RC22245 (W0111-031) November 9, 2001
Computer Science

IBM Research Report

Directory Caching in Compressed Memory Architectures

Peter A. Franaszek, Vittorio Castelli, Caroline D. Benveniste
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Directory Caching in Compressed Memory Architectures

Peter Franaszek, Vittorio Castelli, Caroline Benveniste

November 9, 2001

Abstract

IBM's recently announced Memory eXpansion Technology (MXT) improves computer cost/performance by holding main memory contents in compressed form, to be decompressed/compressed on cache faults/castouts. The current architecture includes a large level three (L3) cache to hide decompression latency. Current design trends, however, are towards incorporating memory control on the processor chip. Here an L3 cache may not be desirable. A possible approach then may be to incorporate a substantial amount of recently accessed uncompressed data in memory, as suggested in earlier studies. A further problem is that memory accesses require directory data, itself stored in main memory. In this report, we consider alternative approaches to caching of such data, and show that a simple combined modification of the memory controller and L2 cache is an effective solution.

1 Introduction

Memory represents the largest cost component of a typical server-class machine. This is despite ever decreasing cost/bit, because faster processors require more data to be available without the massive latency associated with disk storage. A natural approach to reducing the memory cost is to take advantage of the compressibility of representative data and programs. However, doing so required advances in data compression and computer architecture. An example of an approach which has been implemented and for which substantial amounts of performance data are available is IBM's Memory eXpansion Technology or MXT, where data in main memory is held in compressed form, and decompressed/compressed on cache fetches/castouts [1, 2, 3]. Compression/decompression is via a parallel generalization [4] of the popular LZ77 algorithm. As described in more detail below, memory is allocated in fixed-size units of 256B, termed sectors. The unit of compression, which we term a compression line, is 1K B. Data is accessed via a compression-translation table, or CTT, stored in main memory.

The current version of MXT incorporates a large L3 cache, with a line size of 1KB. This effectively hides decompression latency, which is even in the worst case orders of magnitude smaller than that associated with disc access. In fact, decompression is done at the full memory bandwidth, so that, with a compression factor of say 2, the effective bandwidth on access is doubled. However, two issues may affect the desirability of the current design. The first is that the 1KB line size for the L3 cache may be too large for systems with multiple L3 caches, as the unit of data sharing may be substantially lower. A smaller L3 line size poses a number of problems, solutions for which were described in [5]. In particular, for this case it is advantageous to maintain an uncompressed area of recently referenced 1KB units. The uncompressed area (termed a virtual compression cache, or VCC) would not be a separate part of memory, but merely consist of a set of uncompressed 1KB units, along with a data structure (essentially a FIFO list) to determine which units to decompress given a castout. This area could be of variable size, with the size controlled by algorithms [6] which would attempt to optimize overall system performance via a tradeoff between paging and access latency. Also associated with the uncompressed area was the suggestion for a CTT cache. Since each 1KB unit of compression would in this case hold multiple L3 cache lines, it was shown that caching recently accessed CTT entries substantially decreased the effective memory latency.

As mentioned in the abstract, a current trend is the integration of memory control within the processor chip. This is the second reason that the current design may eventually be superseded, as such designs currently do not incorporate an L3 cache. The lack of an L3 cache is however not a serious problem, as latency can be hidden by a VCC, provided that directory accesses are usually serviced without having to go to memory, namely, if some form of caching is included for the directory or CTT. Such directory caching could be done with a separate CTT cache, as suggested in [5]. However, this has the disadvantage of complicating the processor chip, and also of incorporating additional circuitry which would not be used if the applications running on the machine were not compatible with memory compression.

In this report we consider several alternatives for CTT caching. These include using the TLB, and two approaches which utilize a modified L2 cache. The former appears to require substantial complication of the design. The latter two are evaluated using IBM System/390 traces, and compared with having a separate CTT cache. Both offer similar and very good performance. One of these requires very little additional circuitry and thus may be preferred. A result of the evaluation is that memory control references to CTT entries in the L2 cache can be treated similarly to ordinary L1 misses, this despite the expected difference in frequency between the two types of events.

The following is a synopsis of this report. Section 2 provides a description of the relevant aspects of MXT architecture. Section 3 discusses the new caching approaches. Performance data is shown in Section 4. Discussion and conclusions are contained in Section 5.

2 Preliminaries

2.1 The MXT memory organization

Pages in memory are divided into compression lines of 1KB. Compressed lines are stored in sectors of 256B, with the possibility of having two lines in a page share a sector. The location of the compressed lines is described in the compression translation table. Each page has an entry of 64B in the table, divided among the four lines (16B per line); the memory controller, in order to service a cache fault, needs access to a 16B section of the page entry in the CTT. The entry has some flags, and space for 4 pointers to sectors. Data in memory can be in one of three states: uncompressed, compressed and using sectors, or compressed so well as to require no sectors. In the latter case, the compressed line fits into the space otherwise reserved for pointers. Latencies encountered in processing an L3 cache fault in the current implementation, as a function of the compression state, are illustrated in Figure 1. These all assume that the CTT entry is not cached.

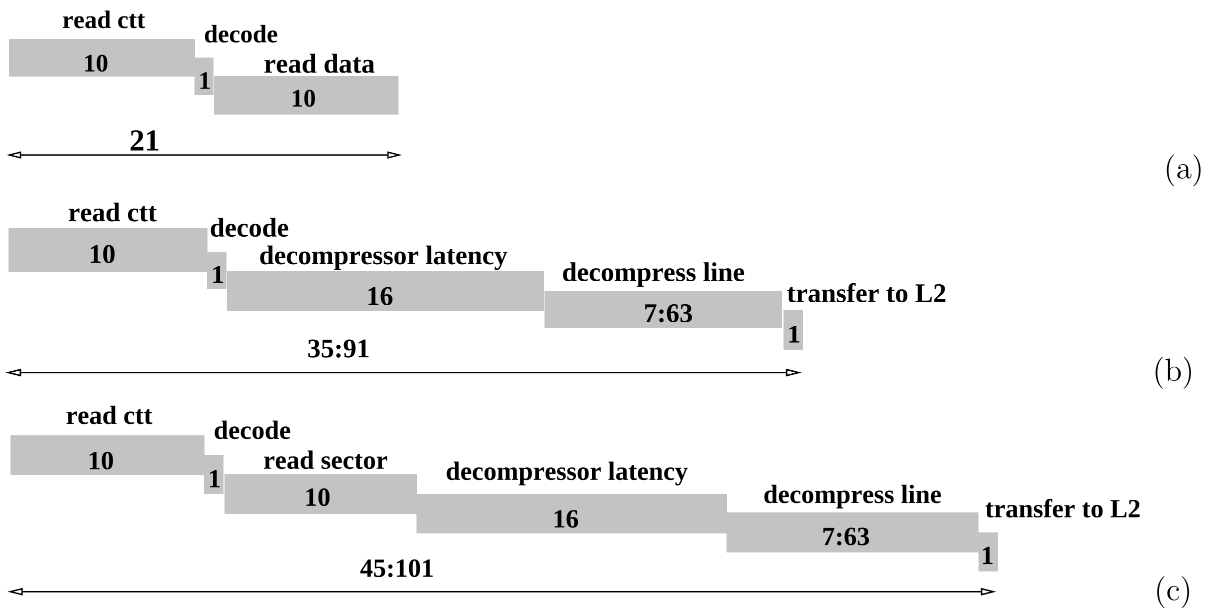

Figure 1: Delays associated with a memory read due to a cache miss, measured in memory bus cycles: (a) the data is poorly compressible stored uncompressed in main memory; (b) the data is highly compressible and stored compressed in the CTT; (c) the data is stored compressed in one or more memory-sectors.

The processor generates what are termed real addresses. The are translated via the CTT into physical addresses. The CTT, in the current implementation, supports a real address space twice the size of the physical memory, with this multiple determined at IPL.

Net compression ratios, that is including overhead for the CTT and fragmentation losses from using 256B sectors, are generally better than a factor of two. The real address space, as mentioned above, is only twice the size of main memory. Thus in principle, there is typically some unoccupied space. In practice, some of this space needs to be reserved for operating system usage [2], so that there is sufficient space to do pageouts when compressibility is degraded. In the current MXT implementation, space not utilized is simply held on a list of available sectors. In [5], a configuration was studied where unused as well as possibly additional space, could be consumed by holding a set of recently referenced lines in uncompressed form, in the data structure mentioned in Section 1. This data structure, termed a virtual compression cache or VCC would in general be of variable size, with the size determined by the current net compressibility of data and programs, as well as tradeoffs between paging rates and the percentage of cache misses which can be serviced by these uncompressed lines. In this study, we assume the existence of a VCC to hide decompression latency.

Management of the uncompressed area is approximately as follows: A compression line which is accessed on a cache miss, and which is not in the VCC, is decompressed and placed in the VCC. The VCC is FIFO ordered, as this is much easier to implement that an LRU structure. The size of the VCC is managed by a set of threshold algorithms [5], with the thresholds set by the operating system.

In practice, if the cache line size is smaller than the compression line, it may pay to include a small line buffer in the controller, to serve as a simple prefetching mechanism.

3 CTT Caching

We now consider candidates structures for CTT caching. The first is simply a separate CTT cache. Another approach, of which two variations are investigated here, allows the on-chip controller to use the L2 processor cache. Another possibility, which we discuss briefly, is use of the TLB as a caching mechanism; this however does not appear to be competitive with the others.

1. **Separate CTT cache**. Here recently accessed CTT entries, each comprising 16B, are cached. The cache organization investigated is 4-way set associative, with LRU replacement within each set. On CTT cache misses (associated with both L2 faults and writebacks), the referenced 16B cache line is entered into the cache. Performance results are also given for the case where the CTT cache has 64B lines; that is, the CTT entry for the entire page is fetched on a CTT cache miss.

2. **Use of the L2 cache**. Here access to a CTT entry can cause the entry into L2 of part of the CTT comprising the referenced 16B, plus neighboring entries so

as to fill an entire cache line. In the analysis below, cache lines are 128 bytes in size, so a cache line would include both 64B for the referenced page, as well as the entry for a page with a neighboring real address. Fetching the entire CTT entry for a page has a prefetch effect in that other 1K compression lines from this page are likely to be used.

A difference between this configuration and the CTT cache is the management of L2 castouts. If a CTT entry is fetched to determine where the castout line is to be stored, there may need to be another cache castout, requiring an additional CTT fetch etc. This potentially disastrous situation is avoided by not entering CTT data into the L2 cache for references associated with castouts. Only CTT references for cache fetches are placed in the cache. Two versions on L2 cache usage are studied. In the first, a CTT entry is treated like all others: LRU status within an equivalence class is affected only on references by the memory controller. This has the potential problem that such references, associated with L2 cache misses, are substantially less frequent than ordinary references, associated with L1 misses. The problem can be expected to more serious with larger L2 caches. An approach to this is to have synthetic updates or refreshes. Here each L1 miss to a line causes an update/promotion to the LRU status of the related CTT entry, provided the entry is in the cache. Yet a third possibility is to have the memory controller operate through the L1 cache. This, however, could at best provide a very limited improvement, reducing the time required to service an L2 miss by at best the difference between L1 and L2 latencies. In the following, data is provided for L2 caching without and with refreshing.

3. **Use of the TLB**. Here the TLB might provide a translation between program virtual addresses and physical addresses. Current TLBs provide a fast translation mechanism between virtual addresses and real addresses. In MXT, real addresses are further translated into physical via the CTT, and the TLB could provide translation for a set of recently referenced pages. The caches would then operate on physical addresses. L2 cache or TLB misses would then require access to the page tables, which could be problematic if such access further resulted in additional L2 misses. Examination of the modifications needed to provide such translation mechanisms suggests that the prior alternatives are probably more practical.

In the following section, approaches 1 and 2 are evaluated via trace-based simulation.

4 Materials and Methods

Simulation of the above alternatives requires an L1 miss trace. This is difficult to obtain for many current architectures because the L1 and L2 caches are usually on the same chip.

The trace used in this report is from a single processor of an IBM System/390 running MVS. This trace contains roughly 31,000,000 L1 misses, but no timing information.

The System/390 from which the trace was obtained has separate L1 data cache and instruction cache. Each is 128 KB with line size of 128B. We analyze L2 caches having line size equal to 128B. Thus when caching a CTT entry into L2, the cache line would contain the entire 64B descriptor for the referenced page, plus the descriptor for a page with a neighboring real address.

In all the experiments, the simulated L2 cache and separate CTT-cache are 4-way set-associative.

The cache simulator is written in C++, and runs on an IBM RS/6000 model 260, running AIX version 4.3.3.

5 Simulation Results

Figure 2 shows the L2 hit rate, as a function of size, and compares the two methods described above for caching CTT entries in L2 to systems that do not cache CTT entries in L2. The figure plots the percentage of L1 misses that are found in L2. The degradation in L2 hit rate due to caching CTT entries is (very roughly) about .04, a number which is insensitive to the L2 cache size. This means that the degradation decreases in percentage terms for larger L2s. The hit rate for L1 misses for the two caching approaches is roughly the same.

Figure 3 shows the fraction of L2 lines holding CTT information. For 1 megabyte L2 caches, the percentage is approximately 14 and 13 percent respectively for the case of refreshing and no refreshing. This translates into approximately 64KB of referenced CTT material, as each line holds 128B, representing a referenced page entry, and its neighbor in real address space. This is an underestimate of the total number of referenced entries, as the neighbor could also be referenced after entry into the cache.

Figure 4 shows the percentage of CTT references, on L2 faults, which are found in L2 or in the CTT cache. These are instances where hits reduce the latency for fetching the faulted L2 line. This shows that L2 caching of CTT entries outperforms a 64KB dedicated CTT cache, with the advantage to the former growing with the size

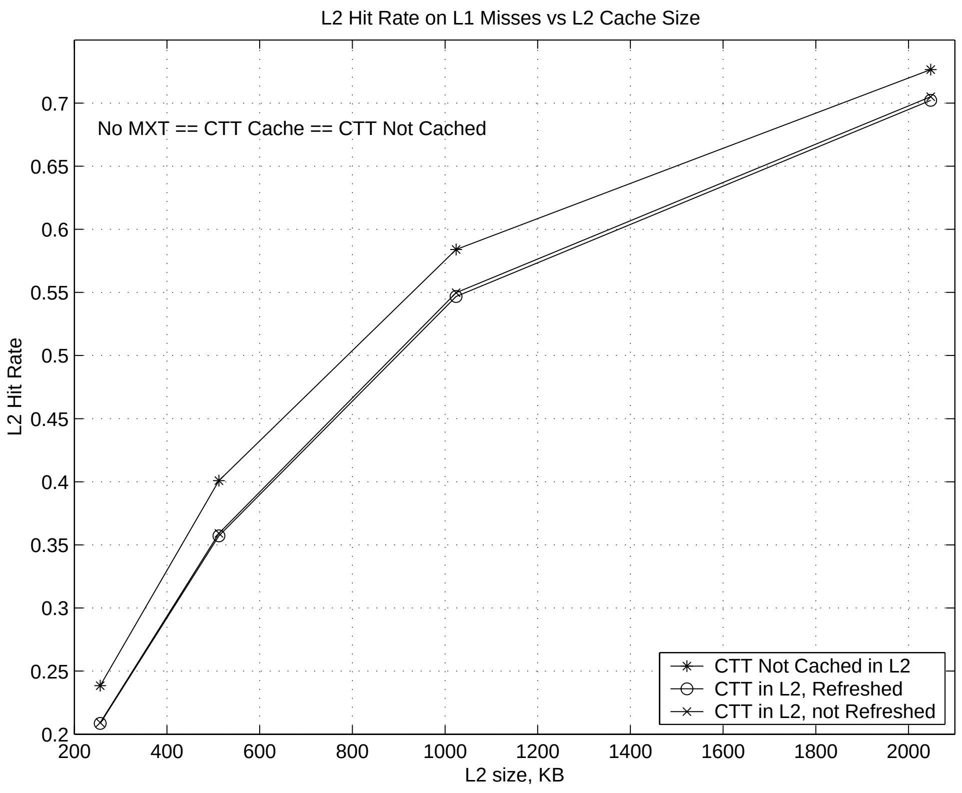

Figure 2:
The hit rate in L2 cache on L1 miss rates. Three curves are shown: one corresponding to cases where the CTT-entries are not cached in L2 (which covers regular, non MXT machines, MXT machines with a separate CTT cache, and MXT machines that do not cache CTT-entries), and two corresponding to the two alternative strategies for storing CTT-entries in L2.

of L2. Interestingly, CTT caching in L2 performs about as well as a 256KB dedicated CTT cache, for an L2 size of 2000KB.

Overall performance of the above approaches is dependent on the details of the chip implementation. Figure 5 shows L1 miss latency numbers, associated with L2 and dedicated CTT caches where the time to retrieve and decode a CTT entry is 1 memory cycle. A curve is also provided for the lower bound where such fetching and decoding is instantaneous. It is assumed that all main memory entries being fetched are held in the above-mentioned uncompressed area. Experimental data [7] suggests that for a variety of applications, an uncompressed area of a few tens of megabytes,corresponding to the L3 in the current implementation of MXT, has a miss ratio of a few percent. In all cases, MXT degrades the perceived memory latency. For example, in the case of an 1000KB L2, the average fetch latency for an L1 miss without MXT is approximately 4.6 memory cycle. This is reduced to less than four with a 2000KB L2. With MXT and no CTT caching, the corresponding latency is

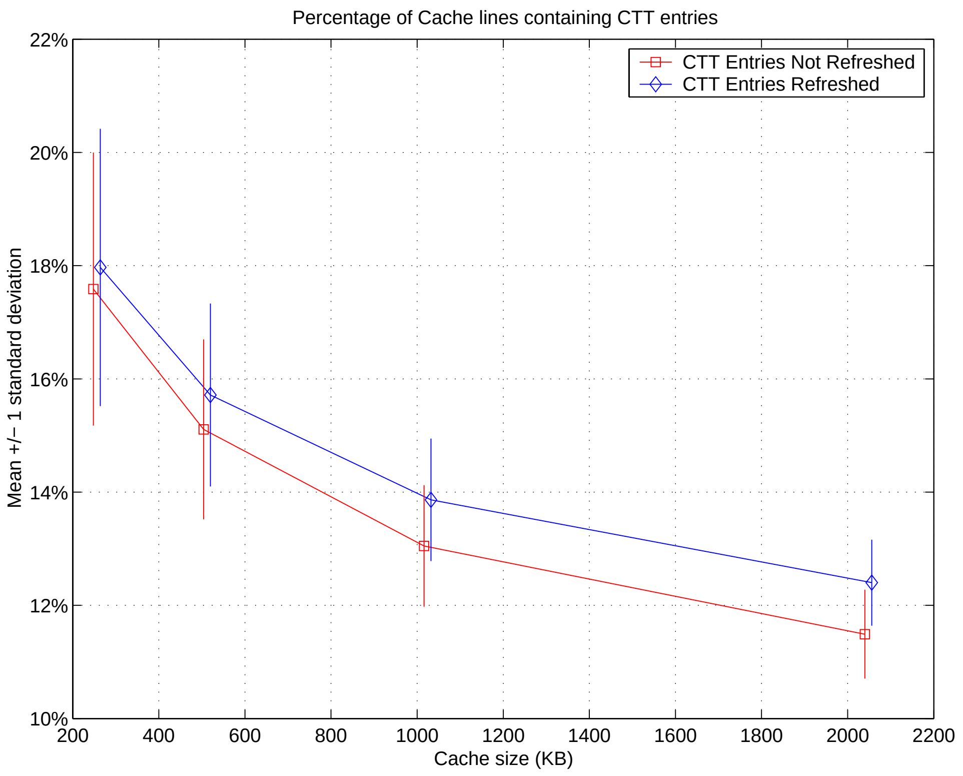

Figure 3: Percentage of L2 cache entries containing CTT-entries. The content of the cache was analyzed at regular intervals and the ratio of the number of lines containing CTT-entries to the cache size was computed for each sampling time. The plot shows the average of the resulting time series. The vertical lines represent the intervals of width equal to 2 standard deviations centered around the mean.

approximately over 10.2 cycles. This is reduced to about 6.3 with L2 caching with refreshing, with the difference from a non-MXT system largely represented by the time required to retrieve and decode a CTT-entry.

The above numbers assumed that all L2 misses could be serviced by the uncompressed area. In practice, the size of this area would be varied so as to optimize the tradeoff between memory latency and paging. It is perhaps instructive to illustrate the latency associated with decompression. Decompression is serial, and is done at the main memory bandwidth. Assuming 8B per cycle from memory, and that on average the required information is in the second 64B within the 1K compression line, the average latency given that the CTT entry has been fetched is 16 plus (assuming a compression ratio of 2 to 1) 128/(16) cycles, or 24 cycles, as compared to (128/8)=16 in an uncompressed memory.

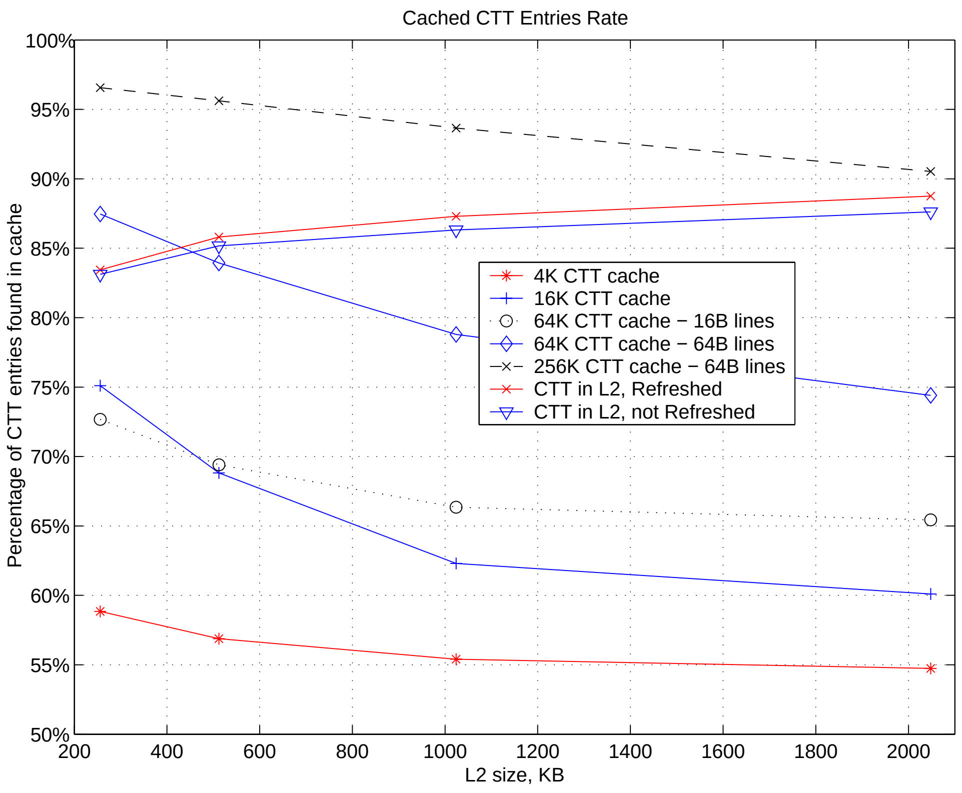

Figure 4:
Hit rate on cached CTT-entries, for separate CTT-caches of various sizes and the 2 methods for caching CTT-entries in the L2.

6 Conclusions

Compression of memory contents offers improved cost/performance tradeoffs. IBM's approach to such compression, MXT, includes in the current implementation a large L3 cache to hide memory latency. Such L3 caches may not be desirable with the current trend towards inclusion of memory controllers on the processor chip. Here hiding the latency of decompression may be done by incorporating a variable size list of uncompressed 1K lines, with the size of this list determined by tradeoffs between latency and paging delays. Access to data in memory is done via a compression translation table or CTT. Access to CTT entries causes additional latencies. These can be reduced substantially by including a caching mechanism for recently used entries. Two promising alternatives are i) a separate and dedicated CTT cache, as described in [5], and ii) a mechanism for including CTT entries in L2. The later configuration is made feasible by the incorporation of memory control on the processor chip. Simulation results were presented which showed that the latter alternative appears to work well, for example since the number of CTT entries cached varies automatically with the application, and also with the L2 size. Two configurations for

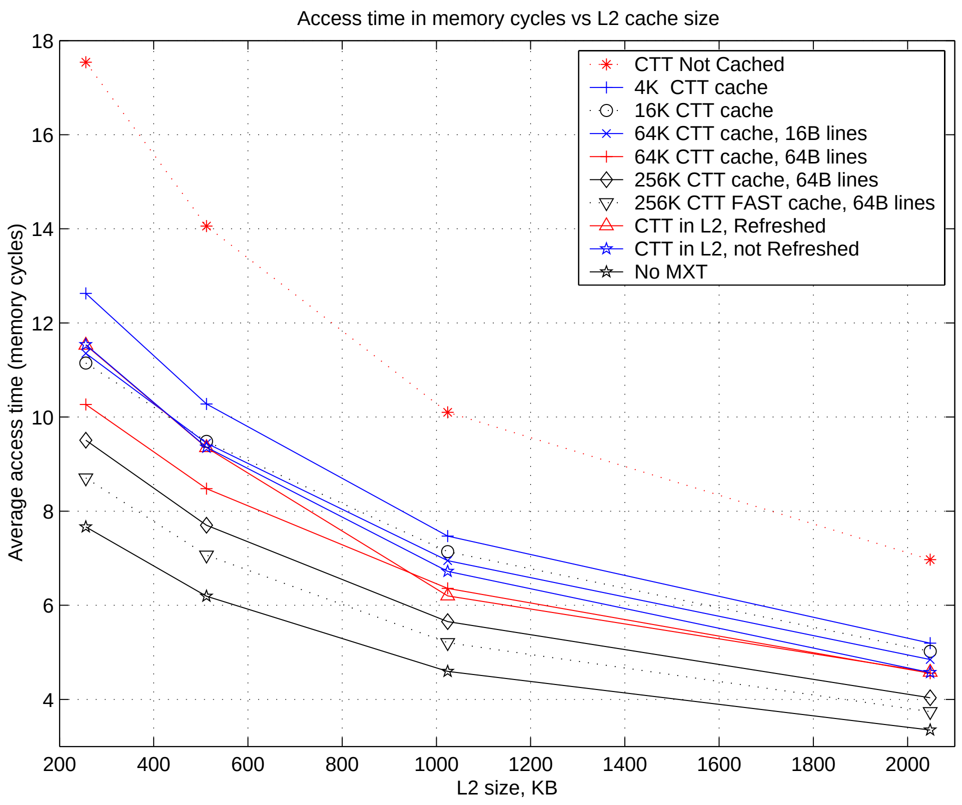

Figure 5: Access time in memory-bus cycles for L1 misses, as a function of the L2 cache size. For MXT machines, with a large uncompressed area, we assume that all memory lines accessed are in the uncompressed area. The timings of Figure 1 are used in the computation. We further assume that the penalty for a hit in L2 and in the CTT cache is 1 memory cycle.

such L2 caching were presented. In the first, CTT references are treated the same as L1 misses. In the second, there is additional refreshing of CTT entries to compensate for differences in inter-reference times. Both approaches appear to work well, with the latter showing a slight performance advantage, but at the cost of additional circuitry.

Acknowledgments

We would like to acknowledge Philip Heidelberger, John T. Robinson, Dan E. Poff, and Charles O. Schulz for insightful discussions.

References

[1] R. Tremaine, P. Franaszek, J. Robinson, T. Schulz, C.O.and Smith, M. Wazlowski, and P. Bland, “Ibm memory expansion technology (mxt),” *IBM J. Res. Devel.*, vol. 45, no. 2, pp. 269–284, 2001.

[2] P. Franaszek, P. Heidelberger, D. Poff, and J. Robinson, “Algorithms and data structures for compressed-memory machines,” *IBM J. Res. Devel.*, vol. 45, no. 2, pp. 243–256, 2001.

[3] P. Franaszek and J. Robinson, “On internal organization in compressed random access memories,” *IBM J.Res. & Devel.*, vol. 45, no. 2, pp. 257–269, 2001.

[4] P. Franaszek, J. Robinson, and J. Thomas, “Parallel compression with cooperative dictionary construction,” in *Data Compression Conference, DCC'96*, pp. 200–209, 1996.

[5] C. Benveniste, P. Franaszek, and J. Robinson, “Cache-memory interfaces in compressed memory systems,” *IEEE Trans. Computers*, vol. 50, p. ????, Nov. 2001.

[6] C. Benveniste, P. Franaszek, and J. Robinson, “Virtual uncompressed cache size control in compressed memory systems.” Patent Pending.

[7] B. Abali, H. Franke, D. Poff, R. J. Saccone, C. Schulz, L. Herger, and S. T.B., “Memory expansion technology (mxt): Software support and performance,” *IBM J. Res. Devel.*, vol. 45, no. 2, pp. 285–300, 2001.