Performance of Hardware Compressed Main Memory

Bulent Abali, Hubertus Franke, Xiaowei Shen, Dan E. Poff, and T. Basil Smith
IBM T.J.Watson Research Center
P.O. Box 218, Yorktown Heights, NY 10598
{abali,frankeh,xwshen,poff,tbsmith}@us.ibm.com

Abstract

A new memory subsystem called Memory Expansion Technology (MXT) has been built for compressing main memory contents. MXT effectively doubles the physically available memory. This paper provides an analysis of the performance impact of memory compression using the SPEC2000 benchmarks and a database benchmark. Results show that the hardware compression of memory has a negligible performance penalty compared to a standard memory. We also show that many applications' memory contents can be compressed usually by a factor of two to one. We demonstrate this using industry benchmarks, web server benchmarks, and contents of popular web sites.

1. Introduction

Data compression techniques are extensively used in computer systems to save storage space or bandwidth. Both hardware and software based compression techniques are used for storing data on magnetic media or for transmission over network links. While compression techniques are prevalent in various forms, hardware compression of main memory contents has not been used to date due to its complexity. The primary motivator of a compressed main memory system is savings in memory cost. Recent advances in parallel compression-decompression algorithms coupled with improvements in the silicon density and speed now make main memory compression practical [1,2,8,9]. A high-end, Pentium based, server class system with hardware compressed main memory, called the Memory eXpansion Technology (MXT), has been designed and built [8]. We ran numerous benchmarks on this new system. In this paper, we present the performance results and main memory compressibility of these benchmarks. Our results show that two to one compression (2:1) is practical for most applications. Results also show that performance impact of compression is insignificant. Two to one compression effectively doubles the amount of memory; or alternatively it provides the expected amount of memory at a reduced cost. Figure 1 illustrates possible savings in the price of a computer system if MXT were used. MXT not only doubles the amount of memory, but allows cheaper and lower density memory modules to be used. Thus savings in memory cost may exceed 50% with MXT.

Observations show that main memory contents of most systems, operating system and application memory included, are compressible. Only few applications' data, which are already compressed or encrypted, cannot be further compressed. In the MXT system, the Compressed Memory/L3 cache controller chip is central to the operation of the compressed main memory [8]. The MXT architecture adds a level to the conventional memory hierarchy. Real addresses are the conventional memory addresses seen on the processor external bus. Physical addresses are the addresses used behind the controller chip for addressing the compressed memory. The controller performs the real to physical address translation and compression/decompression of L3 cache lines. The processors are off-the-shelf Intel processors. They run with no changes in the processor or bus architecture. Common operating systems, Windows NT, Windows

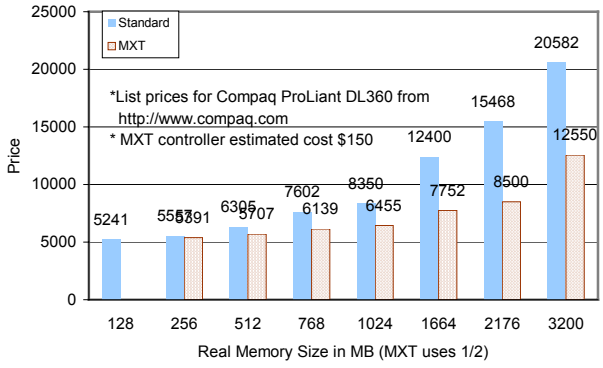

Figure 1. MXT vs Standard System Price Comparison

2000 and Linux run on the new architecture with no changes for the most part. However, a corner case exists where compressed memory may be exhausted due to incompressible data. This corner case is currently handled by small modifications in the Linux kernel [9] and by a device driver in the Windows NT and Windows 2000 operating systems.

Main contributions of this paper are as follows:

1. We present an overview of the MXT architecture and the compressed memory management software.

2. Using SPEC2000 CPU benchmarks, we show that MXT on the average runs 1.3% faster than a standard system without MXT hardware. This is mainly due to a large L3 cache that comprises double data rate (DDR) SDRAM. We provide benchmarks results and L3 cache statistics.
3. Using SPEC2000, we show that on the MXT system, performance impact of compression over no compression is 0.4% on the average, although the effective size of the memory is doubled.
4. We show using a database benchmark that the performance significantly increases by doubling the memory size.
5. We further show that for a number of applications, main memory contents and the contents of several popular web sites can be compressed by a factor of 1.54 to 2.68.

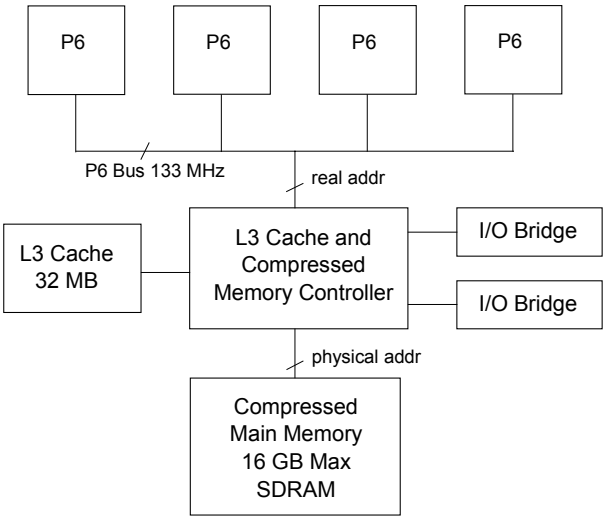

Figure 2. MXT Hardware Organization

In related work [3,4], the authors describe a method of estimating the number of page frames as a function of physical memory utilization. They further model the residency of outstanding I/O as data is transferred into the memory through the L3 cache, thus potentially forcing cache write backs that could increase the physical memory utilization. Using a time decay model they evaluate the system behavior using simulation. In [5], an approach is described where a page-based hardware data compression engine sets aside a part of physical memory as a compressed paging space. In [6], a software solution is simulated that sets aside a part of the physical memory as a compressed paging space. In both cases, if the compressed paging space is filled up, compressed pages are swapped out, thus reducing the I/O overhead incurred. In [9] operating system techniques for managing compressed memory are described and demonstrated.

In the following section we give an overview of the MXT hardware and we describe the memory compression support added to the Linux operating system. In Section 3, we present results of various benchmark runs on the MXT system. In Section 4, we examine the compressibility of various applications' memory contents.

2. Overview of MXT

2.1 The Hardware

The organization of the MXT system is shown in Figure 2. The main memory (SDRAM) contains compressed data and can be up to 8 GB in size (16 GB effective due to compression). The third level (L3) cache is a shared, 32 MB, 4-way set associative write-back cache with 1 KB line size. The L3 cache is made of double data rate (DDR) SDRAM. The L3 cache contains uncompressed cached lines. L3 hides the latency of accessing the compressed main memory. The L3 Cache/Compressed Memory Controller is central to the operation of the MXT system. The L3 cache appears as the main memory to the upper layers of the memory hierarchy and its operation is transparent to the rest of the hardware including the processors and I/O. The controller compresses 1 KB cache lines before writing them to the compressed memory and decompresses them after reading from the compressed memory.

The compression algorithm is a parallelized variation of the Lempel-Ziv algorithm known as LZ1. The compression scheme stores compressed data blocks to the memory in a variable length format. The unit of storage in compressed memory is a 256 byte *sector*. Depending on its compressibility, a 1 KB block of memory (which is the same size as an L3 line) may occupy 0 to 4 sectors in the compressed memory. Due to this variable length format, the controller must translate real addresses on the system bus to physical addresses in the physical memory. Real addresses are conventional addresses seen on the processor chip's external bus. Physical addresses are used for addressing sectors in the compressed memory. The real memory is an abstraction as the virtual memory is in contemporary architectures. The real memory is merely an address space whose small sections reside in the L1/L2/L3 caches for immediate access. Rest of the memory contents reside in the physical memory in compressed form. The memory controller performs real to physical address translation by a lookup in the Compression Translation Table (CTT), which is kept at a reserved location in the physical memory. Each 1 KB block's real address maps to one entry in the CTT, and each CTT entry is 16 bytes long. A CTT entry

includes four physical sector addresses each pointing to a 256-byte sector in the physical memory. For example, a 1 KB L3 line, which compresses by two to one, will occupy two sectors in the physical memory (512 bytes) and the CTT entry will contain two addresses pointing to those sectors. The remaining two pointers will be zero. For blocks that compress to less than 120 bits, for example a block full of zeros, a special CTT format called *trivial line* format exists. In this format, the compressed data is stored entirely in the CTT entry replacing the four address pointers. Therefore, a trivial block of 1 KB occupies only 16 bytes in the physical memory resulting in a compression ratio of 64 to 1. Note that the compression operations described so far are transparent to the processors, I/O devices, application software and device drivers.

Note that the selection of the 1 KB line size was influenced by many factors. L3 directory size, which grows inversely proportional to the cache line size for a given cache size, and the compression block size that affects the compression efficiency were the two most significant factors for the 1 KB line size [8]. Shorter lines may not compress well and longer lines may impact performance due to longer compression and decompression times.

2.2. The Compressed Memory Management Software

Since the compressed main memory concept is new, common operating systems do not have mechanisms to deal with out-of-physical-memory conditions. The compressed memory management software addresses this problem. For Linux, minor changes to the kernel were made [9]. For WinNT and Win2000, since kernel source code was not available, a device driver was implemented. The MXT hardware allows an operating system to use a real address space larger than the physical address space. During the boot process, the system BIOS reports having more memory than the physical memory. For example, the particular MXT system we used has 512 MB of physical memory, but BIOS reports having twice that amount, 1 GB of memory. The main problem in such a system occurs when applications fill the memory with incompressible data while the operating system had committed more real memory than physically available. In these situations, the common OS is unaware that the physical memory may be exhausted. The compressed memory management software uses the following mechanisms to prevent physical memory exhaustion:

1. Receives a warning interrupt from the memory controller that physical memory is running out.
2. Blocks further memory allocation by reserving pages either explicitly (device driver allocation) or implicitly (VMM modifications). Activates the swap daemon to shrink file caches and to swap out some user process pages, hence forcing some memory to be freed.
3. Fills those freed pages with zeros to reduce physical memory utilization, since zero filled cache lines occupy only 1/64 of their actual size in the physical memory.
4. Activates a set of busy spinning threads to stall the execution of processes that exhaust physical memory if items 2 and 3 above cannot offset the increase in the physical memory utilization fast enough.

An MXT system running out of physical memory behaves similar to a conventional system with insufficient memory and therefore may have increased swap activity. We refer readers to [9] which explains these mechanisms in further detail.

3. Performance Impact of Compression

The MXT memory system uses a relatively long 1 KB compression block size to be able to compress efficiently, since shorter blocks may not compress well. Due to the compression and decompression operations performed on these blocks in the memory controller, memory access times are longer than usual. The 32 MB L3 cache contains uncompressed (1 KB) lines to reduce the effective access times by locally serving most of the main memory requests. Since this type of memory organization is new, we used industry standard CPU benchmarks to measure its performance impact. We also present performance results of a database benchmark.

3.1. Methods

In these experiments, we used an MXT system with 512 MB physical memory, effectively having 1 GB real memory. The Compression Translation Table is placed by BIOS at the end of the physical memory and occupies about 8 MB space (16 bytes/cache line or 64 bytes/page) in the physical memory. The processor external bus (P6 bus) speed is 133 MHz. The system is comprised of two 733 MHz PIII processors and a single disk drive. The MXT system used ServerWorks Pinnacle memory controller chip incorporating MXT. We compared this to a fairly standard system using the same processors and same SDRAM, and using the ServerWorks CNB30LE memory controller chip, which is fairly similar to Pinnacle however without compression or L3 support.

We used the SPECint2000 benchmark suite designed by the SPEC consortium to measure the performance of the CPU and the memory (http://www. spec.org/osg/cpu2000/). SPECint2000 suite requires at least 256 MB of memory. There are 12 integer benchmarks in the suite. These are the *gzip* data compression utility, *vpr* circuit placement and routing, *gcc* compiler, *mcf* minimum cost network flow solver, *crafty* chess program, *parser* natural language processing, *eon* ray tracing, *perlbmk* perl utility, *gap* computational group theory, *vortex* object oriented database, *bzip2* data compression utility, and *twolf* place and route simulation benchmarks.

We ran benchmarks three times, once on the standard system, and twice on the MXT system with the compression on and off. The difference between the standard and the MXT with compression off results give the performance impact of L3 cache. The difference between compression off and compression on results gives the performance impact of the compression-decompression hardware. The MXT system has a boot option that permits compression to be turned off. In that case, the system operates as a standard with an L3 cache and with non-compressed memory. Since compression/decompression hardware is disabled, the memory access latency is expected to be different than that of the compression-on case. We also recorded the number of L3 requests and L3 misses using the performance counters built into the memory controller chip.

3.2. Results

Main results of the SPECint2000 experiments are shown in Fig. 3. The right most three columns are the average SPECint rates for the three systems that we considered. Due to strict reporting requirements of the SPEC consortium, we cannot publish the actual execution times. Therefore, we normalized the compression on and off results relative to the standard system results.

Results show that on the average, MXT with compression-on is 1.3% faster than a standard system. The individual benchmarks *twolf*, *vpr*, *parser*, *gcc*, and *bzip2* perform 4.0 to 8.3% faster on the MXT system as shown in Fig. 3. *Mcf* runs 1.1% faster on MXT. The L3 miss rates (Fig.4) and L3 request rates (Fig.5) for these six benchmarks reveal the reason: their miss rates are relatively small, but their L3 request rates are the highest among the twelve benchmarks. In other words, the working set of these six benchmarks fit in the 32 MB L3 cache and since they make large number of L3 requests, the L3 cache comprising double data rate (DDR) SDRAM gives a slight performance advantage over the standard system comprising regular SDRAM. On the contrary the benchmarks *vortex*, *gzip*, and *gap* respectively run 2.1, 2.4, and 10% slower on the MXT system than the standard system. Fig.4 shows that these three benchmarks have the highest miss rates among the twelve benchmarks (*gap* that has the worst performance and the highest miss rate.) It also has the highest misses/second metric, which is 2.6 times greater than the next highest. Another observation is the relatively small L3 request rates for *crafty* and *eon* which indicate that their working sets entirely fit in the L1 and L2 caches. Therefore the L3 cache does not impact the performance of crafty and eon, which reveals itself as a negligible difference in their SPEC rates on Figure 3.

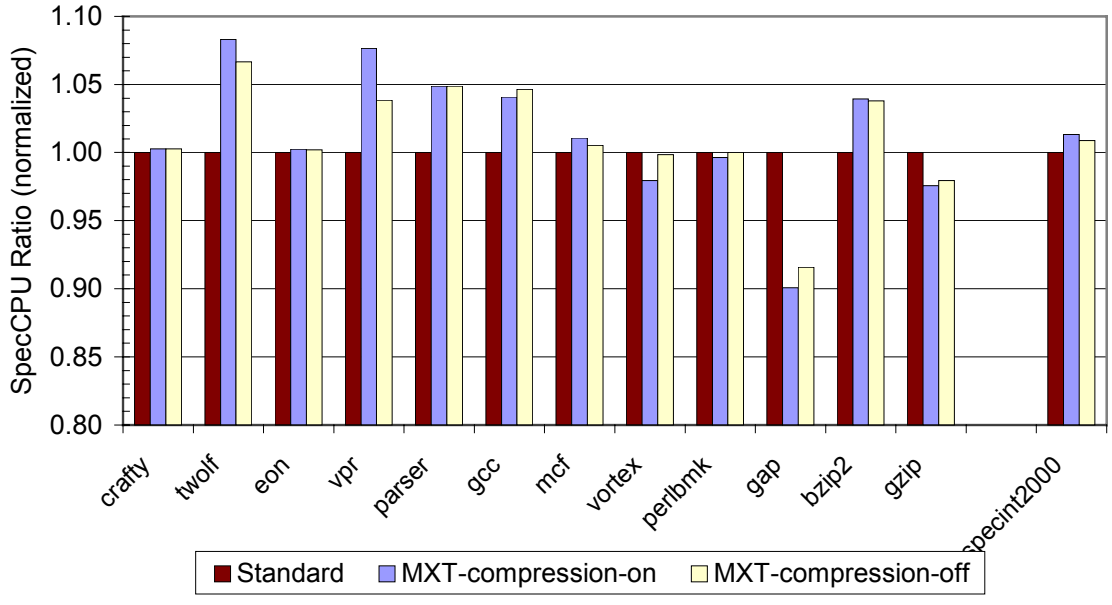

Figure 3. SPECint2000 results for three system configurations

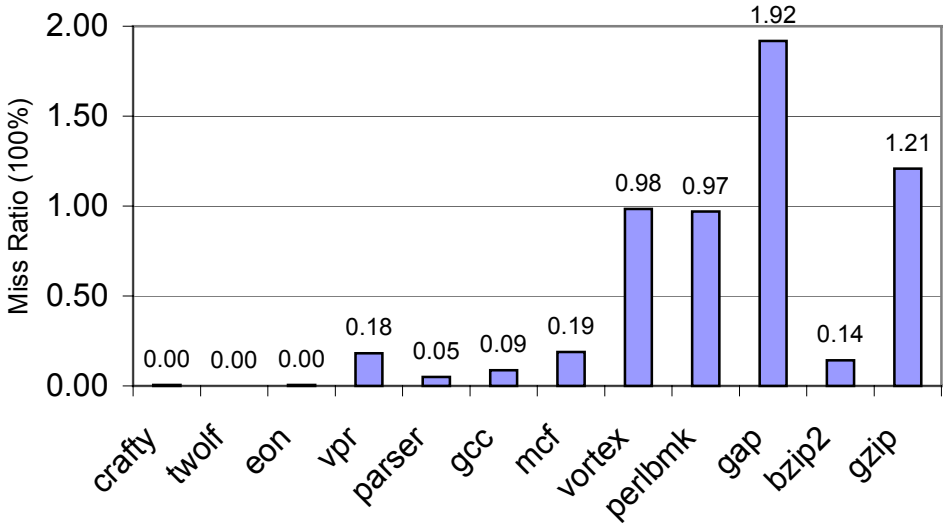

Figure 4. Miss ratios of the benchmarks

Comparisons between MXT with compression and without compression show that the compression on case is 0.4% faster than the compression off case on the average. Figure 3 shows that *vpr* and *twolf* have the greatest difference in favor of the compression on case. One possible explanation is that cache misses of these two benchmarks may result in large number of *trivial* line compression and decompressions. As explained before, a *trivial* line is an L3 cache line, which compresses to less than 120 bits and therefore is stored in a 16 byte CTT entry. Compression and decompression of trivial lines have much less overhead than that of a line occupying a sector (256 bytes) or more in the physical memory. For example, cold cache misses at the beginning of execution will almost always result in trivial line compressions because the memory is filled with zeros initially. The compression off case runs faster than the compression on case for the *gap* and *vortex* benchmarks by 1.5 and 1.9% respectively. They have the first and third highest miss rates, and first and second highest misses per second.

In summary, the impact of the L3 cache and memory compression for the set of benchmarks is negligibly small considering that MXT doubles the amount of memory.

3.3. Database Benchmark Results

The MXT system has been measured running a typical insurance company database schema. This configuration is primarily used within IBM as a quick regression test for ascertaining the impact of DBMS design changes. It is substantially less costly and quicker to run than complex benchmarks such as TPC-C, but is coarsely representative of the performance characteristics that might be expected. Several configurations were run on the prototype hardware: 512 MB with MXT off, 512 MB (1 GB expanded) with

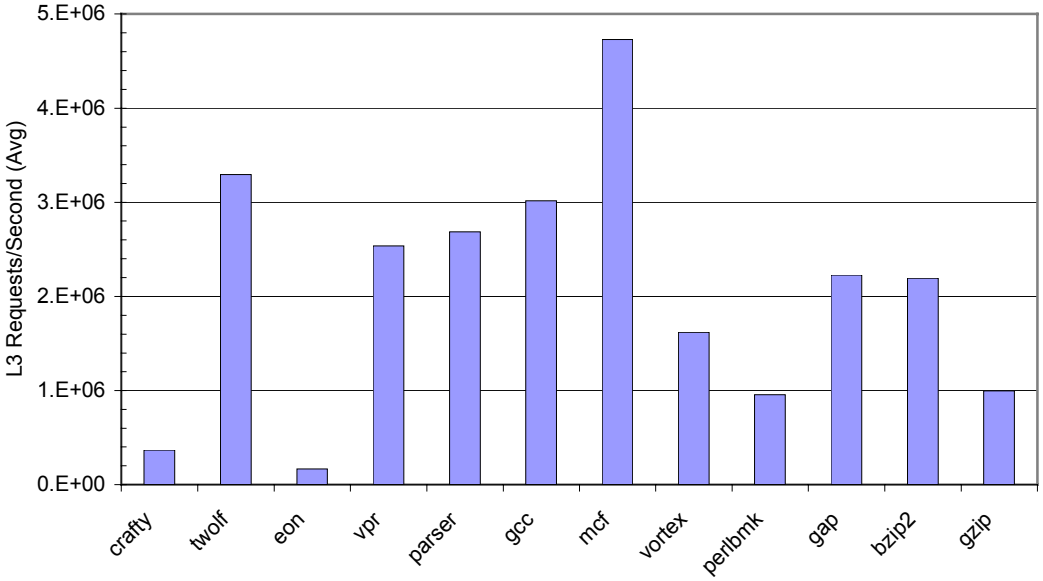

Figure 5. L3 request rates of the benchmarks

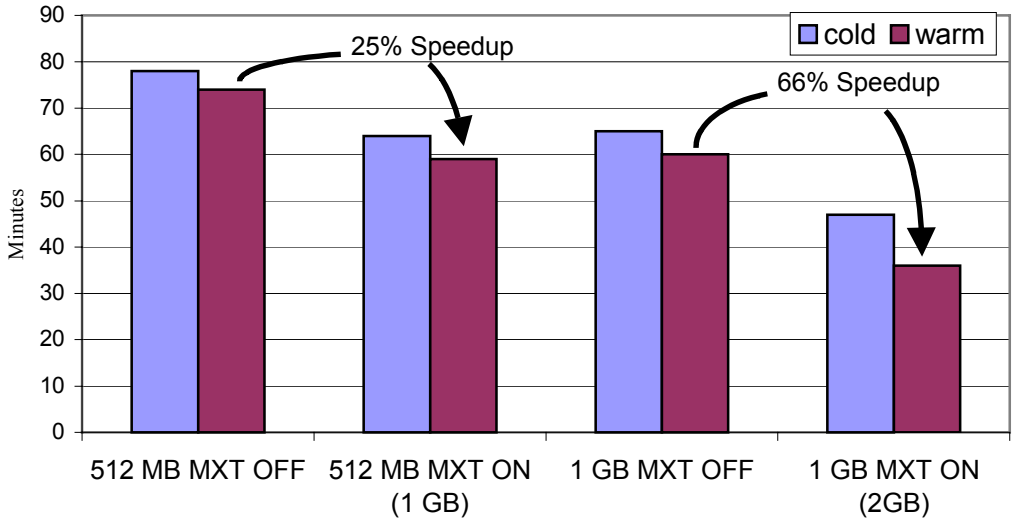

Figure 6. Database benchmark results for four different configurations

MXT turned on, 1 GB MXT off, and 1 GB MXT on (2 GB expanded) configurations. Two runs were made for each configuration, a cold run where the file cache is initially empty, and immediately following that a second warm run, where the buffers have been "warmed" by the preceding cold run.

Figure 6 shows the performance benefits of MXT. For the 512 MB system when compression is on, it doubles the effective amount of memory and the benchmark runs 25% faster than the compression off case. For the 1 GB system when compression is on, it doubles the effective amount of memory to 2 GB and the benchmark runs 66% faster than the compression off case. It is interesting to note that the benefit of larger memory is more pronounced for this workload for larger memory sizes, and is indicative of both the smaller 512 MB memory and 1 GB memory configurations being memory starved. For this workload, system performance is improved by 66%.

4. Compressibility of Applications

Now that the performance of the MXT system is established, we turn our attention to the compressibility of main memory contents of various applications. We analyzed the SPEC2000 CPU benchmarks, the contents of several web sites, and a web server benchmark. We measured the compression ratios on the actual hardware. Results show that most of these applications' main memory contents can be compressed usually by a factor of 2:1, justifying the real to physical memory ratio chosen for the MXT systems.

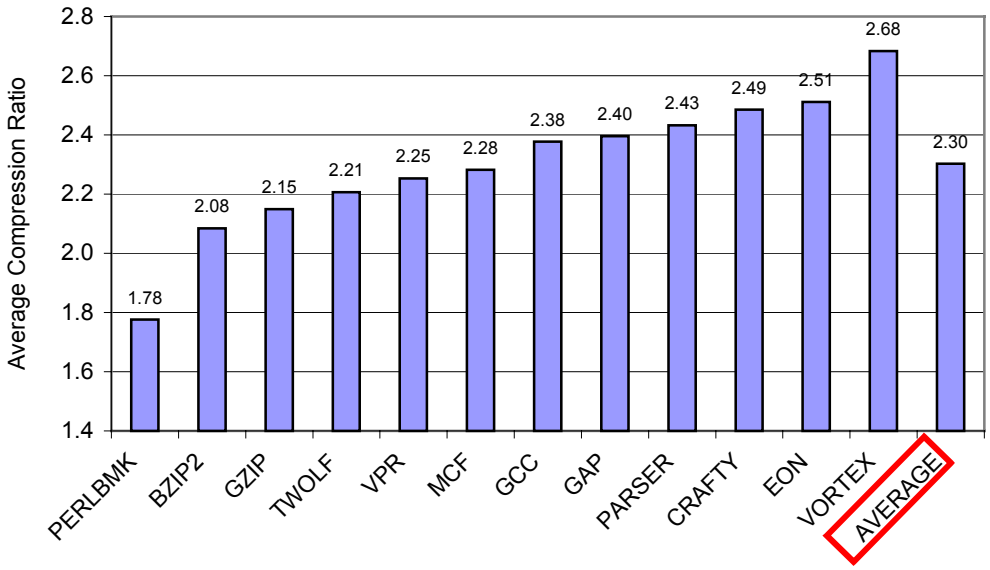

Figure 7. Compressibility of SPEC benchmarks

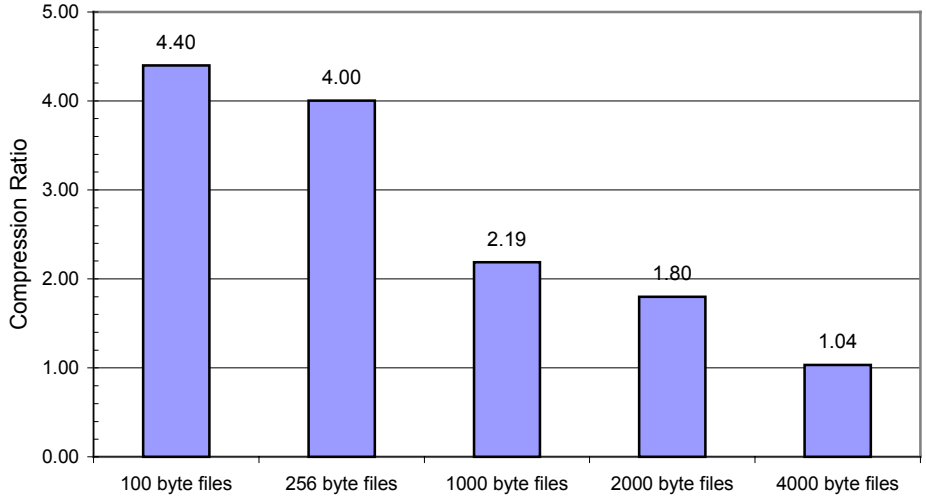

Figure 8. Compressibility of the memory footprints of "incompressible" files

4.1. Methods

For the SPEC2000 benchmarks, the real and physical memory utilizations were sampled every two seconds. The Sectors Used Register (SUR) in the memory controller reports to the operating system physical memory utilization. We exported this register to the user space through the /proc file system of Linux. The sampler program reads every two seconds the SUR register as well as the real memory utilization as reported by Linux and saves them in a file to be processed later. The measured memory values are for the entire memory. Therefore, in addition to the benchmark application's memory utilization, the measurements include possibly large data structures such as file cache and buffer cache that the OS maintains for efficient use of the system. In a post-processing step we took the average of the samples to produce the average compression ratio of a given benchmark.

For web content and web server benchmarks, we used additional methods and we will postpone their discussion until after the next section.

4.2. SPEC Results

Physical memory utilization of applications is generally not constant but varies over time according to the compressibility of memory contents. Figure 7 shows the average compression ratio for each benchmark run. We defined the average compression ratio as the time averaged real memory utilization over time averaged physical memory utilization. In this set of 12 benchmarks, the smallest compression ratio of 1.78 was observed for *perlbmk* and the largest compression ratio of 2.68 was observed for the *vortex* benchmark. The average of all 12 benchmarks was 2.30. Thus, the fixed real to physical memory ratio of 2.0 used by the MXT system is well justified for this set of applications. All benchmarks had a compression ratio better than 2.0 except for *perlbmk*.

4.3. Methods for Web Content and Web Server Benchmarks

Web applications require a special treatment for determining the compressibility, because memory contents of the system depends on the web content as well as caching policies of the web server, and the file system policies of the operating system. We analyzed the web server benchmarks WebBench 3.0, SPECweb99, and the contents of well known Internet portals such as www.yahoo.com, and home.netscape.com. In this section we discuss some of the issues in determining compression characteristics before presenting our results.

A problem that we encountered while analyzing web server benchmarks is the method with which web content has been generated. SPECweb99 benchmark uses synthetic content created during installation. The installation program generates a large number of files filled with random strings that are generally incompressible. Since SPECweb99 content was not realistic we eliminated it from further consideration. WebBench 3.0 benchmark's content, on the contrary, contains a mixture of HTML and GIF files that have been copied from a real web site.

Another issue is GIF and JPEG files widely used on web pages. These are graphics files that use

compressed file formats, and therefore it is generally assumed that they would be incompressible on the MXT hardware. However, our measurements indicated otherwise. The primary reason for the better than expected 1.0 compression ratio is the fact that graphics files on web pages are often small (few hundreds of bytes each), yet they have a larger footprint when mapped to the memory. Operating systems such as Linux and NT generally allocate memory for file objects in increments of a page size (4096 bytes). Therefore, a small file would occupy one page in the memory regardless of its size. To prove this point we did the following experiments:

We populated the file system with thousands of 100 byte size incompressible files. (An incompressible file can be created easily by compressing any file with a software utility such as zip, gzip, or compress.) We forced the 100 byte files into the OS filecache by copying them to /dev/null. We repeated this experiment with 256, 1000, 2000 and 4000 byte size incompressible files. We calculated the compression ratio as the increase in the real memory utilization divided by the increase in the physical memory utilization. Figure 8 shows the compression ratio measured on the MXT hardware for this experiment. For small incompressible files it can be seen that the compression ratio is much larger than 1.0 for the reasons discussed above. For the 4000 byte size incompressible files, the compression ratio is much closer to the expected value of 1.0.

Given our discussions on small files above, one would expect the compression ratio to be much higher than what was measured in Figure 8. For example, a 100 byte file should occupy one 256 byte sector in the physical memory, thereby resulting in a compression ratio of 4096/256=16 vs. the measured 4.40. For small files, certain filesystem characteristics come into play. In Linux, when files are read from a block device (i.e. disk) into the file cache, the smallest unit of transfer is 1024 bytes. Clearly more than 100 bytes are read into the file cache, which may affect the compression ratio. Other factors may also play a role: some disk blocks may be prefetched and the inode structures maintained for of each file take up memory space. In summary, small files are wasteful in real memory due to the file system overhead and therefore memory footprints of small incompressible files can still have better than 1.0 compression ratio.

In order to determine the compressibility of important web sites, we mirrored their content to a local disk. Several software tools exist for mirroring a web site. We used Wget on Linux and WebReaper on Windows (http://www.otway.com/webreaper). Given a root URL, these tools follow all the links in the root page down to a specified depth and create a local copy of all the files retrieved. The local mirror allows browsing of the web site while disconnected from the network. For example wget http://www.yahoo.com will create a mirror of the popular Yahoo web site. Once a local mirror was obtained, we copied all the files to the device /dev/null to force them into the file cache. Then, the compression ratio is simply determined as the real memory utilization over the physical memory utilization as reported by the MXT

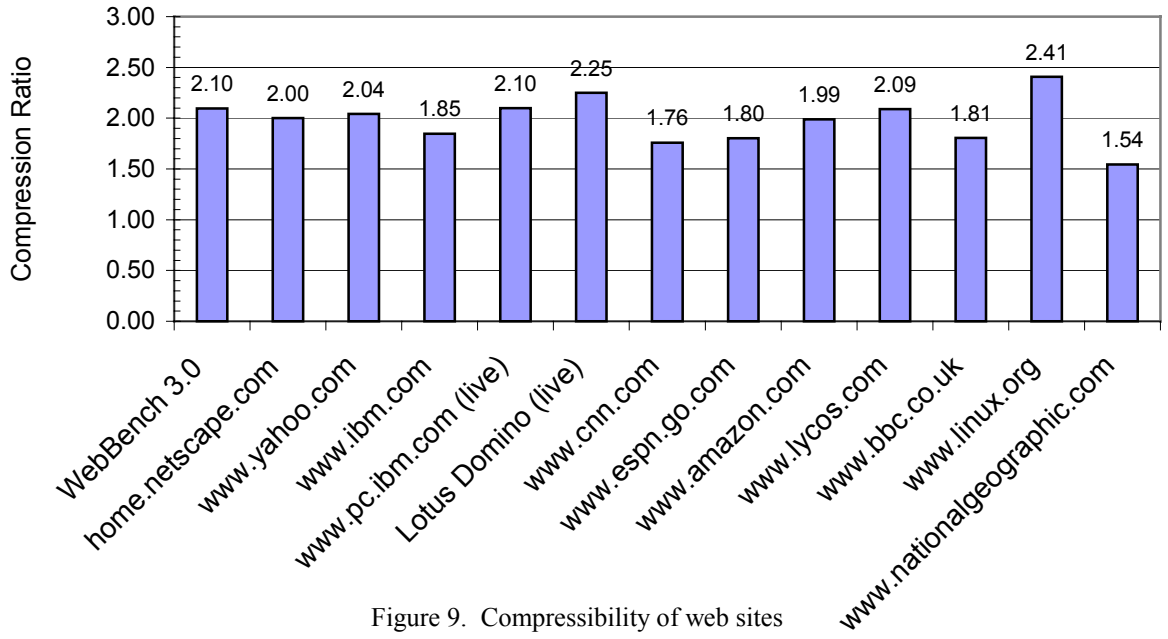

Figure 9. Compressibility of web sites

controller.

It should be mentioned that the approach we used here is an approximation. It is not possible to mirror an entire web site as it would have taken very long time. Depending on the selected web site a link depth between 4 and 10 to obtain a sizeable local copy. Only static pages can be mirrored; for example, pages with dynamic content, executables or pages that require user input are not handled by this approach. Also, all the files on a web server are not necessarily resident in the memory at once, because some files are accessed with much less frequency than others. Therefore the memory contents may be different than the disk contents.

4.4. Web Results

We mirrored well-known Internet portals such as www.yahoo.com and home.netscape.com on the local disk of the MXT system. Figure 9 shows the measured compression ratios for these mirrored web sites. Results show that they compress well and near the real to physical memory ratio of 2.0 chosen for the MXT hardware (1GB/512MB.) Two of the results in Figure 9, www.pc.ibm.com and Lotus Domino with compressibility of 2.1 and 2.25 are actual web servers. We measured these results by sampling memory contents of the servers and then running a simulation of the hardware compression.

5. Conclusion

In this paper we described and evaluated a computer system with hardware main memory compression that effectively doubles the size of the main memory. We gave an overview of the software support for main memory compression. We measured the impact of compression on the application performance using industry benchmarks and determined that the hardware compression has a negligible penalty over a non-compressed hardware. We demonstrated the price-performance advantage of MXT by a database benchmark. We measured real and physical memory utilization of industry benchmarks and determined that main memory contents can be compressed by a factor of 2.3 on the average. We mirrored contents of some web sites and determined that main memory contents of web servers carrying such content can be compressed by a factor of 1.54 to 2.41.

Acknowledgements

We thank Brett Tremaine and Mike Wazlowski who described us the operation of the L3 Cache/Memory Compression Controller chip and Abhishek Gulati and Jiesheng Wu who collected some of the web compressibility data points.

References

[1] Hovis et al., "Compression architecture for system memory application," US Patent 5812817, 1998.

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

[3] Franaszek, P., Robinson, J., "Design and Analysis of Internal Organizations For Compressed Random Access Memory," IBM Research Report RC21146, Yorktown Heights, NY 10598.

[4] Franaszek, P., Heidelberger, P., Wazlowski, M.: "On Management of Free Space in Compressed Memory Systems", *Proceedings of the ACM Sigmetrics*, 1999.

[5] Wilson, P, Kaplan, S., Smaragdakis, Y.: "The Case for Compressed Caching in Virtual Memory Systems", *USENIX Annual Technical Conference*, 1999.

[6] Kjelso, M, Gooch, M., Jones, S.: "Empirical Study of Memory Data: Characteristics and Compressibility," In *IEEE Proceedings of Comput. Digit. Tech*, Vol 45, No. 1, pp 63-67, IEEE, 1998.

[7] Vahalia, U: "Unix Internals, The New Frontiers", Prentice Hall, ISBN 0-13-101908-2, 1996

[8] Arramreddy, S., Har, D., Mak, K., Smith, T.B., Tremaine, B., Wazlowski, M.: "IBM X-Press Memory Compression Technology Debuts in a ServerWorks NorthBridge," *HOT Chips 12 Symposium*, Palo Alto, CA, Aug.13-15, 2000.

[9] Abali, B., and Franke, H.: "Operating System Support for Fast Hardware Compression of Main Memory", Memory Wall Workshop, *Intl. Symposium on Computer Architecture (ISCA2000),* Vancouver, B.C., July 2000.