RC 21799 (98138) 19 July 2000 **ComputerScience/Mathematics**

IBM Research Report

Performance of Hardware Compressed Main Memory

Bulent Abali, Hubertus Franke, Dan E. Poff, T. Basil Smith
IBM T. J. Watson Research Center
P. O. Box 218
Yorktown Heights, NY 10598

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

Performance of Hardware Compressed Main Memory

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

Abstract

A novel memory subsystem called Memory Expansion Technology (MXT) has been built for compressing main memory contents. This allows effectively a memory expansion that presents a "real" memory larger than the physically available memory. This paper provides an overview of the architecture and OS support and in-depth analysis of the performance impact of memory compression using the SPEC2000 benchmarks. Our results show that the hardware compression of memory has a negligible penalty compared to a non-compressed memory. We also show that most applications' memory contents can be compressed usually by a factor of two to one. We demonstrate this using industry benchmarks, webserver benchmarks, and the 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 makes 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 in cost sensitive applications it provides the expected amount of memory at ½ of the expected cost.

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. Standard operating systems, Windows NT, Windows 2000 and Linux, run on the new architecture with no changes for the most part. However, a boundary condition exists where compressed memory may be exhausted due to incompressible data. This boundary condition and the compressed memory management is 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 give an overview of the MXT architecture and the compressed memory management software.
2. Using industry benchmarks, we show that the performance penalty of compression on the MXT hardware is 1.5% on the average, although the effective size of the memory is doubled.
3. We further show that a number of applications main memory contents and the contents of several popular websites can be compressed by a factor of 1.78 to 2.68.

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 they transfer data 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 experimental results of running SPEC benchmarks on the MXT system. In Section 4, we examine the compressibility of various applications' memory contents.

2. Overview of MXT

The Hardware

The organization of the MXT system is shown in Figure 1. The main memory (SDRAM) contains compressed data and can be up to 16 GB in size. The third level (L3) cache is a shared, 32 MB, 4-way set associative write-back cache with 1 KByte line size. The L3 cache is made of double data rate (DDR) SDRAM. The L3 cache contains uncompressed cached lines. It 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 cache lines to the compressed 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 cache line may occupy 0 to 4 sectors in the compressed memory. Due to this variable length format, the controller must translate real addresses to physical addresses. A 1 KB cache line (real) address is mapped to 0 to 4 sector (physical) addresses in the compressed memory. The real address is the conventional address seen on the processor chip's external bus. The physical address is used for addressing the sectors in the compressed memory. 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

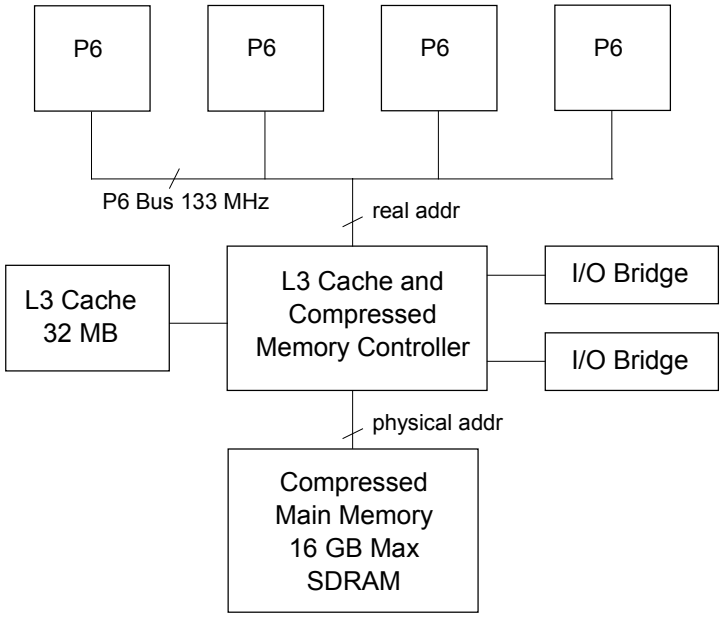

Figure 1: Compressed Memory System Organization

memory. Each 1 KB cache line address maps to one entry in the CTT, and each CTT entry is 16 bytes long. A CTT entry contains control flags and four physical addresses each pointing to a 256-byte sector in the physical memory. For example, a 1 KB cache line, which compresses by 2:1, will occupy two sectors in the compressed memory (512 bytes) and the CTT entry will contain two addresses pointing to those sectors. The remaining two pointers will be zero. For cache lines that compress to less than 120 bits, for example a cache line 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 line of 1 KB occupies only 16 bytes in the physical memory resulting in a compression ratio of 64:1. Another memory saving

optimization implemented in the controller is sharing of sectors by *cohort cache lines.* If two 1 KB cache lines are in the same 4 KB memory region they are called *cohorts.* Two cohorts may share a sector provided that space exists in the sector. For example, two cohorts each compressing to 100 bytes may split and share a sector since their total size is less than the sector size of 256 bytes. The compression operations described so far are entirely done in hardware with no software intervention.

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

The Compressed Memory Management Software

Since the compressed main memory concept is new, common operating systems do not have mechanisms to distinguish between real and physical memory nor do they 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 larger amount of real memory than physically exists. 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 application(s) start filling 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 is being 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 exhaustion is near.
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 freed.
3. Fills those freed pages with zeros to reduce physical memory utilization, since zero filled cache lines occupy only 1/64th 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.

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 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 the 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 present the results of this study in the next section.

Methods

In these experiments, we used an MXT system with 512 MB physical memory,

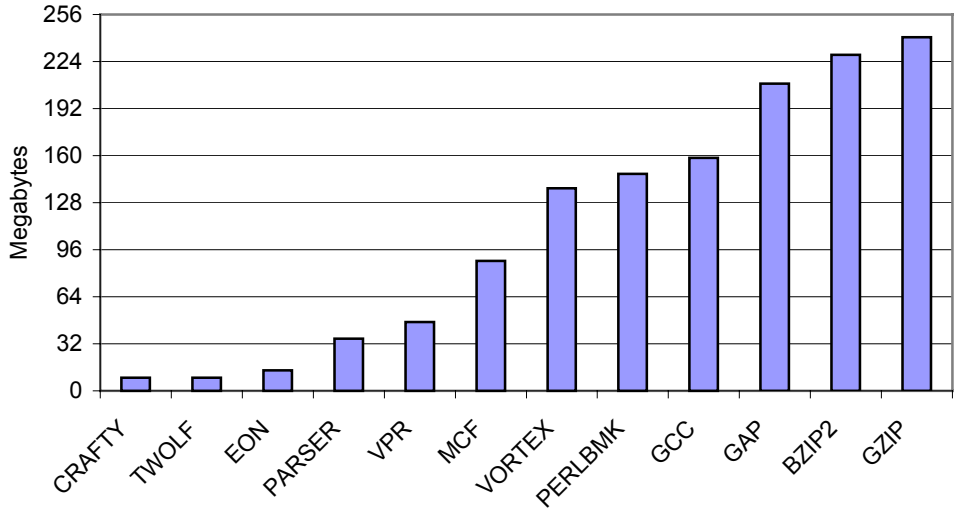

Figure 2: Memory footprints of SPEC2000 benchmarks

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) shown in Fig.1 ran at 91 MHz in this early hardware prototype, although it was designed for a 133 MHz operation. The system is comprised of a single 455 MHz Xeon processor and a single disk drive.

We used the SPEC CPU2000 benchmark suite designed to measure the performance of the memory as well as processor speed (http://www.spec.org/osg/cpu2000/), and in addition requires at least 256 MB of RAM. There are 12 integer benchmarks in the SPEC2000 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. Figure 2 shows the memory utilization of these benchmarks, measured in terms of the increase in the system memory utilization after a benchmark was started. Thus, the memory used by the operating system and other processes is excluded from the values shown in the figure. Figure 2 shows that all but three of the benchmarks have a memory footprint larger than the 32 MB L3 cache. Therefore, they do exercise the compressed main memory.

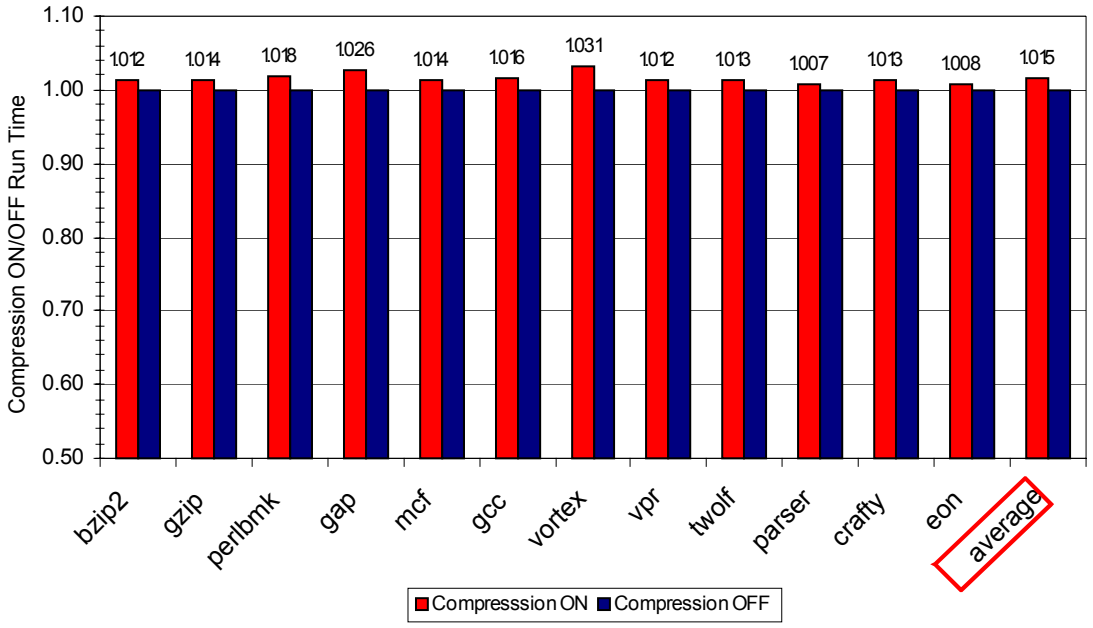

Figure 3: Execution Overhead on Compressed Memory Hardware

Results

Benchmarks were run and execution times were recorded twice with the compression on and off. The difference of the two execution times gives the penalty of compression. The MXT system has a boot option that permits compression to be turned off. In that case, the system operates as an ordinary system with an L3 cache and with non-compressed memory. Since compression/decompression circuitry is out of the way, the memory access latency is smaller in the compression-off case. Figure 3 shows the difference in execution times of compression on and off cases. As expected, in the compression-on mode the system runs slower; the last column shows that the average execution time is 1.5 percent longer for the compression-on case. This is a negligible quantity considering that the memory size is doubled. Throughput increases more than 1.5 percent in many systems when memory size is doubled.

Note that due to strict run rules and reporting requirements of the SPEC consortium, we cannot publish the actual execution times. In Fig.3 results are presented as the ratio of execution times of the compression-on mode over compression-off mode.

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

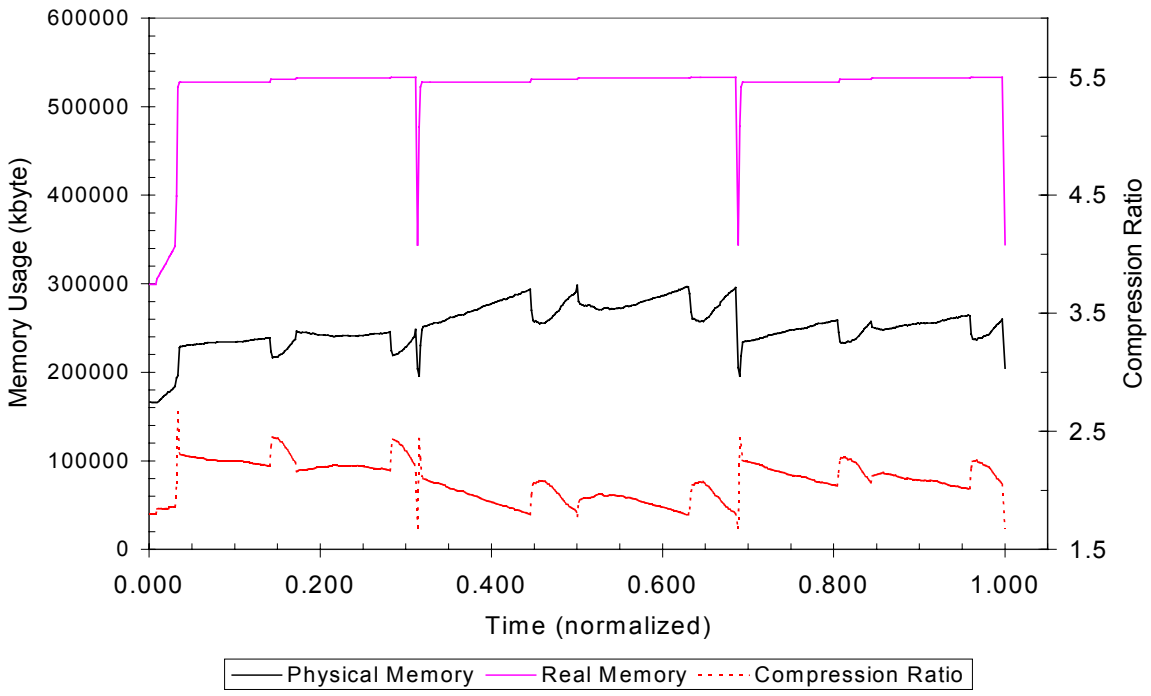

Figure 4: Real and Physical Memory Utilization for the BZIP2 benchmark as a function of time

SPEC2000 CPU benchmarks, the contents of several web sites, and a webserver 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.

Methods

For the SPEC2000 benchmarks, the real and physical memory utilizations were sampled every two seconds using an instrumentation register of the memory controller. The Sectors Used Register (SUR) reports to the operating system the amount of physical memory in use. 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 and the real memory utilization as reported by Linux OS 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 webserver benchmarks, we used additional methods and we will postpone their discussion until after the next section.

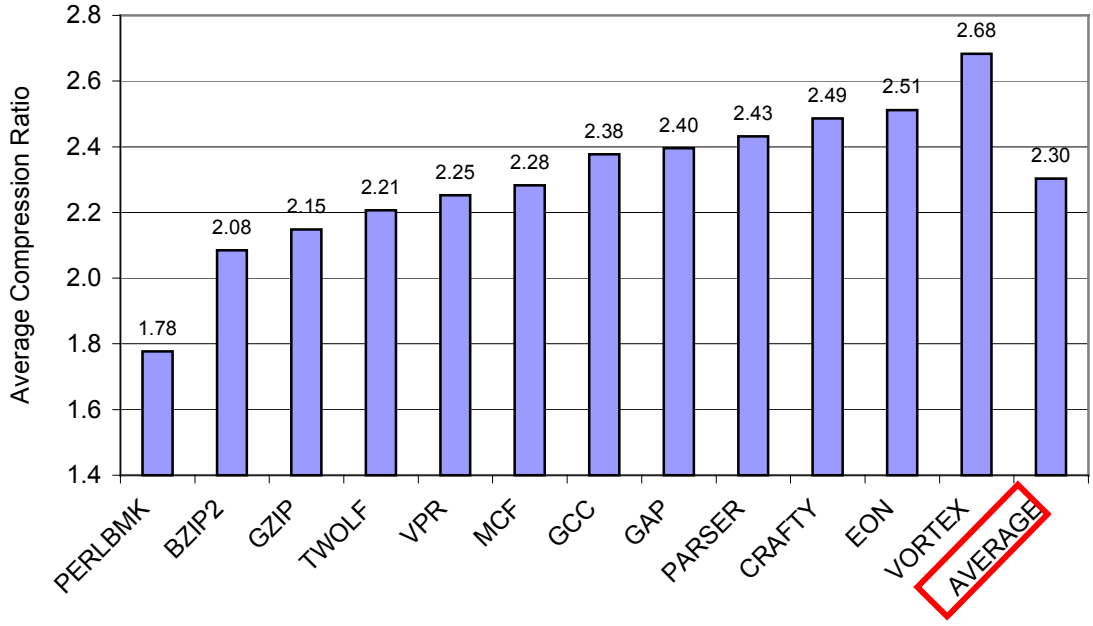

Figure 5: Average compression ratio of SPEC2000 benchmarks

Results

Figure 4 demonstrates the operation of the MXT system while executing the BZIP2 benchmark of the SPEC CPU2000 suite. The system memory utilization is initially at about 300 MB, before the benchmark starts. At time t=0.034, real memory utilization reaches about 520 MB, and remains nearly constant until the end, except for the two points at times t=0.314 and t=0.69 at which memory has been freed and allocated again. It can be observed from Fig.4 that while the real memory utilization has been nearly constant, the physical memory utilization varies between 170 and 300 MB as the application changes the contents of the memory. The ratio of real to physical memory utilization over time gives the *dynamic* compression ratio of this application.

Figure 5 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 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.

Methods for Web Content and Webserver 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 webserver, and the file system policies of the operating system. We analyzed the webserver benchmarks WebBench 3.0, SPECweb99, and the contents of well known sites www.yahoo.com, home.netscape.com, and www.ibm.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 webserver 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 randomly chosen characters. Random strings 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 was generally assumed that they would be incompressible on the MXT hardware. However, our measurements indicate otherwise. The primary reason for the better than expected 1.0 compression ratio is the fact that graphics files on webpages are often small (few hundreds of bytes each), yet they may have a larger memory footprint. Operating systems such as Linux and NT generally allocate memory for file objects in increments of a page size (4096 bytes) in order to organize their file cache. 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 in the following manner: the increase in the real memory utilization after the experiment has been started is divided by the increase in the physical memory utilization. Figure 6 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.

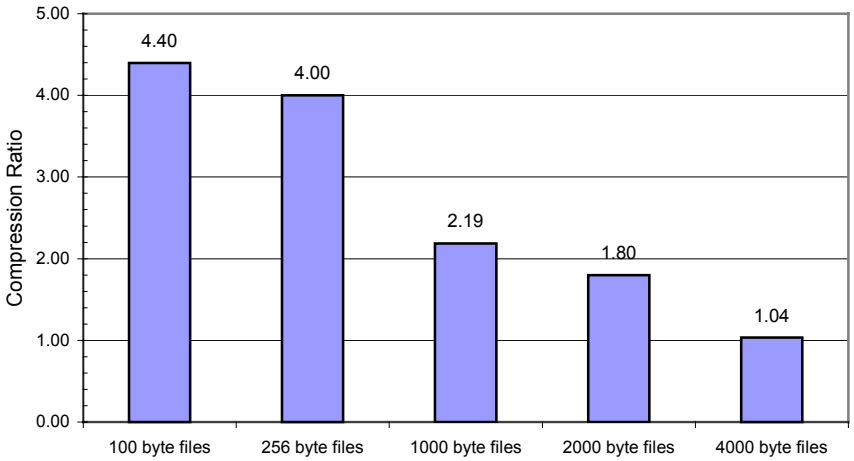

Figure 6: Compressibility of "incompressible" files in memory

Given our discussions on small files above, one would expect the compression ratio to be much higher than what was measured in Fig.6. 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. To summarize, small files are wasteful in real memory due to the file system overhead and therefore the memory footprints of small incompressible files can still have better than 1.0 compression ratio.

In order to determine the compressibility of important websites, 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 controller.

It should be mentioned that the approach we used here is an approximation. It is not possible to mirror an entire website as it would have taken very long time. Depending on the website we selected 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 webserver 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.

Results

We mirrored three well-known sites, www.yahoo.com, home.netscape.com, and, www.ibm.com on the local disk of the MXT system. Figure 7 shows the measured compression ratios for these web sites' content as well as the WebBench 3.0 content. Results show that websites compress well and near the real to physical memory ratio of 2 chosen for the MXT hardware (1GB/512MB.)

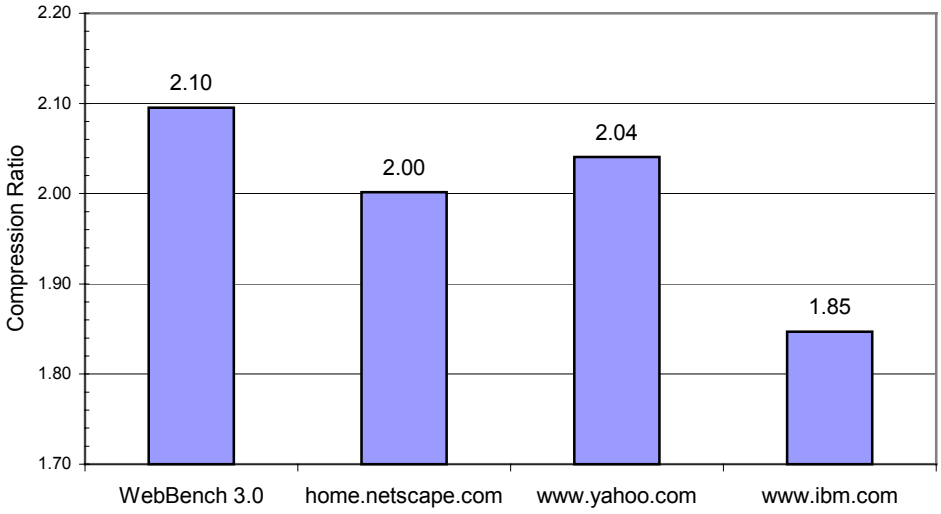

Figure 7: Compression ratios of well-known websites

6. 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 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 few popular web sites and determined that main memory contents of webservers carrying such content can be compressed by a factor of 1.85 to 2.10.

Acknowledgements:

Brett Tremaine and Mike Wazlowski described us the operation of the L3 Cache/Memory Compression Controller chip.

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," To appear at the *HOT Chips 12 Symposium*, 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.