Pinnacle: IBM MXT in a Memory Controller Chip

PINNACLE LEVERAGES STATE-OF-THE-ART TECHNOLOGIES TO ESTABLISH A LOW-COST, HIGH-PERFORMANCE SINGLE-CHIP MEMORY CONTROLLER. THE CHIP USES IBM'S MEMORY EXPANSION TECHNOLOGY SYSTEM ARCHITECTURE, WHICH MORE THAN DOUBLES THE INSTALLED MAIN MEMORY'S EFFECTIVE SIZE WITHOUT ADDING SIGNIFICANT COST OR DEGRADING PERFORMANCE.

R. Brett Tremaine
T. Basil Smith
Mike Wazlowski
David Har
Kwok-Ken Mak
IBM T.J. Watson Research Center

Sujith Arramreddy
ServerWorks

Memory costs dominate both large-memory servers and expansive-computation server environments, such as those operating in today's data centers and compute farms. These costs are both financial and physical (for example, the volume, power, and performance associated with memory system implementation). They often add up to a cost constraint that the IT professional must trade off against computation goals.

The computer industry uses data compression techniques widely to increase the cost efficiency of storage and communication media. Despite some experimental work,[1,2] however, the industry has not exploited system main-memory compression to its potential. IBM's Memory Expansion Technology (MXT)[3] confronts the memory cost issue with a new memory system architecture that more than doubles the installed main memory's effective capacity without significant added cost.

Engineers from IBM and ServerWorks have jointly developed Pinnacle, a low-cost, single-chip memory controller (or north bridge) using MXT. This Intel Pentium III and Xeon bus-compatible chip is the first commercially available memory controller that employs real-time main-memory compression at a performance level competitive with the market's best products. The Pinnacle chip is ServerWorks' flagship product for the commercial server market.

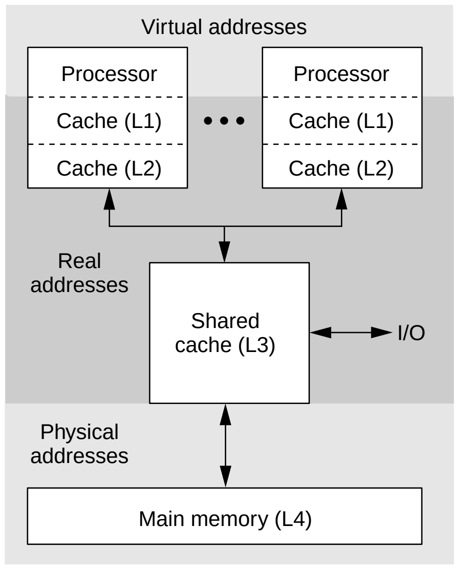


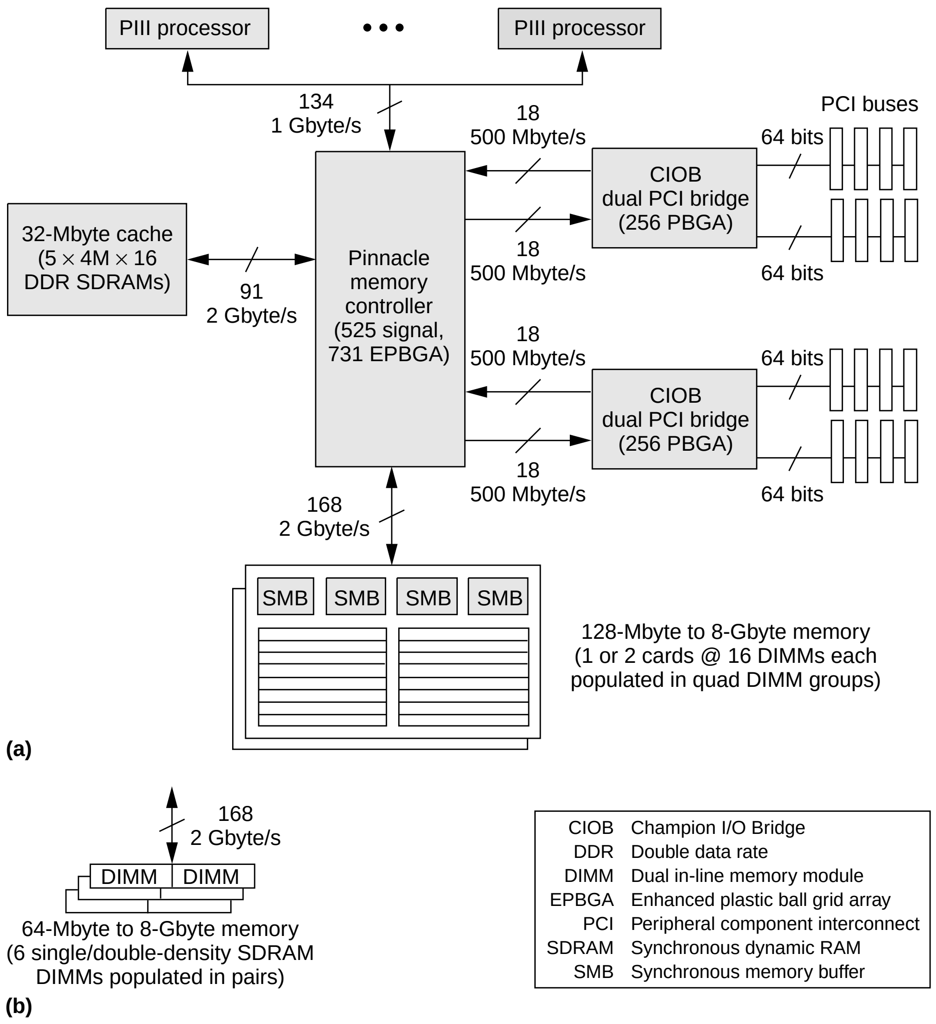
Figure 1. MXT system memory hierarchy.

0272-1732/01/$10.00 © 2001 IEEE

MXT architecture

In the typical architecture of conventional commodity computer systems, a memory controller chip set connects a collection of processors to a common synchronous dynamic RAM (SDRAM)-based main memory. In contrast, MXT uses the two-level main-memory architecture[4] shown in Figure 1, consisting of a large shared cache coupled with a typical main-memory array. The high-speed cache, which contains frequently referenced processor data, architecturally insulates system performance from main-memory access latency. Thus, MXT opens opportunities to trade off increased memory-access latency for greater functionality. For example, system designers can incorporate remote, distributed, very large, or highly reliable features without adversely affecting system performance.

IBM engineers combined the shared-cache architecture with high-density 0.25-micron and smaller application-specific integrated circuit (ASIC) technology in the compressed main-memory architecture. Special logic-intensive hardware engines simultaneously compress and decompress data as it moves between the shared cache and main memory. The compressor encodes 1-Kbyte data blocks into as compact a result as the algorithm permits. A sophisticated memory-management architecture permits storing the variable-size compressed data units in main memory, while mitigating fragmentation effects and avoiding garbage collection schemes. The new architecture halves the main-memory cost, without significant degradation in system performance.



CIOB	Champion I/O Bridge
DDR	Double data rate
DIMM	Dual in-line memory module
EPBGA	Enhanced plastic ball grid array
PCI	Peripheral component interconnect
SDRAM	Synchronous dynamic RAM
SMB	Synchronous memory buffer

Figure 2. Pinnacle chip (second CIOB optional) with DIMM card configuration (a); alternate directly attached DIMM configuration (b).

Pinnacle chip

The Pinnacle host bridge controller connects dual Pentium III or quad Xeon processors; a 32-Mbyte, double-data-rate SDRAM shared-cache memory; a main memory; and up to two independent, remote peripheral-component interconnect bridge chips. We optimized the low-cost single-chip controller for a wide range of high-performance server system applications. The full-featured chip (see the "Pinnacle features" box on page 67) can be used in either of the two primary memory configurations shown in Figure 2.

Figure 3 (next page) shows Pinnacle's internal structure. All primary internal dataflow is 128 bits wide, pipelined, and full duplex. The cache and intermodule bus (IMB) interface operate at double the chip clock rate. Any processor or I/O memory references are directed to the cache controller, which uses cache directory lookup

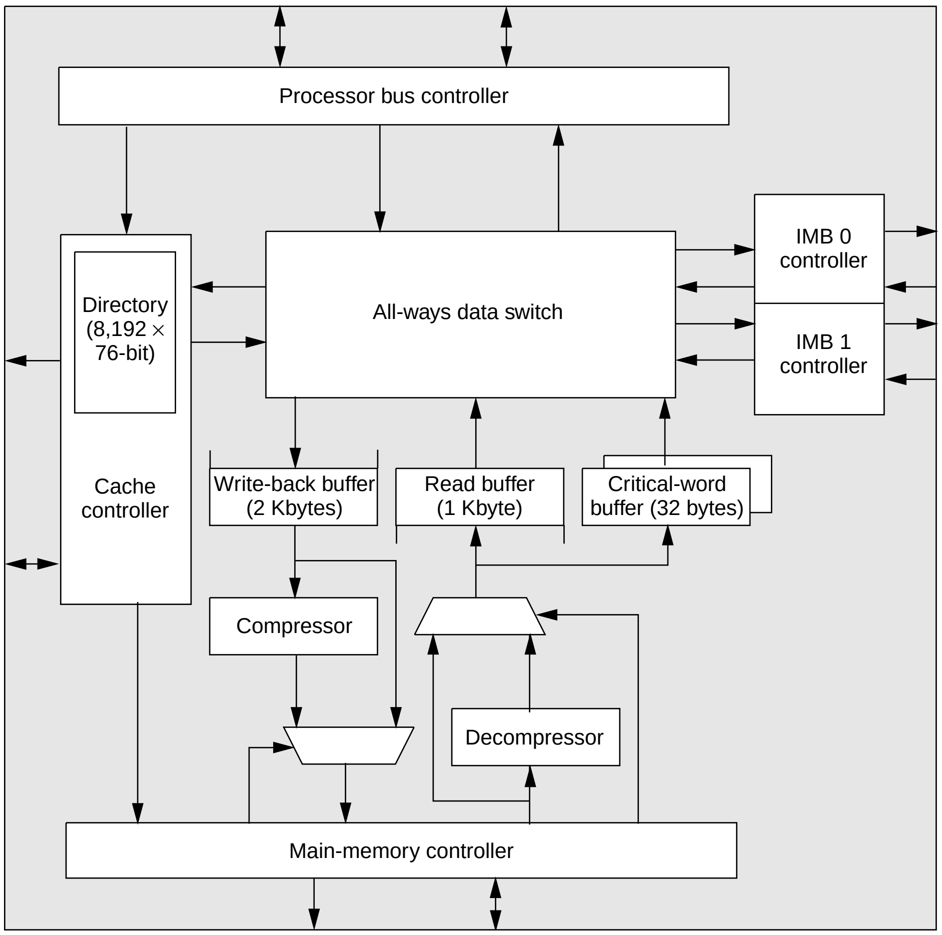


Figure 3. Pinnacle chip block diagram.

to determine whether the cache contains the address. The cache controller services cached references directly from the cache; it defers cache read misses, and it selects the least recently used (LRU) cache line for replacement with the new cache line containing the requested address. The cache controller issues a request for the new cache line from the main-memory controller, while writing back the old cache line to the write-back buffer if the old cache line contains modified data.

To service the new cache line fetch, the memory controller first reads a small address-translation-table entry from memory to locate the requested data. Then it starts reading the requested data. Data streams either around the decompressor when uncompressed or through the decompressor when compressed. In either case, the data then streams through the elastic buffer to the cache. The memory controller provides seven-cycle advance notification of when the requested 32-byte data will be in the critical-word buffer. This lets the processor bus controller arbitrate for a deferred read reply and deliver data without delay.

The chip processes cache write-back activity in parallel with read activity. Once an entire cache line is queued in the write-back buffer, compression begins and runs uninterrupted until it is complete, 256 cycles later. Then, if a spatial advantage exists, the memory controller stores the compressed data; otherwise, it stores the write-back data directly from the write-back buffer. In either case, to allocate the appropriate storage, the memory controller must first read and update the translation-table entry for the write-back address, before writing it back to memory. The memory controller then writes the data itself to memory, within the allocated sectors.

Shared-cache subsystem

The shared cache provides low-latency processor and I/O subsystem access to frequently accessed uncompressed data. The data-, code-, and I/O-unified cache content is always uncompressed and accessed at 32-byte granularity. Write accesses smaller than 32 bytes require the cache controller to perform a read-modify-write operation for the requesting agent.

The cache consists of four banks of 8K (8,192) × 1,024-byte lines, or 8K sets of four lines each. Cache lines within a set are replaced according to the LRU policy. We chose the cache line size to minimize the on-chip cache directory's size and to match the compression algorithm block size.

The shared-cache directory contains a unique 17-bit entry, organized as shown in Figure 4, for each of the 32K (32,768) cache lines. The tag address bits permit caching of the low-order 16 Gbytes of real-address space. We implemented the directory on chip as an 8K× 76-bit dual-port (one read and one write) SRAM. The cache controller accesses the four entries associated with a set concurrently with associated LRU state and parity bits.

The relatively long (1-Kbyte) cache line merits special design consideration. For example, the processor reference bits mitigate extraneous cache coherency snoop traffic on the processor bus. These 4 bits indicate when any processor has referenced one or more quarter cache-line (256-byte) segments. When a cache line is evicted, only referenced cache-line segments, not the entire line, must be invalidated on the processor bus.

Shuttling the wide lines in and out of the cache during cache-line replacement requires at least 64 system clock cycles, or 32 accesses for each write-back and line-fill operation. To alleviate processor access stalls during the lengthy replacement, the cache controller permits two logical cache lines to coexist within one physical cache line. This mechanism permits the cache line to be written back, reloaded, and referenced simultaneously.

During replacement, the cache controller maintains a state vector to indicate old, new, or invalid state for each of the 32 subcache lines within the physical line. As 32-byte subcache lines are invalidated or moved from the cache to the write-back buffer, they are marked *invalid*, indicating that the associated new subcache line can be written into the cache. Each time the cache controller loads a new subcache line, it updates the associated state as *new*, indicating that processors and I/O can access the new subcache-line address. When the associated subcache lines are marked *old*, processors and I/O can access the old subcache-line addresses. The cache controller always optimally fetches and fills cache lines; that is, the subcache line write-back follows the same subcache line order to maximize the amount of valid cache line available at all times.

The cache controller can support up to two concurrent cache-line replacements. The two independent 1-Kbyte write-back buffers facilitate a store-and-forward pipeline to the main memory, and the 1-Kbyte elastic buffer queues line-fill data when the cache is unavailable for access. A write-back buffer must contain the entire cache line before the main-memory compressor can begin its operation. Conversely, the line-fill data stream goes directly to the cache as soon as the buffer contains a minimum 32-byte data granule. The two independent 32-byte critical-word buffers capture the data associated with cache misses for direct processor bus access.

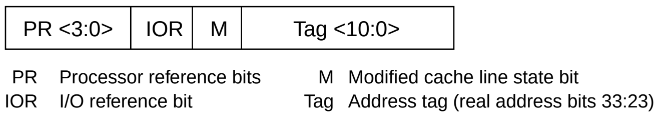

PR <3:0>
IOR
M
Tag <10:0>
PR Processor reference bits
IOR I/O reference bit
M Modified cache line state bit
Tag Address tag (real address bits 33:23)

Figure 4. Cache line directory entry (17 bits).

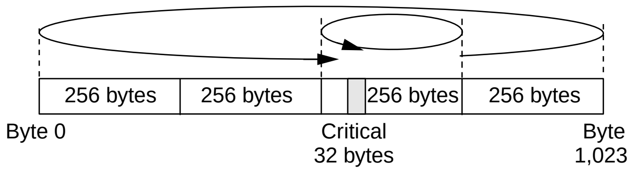


Figure 5. Critical-word fetch order.

Main-memory subsystem

The main-memory subsystem stores and retrieves 1-Kbyte cache lines in response to shared-cache write-back (write) and line-fill (read) requests. It stores data in a 64-Mbyte to 16-Gbyte array of industry-standard PC100 and PC133 SDRAM dual in-line memory modules (DIMMs). The memory controller supports either of two DIMM configurations for optimal large- and small-server product implementations. The large memory configuration (Figure 2a) supports one or two cards with synchronous memory buffer (SMB) chips and 16 DIMMs each, populated in quad-DIMM groups. The directly attached configuration (Figure 2b) supports two, four, or six single- or double-density DIMMs connected to the Pinnacle chip without any glue logic.

In either configuration, the memory controller accesses the array via a 144-bit (16 bytes + error-correcting code) data interface with 32-byte to 256-byte access granularity. For minimal latency, uncompressed data references are always retrieved with the critical 32 bytes first and the 256-byte address wrapped as shown in Figure 5.

Users can configure the main-memory subsystem to operate with compression disabled,

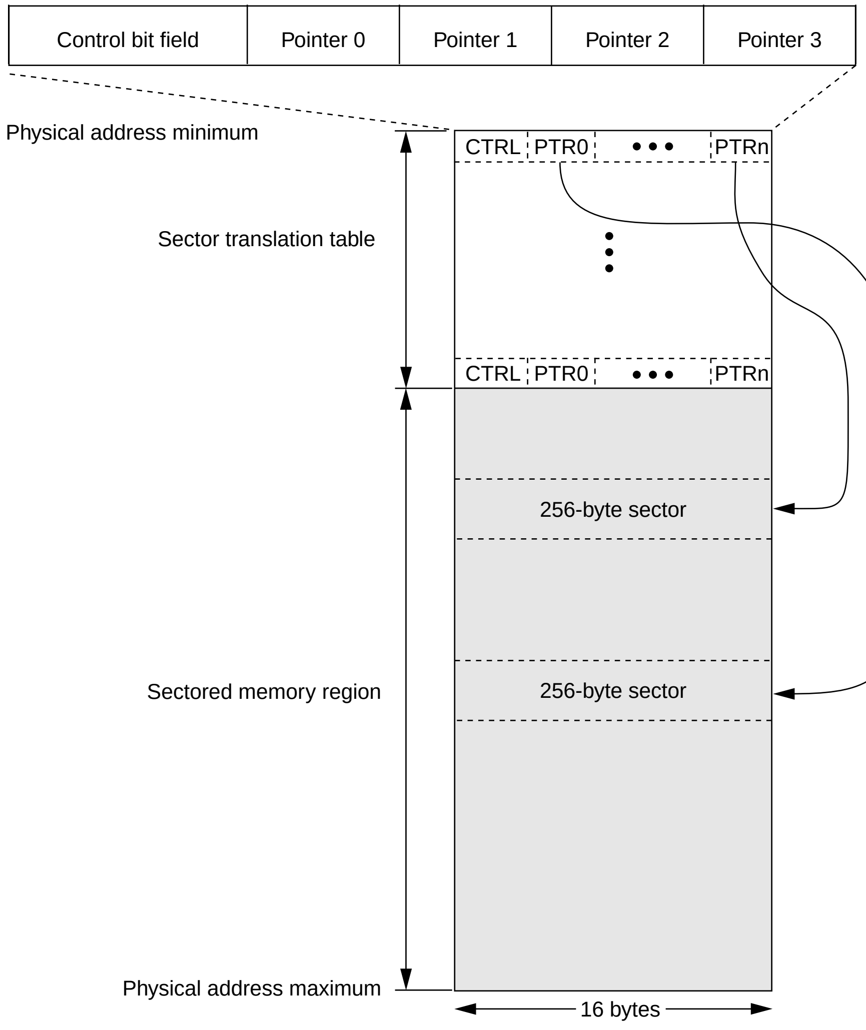


Figure 6. Memory organization.

enabled for specific address ranges, or completely enabled. In disabled-compression mode, the physical-memory-address space maps directly to the real-address space as in conventional memory systems. Otherwise, the memory controller provides real-to-physical address translation to dynamically allocate storage for the variable-size data associated with compressed 1-Kbyte lines. The memory controller carries out the additional level of address translation completely in hardware, using a translation table apportioned from the main memory.

The physical memory consists of two regions, or, optionally, three if uncompressed memory is configured. As Figure 6 shows, the memory contains two primary data structures: the sector translation table (STT) and the sectored memory. The STT consists of an array of 16-byte entries, each entry directly mapped to a corresponding 1-Kbyte real address. Therefore, the number of STT entries is directly proportional (1/64) to the size of the real-address space declared for a given system. We define the real-address space to the operating environment through a hardware register. The basic I/O system (BIOS) firmware initializes the register with a value based on the quantity and type of DIMMs installed in a system. If compression is enabled, the BIOS doubles this value to indicate a real-address space twice as large as that populated with DIMMs.

Each STT entry describes the attributes of the data stored in the physical memory and associated with the corresponding 1-Kbyte real address. Data can exist in one of three conditions: compressed to ≤ 120 bits, compressed to > 120 bits, or uncompressed.

If a 1-Kbyte data block is compressible to ≤ 120 bits, the data is stored directly in the STT entry, together with the control field, yielding a maximum compressibility of 64:1. Otherwise, the data is stored outside the entry in one to four 256-byte sectors, with the sector pointers contained in the STT entry. If the data block is uncompressed, it uses four sectors, and the STT entry control field indicates the uncompressed attribute. If unused sector-memory fragments exist in a 4-Kbyte real page, any new storage activity in the same page can share a partially used sector in 32-byte increments. A maximum of two 1-Kbyte blocks in a page can share a sector. This simple two-way sharing scheme typically improves compression efficiency by 15 percent, nearly all the potential gain attainable from combining fragments by any degree of sharing.[5]

The sectored memory consists of a “sea” of

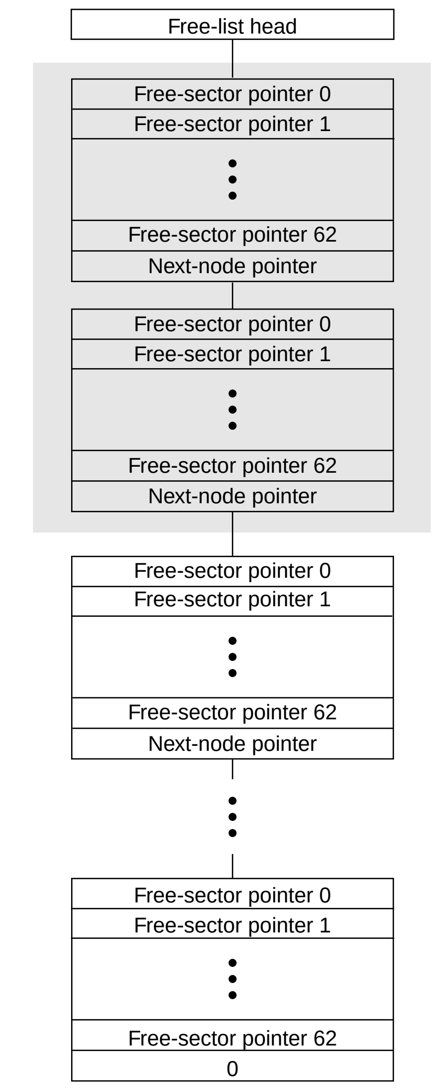


Figure 7. Free-sector list.

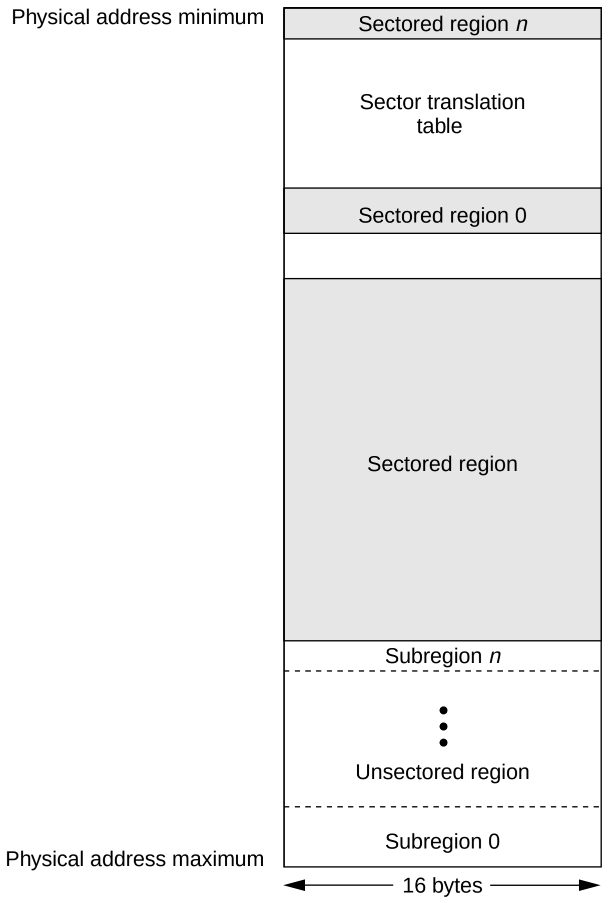


Figure 8. Unsectored-memory organization.

256-byte storage sectors allocated from a "heap" of unused or free sectors available in the sectored memory region. The heap is organized as a linked list of unused sector addresses, with the list head maintained in a hardware register. The free-sector list itself is stored in the free sectors, so sector use oscillates between holding the list and holding data. As Figure 7 shows, each node of the free-sector list contains pointers to 63 free 256-byte sectors and one pointer to the next 256-byte node in the list. Since the node is itself a free or unused 256-byte sector, the free-sector list effectively requires no additional storage.

A small hardware cache contains the leading two nodes (shaded in Figure 7) of the free-sector list, for rapid access during allocation and deallocation of sectors associated with data storage requests.

Unsectored memory regions are areas in a compressed memory's real-address space in which data is never compressed. The Pinnacle chip supports up to four such regions, each configurable as a 32-Kbyte to 256-Mbyte range and aligned on a 32-Kbyte address. The memory controller apportions and direct-maps these regions from the real-address space to the top of the sectored memory, as shown in Figure 8. The access latency to these regions is minimal because data is directly addressable without the intermediate step of referencing an STT entry. The memory controller fetches the data with the requested 32 bytes first, as is always the case for uncompressed data.

STT regions that contain entries for addresses in unsectored regions are never referenced. To prevent wasting them, the Pin-

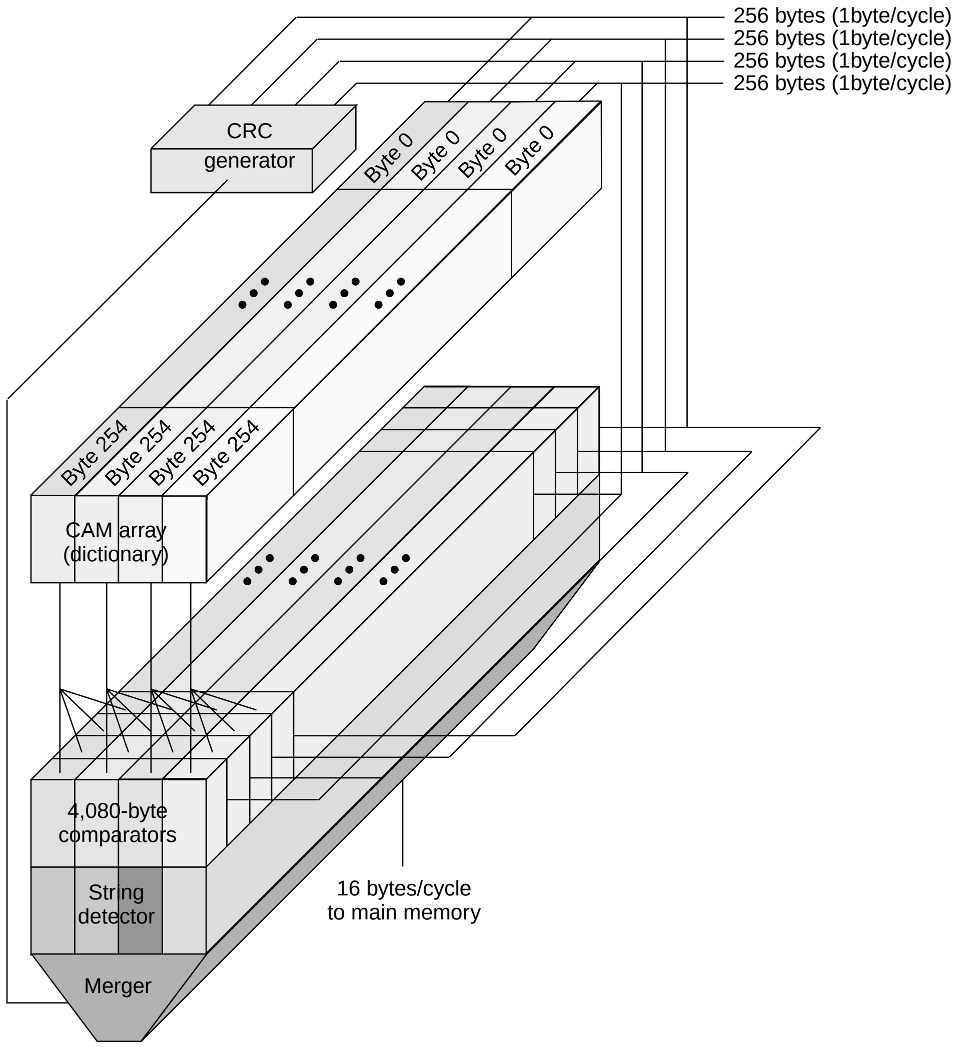


Figure 9. Compressor block diagram.

nacle makes these holes in the STT available as additional sectored storage by incorporating them in the free-sector list.

Page operations

A beneficial side effect of virtualizing the main memory through a translation table is that a simple alteration of a table entry can logically relocate or clear data associated with the entry. We capitalized on this by implementing a user-programmed control mechanism to enable real-memory page (4 Kbytes) manipulation, at speeds from 0.1 to 3.0 microseconds, depending on the amount of processor-bus coherency traffic required. We also provided Pinnacle with a programmed page-tagging mechanism, using a class code field in an STT entry reserved for this purpose. Special hardware counters count the allocated sectors by class. This scheme enables software to establish metrics supporting memory-use optimization algorithms. Pinnacle supports page-swap, clear, invalidate, flush-invalidate, and set-class-code operations.

Compression and decompression

The compression and decompression mechanism is the cornerstone of MXT. Compression, as applied in the main-memory dataflow application, requires low latency and high bandwidth in the read path, and it must be nonlossy. Although many compression algorithms exist, none met our architectural criteria. We chose to leverage high-density

(0.25-micron) CMOS ASIC technology by implementing a gate-intensive, parallelized derivative[6] of the popular Ziv-Lempel (LZ77 version) adaptive dictionary approach. This scheme partitions the unencoded data block into *n* equal parts, each operated on by an independent compression engine, with shared dictionaries. Experiments have shown that parallel compressors with cooperatively constructed dictionaries have a compression efficiency essentially equivalent to that of the sequential LZ77 method.

The Pinnacle embodiment, diagrammed in Figure 9, contains four compression engines, each operating on 256 bytes (one quarter of the 1-Kbyte uncompressed data block), at 1 byte per cycle, yielding a 4-byte-per-cycle aggregate compression rate. Each engine contains a dictionary (a history buffer) consisting of a 255-byte content-addressable memory (CAM) that functions as a shift register.

Attached to each dictionary are four 255-byte comparators for locating the incoming reference byte in the entire dictionary structure. Each clock cycle, one byte from each 256-byte source data block (read from the shared-cache write-back buffer) is simultaneously shifted into a respective dictionary and compared with the accumulated (valid) dictionary bytes. The longest match of two or more bytes constitutes a working string; the copy in the dictionary is the reference string. If a single-byte match or no match is found, as may be the case for random data, the reference byte is a raw character.

Compression occurs when location and length encoding to the dictionary reference strings replaces working strings in the compare data stream. However, the encoding scheme, shown in Table 1, may cause a 256-byte uncompressed data stream to expand to 288 bytes for a given engine. Therefore, we use special logic to detect that the accumulated aggregate compressed output exceeds 1 Kbyte (or a programmed threshold), causing compression to abort and the uncompressed data block to be stored in memory.

Once one of the 255 detectors detects strings in any one of the four dictionaries, it ignores future potential strings until it detects the end of the current string. The detector calculates the length and position of the longest working string or, in the case of multiple strings of the same length, the string starting nearest the beginning of the dictionary. The length field encoding the number of bytes in the working string ranges from 2 to 12 bits, using a Huffman coding scheme. The position field encoding the starting address of the reference string ranges from 2 to 10 bits. Merge logic packs the variable-length bit stream into a word-addressable buffer.

Table 1. Compressed data encoding.

Compressed data type	Encoded vector
Raw character	0, data byte
String	1, primary length, position, secondary length

The much simpler decompressor consists of four engines that cooperatively decode the encoded compressed data block. Each engine can produce 2 bytes per cycle, yielding an aggregate 8 bytes per cycle when single clocked or 16 bytes per cycle when double clocked as in Pinnacle.

Reliability, availability, and serviceability

The importance customers place on reliability, availability, and serviceability compels manufacturers of server-class computers to attain these characteristics at the highest cost-effective level. Main-memory compression adds a new facet to this endeavor,[7] with the primary goal of detecting data corruption in the system. To that end, the Pinnacle chip includes the following features, with appropriate logging and programmable interrupt control:

- shared-cache error-correcting code (ECC),
- shared-cache-directory parity checking and programmable hardware-assisted test,
- main-memory ECC with programmable scrub,
- I/O channel dead-man timer time-out recovery,
- hardware configuration/control register write protection with service processor override via inter-integrated circuit (I^2C) bus,
- processor, cache, and I/O interface protocol checking,
- STT-entry parity checking,
- free-sector-list parity checking,

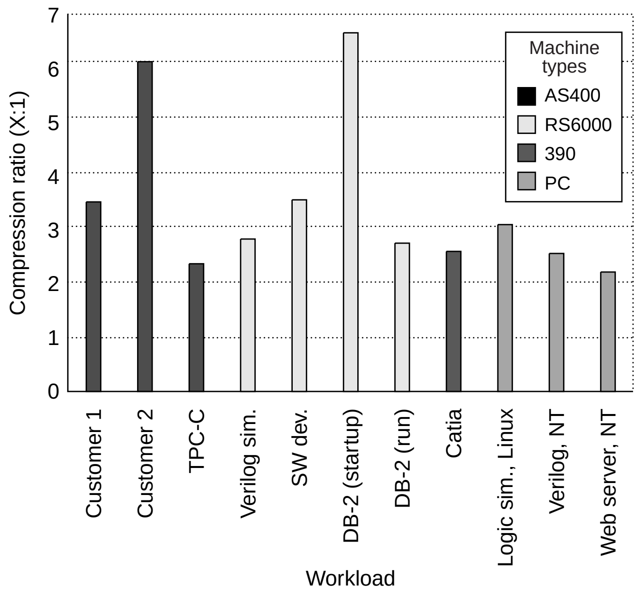


Figure 10. Memory compressibility.

- sector-out-of-range checking,
- sectored-memory overrun detection,
- sectors-used threshold detection (two),
- compressor/decompressor validity checking, and
- compressed-memory cyclic-redundancy-code (CRC) protection.

Since the compression and decompression functions effectively encode and decode system data, any malfunction during the processes can produce seemingly correct, yet corrupted, output. Further, the hardware function implementation requires a prodigious quantity (about 1 million) of logic gates. Although the compression and decompression hardware includes special fault detection mechanisms, they cannot provide complete fault coverage. Consequently, data corruption induced by a logic upset is likely to survive undetected. Therefore, we needed an improved data-integrity-protection method.

We use a standard 32-bit CRC computation over the 1-Kbyte uncompressed data block as it streams into the compressor. When compression is complete, and the data is to be stored in the compressed format, the check code is appended to the compressed data block, and the associated block size increases by 4 bytes. (Data is stored in the compressed format if a spatial advantage exists over storing it in the uncompressed format.) Information stored in the uncompressed format bypasses the compressor and decompressor functions and hence is not covered by the CRC protection.

Servicing a compressed memory-read request results in decompression of the compressed block and concurrent recomputation of the CRC over the uncompressed data stream from the decompressor. Upon completion of the decompression, the hardware compares the appended CRC with the recomputed CRC. If the two are not equal, an uncorrectable-error signal alerts the operating system to the event.

Operating system software

All commercial computer operating system software environments manage the hardware memory as a shared resource for multiple processes. If the memory resource becomes limited (that is, processes request more memory than is physically available), the operating system can take steps for continued system operation. Typically, it moves underused memory pages to disk and then reallocates the memory to the requesting processes. In this scheme, the main memory acts like a cache backed by a large disk-based storage. The scheme works well because the operating system knows the absolute amount of memory. The algorithm applies to MXT-based systems as well.

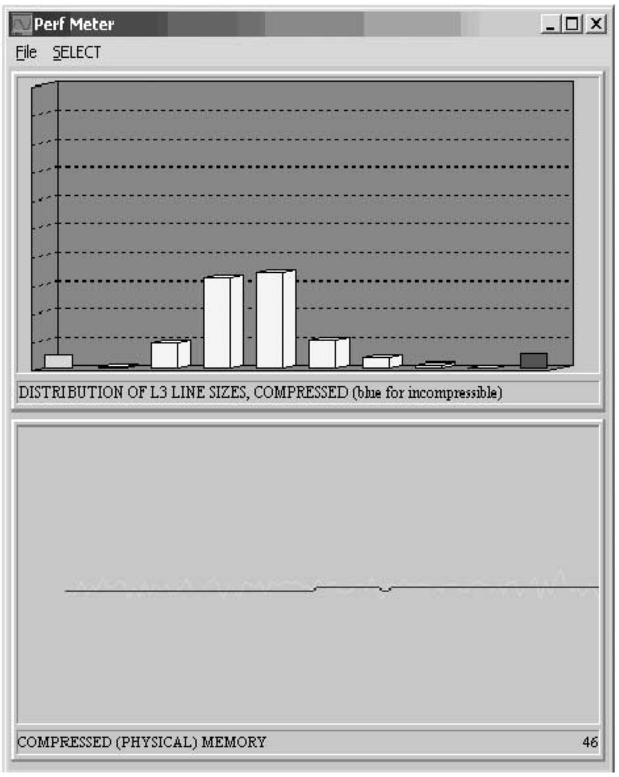
Although current shrink-wrapped operating system software can run on an MXT machine, it cannot yet distinguish an MXT-based machine from a conventional memory-hardware environment. This poses a problem because the amount of memory known to the operating system is twice what actually exists in an MXT machine. Further, the operating system has no notion of compression ratio. Therefore, it cannot detect conditions in which the physical memory might be overused (that is, too few unused or free sectors remain in the sectored memory). So it doesn't invoke the page management software to handle the situation, which may lead to system failure. This condition can occur when the operating system has fully allocated the available memory and the overall compression ratio has fallen below 2:1.

Fortunately, minor changes in its kernel virtual-memory manager are sufficient to make the operating system "MXT aware,"[8] and these changes are under way in most commercial operating systems. MXT awareness can also be accomplished outside the kernel—for example, by a device driver and service—albeit less efficiently. We currently have Pinnacle-based machines in beta deployment running Linux and Microsoft Windows 2000 and NT 4.0.

Performance

MXT compression performance ranges between 1:0.98 (1:1) and 64:1, including translation table memory overhead. Figure 10 shows a representative sampling of the many memory-content-compressibility measurements we took from various machines. We can take measurements either directly on an MXT machine, indirectly via a monitor program running on a non-MXT machine, or by postanalysis of memory dumps. Compressibility drops below 2:1 only in the rare case that most of the system memory contains random or precompressed data.

We observed that a given machine's compression ratio remains relatively constant throughout the operation of an application suite. For example, our monitoring of three IBM Internet online-ordering Web servers for 16 hours found an average compression ratio of 2.15:1 ± 2 percent (from 1 August 2000 at 6:00 p.m. EST through 2 August 2000 at 10:00 a.m. EST; http://www.pc.ibm.com/ibm_us/). Further, Figure 11 shows that the distribution of compressibility over the measurement period is normal. Each bar of the histogram represents the degree of compressibility; the rightmost bar is incompressible (1:1), and the leftmost bar is maximally compressed (64:1). The line in the lower half of the graph represents the degree of change in compressibility over the measurement period.



DISTRIBUTION OF L3 LINE SIZES, COMPRESSED (blue for incompressible)

COMPRESSED (PHYSICAL) MEMORY 46

Figure 11. Web server compression distribution.

We can evaluate MXT system performance from two main perspectives: intrinsic performance measured independently of memory consumption, and as a function of cost versus performance in memory-starved applications. MXT systems stand out in the performance benefits that additional memory provides for memory-intensive applications—customers typically experience system throughput improvements of from 50 to 100 percent. For example, one customer operating a compute farm with several thousand dual-processor servers, each containing 1 Gbyte of memory, ran one job per unit time on each machine. When the customer substituted an equivalent (dual-processor and 1-Gbyte memory) MXT machine, two jobs could run concurrently over the same period because MXT expansion effectively doubled the 1 Gbyte.

We observed similar memory-dependent performance with the SPECweb99 benchmark. Increasing memory from 256 Mbytes to 512 Mbytes yielded a 45 percent performance improvement. Beyond 512 Mbytes, the benefit diminished.

We expect equally significant MXT cost-performance advantages for large, commercial database applications. We compared performance results published by the Transaction Processing Council as a function of memory size (4 Gbytes versus 8 Gbytes) for similarly configured systems. These included the Microsoft Windows 2000 Server Edition and the Microsoft Internet Information Server 5.0 running on quad Xeon processors with 2 Mbytes of cache. Our comparison suggests a 30 to 40 percent MXT advantage for either

Table 2. Pinnacle chip reaction times.

Processor request (with respect to shared-cache hit/miss)	Latency (bus clock cycles) for first and subsequent data transfers
Write	6, 1, 1, 1
Read, hit (SDRAM row open and row hit)	8, 1, 1, 1
Read, hit (SDRAM row open and row miss)	14, 1, 1, 1
Read, hit (SDRAM row closed)	14, 1, 1, 1
Read, hit (SDRAM autoprecharge mode)	12, 1, 1, 1
Read, miss compressed (average)	69, 1, 1, 1
Read, miss not compressed	24, 1, 1, 1
Read, miss compression off	16, 1, 1, 1

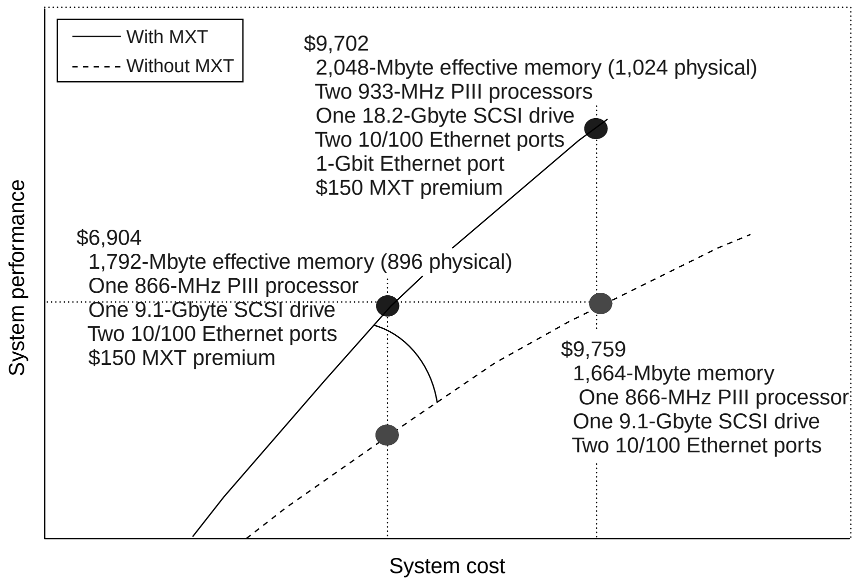


Figure 12. System cost-performance comparison.

a price or performance metric.[9]

We began the MXT project with the primary goal of doubling the effective system memory at a negligible cost without degrading system performance. We based our measurement of the Pinnacle chip's performance on the intrinsic chip reaction times listed in Table 2, as well as the shared-cache hit rate.

The shared-cache hit rate is application dependent, and we usually measure the cache hit rate at roughly 98 percent on most applications. However, the cache hit rate for large database applications such as TPC-C, SAP, and particularly Lotus Notes can range from as low as 94 percent, as measured by quad processor trace-driven performance models. These applications reference some database records infrequently, resulting in a prefetch advantage with the long cache line, but little reuse of data within the line.

In our comparison of an MXT system with a high-performance contemporary system, the two systems had essentially equivalent (within one point) performance for the SPECint2000 benchmark. Both machines were 512-Mbyte IBM 1U commercial servers with Intel 733-MHz PIII processors, executing program code from the same disk drive. The two systems differed only in their memory controllers: the MXT system used the Pinnacle chip; the other used the ServerWorks CNB30LE chip.

Figure 12 illustrates the degree of MXT's system cost leverage with a case in point. The figure shows how the cost-performance metric for conventional machines (dashed line) dramatically improves when we factor in the benefits of MXT (solid line). For a representative product, we configured a ProLiant DL360 commercial server on Compaq's retail-equipment sales Web site (http://www5.compaq.com/products/servers/platforms). This server was priced at $9,759. Then we configured a hypothetical MXT-equivalent system, with half the memory and an estimated $150 MXT cost premium, achieving a near 30 percent savings at $6,904. Finally, we configured a higher-performance hypothetical MXT system at a cost ($9,702) commensurate with that of the reference machine. Either way, the MXT system cost-performance metric compares favorably with any conventional system.

A logical step in the growth of compression technologies, MXT, in products like Pinnacle, empowers customers to gain full advantage of their memory investment. IT professionals can routinely save thousands of dollars on systems ranging from high-density servers to large-memory enterprise servers. We expect MXT to extend its presence to other processor memory controllers, as well as other memory-intensive system applications includ-

Pinnacle features

- Single-chip controller
  —731 (525 signal) EPBGA, 45 × 45-mm package
  —single 2.5-V DC power supply (6 W typical)
  —100- or 133-MHz operation (synchronous to a processor, cache, and memory interfaces)
- Processor interface
  —Pentium III or Xeon, one to four processors, 36-bit address
  —Eight-entry request queue
  —8 × 32-byte burst write buffer and 8 × 4-byte write buffer
- I/O interface
  —dual independent, full-duplex, 533-Mbyte-per-second, remote-I/O bridge intermodule bus (IMB) links
  —16-entry request queue per link
- Shared cache
  —dual-ported on-chip directory supporting 16-Gbyte real-address space
  —32-Mbyte DDR SDRAM four-way set associative, error-correcting-code (ECC)-protected cache with 1-Kbyte line
  —reference state for snoop filtering (256-byte granularity and I/O)
  —2.1-Gbyte-per-second DDR SDRAM cache interface
  —least recently used (LRU) replacement, write-back, and write-allocate policies
- Main memory
  —2.1-Gbyte-per-second access to 16-Gbyte uncompressed or 8-Gbyte (16-Gbyte effective) compressed memory
  —supports PC100 and PC133 DIMMs
  —ECC with scrub for 4- and 8-bit-wide memory device failure protection
  —hardware memory management with hardware-accelerated 4-Kbyte page move, swap, and clear
  —real-time hardware dataflow compression/decompression for 1:1 to 64:1 main-memory content compression
- Hardware performance monitor facility
- Inter-integrated circuit (I^{2}C) bus access to all internal registers

ing disk storage controllers, laptop computers, and information appliances. MICRO

Acknowledgments

MXT and its embodiment in the Pinnacle chip involved significant contributions from many people. We especially thank Peter Franaszek and John Robinson for their work on the compression approach; Dan Poff, Rob Saccone, and Bulent Abali for their work on the operating system and performance measurement; and Michel Hack and Chuck Schulz for their work on data storage and page operations.

References

1. W. Hovis et al., *Compression Architecture for System Memory Application*, US patent 5812817, Patent and Trademark Office, Washington, D.C., 1998.
2. M. Kjelso, M. Gooch, and S. Jones, "Design and Performance of a Main Memory Hardware Data Compressor," *Proc. 22nd Euromicro Conf.*, IEEE Computer Society Press, Los Alamitos, Calif., 1996, pp. 423-430.
3. R.B. Tremaine, T.B. Smith, and M. Wazlowski, "IBM Memory eXpansion Technology (MXT)," to be published in *IBM J. Research and Development*, Apr. 2001.
4. M. Abbott et al., "Durable Memory RS/6000 System Design," *Proc. 24th Ann. Int'l Symp. Fault-Tolerant Computing*, IEEE CS Press, Los Alamitos, Calif., 1994, pp. 414-423.
5. P.A. Franaszek and J. Robinson, *Design and Analysis of Internal Organizations for Compressed Random Access Memories*, research report RC 21146(94535)20OCT1998, IBM Research, Yorktown Heights, N.Y., 1992.
6. P.A. Franaszek, J. Robinson, and J. Thomas, "Parallel Compression with Cooperative Dictionary Construction," *Proc. Data Compression Conf.*, IEEE CS Press, Los Alamitos, Calif., 1996, pp. 200-209.
7. C.L. Chen et al., "Reliability-Availability-Serviceability of a Compressed-Memory System," *Proc. Int'l Conf. Dependable Sys-*

tems and Networks, IEEE CS Press, Los Alamitos, Calif., 2000, pp. 163-168.

8. B. Abali and H. Franke, "Operating System Support for Fast Hardware Compression of Main Memory Contents," research report RC 21964, IBM Research, Yorktown Heights, N.Y., 2000.
9. T.B. Smith et al., "Memory eXpansion Technology (MXT): Competitive Impact," to be published in *IBM J. Research and Development*, Apr. 2001.

R. Brett Tremaine is a senior technical staff member at the IBM T.J. Watson Research Center, where he is responsible for commercial server and memory hierarchy architecture, design, and ASIC implementation. Tremaine received an MS in computer engineering from Syracuse University and a BS in electrical engineering from Michigan Technological University. He is a member of the IEEE.

T. Basil Smith is a senior manager at the IBM T.J. Watson Research Center, where he researches the exploitation of high-leverage server innovations. His work has focused on memory hierarchy architecture, reliability, durability, and storage efficiency enhancements in advanced servers. Smith received a BS, an MS, and a PhD in computer systems from MIT. He is a member of the IEEE Computer Society Technical Committee on Fault-Tolerant Computing and an IEEE Fellow.

Mike Wazlowski is a research staff member at the IBM T.J. Watson Research Center, where he is responsible for high-performance memory system architecture and design. His research interests include computer architecture, memory systems, and ASIC design. Wazlowski received a BS in computer engineering from the University of Massachusetts at Amherst and an MS and a PhD in electrical sciences from Brown University. He is a member of the IEEE.

David Har is an advisory engineer at the IBM T.J. Watson Research Center, where he implemented the MXT compression algorithm and hardware. His technical interests include microarchitecture and low-power DSP design. Har received MS and BS degrees in electrical engineering from MIT.

Kwok-Ken Mak is a senior hardware engineer at Cisco Systems, where he is responsible for high-performance memory architecture and design. Earlier, he was a senior engineer at the IBM T.J. Watson Research Center, where he participated in the MXT project. He received a BS in electrical engineering from Polytechnic University of New York.

Sujith Arramreddy is the chief technology officer and a founding member of ServerWorks, where he is responsible for technology direction. Arramreddy received an MS in computer engineering from Wayne State University and a BS in electrical engineering from Jawaharlal Nehru Technical University in Hyderabad, India.

Direct questions about this article to Brett Tremaine, IBM T.J. Watson Research Center, PO Box 218, Yorktown Heights, NY 10598; afton@us.ibm.com.