IBM Memory Expansion Technology

By:

Jeromie LaVoie

Marist College
Computer Architecture 507L256
Instructor: David Meck

Note: Jeromie wrote this review paper as his homework. IBM is not responsible for it's contents.

Table Of Contents

Introduction

A new memory enhancement system has recently been designed by IBM researchers to essentially double the amount of internal memory capacity within a computer system. MXT is a hardware implementation that stores frequently accessed data and instructions close to the computer's CPU in order to increase access times and improve performance. Less frequently accessed data and instructions are compressed internally and stored in nearby memory locations instead of on a disk drive (Kawamoto). The compression is a direct result from new compression algorithms that provide a great increase over previously designed techniques. Memory contents are stored and accessed using new data structures that use little space and require no reorganization ("New IBM Chip").

The MXT chip appeared first in the popular Intel-based Netfinity line but will eventually be adapted for personal computers and e-business devices. Since memory typically accounts for 40 to 70 percent of the cost of a computer system, the new MXT technology will essentially save ISP's and other larger server-based companies thousands or even millions of dollars. Customers can cut costs by purchasing half the amount of memory to achieve the same performance, or they can receive twice the performance with the same amount of memory (Kawamoto, "New IBM Chip").

Hardware Overview

Extensive observational research has been performed and overwhelmingly demonstrates that the content of a computer's main memory in most systems is compressible to some degree. This includes the operating system, programs, and any related data. Of course,

not all data is compressible because it may already be compressed, reside in an encrypted form, or it may just be uncompressible.

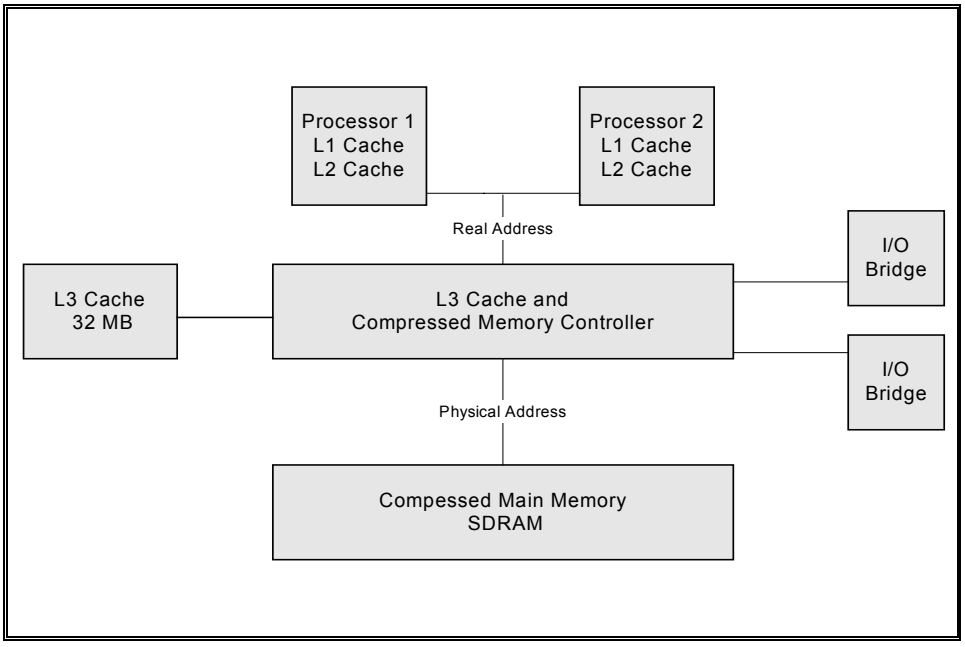
The idea of data compression is not new; researchers have actively been perfecting ways to utilize compression techniques within a computer system. These techniques have effectively been utilized in most of today's computer systems to save storage space, increase bandwidth on a network, or as a means to package data together. Although many forms of compression have been utilized, developing a hardware implementation of memory compression was not an easy task. The very idea may have seemed simple; compress the data and store it in physical memory. The issue was how to feed the processor quick enough with frequently accessed data and how to find the required data locations after it has been compressed (Abali, 1).

Typical computer systems share the same architectural features. They will have one or more CPUS connected to main memory (SDRAM) through a memory controller. The MXT architecture takes that one step further by using two different levels. One is for accessing the memory array and the other is actually a large L3 high-speed cache. This cache acts as a buffer to the main memory. As described by Tremaine, Smith, and Wazlowski, frequently accessed data can be stored in this location and required stored data can be uncompressed and made ready for execution by the system and help alleviate the dreaded wait time for data to arrive in its uncompressed form (1).

The total amount of main memory in a system can contain as much as 8 Gigabytes. After MXT is used, this size is increased up to 16 Gigabytes (doubled). The L3 cache is shared

and is 32 megabytes of double data rate SDRAM with a one-kilobyte line size (Abali, 2). The L3 Cache and Compressed Memory Controller connects the different components together and performs the main operations. The L3 cache appears as the main memory to upper memory and its operations are hidden from the rest of the hardware. Neither the processors nor the I/O can see the differences in computer activity. The controller compresses the data delivered from the 1 KB cache lines before writing them to the compressed main memory. Vice Versa, it also decompresses the data after reading it from the compressed main memory (Abali, 2). Therefore, the real addresses illustrated in Figure 1 are the addresses as perceived by the CPU. The physical addresses are the actual addresses of the compressed data stored in memory. The controller handles this translation process.

Figure 1. Hardware Structure Overview



Architecture

Shared Cache – Multiple possessors can access the shared L3 Cache system. This cache provides low latency access to data that is used often by either a CPU or the I/O constructs. As illustrated by Tremaine, Smith, and Wazlowski, this data is not compressed and is usually 32 bits (3). The cache is partitioned into lines that correspond to a 1KB storage unit. A directory is maintained to relate the real cached addresses with those that can be stored in the lines. The cache must be able to access the memory for both accessing and storing data.

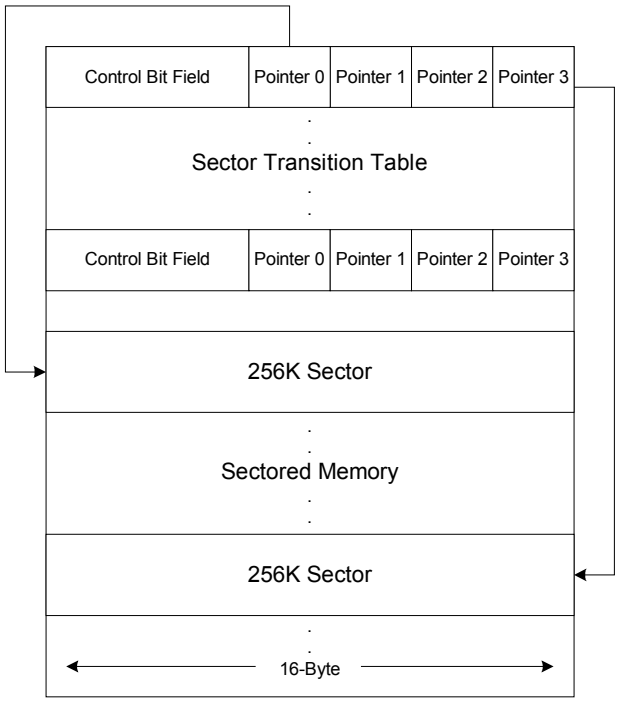
Controller – Any references to memory by either the CPU or the I/O constructs are directed to the MXT cache controller. It is here where the lookup table exists to determine whether or not the requested data is stored within the cache (cache hit) or not (cache miss). In the event of a cache miss, the least recently used cache line is written back to main memory write back buffer and a request is sent to the memory controller to retrieve the necessary data. To service this new request, the memory controller will first need to determine the address location of the data by utilizing a small lookup table. The data will then be read from memory. Two paths may be taken; data may be compressed and need the services of the decompression controller or may be streamed around it if the data is not in compressed form. Either way, the data will then be sent first to a line fill buffer, and then on to the cache (2). Cache write backs can be processed in parallel with a read. The architecture includes a set of individual 1KB write back buffers to enable both the storing and forwarding of data to the main memory, and another 1KB elastic buffer to fill with data when the cache access is unavailable. All of the data must be received

before any compression can commence (4). Data is read from the write back buffer and sent either to the compressor or to the memory location.

Main Memory – The main memory is made up of the standard SDRAM DIMMs used in today's computer market. The memory system stores and retrieves data in response to the cache line requests. The memory controller is set to support two types of configurations; DIMMs connected to the system without any built in logic, or the larger memory modules with memory buffering chips included. Both can be accessed at a bandwidth of 32B – 256B access granularity (4). For increased access times, uncompressed data is accessed first with the important 32B first and the remaining bytes wrapped around so that the first 32B is actually the center of the memory being accessed.

Main memory can be used in three different modes; disabled, enabled, or enabled only in a specified address range. Disabled means that the memory is mapped to the real address. Enabled means that the memory controller provides the real to physical address translation in order to utilize the variable length data in the compressed 1KB lines. After this translation is complete, the main memory lookup table is used to find the actual place for the data within memory. The physical memory is partitioned into either two or three regions depending on whether or not uncompressed data exists. The memory consists of two main data structures; the sector translation table (STT) and the sectored memory (4). The STT basically consists of an array of 16B pointers to a corresponding 1KB region in the sectored memory region. Each of these entries consists of attributes for the data as well as the particular address. If the stored data is compressed to less than 121 bytes, then the data will be contained within the STT itself. If it is longer than 120B, then the data

Figure 2. Physical Memory Partitions



will be placed in the sectored memory region in one to four 256B blocks. The sector pointer will show the start address of the block and is maintained in the STT entry as shown in Figure 2 (5). The sectored memory is allocated from a heap of available sector addresses stored in a linked-list format. A hardware cache maintains the first two nodes of the list, so that allocation and deallocation of free sectors can happen rapidly.

Compression/Decompression – The key to the success of MXT is with the compression of the data. The compression algorithm used is a parallel version of the Lempel-Zov algorithm known as LZ1. Four compression engines are used to access 256B of data each (a quarter of the 1KB block of uncompressed data, which is the same size as the L3 cache line) (6). For each clock cycle, one byte is read into each engine from the write back buffer and is shifted into a buffer called a dictionary (total of four bytes per CPU cycle). The longest match of consecutive data is called the working string. After the matching is complete, a copy of the raw character and its length is copied from the working string to a reference string (7). This process will continue until each engine processes their block of data. The net result is a possible smaller string to be forwarded into main memory.

Pages – Another benefit of using virtual memory is that the memory translation table can be altered to quickly relocate or clear data. Therefore, a program control mechanism was initiated to handle page operations at a high rate of 0.1 to 3.0 microseconds. The mechanism has a page register to contain the page address and command bits. It also has a page compliment register "for specifying the second page address for commands involving two separate pages" (6). The commands are interpreted and will allow the page controller to perform specific operations. Pages can also be grouped into classes and listed in the STT for faster processing.

Performance

In order to test the compression ratios and performance impacts, several benchmarks were used and analyzed from the SPEC CPU2000 benchmark set (Abali and Franke, 11). Table 1 illustrates what they are and the type of application it represents (part of the CINT2000 application set).

Table 1. CINT2000 Programs Tested.

GZIP	Compression utility.
VPR	Circuit placement and routing.
GCC	Compiler program.
MCF	Minimum cost network solver.
CRAFTY	A computerized chess application.
PARSER	A natural language processor.
EON	Ray tracer.
PERLBMK	A utility written in Perl.
GAP	A computational group theory application.
VORTEX	An object-oriented database.
BZIP	A data compression utility program.
TWOLF	A place and router simulation program.

Since the applications require an operating system to function, part of the memory utilized belonged to the operating system and had to be included in the measuring process. An instrumental interface of the memory controller was used to monitor both the real and physical memory usage during execution times. The hardware contained 512MB of physical memory leaving 1GB of real memory to test with. Compression ratios were sampled at 2.0-second intervals and properly recorded. The results were convincing: PERLMBK produced the lowest result of 1.78% while the largest was Vortex with a 2.68%. The average result of the total sample produced a whopping 2.30%. This is well above the boosted 2.0 ratio (doubled) for the MXT controller.

A second test was conducted to measure the speed impact of the running the compression utility. In order to make an accurate comparison, all programs were run two times; once with compression turned off and a second time with it turned on. A selection was made available in the BIOS that made turning the compression off rather simple. The L3 Cache Compressed Memory Controller chip now ran without the compression feature, but the L3 cache was still fully operational. The large 32 MB L3 cache itself helped improve performance over industry standard, as to be expected. VORTEX showed to be the worst case, with a 3.1% slower performance with compression turned on. The best case was PARSER, it was only 0.7% slower. The average difference with all applications combined was 1.5%. Considering the data was effectively compressed to more than half of the original amount, a mere 1.5% is considered minimal (Abali and Franke, 12).

Operating System Support

Now that the physical hardware is in place, let us take the operating system into consideration. As already discussed, the compressed memory controller chip performs the real to physical address translations. Therefore, the I/O devices, CPU, and the operating system are oblivious to its operations. The only exception is within the BIOS; the BIOS believes the memory within the system is actually double what it really is. The problem with this is when a large amount of uncompressible data is sent into memory. The memory will fill fast and cause an out of memory situation (Abali, 1).

Neither Linux nor Windows NT (4.0 or 2000) was developed to handle the new out of physical memory operations. Therefore, special compressed memory software was developed by the IBM research team to handle the translations. Because Linux is an open source operating system, changes could be made directly to the kernel. The Microsoft Windows operating systems are not open source, so a device driver was created to work with the hardware. As described by Abali et al., the compressed memory software must perform the following tasks:

1. Accept a warning interrupt sent by the Compressed Memory Controller indicating the main memory capacity is getting full.
2. Stop any more memory allocations from taking place. Pages will be stopped and swapped with existing memory data. This essentially "activates the swap daemon to shrink file caches and to swap out some user process pages, hence forcing some memory to be freed (Abali et al., 3)."

3. Replace freed pages with all zeros to help free available memory. This is possible because zero-filled cache lines can occupy only 1/64 the actual size in the main memory locations.
4. Start a thread-spinning routine to stall programs. This will actually keep memory-intensive operations from accessing the memory so quickly.

The MXT memory saving routines are not new to computer operations. These routines are basically the same as current computer functionality. You have probably noticed your CPU going crazy swapping data when your system memory is getting full at some point in time.

According to the MXT project site, several programs and patches have been developed to support the Linux operating system; RedHat 7.2 now supports the MXT hardware and was shipped last fall, RedHat 7.1 will upgrade the Linux kernel to support the hardware, and several patches have been developed to upgrade the Linux 2.4 kernel to the proper functioning level. Please see http://oss.software.ibm.com/mxt for more information.

Competitive Impact

MXT is such a major improvement that the impact in the server market is extensive. If you placed two identical systems side by side with only the addition of the MXT controller, you would have a profound difference in performance. No known technologies exist that would produce as great of a variation as this has. It has been mathematically estimated that MXT improves both price and performance by 25% to 70% (Smith et al., 1).

Prices were analyzed by taking two almost identical systems and running price comparisons. The only difference in these base systems would be the MXT technology installed on only one system. Believe it or not, the MXT system was priced at thousands less than the counter-part because half of the memory would not be needed to produce the same results (Smith et al., 3). As an example, computer A may cost $1,000 and run with 512 MB of RAM. Computer B might cost only $750.00 because it requires only half the amount of physical memory, yielding a 25% savings amount. For larger ISP companies, savings could reach as high as one million dollars!

Conclusion

MXT has already found a home within IBM's line of data transaction and Wed-based servers, such as the IBM xSeries 330 1U Rack Mount Servers. In the future, this technology may find its way into desktop and laptop PCs, handheld devices, and cell phones ("New IBM Chip").

You can fully expect this technology to be refined, enhanced, and eventually make a permanent home within the computing industry. The market has effectively demonstrated the desire for faster and more efficient technology. This is clearly just a major advancement in the right direction. Memory enhancements have always risen since the birth of the PC. Technology has always given way to more memory with quicker access rates. Let us take the new DDR memory for example; the memory speeds are now

running at 266 and 300MHz with a minimum of 256 MB! The beauty of MXT is that it can be incorporated with those new enhancements! Other vendors will be forced to try to model a better technology and compete in the marketplace. Maybe a faster algorithm can be used or maybe more compression engines can be applied? Either way, it is undisputed how great of an achievement it was for IBM to double the memory used in conventional systems while maintaining system performance.

Works Cited

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

Abali, Bulent, et al. "Performance of Hardware Compressed Main Memory." IBM T.J. Watson Research Center. 5 Nov. 2000.

Kawamoto, Wayne. "IBM's Memory eXpansion Technology." ISP-Planet 10 July 2000. http://www.isp-planet.com/equipment/ibmx.html (21 Apr. 2002).

"MXT Support Patch for Linux." Open Source: MXT Project Web Site. http://oss.software.ibm.com/mxt (21 Apr. 2002).

Smith, Basil T., et al. "Memory Expansion Technology (MXT): Competitive Impact." IBM T.J. Watson Research Center. 12 Feb. 2001.

Tremaine, Brett R., Basil T. Smith, and Mike Wazlowski, "IBM eXpansion Technology (MXT)." IBM Journal of Research and Development. Vol. 2, 2 Nov. 2001.

"New IBM Chip Doubles Computer Memory." IBM Think Research. 24 Oct. 2000. http://www.research.ibm.com/thinkresearch/pages/2000/200008_mxt.shtml (31 Apr. 2002).