Delta-Compressed Caching for Overcoming the Write Bandwidth Limitation of Hybrid Main Memory

YU DU, MIAO ZHOU, BRUCE CHILDERS, RAMI MELHEM, and DANIEL MOSSÉ,
University of Pittsburgh

Limited PCM write bandwidth is a critical obstacle to achieve good performance from hybrid DRAM/PCM memory systems. The write bandwidth is severely restricted in PCM devices, which harms application performance. Indeed, as we show, it is more important to reduce PCM write traffic than to reduce PCM read latency for application performance. To reduce the number of PCM writes, we propose a DRAM cache organization that employs compression. A new delta compression technique for modified data is used to achieve a large compression ratio. Our approach can selectively and predictively apply compression to improve its efficiency and performance. Our approach is designed to facilitate adoption in existing main memory compression frameworks. We describe an instance of how to incorporate delta compression in IBM's MXT memory compression architecture when used for DRAM cache in a hybrid main memory. For fourteen representative memory-intensive workloads, on average, our delta compression technique reduces the number of PCM writes by 54.3%, and improves IPC performance by 24.4%.

Categories and Subject Descriptors: C.2.2 [Computer Architecture]: Cache memories

General Terms: Design, Performance

Additional Key Words and Phrases: Phase change memory, memory compression

ACM Reference Format:

Du, Y., Zhou, M., Childers, B., Melhem, R., and Mossé, D. 2013. Delta-Compressed caching for overcoming the write bandwidth limitation of hybrid main memory. *ACM Trans. Architec. Code Optim.* 9, 4, Article 55 (January 2013), 20 pages.

DOI = 10.1145/2400682.2400714 <http://doi.acm.org/10.1145/2400682.2400714>

1. INTRODUCTION

Driven by multicore processors and datacenter applications, there is an increasing demand for main memory capacity to consolidate computation tasks and solve large problem sizes. For example, Intel's 10-core 20-thread server processor can support 4TB of memory in a 8-socket system [Intel 2011]. Facebook uses thousands of servers supplying over 28TB of memory as a distributed cache for their database to mitigate expensive disk access latency [Saab 2008]. With increasing capacity, DRAM has become a major source of datacenter energy consumption [Hoelzle and Barroso 2009; Lefurgy et al. 2003]. Specifically, at low load, energy consumption is dominated by DRAM background (static) power. Thus, there is a critical need to find new memory architectures that have high capacity (density) and consume little energy [Malladi et al. 2012].

This work was supported by NSF awards CNS-1012070. This work was done in the PCM@Pitt group (www.cs.pitt.edu/PCM).

Author's addresses: Y. Du (corresponding author), M. Zhou, B. Childers, R. Melhem, D. Mossé Computer Science Department, University of Pittsburgh, Pittsburgh, PA; email: fisherdu@cs.pitt.edu.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.

© 2013 ACM 1544-3566/2013/01-ART55 \$15.00

DOI 10.1145/2400682.2400714 <http://doi.acm.org/10.1145/2400682.2400714>

Phase change memory (PCM) is a promising technology for constructing energy-efficient memory [Lee et al. 2009; Qureshi et al. 2010b; Ramos et al. 2011]. When used as a large-capacity main memory, PCM has two major advantages. First, PCM cells have good scalability and can achieve excellent memory density. A high-performance PCM cell design has been demonstrated in sub-20nm technology [Kim et al. 2010]. With a multilevel cell design (MLC), PCM can achieve even greater densities by storing multiple bits in one bit cell. Second, and most importantly, PCM has negligible background power due to its nonvolatile nature. Therefore, PCM can provide much higher capacity than DRAM within the same budget [Qureshi et al. 2009].

Despite advantages, PCM also has drawbacks. It has long write latency, high write energy, and limited write endurance in comparison to DRAM. In addition, PCM devices have severely constrained write bandwidth. For example, a prototype 20nm 8Gb PCM chip [Choi et al. 2012] has been demonstrated that has a read bandwidth of 800MB/s and a write bandwidth of 40MB/s. The limited write bandwidth is due to two reasons. First, it takes much longer to program (i.e., write) a PCM cell than a DRAM cell. Second, the programming of a PCM cell takes more write energy than a DRAM write. To provide the same write bandwidth, PCM may require five times more power than DRAM [Chen et al. 2011]. This large difference between read and write bandwidth makes PCM not suitable for storing frequently modified data.

Since PCM cannot fully replace DRAM, *hybrid memory* systems [Zhang and Li 2009; Mogul et al. 2009; Ramos et al. 2011] have been proposed with DRAM and PCM. Common hybrid organizations use DRAM as a cache or an extension to PCM. When used as a cache, the DRAM's address space is not visible to the software. Frequently modified data are transparently cached in DRAM to offload write traffic. It is important for hybrid memory systems to cache the frequently modified data to avoid the limited PCM write bandwidth becoming a performance bottleneck.

In this article, to allow more data to be cached in DRAM, we propose to increase the *effective capacity* of the DRAM cache with compression that mimics DRAM-only memory compression. For hybrid memory, however, only frequently modified data are suitable for compression because the read latency advantage of DRAM over PCM is offset by the extra decompression cost. We propose a *delta compression* scheme that is specifically designed to compress only modified data for hybrid memory. To improve delta compression's efficiency and reduce its overhead, we also propose *selective and predictive compression* that avoids unnecessary delta compression operations. We evaluate our design on fourteen different memory-intensive workloads. The results show that the proposed DRAM delta compression reduces, on average, the number of PCM writes by 54.3% and improves IPC performance and system energy consumption by 24.4% and 11.0%, respectively.

The rest of the article is organized as follows. Section 2 motivates the use of DRAM delta compression in hybrid memory. Section 3 describes our new DRAM compression organization that specifically targets hybrid memory. Section 3.3 describes an instance of using our technique in IBM's MXT compression framework, when applied to DRAM cache in a hybrid memory. Section 5 details our experimental methodology and Section 6 evaluates delta compression's effectiveness, overhead, and architecture sensitivity. Sections 7 and 8 summarize related work and conclusions.

2. MOTIVATION

2.1. PCM Write Bandwidth as Performance Bottleneck

Although integrating PCM with DRAM has a standby power advantage over a DRAM-only design, PCM can harm performance due to two reasons. First, PCM read latency

Fig. 1. Normalized IPC with different PCM read latencies and write bandwidths.

is higher than DRAM. Second, PCM has acutely restricted write bandwidth. Figure 1 shows how performance can degrade when DRAM is replaced by hybrid main memory. The results are collected for twelve memory-intensive benchmarks from the SPEC CPU2006 and PARSEC benchmark suites (see Section 5). Each simulated benchmark has 64MB DRAM cache quota. Data that is not cached in the DRAM cache is accessed from PCM. The figure shows the average impact on Instructions Per Cycle (IPC) from different PCM read latency and write bandwidth assumptions. The IPC values are normalized to a DRAM-only system with 50ns read latency and 6.4GB/s write bandwidth. The figure shows that the performance degradation due to longer PCM read latency is not as significant as the impact of writes. When the PCM read latency doubles, the performance slowdown is about 10%.

In comparison, PCM write bandwidth has a significant impact on performance. With 400MB/s PCM write bandwidth, the performance drops by 65%. When PCM write bandwidth is a bottleneck, performance is determined by the available PCM write bandwidth and is not sensitive to read latency. Given constrained PCM write bandwidth, it is critical for hybrid memory to cache more *frequently modified data* to reduce PCM write traffic, even at the cost of increased read latency.

2.2. Compressed DRAM Caching in Hybrid Memory

To cache more memory data, compression can be used to increase the effective capacity of a DRAM cache. In a DRAM-only system, compression can help avoid expensive disk accesses. In hybrid memory, DRAM compression can reduce PCM write traffic and avoid PCM write bandwidth becoming a bottleneck. Figure 2 shows our organization to support compressed DRAM caching in hybrid memory. DRAM is used as a cache to store both *performance-critical data* and *frequently modified data*. The cache's capacity is divided into *uncompressed* and *compressed* regions. The amount of capacity allocated to each region can be dynamically changed. The uncompressed region has low access latency (it does not require decompression and compression), and thus, it is used to cache performance-critical data. The compressed region has higher access latency than the uncompressed region; it caches only frequently modified data that are not in the uncompressed region. Nonmodified data are not cached in the compressed region because the potential gain to read data from the compressed DRAM region instead of PCM is limited due to the latency overhead to locate and decompress the data.

Fig. 2. Compressed DRAM caching in hybrid memory.

Fig. 3. An example to illustrate delta compression.

2.3. Delta Compression for Written Data

For compression in a DRAM-only system, the data in a *memory line* itself should be compressed¹. A higher compression ratio can be achieved with delta compression. Delta compression requires the data having a reference copy. The modification to the reference copy is compressed instead of the data itself. Figure 3 shows how to apply delta compression in hybrid memory. First, the old data is read from PCM. Then the difference between the new and the old data (called *diff data*) is computed using an XOR operation. Lastly, the diff data are compressed and stored in DRAM. Delta compression tends to have a higher compression ratio than conventional compression because delta compression converts unmodified data bits to zeros before compression.

The challenge is reading delta compressed data (i.e., the *frequently modified data*): it is more time consuming because the decompressed diff data cannot be accessed directly from DRAM. Instead, the old PCM data must be read from main memory (PCM) and XORed with the decompressed diff data. Delta compression in DRAM provides a way to trade more PCM reads off against reduced PCM writes (caching more modified data in a more compressed format).

In summary, for delta compression in hybrid memory, frequently modified data are compressed and partially stored in both DRAM and PCM. The memory controller splits and combines data to/from DRAM and PCM to provide correct data. Given this unique requirement on memory compression in hybrid memory, we designed a compression scheme specially tailored to the DRAM cache in hybrid memory, as described next.

3. COMPRESSED DRAM CACHING

To support compressed DRAM caching, we introduce three components to control the DRAM cache. The *page classification* module identifies memory pages that are suitable for the uncompressed DRAM region and compressed DRAM region. The *partition*

¹A memory line is defined as the minimum unit for memory compression.

adjustment module determines how much DRAM capacity is used by each region. The *compression and decompression* module accesses data in the compressed region.

3.1. Page Classification

Page classification is used to determine whether data should be kept uncompressed (frequently accessed), compressed (frequently modified), or uncached (infrequently accessed). The monitoring is done at page granularity instead of line granularity to amortize the storage overhead. Recent techniques for PCM data placement use variations of a MultiQueue (MQ) algorithm to identify hot pages to cache in DRAM [Zhang and Li 2009; Ramos et al. 2011]. In an MQ algorithm, a memory page's access frequency is monitored. Pages are placed into a queue based on their access frequency. Queues are classified into *hot* and *cool*, and pages in the hot queues are cached in DRAM.

In our technique, the DRAM cache is partitioned into uncompressed and compressed regions. Therefore, we introduce an additional classification category to the MQ algorithm: *warm queues*. Using this classification, pages in hot queues are cached in the uncompressed region, pages in warm queues are cached in the compressed region, and pages in cool queues are not cached in DRAM. Because the compressed region is suitable for storing frequently modified pages, page write frequency should be used to identify warm pages, while page access frequency should be used to identify hot pages. However, this increases hardware cost to monitor multiple metrics; we assume that the memory controller uses only page access frequency to determine page type.

3.2. DRAM Partition Adjustment

When data is cached in the compressed region, its access latency inevitably increases due to extra indirect memory accesses and compression/decompression operations. Therefore, it is beneficial to control the capacity of the uncompressed and compressed cache regions. For workloads with low PCM write traffic, the capacity allocated to the uncompressed region should be increased to get the benefit from low access latency. When PCM write traffic is high, the compressed region's size should be increased to cache more modified data in DRAM (avoiding writebacks and hitting the "write bandwidth wall" of the PCM devices). We assume that the capacity allocated to the uncompressed and compressed regions are parameters that are fixed at system boot-up (e.g., set in the BIOS). We investigate the trade-off of how much capacity to allocate to each region in the experimental results. The allocation decision could be made online, according to monitored workload behavior. As this article addresses the detailed design of incorporating compressed DRAM cache in hybrid memory, online allocation is left for future work.

3.3. Selective and Predictive Compression

Any memory compression scheme can be used for the compressed region of the DRAM cache. In our scheme, we choose to use conventional and delta compression together. Delta compression leads to a higher compression ratio and fewer PCM writes, but it may not improve performance because delta compression can generate extra PCM read traffic (to read the old reference data) for both memory reads and writes. Due to this additional cost, we propose *selective compression*, which sets a threshold on when delta compression should be enabled. Specifically, for each memory line, the memory controller selectively enables delta compression only if it leads to a minimum gain in storage (which we call a *Gain Threshold*, GT) over conventional compression. The GT captures a trade-off between gain in memory capacity due to compression and extra memory traffic to access delta-compressed data. Also, a line is kept uncompressed if neither compression nor delta compression can reduce the storage needed for the line.

When writing a memory line, selective compression needs to read the old data in PCM to evaluate whether the line should be delta-compressed. If the line is evaluated as not suitable for delta compression, the extra PCM read becomes an unnecessary overhead. We propose *predictive compression* to address this problem. Predictive compression relies on the observation that delta compression is probably not advantageous for a line if delta compression was not advantageous the last time that the line was written. In other words, the memory controller applies delta compression to lines that can be consistently delta-compressed. For a line that was not delta-compressed last time it was written, the memory controller typically stores it not delta-compressed to avoid unnecessary PCM read cost. To give the line a chance to recover delta-compressed status, with a small probability (which we call *Recovery Probability*, RP) the memory controller will evaluate whether the line should be delta-compressed.

4. IMPLEMENTATION OF DRAM CACHE COMPRESSION

Delta compression is a unique feature for hybrid memory because it needs both the DRAM and the PCM (for reference data). In this section, we choose to use a DRAM-only compression scheme as our baseline and extend it to support delta compression for hybrid memory. This approach allows us to focus our efforts on enabling the unique delta compression without reinventing the nontrivial mechanisms that are germane to any memory compression scheme. Our proposed techniques can be tailored to other compression schemes that are used for hybrid memory systems.

4.1. IBM MXT Compression for DRAM-Only Systems

IBM Memory Expansion Technology (MXT) [Tremaine et al. 2001] is a high-performance hardware-based memory compression scheme for DRAM-only systems. Figure 4(a) shows MXT's architecture; 4KB pages are stored in DRAM in compressed form. On every memory access, the memory controller (MC) compresses or decompresses the data. MXT offers a framework for our D-COMP algorithm and its variants. Other compression frameworks could also be used.

MXT partitions DRAM into two regions: the Sector Translation Table (STT) and the sector memory². STT contains the metadata to locate compressed lines. Sector memory is a collection of 256-byte memory sectors that hold compressed data. Each 4KB page is divided into 4 *memory lines*. Figure 4(b) shows how each memory line is compressed in MXT. A 16-byte STT entry is associated with each 1KB memory line and contains up to 4 pointers to memory sectors. The memory controller uses the pointers in the STT entry to access the data. The control field of the STT entry contains a line state to indicate whether the line is Uncompressed (U), Compressed (C), or Invalid (I). If a line is Uncompressed, the STT entry uses all pointers to 4 memory sectors. If a line is Compressed, the STT entry may have fewer than 4 valid pointers to compressed memory sectors. The control field also contains the compressed size of the line. To avoid repeated accesses to STT entries, a small dedicated on-chip storage in the memory controller (*STT Cache*) contains recently accessed STT entries.

4.2. Hierarchical Compression Metadata

The most important change to MXT for hybrid memory involves the compression algorithm. MXT uses a parallel derivative [Franaszek et al. 1996] of the Ziv-Lempel (LZ77) algorithm. Because LZ77 is a dictionary algorithm, the memory line size is 1KB to achieve a good compression ratio. The whole 1KB memory is stored in DRAM. It takes 128 CPU cycles for MXT to decompress a memory line.

²This is a common design for a memory compression scheme.

Fig. 4. IBM MXT compression.

In contrast, our new approach for hybrid memory needs to cache only modified data in the compressed region of DRAM. Therefore, we divide the 1KB line into sixteen 64-byte *blocks* to allow a finer granularity of compression (i.e., 64 bytes rather than 1KB). The memory controller uses *Frequent Pattern Compression* (FPC) [Alameldeen and Wood 2004b] to compress memory blocks. FPC is the underlying compression algorithm for both conventional and delta compression. FPC compresses data on a word-by-word basis by storing common patterns in a compressed format accompanied by an appropriate prefix. FPC has a low-cost hardware implementation [Alameldeen and Wood 2004b] and takes only 5 CPU cycles to encode/decode a 64-byte block [Alameldeen and Wood 2004a]. In our implementation, the sixteen blocks in a memory line can be compressed or decompressed in parallel.

Figure 5(a) shows an example in which nine modified blocks are compressed and stored in DRAM. For each block, both conventional compression and delta compression are tried. The compression method that achieves the smallest compressed block size is selected. If both schemes give a compressed block size equal or larger than the uncompressed size, neither scheme is selected and the block is stored in uncompressed form. Each block has a state to indicate its format. There are four block states: Invalid (I), Uncompressed (U), Compressed (C), and Delta-compressed (D). If a block state is Invalid, data is written and read from PCM. If a block state is Uncompressed or Compressed, data is written and read from DRAM. If a block state is Delta-compressed, then the DRAM stores only the diff data. To read the content of the block, the PCM data is read and XORed with the diff data.

(a) an example of a compressed line

Superblock	Block
Invalid (I)	I
Uncompressed (U)	U
Compressed (C)	U, C
Delta-compressed (D)	I, U, C, D

(b) Block and superblock states

Fig. 5. FPC-based delta compression algorithm.

FPC-based compression allows a line to be partially decompressed to access some of its data blocks. To achieve this capability, additional metadata are stored in the STT entry of each compressed line. At a minimum, each 64-byte block requires 5-bit metadata: a 2-bit block state and a 3-bit block size (if data block is aligned at 8-byte boundary). Hence, for all 16 blocks in a line, there is an extra 80-bit metadata in each STT entry versus the original MXT scheme. To avoid actually increasing the STT entry size, we reuse the PTR3 field of the STT entry, as follows.

For our compression scheme, a 1KB line is not compressed if compressed data needs more than three memory sectors. Therefore, for compressed lines, the PTR3 field of the STT entry is unused. However, the PTR3 field is only 30 bits, which is not enough to store the 80-bit metadata. To work around the problem, we use a hierarchical design for the metadata. We treat every 4 blocks in a line as a *superblock*. Each 256-byte superblock requires 4-bit metadata: a 2-bit superblock state and a 2-bit superblock size (superblocks are aligned at 64-byte boundary and the superblock size field is valid only for compressed superblocks). The full 16-bit metadata for 4 superblocks can be stored in the PTR3 field.

The *superblock state* is a summary of the states of the associated blocks (see Figure 5(b)). For an uncompressed (U) superblock, all of its blocks are uncompressed and there is no need to store block states. Similarly, for an invalid (I) superblock, all of its blocks are invalid and the block states are unnecessary. If all blocks in a superblock are stored in DRAM and at least one block is compressed, the state of the superblock is compressed (C). If the state of a superblock is not U, I, and C, the state of the superblock is delta-compressed (D). For compressed or delta-compressed superblocks, the states of constituent blocks are stored along with the compressed data (described in the next section). By keeping superblock states rather than block states, we trade the granularity of state information off against the storage overhead of the metadata.

Fig. 6. Data layout of an example memory line.

4.3. Compressed Data Layout

In MXT, compressed data are stored contiguously in the logically linear storage implemented by the pointers in an STT entry. To avoid accessing two memory sectors for each uncompressed superblock, we use a revised compressed data layout in the logically linear storage. In our layout, uncompressed superblocks are placed before compressed superblocks. By doing so, each uncompressed superblock can align to a 256-byte boundary and be stored in a dedicated sector³. Compressed superblocks are sequentially arranged after uncompressed ones. Because our design uses a 64-byte read size, a compressed superblock is aligned to a 64-byte boundary to avoid extra memory reads for crossing a 64-byte boundary.

Figure 6 shows the compressed data layout of an example memory line. Uncompressed superblock 1 is placed at the beginning of the logically linear storage, followed by two compressed superblocks. There is a gap between the two compressed superblocks because superblocks are aligned to a 64-byte boundary. The example also shows the detailed layout of superblock 2. Since this block is delta-compressed, it has a block state header before the compressed data. The 1-byte header contains the block states of the superblock. The data of the blocks are compressed by the FPC algorithm. The size of each compressed block is indicated by the metadata inside the FPC-compressed data.

4.4. Operations

4.4.1. Memory Read. To read data from the cache, the memory controller needs to determine the location of the data. Figure 7(a) shows how to determine the data location from the line state and superblock state. If line state is invalid or uncompressed (first two rows of Figure 7(a)), the data will be read from PCM and DRAM only, respectively. If line state is compressed, the location of data depends on the superblock state as follows. If a superblock X is invalid (I), the data will be read from PCM. If a superblock X is uncompressed (U) or compressed (C), the data will be read from DRAM. If superblock X is delta-compressed (D), the data will be read from both DRAM and PCM. The data from the DRAM and PCM also need to be merged to get actual memory data. The merge operation is performed at block granularity based on the state of each block.

³Recall that the logically linear storage space is represented by multiple 256-byte sectors.

Line State	Superblock State	DRAM	PCM
I	-	No	Yes
U	-	Yes	No
C	I	No	Yes
C	U	Yes	No
C	C	Yes	No
C	D	Yes	Yes

(a) determine data location

Block State	Action on Data
Invalid (I)	Read PCM
Uncompressed (U)	Read DRAM
Compressed (C)	Read Decompressed DRAM
Delta-compressed (D)	Read Decompressed DRAM XORed with PCM

(b) merge block data

Fig. 7. Rules for reading data from compressed hybrid memory.

Figure 7(b) shows the merging rules. For example, if a block is delta-compressed, the data from the DRAM is decompressed and XORed with the data from the PCM.

To read PCM data, the address is the same as the address of the memory request. To read DRAM data, the memory controller needs to calculate the start position of the superblock from the superblock state and superblock size in the STT entry.

If superblock state is U, its relative position is the sum of the sizes of the uncompressed superblocks stored before it, that is

$$StartPos(X) = \sum_{j=1}^{X-1} Uncomp(j) \cdot Size(j). \quad (1)$$

If a superblock state is C or D, its relative position is the sum of the sizes of all superblocks stored before it and all other uncompressed superblocks, that is

$$StartPos(X) = \sum_{j=1}^{X-1} Size(j) + \sum_{j=X+1}^4 Uncomp(j) \cdot Size(j). \quad (2)$$

To implement this calculation, the simple logic only needs to conditionally sum the size of four superblocks.

4.4.2. Memory Write. To store a compressed superblock, the memory controller first evaluates the different sizes of the data when the superblock is stored as uncompressed (U), compressed (C), or delta-compressed (D). To calculate the size of a delta-compressed superblock, the current PCM data of the superblock are read. After the evaluation, the memory controller chooses the superblock state that can achieve the smallest compressed size. If the storage size of the superblock is unchanged (aligned on a 64-byte boundary), then the new data will be directly written to DRAM. If the storage size of the superblock is changed (overflow/underflow), which is a common condition for compressed memory, the memory controller will read from DRAM all superblocks stored after the requested superblock. The memory controller writes all affected superblocks to their new DRAM location. We evaluated this extra data movement cost in our experiments. With a write buffer, the data movement operation is not on the critical path, and accounts for only a small portion of total DRAM memory traffic. Similarly to reading PCM, writing PCM is straightforward. The data is directly written to PCM using the address from the write request. The memory controller needs to write data to PCM in two cases: a memory line is written but not cached in DRAM or a memory line is evicted from DRAM. Next, we describe the cache replacement policy for the compressed DRAM cache.

4.4.3. Cache Replacement Policy. For MXT, unused memory sectors are organized as a linked list. When extra memory storage is needed, unused sectors are allocated from the list by a small hardware circuit [Tremaine et al. 2001]. The memory controller tries

to maintain a minimum quota of unused DRAM sectors (16MB in our experiments). Once the quota is below this threshold, the memory controller walks the STT to evict rarely accessed pages.

Our design uses a MQ algorithm to monitor page access frequency. Recall that hot pages are placed in the uncompressed region, only warm pages are candidates for cache replacement in the compressed region. The memory controller tracks the number of pages in each warm queue and divides warm queues into high-rank queues and low-rank queues. A high-rank warm page is guaranteed to be cached in the compressed region. A low-rank warm page with low access frequency is evicted first. To avoid thrashing, modified data of a low-rank warm page is inserted into the compressed region with a throttled insertion policy [Qureshi et al. 2007].

4.5. Hardware Cost and Overhead

The hardware cost to support compressed DRAM depends on the underlying compression algorithm and page classification mechanism. Delta compression by itself only requires implementing a tailored delta compression algorithm for both memory reads and writes in the memory controller. We use FPC as the building block for delta compression. FPC is easily implemented in hardware and is used for on-chip cache compression [Alameldeen and Wood 2004a].

For MXT, the STT is the major memory data structure. The size of the table can be dynamically adjusted based on the size of the compressed region. For every 2GB of compressed capacity, 32MB DRAM is needed for the STT. We assume a 16KB STT cache to hold the 256 most recently accessed STT entries. We use Rank-based Page Placement (RaPP) [Ramos et al. 2011], a hardware-assisted variation of MQ, for page classification. RaPP requires 126KB on-chip storage and 12MB DRAM to maintain data structures for the MQ algorithm for a 2GB DRAM and 16GB PCM hybrid memory.

When data is stored in compressed form, its access latency is increased. There are two reasons for the increase: STT translation delay and decompression latency. We assume 5 CPU cycles for STT translation delay, for a hit in the STT cache. We assume 15 CPU cycles to decompress a 256-byte superblock and merge the data from both DRAM and PCM if the data is delta-compressed. We also model the extra memory delay due to STT misses. For uncompressed data, the memory latency is the time to read the critical 64-byte block. For compressed data, all compressed blocks of a superblock are read before decompressing the superblock.

Because FPC is simple, an on-chip FPC compression/decompression engine has low power overhead. The power of a 4GHz implementation is estimated to be 0.28W [Das et al. 2008]. We add 0.3W for the compression and decompression logic. We assume 0.3W for STT logic and 0.3W for buffering.

5. EXPERIMENTAL METHODOLOGY

5.1. System Configuration

We use Virtutech Simics [Magnusson et al. 2002] to collect memory traces. To evaluate memory compression, we use a trace-driven simulator that takes traces as input files with the command and address of each memory request, and the data before and after every memory write.

We model an 8-core 2GHz CMP with in-order cores and a cache hierarchy similar to the IBM Power7 [Sinharoy et al. 2011] with a 32MB L3 cache. Since we execute multiprogrammed workloads with one program per core, we assume that each core only uses its local 4MB L3 cache region. To alleviate the miss penalty, we also modeled a simple sequential data prefetcher for the L3 cache.

Table I. System Settings

CPU	8-core, 2GHz, 2-issue, in-order, 32KB L1.
L2 Cache (private)	512KB, 8-way, 64-byte linesize, write-back.
L3 Cache	32MB total, 4MB per core, 256-byte linesize, 20-cycle local L3 hit, write-back, sequential data prefetcher
Memory Controller	2 PCM channels + 2 DRAM channels, 12.8GB/s per channel, 2 DIMMs per channel, 8 banks per DIMM, 32-entry read- and write-queue per bank, read priority scheduling for DRAM, write pausing scheduling for PCM, FPC decoding latency 15 cycles. STT translation latency 5 cycles
DRAM	2GB, 50 ns (100 cycles) for first 64-byte read, 15 ns (30 cycles) for each consecutive 64-byte read. Background Power: 0.93W/GB (leakage + refresh) Read Energy: 0.8J/GB, Write Energy: 1.2J/GB
PCM	16GB, 80 ns (160 cycles) for first 64-byte read, 15 ns (30 cycles) for each consecutive 64-byte read, 0.75GB/s write bandwidth per channel, (2600-cycle write service time per 64-byte write), 50ns epoch, 26 epochs per PCM write. Background Power: 0.10W/GB (leakage) Read Energy: 1J/GB, Write Energy: 6J/GB

We model a hybrid memory system with 2GB DRAM and 16GB PCM. Similar to Ramos et al. [2011], we used a small DRAM size to match the memory footprint of the workloads. The hybrid memory has four 12.8GB/s memory channels. Based on the measured bandwidth requirements, DRAM and PCM have two dedicated memory channels, each with two DIMMs, and each DIMM with eight banks. We use a memory controller configuration similar to Qureshi et al. [2010a], where each bank has a separate 32-entry read queue and a 32-entry write queue. Read requests are given higher priority, as long as the write queue is not full. For PCM memory scheduling, we use Write Pausing [Qureshi et al. 2010a]: a PCM write is divided into multiple 50ns epochs and the memory controller can suspend an active PCM write at the beginning of an epoch to schedule a read request to the memory bank. In the baseline, we assume 0.75GB/s PCM write bandwidth per channel, which is equivalent to PCM write service time of 2600 cycles per write. We also perform a sensitivity analysis on PCM write bandwidth in Section 6.

We calculate memory power as in David et al. [2011], adding background power and operation power. Background power is determined by memory type, capacity, and power state. Operation power is assumed to be proportional to memory bandwidth. Memory power is estimated using the parameters in Chen et al. [2011]. The simulation parameters are summarized in Table I. Our experiments confirmed the results in Qureshi et al. [2009] that a hybrid memory can have better performance and energy than a DRAM-only system due to the increased capacity enabled by better cell density from PCM.

5.2. Workloads

Because our work focuses on memory, we use only memory-intensive benchmarks that have large memory footprints from the SPEC CPU2006 [CPU2006] and PARSEC

Table II. Benchmarks with Scaled Problem Sizes

Name	Ref Problem Size	New Problem Size
libquantum	1397	5953 (1GB)
milc	20x20x20x20	26x26x26x26 (1.5GB)
leslie3d	121x121x15	331x331x35 (1.2GB)

Table III. Simulated Workloads

Name	Size (GB)	Read PKI	Write PKI	Description
Gems.r	6.3	5.34	2.40	8 copies of GemsFDTD
leslie.r	9.2	3.71	1.12	8 copies of leslie3d
mcf.r	12.8	15.10	7.24	8 copies of mcf
milc.r	11.8	6.18	1.84	8 copies of milc
mix.1	8.3	8.10	4.09	lbm-libq-mcf-milc
mix.2	8.2	5.43	1.66	libq-mcf-milc-zeusmp
mix.3	7.9	6.60	3.39	lbm-leslie-libq-mcf
mix.4	7.7	8.52	4.57	lbm-libq-mcf-canoeal
mix.5	5.7	8.59	3.27	gcc-mcf-zeusmp-canoeal
mix.6	5.4	5.14	1.98	leslie-omnetpp-zeusmp-canoeal
mix.7	5.2	5.65	3.16	lbm-libq-zeusmp-canoeal
mix.8	5.0	13.99	5.97	astar-libq-milc-omnetpp
mix.9	3.6	3.91	2.06	astar-gcc-milc-zeusmp
mix.10	2.5	3.23	2.06	astar-gcc-Gems-wrf

[Bienia et al. 2008]. All benchmarks are 64-bit binaries, compiled with gcc 4.1.2. To our surprise, most PARSEC benchmarks are computation intensive or have a very small memory footprint. For SPEC CPU2006, we use the reference inputs. We scale the memory footprint of benchmarks when it is possible to change the input parameters to avoid results that are skewed by *mcf*, which has a 1.6GB memory footprint. Table II gives the detailed parameters of the scaling. For *canoeal* from PARSEC, we run it in single-threaded mode and use native input with 940MB memory footprint.

Table III shows the memory footprint (size, in GB), the number of memory read requests per 1000 instructions (Read PKI), and the number of memory write requests per 1000 instructions (Write PKI) of each workload. We choose ten representative multiprogrammed workloads, each containing two copies of four unique benchmarks. We also select four single applications and run them in a rate mode, where eight instances of the same benchmark are concurrently executed. The simulator requires a long warm-up phase to populate the large 2GB DRAM cache. We switch from the warm-up phase to the timing phase after 160 million write references are simulated or after one benchmark completes 30 million write references. The simulation stops when one of the benchmarks completes 60 million write references.

6. RESULTS

The baseline for our evaluation is a hybrid memory system using *Rank-based Page Placement* [Ramos et al. 2011] without compression (RaPP-RW). RaPP-RW uses a tailored MQ algorithm to identify and store *frequently accessed* pages in DRAM. We also evaluated a variation of the MQ algorithm (RaPP-WO), which only identifies and stores *frequently modified* pages in DRAM.

We compare the baseline with four schemes that use compression: COMP applies FPC compression only on the written data; D-COMP is a delta compression scheme that applies FPC on the difference between the new and old data; SD-COMP applies selective compression and PSD-COMP adds predictive compression to SD-COMP (both

Fig. 8. Compression ratios (original size / compressed size) for FPC compression (COMP) and delta compression (D-COMP).

described in Section 3.3). Unless explicitly specified, COMP, D-COMP, SD-COMP, and PSD-COMP all have a 1.5GB compressed region and a 0.5GB uncompressed region.

For SD-COMP and PSD-COMP, we performed a sensitivity study on Gain Threshold (GT, defined in Section 3.3). If GT is small, more lines are delta-compressed, and vice versa. We find optimal performance results when GT is between 48 bytes and 96 bytes. In the evaluations, we choose 64 bytes as the value of GT. For PSD-COMP, we also did a sensitivity study on the value of the recovery probability (RP, defined in Section 3.3). We found that the workloads are not sensitive to the value when RP is small. Consequently, we choose $RP = 1/32$, which means that for every 32 memory writes (lines are not delta-compressed), the memory controller evaluates one write to check whether the corresponding line should be delta-compressed.

6.1. Compression Ratio

First, we evaluate the compressibility of the written data in the benchmarks. Figure 8 shows the compression ratio for FPC compression (COMP) and delta compression (D-COMP). For the twelve studied benchmarks, while COMP achieves an effective average compression of 2.2X, D-COMP can achieve an average of 3.1X. For six out of twelve benchmarks, D-COMP achieves at least 35% more effective capacity than COMP because delta compression converts unmodified data bits to zeros, which are more compressible. The results also show that some floating-point benchmarks, like *GemsFDTD* and *leslie3d*, are difficult to compress, even with delta compression. It is our future work to detect such applications and reduce unnecessary compression overhead by not putting them into the compressed DRAM region.

6.2. PCM Write

Figure 9 shows the number of PCM writes normalized to RaPP-RW baseline. On average, RaPP-WO, COMP, and D-COMP reduce the number of PCM writes by 8.2%, 35.5%, and 64.5%, respectively. It turns out that using only D-COMP more than 90% PCM write requests are absorbed by the DRAM cache for five out of fourteen workloads. This is because D-COMP can achieve a much higher compression ratio than COMP.

For *GemsFDTD*, compression is not effective and cannot reduce the number of PCM writes. For *leslie3d*, the number of PCM writes will be reduced significantly if only modified data are cached in DRAM. In general, SD-COMP results in more PCM writes than D-COMP because SD-COMP only enables delta compression on lines that have a minimum 64-byte storage gain (GT). PSD-COMP has almost the same number of PCM writes as SD-COMP because PSD-COMP only disables delta compression for lines that

Fig. 9. PCM writes normalized to RaPP-RW.

Fig. 10. IPC normalized to RaPP-RW.

are difficult to be delta-compressed, which will not change overall compression ratio. When DRAM compression is enabled, PCM endurance is proportionally improved with the reduction in PCM writes. On average, COMP, D-COMP, SD-COMP, and PSD-COMP achieve 1.6X, 2.8X, 2.2X, and 2.2X improvements in PCM lifetime over RaPP-RW⁴.

6.3. Performance

Figure 10 shows IPC improvement of 15.6% and 22.3% for COMP and D-COMP, respectively, when normalized to RaPP-RW. The difference shows the importance of enabling delta-compressed caching for hybrid memory. SD-COMP and PSD-COMP have similar improvement as D-COMP, but they bring higher power and energy savings (see next section).

Notice that blindly enabling compression will not always improve performance. This is because the performance gain on reducing PCM writes can be offset by the performance loss of increased accessing latency. A typical example is *mix-2*. With COMP, *mix-2* has no reduction on PCM writes but 25% performance penalty over RaPP-WO due to the extra read latency.

⁴Assuming an ideal wear leveling mechanism.

Fig. 11. Memory power consumption.

Fig. 12. System energy consumption.

Our results also show that on average RaPP-WO has similar performance as RaPP-RW. For *mix-2* and *mix-3*, RaPP-RW is better because more frequently accessed data are cached in the DRAM cache. For *leslie_r*, RaPP-WO is better because more frequently modified data are cached in the DRAM cache.

6.4. Power and Energy Consumption

Figure 11 shows memory consumption that indicates that D-COMP has higher power consumption than COMP, mainly due to the extra PCM read traffic for delta compression and the reduction in execution time which increases the rate of memory operations (number of L3 misses in a program is fixed). On average, SD-COMP and PSD-COMP save 1-1.5W on memory power over D-COMP, or approximately 5%.

In addition to considering memory power, we also consider system energy consumption. Although compression increases power consumption, it reduces system total energy consumption due to the reduction in execution time. To be conservative in estimating system power, we modeled the 8-core processor with an average processor power of 25W. The energy consumption is computed by multiplying the execution time with system power. Figure 12 shows that system total energy consumption is reduced 4.6% and 6.6% with COMP and D-COMP, respectively. The reduction further goes to 7.5% and 11.0% with SD-COMP and PSD-COMP, respectively. *Mcf_r* and *mix-1* have large

Fig. 13. Sensitivity analysis of varying PCM write bandwidth on IPC.

Fig. 14. Speedup with different compressed region capacities (normalized to RaPP-RW).

energy savings with compression, which is consistent with their IPC improvements, even though the power is much higher.

6.5. Sensitivity Studies

6.5.1. Impact of PCM Write Bandwidth. To analyze the impact of the PCM write bandwidth on the DRAM compression schemes, we carried out experiments with different PCM write bandwidth values. Figure 13 shows IPC normalized to the RaPP-RW baseline. As expected, the benefits of compression are higher when PCM write bandwidth is lower. When PCM write bandwidth is decreased to 1GB/s, IPC performance benefit of PSD-COMP is increased to 37.3%. When PCM write bandwidth is increased to 2GB/s, IPC performance benefit of PSD-COMP is reduced from 24.4% to 15.7%. The performance gain from the reduction of PCM writes is offset by the increased latency of compressing memory. If PCM write bandwidth is the only bottleneck of the system, D-COMP will always get the best performance, because it maximizes the number of PCM writes that can be reduced. These results show the write bandwidth gap between DRAM and PCM is an important factor to determine whether memory compression should be enabled for hybrid memory.

6.5.2. Impact of Size of Compressed DRAM Region. We evaluate PSD-COMP with different sizes of the compressed region. The total DRAM capacity is fixed at 2GB. From Figure 14, we observe that the optimal partition size highly depends on the workloads. For *mix-5* and *mix-8*, it is better to use 1GB DRAM as compressed memory. For *milc_r*

and *mix-3*, it is better to use 2GB DRAM as compressed memory. To achieve better performance, the memory controller should dynamically adjust the sizes of compressed and uncompressed regions of the DRAM (about 7% potential for IPC improvement), which is an aspect of our future work.

7. RELATED WORK

DRAM/PCM hybrid memory systems have been proposed to combine the high density and low standby power of PCM with the good write performance of DRAM. Qureshi et al. [2009] proposed to add a DRAM buffer between the CPU and the PCM main memory to cache frequently accessed data. Ferreira et al. [2010] proposed PMMA architecture which integrates DRAM page cache with PCM main memory using an improved page replacement policy. Zhang and Li [2009] proposed a 3D-stacked DRAM/PCM hybrid memory architecture which uses an OS-based page migration mechanism to cache hot-modified pages in DRAM. Ramos et al. [2011] proposed a hardware-based page migration mechanism for hybrid memory which puts both frequently modified and frequently read pages into DRAM. Qureshi et al. [2012] proposed PreSET to use spare memory bandwidth to reduce PCM read latency, which should not be triggered when PCM write bandwidth is a bottleneck. Our article considers the use of memory compression to improve DRAM effective capacity and reduce PCM write traffic in hybrid systems which is orthogonal to the preceding works.

Douglis [1993] studied using compression to free up memory pages to reduce paging overhead. Ekman and Stenstrom [2005] proposed a low-latency main memory compression scheme based on the FPC compression algorithm. IBM MXT technology [Tremaine et al. 2001] was introduced as a hardware-based high-performance architecture and is implemented in commercially available chips. Suel and Memon [2002] use delta compression to reduce data traffic for remote file synchronization. Zhang and Li [2009] proposed to apply compression to PCM data. Their proposal is not to improve DRAM effective capacity, but to reduce the number of PCM bits that need to be updated. The studies in Douglis [1993] and Tremaine et al. [2001] focus on DRAM-only systems. Our work explores the unique characteristics of hybrid memory. We propose delta compressed caching which is not discussed by traditional DRAM-only memory compression. We give a detailed example to show how to extend an existing compression architecture to support delta compression for hybrid memory.

8. CONCLUSION

Given that PCM write bandwidth is a major performance bottleneck for hybrid memory, compression has been used to reduce PCM write traffic by increasing DRAM effective capacity. We designed a novel DRAM cache compression scheme that is flexible and tailored to the specific challenges of hybrid memory. Our *delta compression* algorithm stores a compressed version of the modified bits in the updated data in DRAM. We also proposed two extensions to further improve performance and efficiency of our compression scheme, namely *selective compression* and *predictive compression*. Our results demonstrate that compression can significantly improve DRAM cache effective capacity by up to 6.3X and improve PCM lifetime by 2.2X on average. We observed 24.4% IPC improvement from our compression scheme over a noncompression design. We conclude that compression for the DRAM cache can be an effective mechanism for improving the performance and increasing the endurance of hybrid memory. In our future work, we will study adaptive mechanisms for our compression scheme.

REFERENCES

- ALAMELDEEN, A. AND WOOD, D. 2004a. Adaptive cache compression for high-performance processors. In *Proceedings of the 31st International Symposium on Computer Architecture*. 212–223.

- ALAMELDEEN, A. R. AND WOOD, D. A. 2004b. Frequent pattern compression: A significance-based compression scheme for 12 caches. Tech. rep., University of Wisconsin-Madison.
- BIENIA, C., KUMAR, S., SINGH, J. P., AND LI, K. 2008. The parsec benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*.
- CHEN, S., GIBBONS, P. B., AND NATH, S. 2011. Rethinking database algorithms for phase change memory. In *Proceedings of the 5th Biennial Conference on Innovative Data Systems Research*. 21–31.
- CHOI, Y., SONG, I., PARK, M.-H., CHUNG, H., CHANG, S., CHO, B., KIM, J., OH, Y., KWON, D., SUNWOO, J., SHIN, J., RHO, Y., LEE, C., KANG, M. G., LEE, J., KWON, Y., KIM, S., KIM, J., LEE, Y.-J., WANG, Q., CHA, S., AHN, S., HORII, H., LEE, J., KIM, K., JOO, H., LEE, K., LEE, Y.-T., YOO, J., AND JEONG, G. 2012. A 20nm 1.8v 8gb pram with 40mb/s program bandwidth. In *IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*. 46–48.
- CPU2006. Spec cpu2006: <http://www.spec.org/cpu2006/docs/readme1st.html>.
- DAS, R., MISHRA, A., NICOPoulos, C., PARK, D., NARAYANAN, V., IYER, R., YOUSIF, M., AND DAS, C. 2008. Performance and power optimization through data compression in network-on-chip architectures. In *Proceedings of the 14th International Symposium on High-Performance Computer Architecture (HPCA'08)*. 215–225.
- DAVID, H., FALLIN, C., GORBATOV, E., HANE BUTTE, U. R., AND MUTLU, O. 2011. Memory power management via dynamic voltage/frequency scaling. In *Proceedings of the 8th International Conference on Autonomic Computing (ICAC'11)*. 31–40.
- DOUGLIS, F. 1993. The compression cache: Using on-line compression to extend physical memory. In *Proceedings of the 1993 Winter USENIX Conference*. 519–529.
- EKMAN, M. AND STENSTROM, P. 2005. A robust main-memory compression scheme. In *Proceedings of the 32nd International Symposium on Computer Architecture (ISCA'05)*. 74–85.
- FERREIRA, A. P., ZHOU, M., BOCK, S., CHILDERS, B., MELHEM, R., AND MOSSÉ, D. 2010. Increasing pcm main memory lifetime. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE'10)*. 914–919.
- FRANASZEK, P., ROBINSON, J., AND THOMAS, J. 1996. Parallel compression with cooperative dictionary construction. In *Proceedings of the Data Compression Conference*. 200–209.
- HOELZLE, U. AND BARROSO, L. 2009. The datacenter as a computer. www.intel.com.
- INTEL. 2011. Transform mission-critical computing. www.intel.com.
- KIM, I., CHO, S., IM, D., CHO, E., KIM, D., OH, G., AHN, D., PARK, S., NAM, S., MOON, J., AND CHUNG, C. 2010. High performance pram cell scalable to sub-20nm technology with below 4f2 cell size, extendable to dram applications. In *Proceedings of the Symposium on VLSI Technology (VLSIT'10)*. 203–204.
- LEE, B. C., IPEK, E., MUTLU, O., AND BURGER, D. 2009. Architecting phase change memory as a scalable dram alternative. In *Proceedings of the 36th International Symposium on Computer Architecture (ISCA'09)*. 2–13.
- LEFURGY, C., RAJAMANI, K., RAWSON, F., FELTER, W., KISTLER, M., AND KELLER, T. 2003. Energy management for commercial servers. *Comput.* 36, 12, 39–48.
- MAGNUSSON, P., CHRISTENSSON, M., ESKILSON, J., FORSGREN, D., HALLBERG, G., HOGBERG, J., LARSSON, F., MOESTEDT, A., AND WERNER, B. 2002. Simics: A full system simulation platform. *Comput.* 35, 2, 50–58.
- MALLADI, K., NOTHAFT, F., PERIYATHAMBI, K., LEE, B., KOZYRAKIS, C., AND HOROWITZ, M. 2012. Towards energy-proportional datacenter memory with mobile dram. In *Proceedings of the 39th International Symposium on Computer Architecture*. 37–48.
- MOGUL, J. C., ARGOLLO, E., SHAH, M., AND FARABOSCHI, P. 2009. Operating system support for nvm+dram hybrid main memory. In *Proceedings of the 12th Conference on Hot Topics in Operating Systems (HotOS'09)*. 14.
- QURESHI, M., FRANCESCHINI, M., AND LASTRAS-MONTANO, L. 2010a. Improving read performance of phase change memories via write cancellation and write pausing. In *Proceedings of the 16th International Symposium on High-Performance Computer Architecture (HPCA'10)*. 1–11.
- QURESHI, M. K., FRANCESCHINI, M. M., JAGMOHAN, A., AND LASTRAS, L. A. 2012. Preset: Improving performance of phase change memories by exploiting asymmetry in write times. In *Proceedings of the 39th International Symposium on Computer Architecture (ISCA'12)*. 380–391.
- QURESHI, M. K., FRANCESCHINI, M. M., LASTRAS-MONTAÑO, L. A., AND KARIDIS, J. P. 2010b. Morphable memory system: a robust architecture for exploiting multi-level phase change memories. In *Proceedings of the 37th International Symposium on Computer Architecture (ISCA'10)*. 153–162.
- QURESHI, M. K., JALEEL, A., PATT, Y. N., STEELY, S. C., AND EMER, J. 2007. Adaptive insertion policies for high performance caching. In *Proceedings of the 34th International Symposium on Computer Architecture (ISCA'07)*. 381–391.

- QURESHI, M. K., SRINIVASAN, V., AND RIVERS, J. A. 2009. Scalable high performance main memory system using phase-change memory technology. In *Proceedings of the 36th International Symposium on Computer Architecture (ISCA'09)*. 24–33.
- RAMOS, L. E., GORBATOV, E., AND BIANCHINI, R. 2011. Page placement in hybrid memory systems. In *Proceedings of the International Conference on Supercomputing (ICS'11)*. 85–95.
- SAAB, P. 2008. Scaling memcached at facebook, engineering note. Facebook.
- SINHARROY, B., KALLA, R., STARKE, W. J., LE, H. Q., CARGNONI, R., VAN NORSTRAND, J. A., RONCHETTI, B. J., STUECHELI, J., LEENSTRA, J., GUTHRIE, G. L., NGUYEN, D. Q., BLANER, B., MARINO, C. F., RETTER, E., AND WILLIAMS, P. 2011. Ibm power7 multicore server processor. *IBM J. Res. Devel.* 55, 3, 1:1–1:29.
- SUEL, T. AND MEMON, N. 2002. Algorithms for delta compression and remote file synchronization. In *Lossless Compression Handbook*.
- TREMAINE, R. B., FRANASZEK, P. A., ROBINSON, J. T., SCHULZ, C. O., SMITH, T. B., WAZLOWSKI, M. E., AND BLAND, P. M. 2001. Ibm memory expansion technology (mxt). *IBM J. Res. Dev.* 45, 271–285.
- ZHANG, W. AND LI, T. 2009. Exploring phase change memory and 3d die-stacking for power/thermal friendly, fast and durable memory architectures. In *Proceedings of the 18th International Conference on Parallel Architectures and Compilation Techniques*. 101–112.

Received June 2012; revised November 2012; accepted November 2012