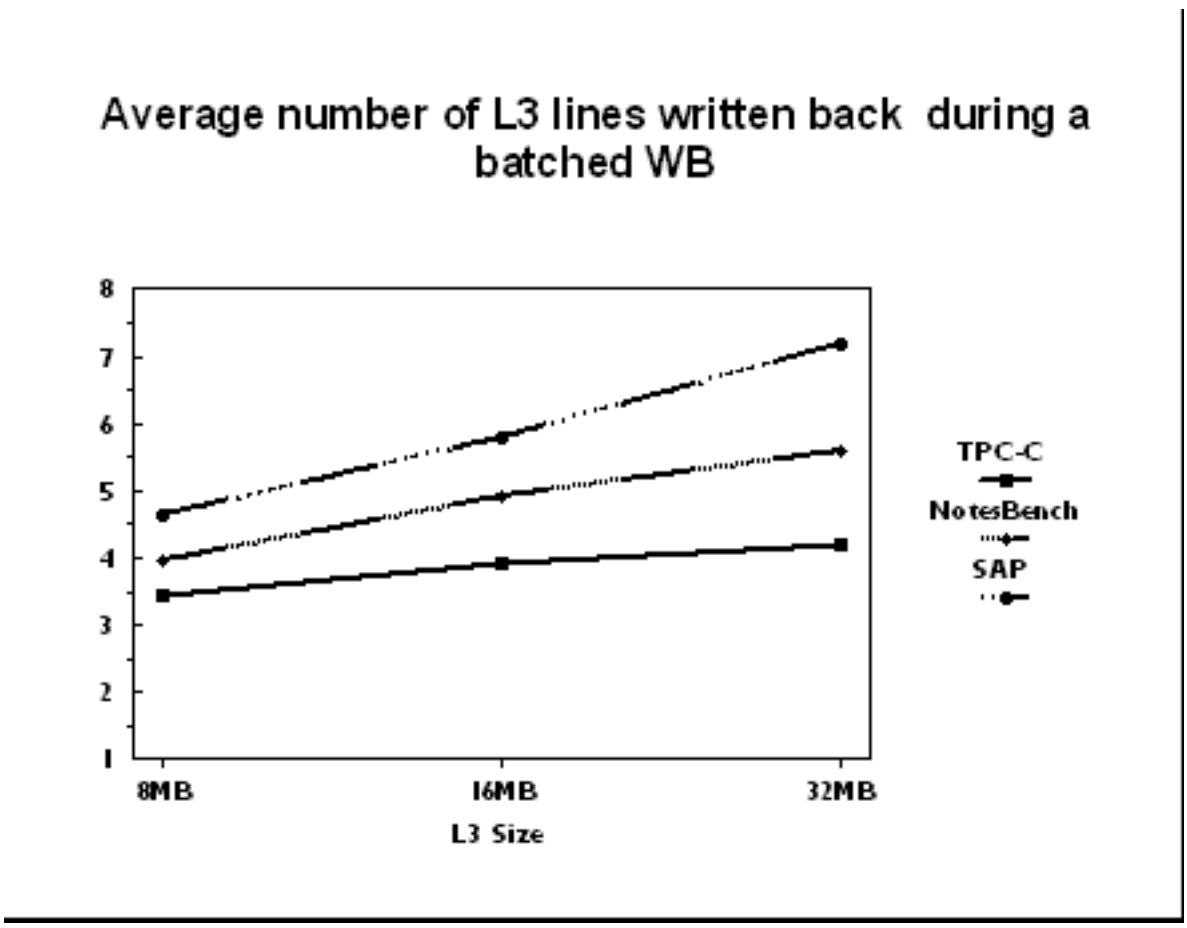
Average number of L3 lines written back during a batched WB
8
7
6
5
4
3
2
1
8MB
16MB
32MB
L3 Size
TPC-C
NotesBench
SAP

Figure 15.

5. Conclusion

Compressed memory systems, in which compression and decompression take place automatically to and from main memory as a result of cache writebacks and misses, offer the potential to provide a real memory space that is 2-3 times the physical main memory size (these are in fact typical compression ratios) with very little increase in system cost. In this approach, unlike designs based on decompression on page faults for example, under normal operation there is no additional software overhead due to the memory compression architecture (the operating system need only respond to significant changes in overall compressibility, decreasing the logical real memory space if compression becomes worse for example, or conversely adding more logical real memory if compression improves – early experience indicates that for a given workload, compressibility tends to be stable). Therefore, since the overall system performance, to a first approximation, is largely determined by the processor speeds, cache access times, and cache hit-ratios, it is to be expected that the use of a compressed memory system architecture of this general type will have performance similar to that of a far more costly non-compressed memory system with 2-3 times the main memory size of the compressed memory system.

However, when the cache line size of the lowest-level cache above main memory has a line size that is different than the unit of compression, there is a potential for some performance problems. For example, a sequence of cache misses to cache lines residing in the same unit of compression could require repeated decompressions of the same main memory data.

Here, we have used trace-driven analyses to investigate these issues, and studied several alternative designs for improving performance. These include the "compresssion cache" approach, the virtual uncompressed cache (VUC) design (including a memory controller cache for CTT, that is main memory directory, entries), and the use of memory controller line buffers to hold a small amount of recently uncompressed main memory data. In terms of memory access times, significant improvement is seen using the VUC design as compared to the compression cache, and a small additional improvment is provided by the use of line buffers.

Related problems in the case that the cache line size is different than the unit of compression are due to cache writebacks. Just as a sequence of cache misses residing in the same unit of compression could require repeated decompressions of the same main memory data, a sequence of writebacks residing in the same unit of compression could require repeated decompressions and also subsequent re-compressions of the same main memory unit of compression. As in cache misses, this effect is greatly reduced by the use of the VUC design (together with, optionally, line buffers). However, there is now an additional optimization that can be made, since all modified cache lines residing in the same unit of compression can be found from the cache directory (using a logical "dual size directory", which in practice could be implemented as a single cache line size directory with additional logic allowing all cache lines residing in the same unit of compression to be found in the time taken for one cache directory access); this allows all such modified lines to be grouped and written back

together, that is, writebacks can be batched. Trace-driven analyses showed that the use of this technique leads to reduced VUC miss rates, which results in a further reduction of memory access times. An additional effect is increased VCC hit rates (that is, that part of the VUC for which CTT entries are cached), which also contributes to the performance improvement.

Batched writebacks can also alleviate problems associated with maintaining enough free space in the compressed main memory to guarantee forward progress (that is, to guarantee that all modified cache lines could be written back even if every such line caused a decrease in compressibility). This is simply because batching writebacks results in fewer modified cache lines: for the traces of this study, the number of modified cache lines was reduced by a factor of two or more.

Finally, although at the current time it is difficult to quantify, the VUC approach has certain properties that could prove to be valuable from the standpoint of a robust system design. Since the VUC is a logical entity (the storage used for the VUC consists of a number of sectors taken from the same pool as that used to store compressed data), as opposed to a memory partition for example, it may be designed so that its size varies dynamically without requiring memory reorganization. This has two effects: (1) workloads generating highly compressible data can gain a performance benefit from a larger VUC made possible by the high degree of compression; (2) when free space in main memory (required to guarantee forward progress, as previously discussed) runs low, decreasing below a threshold for example, then prior to initiating OS action, additional free space can be generated by dynamically decreasing the size of the VUC; similarly, given an excess of free space, the VUC size can be increased; thus in many cases it may be possible to adapt to changes in compressibility with no software OS interaction at all, that is entirely by hardware means.

Acknowledgments: the authors gratefully acknowledge the assistance of Dan Colglazier, Mike Wazlowski, Chuck Schulz, Scott Clark, and Basil Smith in the work described here.

References

[1] Kjelso, M., Gooch, M., and Jones, S. Performance evaluation of computer architectures with main memory data compression. *Journal of Systems Arch. 45*, 571-590, 1999.

[2] MacDonald, J. R., Dutton, D., and Cox, S. Memory paging system and method including compressed page mapping hierarchy. U.S. Patent 5,696,927, Advanced Micro Devices Inc., Dec. 9, 1997.

[3] Douglis, F. The compression cache: using on line compression to extend physical memory. In *Proc. Winter 1993 USENIX Conf.*, pp. 519-529, USENIX Assoc., 1993.

[4] Franaszek, P., Robinson, J., and Thomas, J. Parallel compression with cooperative dictionary construction. In *Proc. DCC '96 Data Compression Conf.*, pp. 200-209, IEEE, 1996.

[5] Franaszek, P., Robinson, J., and Thomas, J. Parallel compression and decompression using a cooperative dictionary. U.S. Patent 5,729,228, International Business Machines Corp., Mar. 17 1998.

[6] Franaszek, P., and Robinson, J. Design and analysis of internal organizations for compressed random access memories. Research report RC 21146, IBM Watson Res. Ctr., Yorktown Hts., NY, October 20, 1998.