# Compressed NVRAM based Memory Systems

***Abstract*—In this paper we examine a 2-level memory system that uses emerging nonvolatile RAM (NVRAM) technologies to build a main memory system with DRAM acting like a cache. We discuss the architectural aspects of such a structure including power-performance tradeoff and how compressed NVRAM storage can help. The analysis indicates that the power benefits of using emerging NVRAM technologies can be very substantial; however, latency sensitive workloads can suffer significant performance impact compared with same sized DRAM only memory. The analysis also shows that compression can provide significant benefits in terms of reducing the NVRAM read/write bandwidth and either enhancing the NVRAM life or reducing the size of NVRAM required.**

## I. Introduction

The substantial increase in computing and memory capacities have fueled growth in application areas such as on-line data mining, activity recognition, large scale data fusion, etc., which require access to huge amounts of data efficiently and often in real-time. Servers configured to handle such workloads typically require large amounts of DRAM. Consequently, DRAM power is already becoming a close second power consumer after CPU's in current platforms and could even move to the first position. The main problem is that the active DRAM power is very high (about 15W for 2GB DIMMS at 100% utilization). Most power management techniques take advantage of the idle periods to reduce the idle power (e.g., fast/slow CKE, self-refresh, etc.) but the active power must be expended in order to get the required work done.

One potential method to address memory power is to use a 2-level memory hierarchy where the higher levels of memory have significantly lower power consumption per GB than DRAM. Some potential technologies for second level memory are slow DRAMs and the emerging non-volatile RAM (NVRAM) technologies such as phase change memory (PCM), magneto-resistive RAM (MRAM), Ferro-electric RAM (FeRAM or FRAM), etc. (For a quick overview of NVRAM technologies see http://en.wikipedia.org/wiki/NVRAM and references to many technologies.) Each of these technologies has its own unique characteristics and implications, and there is no suggestion here that most of them will turn out to be viable for second level memory. Also, while in theory we could have more than 2-levels (e.g., flash memory as the third level), we shall confine ourselves to 2-level memory only in this paper.

In this paper, we examine Phase Change Memory (PCM) as the second level memory technology of choice since it appears to be the most promising among the NVRAM technologies in terms of achievable density and state of current development. A slow DRAM as second level memory is also a good short-term choice but would not provide very substantial power savings. In this paper we consider how the non-volatility of NVRAM can be exploited and how to deal with the unique characteristics of NVRAM technologies, including significantly slower speed than DRAM. We propose compressed storage as a way to deal with NVRAM speed issues and examine performance benefits of compression.

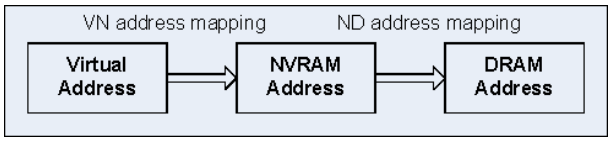


Fig. 1. Address Translation Illustration

The paper is organized as follows. Section II defines the basic 2-level memory architecture. Section III addresses the power-performance tradeoff this basic architecture. Section IV discusses whether compressibility can play into this to improve performance. Section V presents a simple analytic model to estimate the latency and performance impact of compression. It also presents some sample results based on the analysis. Finally section VI concludes the discussion.

## II. Two-level Memory Architecture

The basic architecture considered in this paper is as follows: The NVRAM storage is the de facto "main memory" that the resident OS sees and the DRAM merely becomes a cache, managed underneath the OS. The main distinction between NVRAM used as main memory vs. secondary storage is that the former is far more efficient and does not involve any of the complexities of file-system interfaces. It is well known that file-system interfaces typically introduce a huge amount of overhead and latency. Elimination of this complexity also means that at least portions of NVRAM/DRAM management can be implemented in firmware/HW which further reduces the access latencies and can be significantly more efficient than NVRAM used as fast secondary storage.

NVRAM technologies can generally deliver data more efficiently if the access size is a large block. At the same time, the management of NVRAM to DRAM transfers becomes easier with larger block sizes. As a result, we shall assume NVRAM access size to be a fairly large chunk of a page which we call as "sector", instead of the CPU level cacheline size. Thus, when the CPU misses on a cacheline in DRAM, an entire sector is brought in from NVRAM. In fact, to simplify DRAM management we bring the entire page, except that the requested sector (the "critical sector") is delivered first. On a writeback from DRAM to NVRAM, we write out *only* the dirty sectors of the entire page.

A smaller sector size can reduce the NVRAM read latency; however, it requires more bits to keep track of individual dirty

sectors. From these considerations, a sector size of 1KB (for a 4KB page size) appears to be good compromise.

### A. Address Mapping

In theory, DRAM can be managed like a processor cache (e.g., the standard set associative mechanism backed by CAMs for tag management); however, a page table based approach used in virtual memory management is likely to be lot more flexible and will require less hardware support. With the latter management scheme there are three address spaces, as shown in Fig. 1. The virtual to NVRAM address mapping (VN-AM) is done by OS, and the NVRAM to DRAM address mapping (ND-AM) is done underneath OS. However, they both involve essentially similar approaches. Thus, ND-AM will also use free page list, modified page list, TLB (translation lookaside buffer) for fast lookup, etc. Much of this is routine – the only issue to consider is handling of sectored transfers. On NVRAM read, an entire page must be allocated in DRAM, but the transfer engine needs to ensure that the critical sector is read and transferred first. For efficient writebacks, we need to keep track of dirty sectors, which requires as many bits as number of sectors per page. These bits will be a part of the page table entry for ND-AM scheme.

As a matter of further optimization, it is possible to combine VN-AM and ND-AM into one, but this will have substantial impact on OS and may not be practical. It is also possible to bypass NVRAM and load some of the data in DRAM directly from the secondary storage. In this case, it becomes necessary to tag each page to determine where to load it from and where to save an evicted page from DRAM. Bypassing NVRAM may be useful for heavily used but read-only data, but delving into such issues is beyond the scope of this paper.

### B. Exploiting Nonvolatility

With normal DRAM, protecting a system against crashes requires that all important state information be saved in a stable storage in some way (e.g., periodic checkpointing, write ahead logging, copy on write, etc.) With NVRAM as the main memory, much of the state is automatically safe, and we only need a way of handling dirty sectors in DRAM, which are expected to be much smaller in size than the entire modified main memory. Furthermore, the memory access model for NVRAM (as opposed to file access model) allows for efficient ways of implementing crash resistance. In particular, it may be possible to do periodic checkpointing of an entire application running on the server without significantly affecting the user experience. Since checkpointing techniques are well studied in the literature [9], [10], we do not actually propose a checkpointing mechanism here.

## III. Power vs. Performance of 2-level Memories

The power savings potential of the proposed architecture is driven by the fact that the power consumption of PCM per GB of memory is far lower than that of DRAM. For example, for a 4GB DIMM, the read power is only 0.64W and write power is 1.04W. In contrast, the DRAM read/write power is at least an order of magnitude higher. Furthermore, PCM requires power mostly for the actual read and write operations, the idle power consumption is negligible (about 40mW). It is clear that a system with some amount of NVRAM+DRAM memory will have substantially lower power consumption than a system with equal amount of DRAM memory only. This generally holds in spite of the fact that a 2-level system will put additional power burden on DRAM because of the NVRAM to DRAM data transfers.

Although *a 2-level system will almost always save power, the main issue to examine is the performance impact*. The performance impact obviously depends on three factors: (a) Relative speed of NVRAM compared with DRAM, (b) DRAM hit ratio, and (c) latency sensitivity of the workload. In this section we discuss the simulation modeling of the 2-level system and some results.

### A. Simulation Model

Given the current state of PCM technology, it is not possible to obtain any measurement results. Consequently, we used a detailed simulation model that captures essential aspects of a 2-level memory system discussed here. Instead, the results were produced by the detailed simulation model of a 2-socket (or 2 "node") SMP system called LMPWR [4]. The two sockets are connected via a coherent link and each socket has its own memory controller, each sporting equal amount of DRAM. For a 2-level system, we assume that each socket is augmented with NVRAM memory.

LMPWR was originally developed to study power-performance tradeoffs related to idle power management of both memory and processor-memory link. It has a fairly sophisticated model of link (modeled after Intel's QuickPath™ link) which implements flow control and various transaction types (read, write, remote snoop, remote cache invalidation) along with both reactive and proactive link power control algorithms [5]. It also has a rather sophisticated DDR3 memory model that explicitly models DRAM channels, ranks, banks, interleaving schemes, refresh policies, etc. It also accounts for a variety of latencies inherent in DRAM access including RAS and CAS latencies, asynchronous ODT (on-die termination) overhead, IBT (input bus termination) overhead, rank switch overhead, read-write switch overhead, bank conflicts, channel queuing, transaction blocking for ranks that are being refreshed, etc. The model further implements, in detail, power management of ranks including fast and slow CKE (clock enable) states and switching overheads between them. The reactive and proactive power control algorithms apply to memory ranks as well and admit complexities such as up-front selection between slow and fast CKE or time-based promotion from fast to slow CKE. We believe that the memory and link models are quite sophisticated to accurately reflect power and performance tradeoffs in these subsystems. The models were developed in close collaboration with link and memory experts and are believed to be accurate.

In order to capture the impact of link and memory subsystem latencies on application performance, it is necessary to model the extent to which the HW CPU threads will stall. This part is model somewhat simplistically, but should be adequate for relative performance comparisons. Each HW thread executes for a while (depending on the modeled offered load), and then issues a memory transaction which is placed in the thread-level transaction buffer. This transaction will stay in the buffer until the memory access request (which could be either to the local memory in the current socket or to the remote memory in the other socket) is completed. If the buffer is not full, the thread can still proceed with execution and subsequently generate another memory request. However, if the transaction buffer becomes full, the thread must stall until at least one of the pending memory transactions is done. Note that in case of a memory write, the transaction buffer is released as soon as the transaction is posted to the (local or remote) memory. That is, writes occupy the transaction buffer for shorter periods than reads.

With the above modeling, the size of the transaction buffer represents the *inherent parallelism* of the workload. In reality, the number of transactions that can be issued before the thread must stall varies depending on the phase of the computation. The model allows for this behavior directly: there is a notion of phase duration and at the end of a phase, both a new phase duration and a new buffer size are chosen according to the specified distributions. The net effect of this model is that the latency sensitivity of the workload can be controlled by appropriately choosing the buffer size distribution. In other words, if we know the latency sensitivity of the workload (as is the case with popular benchmarks such as TPC-C), it is possible to dial that into the model.

The basic model was enhanced with NVRAM as the second level memory. This involved a detailed implementation of the virtual memory management scheme as indicated above. In order to properly implement the additional latency, a TLB was also implemented so that if the translation entry is not found in TLB, the overhead of an additional DRAM access (not implemented explicitly) is accounted for. A DRAM can stream a certain maximum number of cachelines with a single CAS command, called *segment size*. A typical value of segment size is 512B. As in actual systems, the LMPWR simulator keeps track of the evicted pages in the modified and free lists, so that if an evicted page is referenced before the page is overwritten with new data, it can be restored without any NVRAM access.

The model can be driven both using analytically generated traffic as well as actual workload traces. The results shown here focus on analytic traffic with a finite horizon Zipf($\alpha = 2.25$) inter-arrival time distribution. (The inter-arrivals times are integers since the entire system operates at the granularity of an underlying "time-slot"). Note that the carried load can be smaller than the offered load due to CPU stalls; therefore, the actual inter-arrival time can be longer than the specified value.

One important issue in generating the traffic analytically is the generation of memory addresses to be referenced. To allow full control over locality of reference, each HW thread independently uses the following algorithm to generate the next address to be referenced:

1) With probability $p_R$, jump to the next page randomly over the entire address space.
2) If the random jump is not taken, with probability $p_V$, jump to a page randomly in the "vicinity" of the current page. The vicinity, denoted $v_f$, is the range around the current page (as a fraction of total address space).
3) If jump in the vicinity is not taken, stay on the same page and move to a random cacheline within that page. (A semi-random model is perhaps more appropriate here, but intra-page behavior does not affect the performance much).

This model is intended to approximate the phase behavior where the random jump outside the vicinity corresponds to (macro-level) phase changes that would essentially require rebuilding the working set in the cache. By varying the model parameters, we can study the impact of NVRAM access latency on the overall performance.

Not surprisingly, the model has a large number (about 200) total parameters. We do not make any attempt to list or explain those parameters. All run-time and compile-time parameters are explained in [4].

### B. Model Calibration

For illustrating results, we choose DDR3-1600 (i.e., DDR3 running at 800 MHz clock). It has a base access latency of about 45ns and full data rate of 12.8 GB/sec per channel. We assume dual-rank 1 GB DIMMs with 8 banks per rank, one DIMM per channel and 2 independent memory channels per socket. Detailed parameters for the memory such as various timings, power values, latencies, etc. were obtained from the DRAM specifications and are not included here to avoid clutter.

A PCM chip comparable with DDR3-1600 DRAM is expected to roughly correspond to DDR3-800 (i.e., DDR3 at 400 MHz clock) with cacheline access time in 90ns range. This is not a huge speed disadvantage per se, but the way NVRAM is used results in significant extra latency: (a) NVRAM latency is in addition to the DRAM latency (upon a DRAM miss), (b) an entire critical sector (1KB) rather than just the critical cacheline (64KB) is retrieved from NVRAM before the requested cacheline can be given to the CPU, and (c) after retrieving the critical sector, the access continues for the rest of the page – some of which may not be needed. Consequently, if accesses to successive cachelines do not show much locality, the system may suffer due to a lot of unnecessary data being loaded into DRAM. Obviously, reducing the access size would help, but would result in more management overhead and less efficient NVRAM accesses.

Although read latency affects performance directly, write latencies can also be problematic. PCM writes (like writes in most NVRAM technologies) are extremely slow – 20-40X slower than reads. Although a write doesn't affect access

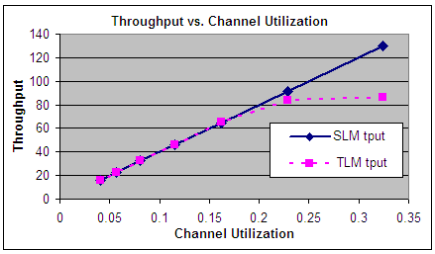


Fig. 2. Throughput vs. channel utilization

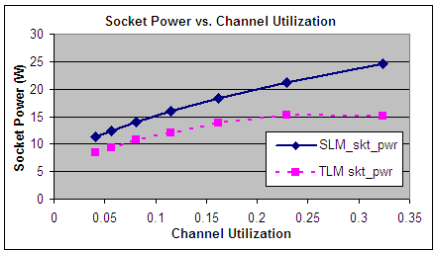


Fig. 3. Power consumption vs. channel utilization

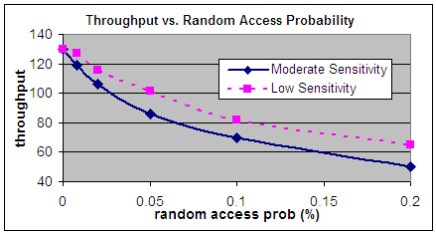


Fig. 4. Impact of decreasing locality of access on throughput

latency directly, a read posted to a PCM bank which has an ongoing write will be stuck there for a long time. (Mechanisms do exist to pause an ongoing write and sneak in a read, but they are complex and have their own associated overheads in terms of managing partial writes). Consequently, the real read access latencies for PCM could be significantly higher. Long write latencies also imply that a larger fraction of DRAM remains occupied with dirty but evicted pages.

We assume a write speed that is effectively 1/40th of read speed. We say "effectively", since PCM writes happen on a bit-by-bit basis and changing 1 to 0 is not the same thing as changing 0 to 1. The simulations do not actually use any contents in the pages; therefore, actual bit patterns are not modeled. Instead, we assume that on the average 1/2 of the bits stay the same and 0 to 1 flips are approximately same as 1 to 0 flips. We assumed 4 GB of NVRAM per socket and 2 GB of DRAM cache. This represents a rather small system (chosen for quick simulation) and a rather large amount of DRAM cache relative to NVRAM.

The workload was analytically generated and used a buffer size corresponding to a fairly moderate latency sensitivity. We assumed that each socket has 8 HW threads (in fact, 4 CPU cores, each with 2 HW threads). To calibrate the locality model, we used nominal values of $p_R = 1/2000$, $p_V = 1/400$, and $v_f = 1/64$. This choice means that a random jump is effected every 2000 accesses, and every 400 accesses, a new page in the vicinity is referenced. This is a rather poor access locality.

### C. Power and Performance Results

Fig. 2 compares the throughput for single level memory (SLM) and two level memory (TLM) systems as a function of memory channel utilization. The units for throughput are not important and can be ignored (only relative values are of interest). The channel utilization only goes up to about 30% since we are modeling a workload where 30% memory channel utilization corresponds to 100% CPU utilization. The important point to note is that when the memory channel is quite reasonably busy (e.g., 30% utilization), the performance impact of the additional latency can be quite severe. At low channel utilizations, the additional latency does not matter as much since the 8 independent HW threads provide enough capacity to support the desired workload throughput.

Fig. 3 is a companion graph to Fig. 2 and shows the socket power consumption vs. memory channel utilization. The power consumption includes only the memory and interconnect, but not the CPU. It is seen that TLM can indeed save a very significant amount of power, i.e., 2.0 to 9.0 Watts per socket from the baseline socket power (excluding CPU) between 11 to 25 Watts. As expected, the power savings go up with channel utilization, and at around the 23% channel utilization point, the power savings are substantial (6.0 Watts) but without a significant impact on throughput (about 9%). However, beyond the 23% utilization point, the power savings increase at the cost of performance degradation.

In order to further examine the issue of throughput degradation, Fig. 4 shows throughput vs. the random access probability $p_R$. Since it is reasonable to expect that $p_V$ will always remain significantly higher than $p_R$, we maintain the same ratio $p_V/p_f = 5$ throughout. Two graphs are shown one for the moderate latency sensitivity considered in Figs. 2 and f:pwr-vs-util, and another one whose latency sensitivity is "low", chosen as 1/4th as much of the moderate case. It is clear that the throughput goes down rapidly at first as the locality deteriorates (and consequently the NVRAM latency and traffic increase). When the locality becomes really poor, the additional degradation is controlled directly by the latency (with the effect of inherent parallelism in the workload essentially having disappeared).

As noted earlier, the nominal value of $p_R = 0.0005$ in Figs 2-4 means that the program branches to a completely random page after 2000 references on the average. In many programs, such branches would occur much less frequently. Consequently, the performance experienced by many applications will lie at the upper part of the curve in Fig. 4. Another point to note is that as we move into an era where power consumption, rather than the performance becomes increasingly important, significant power savings at the cost of, say 10-15% performance degradation, could even be a desirable compromise. Finally, as the NVRAM technologies continue to improve in speed, the performance impact is expected to shrink. In fact, many researchers are beginning to speculate eventual replacement of DRAM with NVRAM technologies, although this might not happen soon.

## IV. Role of Compression

### A. Memory Compressibility and Algorithms

Compressed storage has been explored extensively for the entire storage hierarchy in computer systems including L1 cache, L2 cache, main memory, file cache, etc. [8], [3], [12]. As expected, as we go up the hierarchy the compression/decompression latency requirements become less stringent and better compression becomes possible. Unlike file compression, memory compression must happen independently in the units of "compression blocks" or (CBlock). A larger CBlock size allows better compression but results in higher latency. For reasonable latencies, DRAM compression may require a CBlock size of 4 cachelines (256B) or less and a fast HW implementation. In this regard, NVRAM compression falls in a more comfortable range: CBlock sizes of 1-4KB and compression latencies in 1 $\mu$s range.

One very fast block-based algorithm derived from the traditional LZ77 file compression algorithm is LZO (http://www.oberhumer.com/opensource/lzo/lzodoc.php). LZO really implements many variants; the one referred to here is LZO-1X, which is skewed towards speed rather than compressibility. The data in [14] shows that LZO decompression runs $\tilde{2}$0x faster than zlib (basically gzip) and the compression runs about 10x faster. This is confirmed by the instruction count comparison shown in [11]. In terms of compressibility, LZO is about 20% worse than zlib. Because of these characteristics, LZO is a suitable candidate for NVRAM compression.

Several prior studies have shown that memory contents are generally quite compressible even for reasonably small CBlock sizes [6], [12]. The main reasons for this are preponderance of zeros, small numbers, repeated use of same data in computations, etc. For example, even though programs routinely use 32-bit integers, the stored values mostly fit in 1 or 2 bytes. Some algorithms attempt to exploit this, e.g. see [13], [7]. It is possible to enhance LZO or other algorithms with similar approaches (e.g., by using a tiny dictionary of frequently occurring patterns). However, since the focus of this paper is not compression per se, we do not explore this direction here.

We implemented LZO compression/decompression on a 2.66 GHz Intel Core-2 duo platform and ran it on a number of memory dumps for both client and server workloads. We omit the details here but only note some salient points about these results.

1) Decompression is substantially faster than compression, with the gap generally widening with the block size. This is a highly desirable characteristic since decompression latency is a lot more important than compression latency (read path vs. write path latency).
2) For a given CBlock size, the compression speed generally increases with compressibility, and for a given compressibility, the compression speed decreases with CBlock size.
3) Decompression speed doesn't necessarily track with compressibility or block size, and could vary depending on the precise data involved.

From these results, a CBLOCK size of 2KB appears quite reasonable. The actual implementation of the compression-decompression engine (CDE) can be done in one of the following three ways: (a) Using a HW state machine, (b) Via microcode in a specialized NVRAM controller, and (c) In software running on a dedicated CPU core. It is possible to work out the architectural details of each of these schemes, but we shall not dwell on that aspect. However, we note that the achieved compression/decompression speed would depend strongly on HW vs. SW/FW implementation, and thus it is important to examine the impact of compression/decompression speed on the overall performance. This is taken up in section V-B.

### B. Storage of Compressed Data

The main motivation behind compression is to reduce latency and increase effective bandwidth for NVRAM operations (by reducing the number of bits to be read or written). In other words, we may or may not attempt to reduce the amount of NVRAM actually used to store the compressed data. In the following we discussed both of these possibilities.

*1) Direct Mapping:* In this case, a compressed page is simply stored in the entire 4KB block allocated for the page. This is a simple alternative that does not reduce NVRAM cost, but can still accrue other advantages.

A unique property of all NVRAM technologies is that writes tend to degrade the media slowly. Both PCM and Flash devices support a "wear leveling" mechanism to even out writes across the physical address space. Wear leveling requires a distinction between the "logical" page address provided by the CPU (following the OS-level virtual to physical address translation), and the true "physical" address in the NVRAM. This address mapping allows the writes to be moved to a different page when the wear count of the currently used page goes significantly beyond the current minimum count.

The above wear leveling operates at the level of complete pages. Wear leveling is also a useful concept within a page, since certain "hot" sectors may be written more frequently than others. In case of direct mapping, this becomes an even more pronounced issue. Suppose, for example, that 4KB page compresses to 2KB or less. With direct mapping, the upper half of the block will not see any writes at all. It is easy to fix this by shifting the starting point of the write by one sector on successive writebacks (while treating the block as a ring buffer). Supporting such an *intra-page wear leveling* requires only a few additional bits in the NVRAM metadata and can be done easily. Although this leveling can be done without compression as well, but it is harder and does not provide the 2x or so improvement in NVRAM life as the compressed scheme discussed here.

*2) Compressed Mapping:* In order to reduce the NVRAM storage required, we need an explicit mapping between the uncompressed and compressed NVRAM addreses. The "logical" to "physical" address mapping required by wear leveling

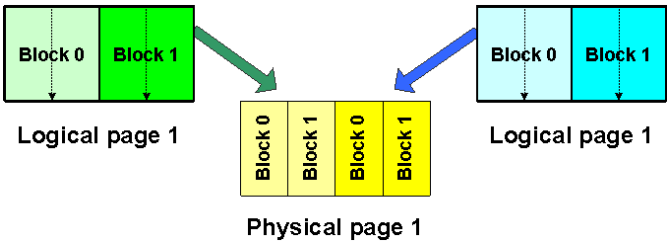


Fig. 5. Illustration of 2 level memory with compression

already provides this functionality. Fig. 5 illustrates this for the case of 1KB sectors, 2KB CBlock and a 4KB page. Here two different logical pages are mapped to the same physical page under the assumption that the compression is 2x or better. If either of the two CBlocks of a logical page experiences a poorer compression, that page will be mapped to a physical page by itself.

We now discuss some details of the NVRAM management for 2:1 target compression situation shown in Fig. 5. Although one could design schemes that allow higher target compression ratios (e.g., 4:1), they are likely to be useful only in some very special cases. Furthermore, an aggressive target compression ratio increases chances of thrashing, out-of-memory conditions and even deadlocks. This happens when the compressibility varies substantially such that during certain periods much of the workload has very poor compressibility. For large memory environments considered here, thrashing is unlikely with a modest 2:1 compression.

In addition to the usual data structures (e.g., free list, modified list, page tables, etc.) the compressed mapping also requires a partially free list (PFL) that maintains partially occupied blocks. For a 2:1 target compression, PFL refers to blocks that store exactly one logical page with compressed size of 1/2 page or less. Blocks from PFL can be used to allocate additional logical pages whose compressed size is 1/2 page or less.

Whenever a single block can hold two compressed logical pages, it is desirable that these pages are "buddies" so that locality of access can be exploited. The buddy page is simply the adjacent page belonging to the 2x page-size block. In case of a writeback, a quick way to check if the buddy page is in PFL is via a hash table that hashes addresses at 8KB level. This hashing may be implemented in hardware for speed.

The various mapping scenarios are now straightforward to handle and are omitted.

## V. Compression Latency Analysis

In the following, we present a simple latency analysis of 2-level memory system using compression in the second level and show some sample results. The analysis here does not focus on the queuing aspects of the problem – it is assumed that by using a suitable model (e.g., analytic, simulation or measurement) we can determine the waiting time (i.e., time from request arrival to start of data delivery). The analysis instead focuses on determining the net impact of pipelined operation of DRAM system, compression/decompression engine and NVRAM system. Assuming the DDR3 type of setup for both DRAM and PCM, it is expected that the BW-latency curve will remain relatively flat except at high channel utilizations (40% or more). Since each channel in a DDR3-1600 DRAM system can deliver 12.8GB/sec, high channel utilizations are rare for most workloads and a constant access time is not unreasonable.

### A. Latency Modeling

We start with some notation where the subscripts have following interpretation: $M$: second level memory (e.g., PCM), $m$: first level memory (DRAM), $r$: read, $w$: write, $b$: base, $c$: compression, $d$: decompression

$S_M$ : Page size. Assumed to be fixed (e.g., 4KB)
$S_s$ : Sector size with values between 512B to $S_M$.
$N$ : Sectors per page = $S_M/S_s$.
$n$ : CBlock size in number of sectors, $1 \leq n \leq N$.
$\alpha(n)$ : Compression fraction (ratio of compressed to original data) when the compression unit is $n$ sectors.
$L_{cb}$ : Base compression latency (i.e., latency to start delivery of first compressed byte).
$L_{db}$ : Base decompression latency (i.e., latency to start delivery of first decompressed byte).
$\eta_c$ : Compression rate in bytes/s excluding startup latency
$\eta_d$ : Decompression rate in bytes/s excluding startup latency
$L_{Mrb}$ : Latency from miss in DRAM to the beginning of delivery of first byte from NVRAM.
$\eta_{Mr}$ : Data read rate in bytes/sec for NVRAM channel.
$L_{Mwb}$ : Latency for writing 0 bytes of data to NVRAM.
$\eta_{Mw}$ : Data write rate in bytes/sec for NVRAM channel.
$L_{mrb}$ : Base latency of a DRAM read (0 bytes).
$\eta_{mr}$ : Data read rate in bytes/sec for DRAM channel.
$S_m$ : Max #bytes that can be streamed on a DRAM read per RAS.

We assume that the NVRAM controller can directly place the critical sector on the link or bus. Then, for an *uncompressed sector*, the overall latency of delivering a sector from NVRAM to the CPU, denoted $R_u$ is given by:

$$R_u = L_{Mrb} + S_s/\eta_{Mr} \quad (1)$$

Note that $R_u$ does not capture the latency of reading the entire page from NVRAM and writing it to DRAM, since only the latency for delivering data to CPU is relevant.

To read a compressed sector, we need to first read compressed block, decompress it and extract the desired sector. The NVRAM data directly goes into the decompressor; therefore, the delivery of compressed block and decompression are pipelined. The overall read latency for a compressed sector $R_c$ is given by

$$R_c = L_{Mrb} + \max\left[\alpha(n)\frac{n\,S_s}{\eta_{Mr}}, L_{db} + \alpha(n)\frac{(n+1)\,S_s}{2\eta_d}\right] \quad (2)$$

where the first term under $\max()$ is time required to deliver the entire compressed CBlock to the decompressor, and the second

term is the time to output 1/2 of the decompressed data. The factor $(n+1)/2$ comes from the assumption that each sector in the block is equally likely to have been requested.

From eqn(2) it is clear that we only need to make decompression fast enough so that the two terms of $\max()$ are equal. That is, the actual $\eta_d$ need not exceed some maximum rate $\eta_d^{\max}$ defined as follows:

$$\eta_d^{\max} = \frac{n+1}{2n}\left[\frac{1}{\eta_{Mr}} - \frac{L_{db}}{\alpha(n)\, n\, S_s}\right]^{-1} \tag{3}$$

Now, let's turn to writes. We assume that the entire page is a unit of writeback, but only the modified sectors need to be actually written out. Let $\beta_c$ denote the probability that a cacheline evicted from DRAM is in modified state. Let $K$ denote the number of cachelines per sector. Then the average number of modified sectors $M_{sp}$ in a page is $N$ times the probability that a sector is dirty.

$$M_{sp} = N[1-(1-\beta_c)^K] \tag{4}$$

NVRAM writes require DRAM reads and we assume the two are pipelined. However, given that NVRAM writes are generally extremely slow, the write time is given by:

$$W_u = L_{mrb} + L_{Mwb} + \frac{S_M}{\eta_{Mw}}[1-(1-\beta_c)^K] \tag{5}$$

With compression, we need to write the entire CBlock if any sector in it is modified. The average number of modified blocks per page is given by:

$$M_{bp} = (N/n)[1-(1-\beta_c)^{nK}] \tag{6}$$

Therefore, the overall write latency $W_c = L_{mrb} + \max[R_m - L_{mrb}, L_{cb} + W_c']$ where $W_c'$ is the maximum of compression and NVRAM writeback latencies, i.e.,

$$W_c' = \max\left[\frac{n\, S_s}{\eta_c} M_{bp}, L_{Mwb} + \frac{\alpha(n)\, n\, S_s}{\eta_{Mw}}\, M_{bp}\right] \tag{7}$$

Given that the DRAM reads are blazingly fast, $W_c$ reduces to $W_c = L_{mrb} + L_{cb} + W_c'$. It then follows that if we do not want to add additional latency (beyond the startup) due to compression, the required compression speed need not exceed the some maximum value $\eta_c^{\max}$ given by:

$$\frac{1}{\eta_c^{\max}} = \frac{L_{Mwb}}{S_M[1-(1-\beta_c)^{nK}]} + \frac{\alpha(n)}{\eta_{Mw}} \tag{8}$$

Since either compression or decompression rates could be the limiting factor, the *optimal rates*, denoted witha superscript "*", is given by:

$$\eta_d^* = \max[\eta_d^{\max}, \eta_c^{\max}/\alpha(n)], \qquad \eta_c^* = \alpha(n)\, \eta_d^* \tag{9}$$

We call $\eta_d^*$ and $\eta_c^*$ as **optimal** decompression and compression rates, respectively, because doing decompression/compression any faster than these will not accrue any further advantages.

### B. Performance Impact Modeling

In section III, we showed detailed simulation results on power consumption and performance. It is also possible to estimate the performance impact of additional latency in the memory path analytically. This is done via a well-known and heavily used technique in platform modeling that involves a couple of basic parameters from low-level cycle accurate simulations. These parameters include: (a) $CPI_{base}$, the cycles per instruction (CPI) for the desired platform on the desired workload when the platform has an infinite amount of cache, (b) Path length ($PL$) or average number instructions per high level transaction, (c) Misses per instruction (MPI) from the highest level cache, and (d) Blocking Factor (BF) which is the fraction of memory access latency that is visible to CPU and thus contributes to CPU stalls. Let $f_{cpu}$ denote the CPU frequency and $R_{tot}$ the total memory access latency. Then the overall CPI ($CPI_{tot}$) that takes into account the memory access latency is given by:

$$CPI_{tot} = CPI_{base} + \text{BF} \times \text{MPI} \times R_{tot} f_{cpu} \tag{10}$$

Now if $\lambda$ is the throughput measure for the workload of interest, then $\lambda$ can be directly related to $CPI_{tot}$ as:

$$\lambda = f_{cpu}/(CPI_{tot}\, PL) \tag{11}$$

To apply this equation to situation with NVRAM memory, let $R_{\text{dram}}$ denote the base DRAM read latency and $R_{\text{read}}$ the NVRAM read latency. If the miss probability out of DRAM cache is $f_{\text{dram}}$, the overall memory access latency is given by $R_{tot} = R_{\text{dram}} + f_{\text{dram}}\, R_{\text{read}}$.

Using these expressions, we can easily project the additional advantage of using compression, provided that we know the basic architectural parameters introduced above. The same equations can also be used for estimating the impact of second level memory; however, it is often essential to run detailed simulations to take into account all the complexities of a 2-level memory system.

| Sector size(B) $S_s$ | Block size $n$ | cmpr ratio $1/\alpha$ | Uncompr read lat. $R_u$ | compr rate $\eta_c^*$ | Relative latency read $R_c/R_u$ | Relative latency write $W_c/W_u$ |
|---|---|---|---|---|---|---|
| 512 | 1 | 1.694 | 170 | 1.461 | 0.807 | 0.670 |
| 512 | 2 | 1.826 | 170 | 1.300 | 1.045 | 0.826 |
| 1024 | 1 | 1.826 | 250 | 1.300 | 0.710 | 0.613 |
| 1024 | 2 | 1.948 | 250 | 1.227 | 1.017 | 0.665 |
| 2048 | 1 | 1.948 | 410 | 1.227 | 0.620 | 0.574 |
| 2048 | 2 | 2.051 | 410 | 1.190 | 0.981 | 0.566 |

TABLE I
READ & WRITE LATENCIES W/ OPTIMAL COMPRESSION

### C. Performance Impact Results

In this section we present some numerical results based on the equations in the last section and the same parameters as for the simulation. To compare performance we use sector sizes from 512 bytes up to the full page size of 4KB and all possible values of CBlock size (same as $n$). The compressibility $\alpha(n)$ was chosen from our compression results on various memory dumps. We chose a workload that led to fairly conservative compressibility of around 2.0.

Table I shows some of the more interesting cases from the analysis for the "optimal" compression case, i.e., where the compression and decompression rates are $\eta_c^*$ and $\eta_d^*$ computed above.

| DRAM miss prob | Performance relative to single level memory | | |
|---|---|---|---|
| | No Compression | Optimal Compression | |
| | | direct mapping | compr. mapping |
| 0.005 | 0.992 | 0.994 | 0.982 |
| 0.01 | 0.984 | 0.988 | 0.965 |
| 0.02 | 0.968 | 0.977 | 0.932 |
| 0.04 | 0.938 | 0.955 | 0.873 |
| 0.08 | 0.789 | 0.841 | 0.775 |

TABLE II
RELATIVE PERFORMANCE OF 2-LEVEL MEMORY

It is seen that a large sector size (e.g., 2KB) significantly increases the read latency w/o any appreciable gain in compressibility. From this perspective, a 2KB sector size is undesirable. A sector size of 512B obviously provides the lowest latency and it still provides a reasonable compression ratio in this case. However, a 1KB sector may be preferable from the management overhead perspective and we choose this for the results shown below.

Now coming to the CBlock size, it is clear that a large $n$ is undesirable from decompression latency perspective; therefore, Table I only shows cases with $n = 1$ and $n = 2$. *These correspond, respectively, to direct and compressed mapping cases discussed earlier*. In either case, the compression leads to a reduction of read and write bandwidths over the uncompressed case – by factors of $1/1.826$ and $1/1.948$ respectively. With $n = 1$, the read and write latencies also go down substantially – to $71.0\%$ and $61.3\%$ respectively. This reduction in latency comes directly from the fact that the amount of data transferred between DRAM and NVRAM is reduced. Now with $n = 2$, there is no reduction in read latency (in fact, about $1.7\%$ net increase) because of the additional overhead of decompression. However, the write latency still goes down to $66.5\%$. The main advantage of $n = 2$ case is, of course, that the amount of NVRAM required is reduced by a factor of 2.

It is seen that in both cases, the optimal compression rate is about 1.3 GB/sec (1.3 and 1.23 respectively, to be precise). Based on an extensive set of experiments (not included here), such a compression rate is currently difficult to achieve with SW based compression on CPU core; however, it should be possible to achieve in near future. Nevertheless it is worth considering what would happen if the compression rate was slower. The numbers are not shown here for brevity, but the general result is an increase in read latencies, but no significant impact on write latencies. The bandwidth reduction – being related to compressibility – is not affected. Thus compression reamins useful even if it is done somewhat slower than the optimal rate.

In conclusion, compression offers the following advantage for 2 level memory systems: (a) Substantial reduction in NVRAM BW requirements, (b) substantial reduction in NVRAM write latency, (c) significant reduction in read latency for $n = 1$, and (d) no read latency reduction but approximately 2x reduction in the required NVRAM space.

Next we consider the impact of compression on application performance by exploiting eqs 10 and 11. We also conservatively assume that the compressed mapping increases the application path length by 25%. Table II shows performance as a function of DRAM miss probability $f_{\mathrm{dram}}$ for SpecWeb 2005 like workload with a sector size of 1KB. The performance is listed related to a single level memory with same overall DRAM size. That is, in all cases, we do get the substantial power savings as discussed in section III-C, but here we are focusing on relative performance which is expected to be lower. Column 2 shows results without NVRAM compression, column 3 with compression and direct mapping ($n = 1$), and column 4 with compression and compressed mapping ($n = 2$).

It is seen that with direct mapping, the compression can bring performance closer to the DRAM only system. Not surprisingly, the advantage is negligible if the workload has strong locality and doesn't miss out of DRAM very much. However, for a more moderate miss rate, such as 4%, an uncompressed system suffers 6.2% performance drop over the DRAM-only system, whereas the compressed system reduces the drop to 4.5%. With compressed mapping, the compression actually reduces the performance due to the overhead of additional memory management. However, performance is not the only issue to consider. With direct mapping the compression allows us to approximately double the NVRAM life, whereas with compression mapping it allows us to halve the NVRAM size. In both cases, compression also reduces the NVRAM bandwidth requirements.

## VI. CONCLUSIONS AND FUTURE WORK

In this paper, we explored the idea of using emerging fast Non-volatile memory (NVRAM) technologies for implementing OS visible memory on a system, with the traditional DRAM "pushed inwards" as a cache to be managed below the OS. We find that such an approach can result in substantial memory power savings but with performance impact that depends on the locality of the workload. For workloads that show good locality, the performance impact can be quite acceptable particularly as we move to an era where power becomes as important as performance. It is also expected that the speedier NVRAM technologies of the future would result in lower performance impact and thus allow a larger class of workloads.

We also investigated the idea of compressing the data stored in NVRAM. Although compression adds some complexities, it can also be useful for reducing the NVRAM bandwidth requirements and write latencies, and at the same time either improve NVRAM life or provide cost savings on NVRAM capacity.

In the future, we intend to do a more detailed analysis of multi-level memory systems for a much larger variety of workloads and configurations using the simulation tool in [4].

## VII. Related Work

The notion of multiple levels in the memory hierarchy is well known – the multiple levels of cache in a typical CPU is an example of this. However, to the best of author's knowledge, the exploration of NVRAM based main memory with DRAM as cache is new. Solid state disks (SSD's) and hybrid storage that exploits Flash technology for much lower latency and higher performance storage has been examined extensively in both industry and academia and is already beginning to appear at commercial scale.

Compression of main memory has been explored by many researchers in the past including studies of compressibility [6], fast compression algorithms [7], [13], hardware implementations [8], [12], and overall schemes [1], [2], [3]. In early 2000's, IBM produced a line of server that utilized memory compression. They designed a specialized chip called pinnacle to support compressed memory management [12]. However, DRAM compression did not gain traction because of small access size (cacheline) and high data rates. We believe that because of page level accesses and lower data rates, NVRAM is a promising technology to exploit the compression.

## References


[1] F. Douglis, "The compression cache: Using on-line compression to extend physical memory", Proc of 1993 Winter USENIX Conference, pages 519-529, San Diego, California, January 1993.

[2] M. Ekman, P. Stenstrom, "A Robust Main-Memory Compression Scheme", ACM Computer Architecture News, v.33 n.2, p.74-85, May 2005

[3] K. Kant, "Performance Evaluation of Memory Compression Alternatives", Proc. of CAECW, Feb 2003, Anaheim, CA.

[4] K. Kant, "LMPWR: A comprehensive link-memory model for studying power performance tradeoffs", at www.kkant.net/download.html

[5] K. Kant and J. Alexander, "Proactive vs. Reactive Idle Power Control", Proc. of Intel Design and Test Technology Conference, Aug 2008.

[6] M. Kjelso, M. Gooch, S. Jones, "Empirical study of memory-data: characteristics and compressibility", IEE Proc. on Computers & Digital Techniques, Vol 145, No 1, Jan. 1998, pp. 63 -67

[7] M. Kjelso, M. Gooch, S. Jones, " Design and performance of a main memory hardware data compressor", Proc. of the 22nd EUROMICRO Conf, Beyond 2000: Hardware and Software Design Strategies, 1995, pp. 423 -430

[8] J-S. Lee, W-K. Hong, & S-D. Kim, "An on-chip cache compression technique to reduce decompression overhead and design complexity", J. of syst. Arch, Vol 46, 2000, pp 1365-1382.

[9] K. Li, J.F. Naughton, J.S. Plank, "Low-Latency, Concurrent Checkpointing for Parallel Programs", IEEE trans. on parallel & dist systems, Vol 5, No 8, Aug 1994, pp 874-879.

[10] E.A. Roman, "Survey of Checkpoint/Restart Implementations", Berkeley Lab Tech Rep. LBNL-54942, July 2002.

[11] C.M. Sadler and M. Martonosi, "Data Compression Algorithms for energy-constrained devices in delay tolerant networks", Proc. of IEEE SenSys'06, Nov 2006, Boulder, CO.

[12] R.B. Tremaine, T.B. Smith, et. al., "Pinnacle: IBM MXT in a memory controller chip", IEEE Micro, March-April 2001, pp 56-68.

[13] L. Yang, H. Lekatsas, R.P. Dick, "High performance operating system controlled memory compression", Proc of design automation conf, July 2006, San-Francisco, CA.

[14] L. Yang, R.P. Dick, et.al., "Online memory compression for embedded systems", to appear in ACM trans on embedded systems.