A Multi-Core Memory Organization for 3-D DRAM as Main Memory

Jared Sherman[1], Krishna Kavi[1], Brandon Potter[1], and Mike Ignatowski[2]

[1] University of North Texas, Denton, Texas, USA,
{JaredSherman, BrandonPotter}@my.unt.edu, Krishna.Kavi@unt.edu,
[2] Advanced Micro Devices, Austin, Texas USA
Mike.Ignatowski@amd.com

Abstract. There is a growing interest in using 3-D DRAM structures and non-volatile memories such as Phase Change Memories (PCM) to both improve access latencies and reduce energy consumption in multi-core systems. These new memory technologies present both opportunities and challenges to computer systems design.
In this paper we address how such memories should be organized to fully benefit from these technologies. We propose to keep 3-D DRAMs as main memory systems, but use non-volatile memories as backing store. In this connection, we view DRAM based main-memory both as a cache memory and as main memory. The cache like addressing allows for fast address translation and better memory allocation among multiple processes. We explore a set of wide-ranging design parameters for page sizes, sub-page sizes, TLB sizes, and sizes of write-buffers.

Keywords: 3-D stacked DRAM, non-volatile memories, phase-change memories, virtual memory management, set-associative indexing

1 Introduction

An emerging DRAM technology, die-stacked DRAM (3-D DRAM), reduces access latencies for large amounts of memory due to its integration with the processor. In the near term, die-stacking may be used as a large last-level cache [1], however as capacity increases it may be an attractive option to use as a system's main memory. We also consider the use of non-volatile solid-state memories, such as phase-change memory (PCM),to replace the disk drive of a traditional computer.

A system where 3-D DRAM is main memory and PCM is the backing store could have significant overall performance benefits due to their improved latencies, however it would be necessary to rethink memory organizations for such a system. For example, 4K or 8K bytes (pages) are used as the units of transfer between DRAM and disks because of large disk latencies. Should we continue to transfer such large amounts of data when the PCM latencies are much smaller? Another issue to consider is the cost of context switches on page faults when the access latencies to PCM are in the same range as the access latencies to DRAM.

Current systems use hierarchical page tables for virtual to physical address translation. With 64-bit virtual addresses, systems need to look up multiple levels of page tables to translate a virtual address into a physical address and the traversal can be relatively slow [2]. If address translation latencies can be minimized, context switches on page faults may be unnecessary. We explore changes to virtual memory management and structure of pages to improve address translation and minimize OS kernel intervention on page faults. When using 3-D DRAMs and PCMs, TLB performance becomes even more critical since delays in address translation can undercut the performance advantages. We attempt to increase TLB performance using large page sizes, say 32KB, 64KB pages, but still transfer only small amounts of data between DRAM and PCM via subpages. An additional challenge is the limited write-endurance of non-volatile memories. We address this by tailoring CMM parameters to consider write-back and using a victim buffer.

This paper makes the following contributions:

- Evaluates new memory organizations for 3-D DRAM as main memory and non-volatile PCM as secondary memory.
- Evaluates treating 3-D DRAM as both main memory and cache for the purpose of speeding up virtual to physical address translation.
- Evaluates the use of large pages with subpages to benefit from small access latencies to PCM, while improving the utilization of TLB and page tables.

Previously we explored our ideas in a single-core environment using Sparc (Serengeti) architecture [3]. In this paper we explore our ideas for multicore systems using x86-64 architecture running Ubuntu.

The remainder of this paper is organized as follows. Section 2 contains our approach to memory organization. Section 3 describes our experimental setup and Section 4 includes results and analysis of the experimental data. Section 5 details future work. Section 6 describes research that is closely related to our work. Section 7 concludes with ending remarks.

2 Memory Organization

Given the physical proximity to the processor, as well as the low access latencies, the 3-D DRAM memory is viewed both as a cache and a primary memory of a computing system. We utilize cache-like addressing using set-associative indexing schemes with main memory. For this reason, we call our main memory a Cache-Main Memory (CMM). Additional design details about our memory architecture and necessary hardware structures can be found in [3].

2.1 Virtual-Physical Address Translation

Our design relies on both page tables and cache-like indexing for addressing DRAM entries. In conventional main memories, virtual addresses are mapped to physical addresses (DRAM frames) using hierarchical page tables. Typically, a virtual address is divided into several parts where each part indexes into a

Fig. 1. Page Table Bit Divisions

different page table, and the page table entries provide addresses of the next level page table.

Some of the page tables can be eliminated if we use a portion of the virtual address as a set index, similar to addressing a set-associative cache. For example, the *efg* portion of the virtual address can be used as a set index into DRAM (see Figure 1). This idea can be viewed in one of two different ways. We can view this as using very large pages, thus *efg* along with page offset will become the offset into a very large page. Or we can also view this as using very large segments with pages, and *efg* constitutes the address of a segment. By using a set-associative index for the segment, we imply associating a working set of DRAM pages with a segment, and the pages within that segment compete for DRAM pages within the working set. The address of the working set for a segment is given by page tables identified by the higher order bits of the virtual address (*abcd* in Figure 1).

This allows us to satisfy the need for physical addresses (e.g., to eliminate address aliases for common or shared pages), but also speeds up address translation. Furthermore, it allows higher memory levels (L1, L2, and Last Level Caches) to use virtual address indexing. This approach is similar in principle to page coloring [4]. By using large segments with reasonably sized pages, we maintain fine-grained protection at page level. We still use a translation look-aside buffer (TLB) that speeds up address translation.

Page Structure. The CMM consists of large pages which are divided into subpages. Large pages are used to reduce the number of TLB entries while subpages allow us to transfer small chunks of data between DRAM and backing store. It will be necessary to keep track of valid and dirty subpages within a page. A subpage bitmap can be used to track valid and dirty status of subpages within a page.

Page Lookup. Pages are located in the CMM using the virtual address, in combination with an address space identifier (ASID). In addition to using set-associative mapping of addresses to CMM pages, we use a specialized TLB-like structure to accelerate this process. Our TLB is fully-associative and contains a small number of entries. In addition, it also contains the subpage bitmaps for its resident pages which may be rapidly accessed to determine a hit or miss. A page may still reside within the CMM even if there is no TLB entry for the page. In such cases, the CMM Tag Structure is searched using set-associative indexing. Bitmaps are not stored with tags but stored in the CMM page itself as header information. Only the actual CMM pages reside in the 3-D DRAM layers. The other structures can reside in the memory controller or MMU, possibly on the same layer as the processing cores.

Mix	Bench1	Bench2	Bench3	Bench4	Total (MB)
Small1	Gobmk	Hmmer	H264Ref	Gromacs	41.7
Small2	Gamess	Sphinx3	Tonto	Namd	27.6
Medium1	Sjeng	Libquantum	Leslie3d	Astar	191.6
Medium2	Omnetpp	Astar	Calculix	Gcc	139.9
Large1	Milc	Wrf	Zeusmp	Soplex	865.9
Large2	Zeusmp	Leslie3d	Gcc	CactusADM	718.3
VeryLarge1	GemsFDTD	Mcf	Bwaves	CactusADM	2262.2
VeryLarge2	Mcf	Zeusmp	Milc	Bwaves	1656.0

Table 1. Benchmark mixes. Total (MB) represents the combined working set size of all applications in a given mix.

3 Experimental Setup

We use Simics [5], a robust full-system simulator. Simics includes a timing module called G-Cache to gather cycle-accurate timing data. We modified G-Cache in order to simulate our 3-D DRAM CMM system. The target platform we use Ubuntu 8.04 running on a x86-64 Hammer processor running at 3GHz. We evaluate our design using mixes of SPEC CPU2006 benchmarks which are shown in Table 1, which are comparable to those of [6].

Each mix contains four benchmarks. Though we ran simulations for 2, 4 and 8 cores, due to space restrictions and the similarity of results we only include 4-core results here. Processes are statically bound to a core so we could more easily track how many instructions each benchmark executed. We are currently investigating the use of better process tracking so that benchmarks can be tracked regardless of the core on which it is scheduled.

Benchmark mixes are grouped by working set sizes, a measure of how much (total) memory is being actively used by an application. This is different from the amount of address space the OS has reserved for an application and resident set size which is how much physical memory an application uses. Working set size has been often considered by researchers looking at cache misses, TLB misses and page faults, and the working set sizes we use were determined by [7]. We felt that grouping our benchmarks by this measure would allow us to see how different applications with similar working set sizes might operate in our CMM design, and also see how benchmarks with small to very large working sets perform. We set the CMM size to 256 MB, so that we may be able to see the effects of heavy workloads up to ten times greater than the size of CMM. In time, 3-D DRAM modules will be able to store 8 to 32 GB and perhaps more, and we are confident that the trends which we report here will scale appropriately for larger CMMs.

Latencies and other fixed L1, L2 and CMM parameters used in our study are the same as reported previously in [3], except that here we are dealing with multiple cores and we use 32 KB L1s and 256KB L2s per core. These sizes are chosen to be representative of current L1 and L2 caches, and these values are fixed throughout our experiments. Design parameters we varied in experiments will be explained in the following section. The caches and CMM are warmed for 300 million user-level transactions to core-0, after which simulation data is gathered for all cores during the time taken for 500 million benchmark instructions executed by core-0. Several parameters are explored here including number of

memory banks, associativity, TLB size, page size, subpage size, number of subpages to pre-fetch and size of victim buffers.

4 Results and Analysis

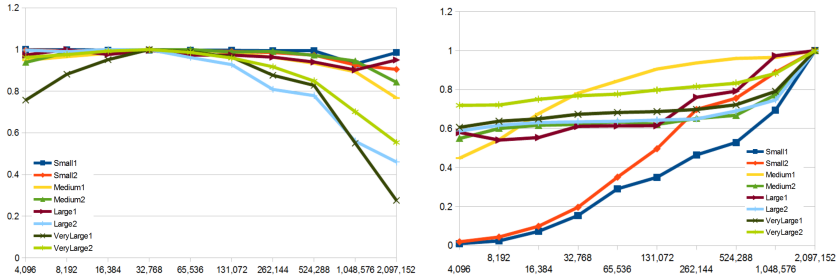
In this section we detail the results of our experiments. The most common metrics used for comparing different design parameters are *relative IPC* and relative amount of data copied back. IPC, or instructions per cycle, is a measure of how many instructions are executed per clock cycle, and data copied back measures the amount of data that is written back from CMM to PCM backing store. The graphs that follow, unless otherwise specified, use relative numbers ranging between 0 and 1, and display the IPC relative to the best IPC possible in that experiment. This is intended to provide a clearer picture as one can view the range of performance possible for different values of the design parameter explored; than using raw numbers, since we use a wide ranging workloads.

Page Size. Larger pages are beneficial in that TLB and page tables can be smaller, while having small subpages grants us the ability to have smaller, quicker transfers between CMM and PCM, eliminating the excessive CPU stalls on a page/subpage miss. Page size affects the number of CMM entries and larger pages may cause more conflicts for (fewer) CMM entries.

We explored page sizes between 4KB and 2MB. In terms of IPC, Figure 2(a) shows that pages between 4KB to 64KB show minor performance changes, with a marginal peak at 32KB. IPC declines sharply after 128KB, particularly for benchmarks with large working sets; a result of having fewer pages that can reside in memory at any given time. Further, Figure 2(b) shows a general trend where page size increases leads to an increase in write-backs. Smaller benchmark mixes display more sensitivity to page size variation, and a linear increase in the amount of data written back with page sizes. *Of particular interest, for small working sets, CMM is not fully utilized and thus in case of 4KB pages, no data is written back, because page conflicts are eliminated.* However such page sizes require very large TLBs; thus we believe a 32KB page provides a reasonable trade-off between the amount of data written back and the cost of memory structures (e.g., TLB).

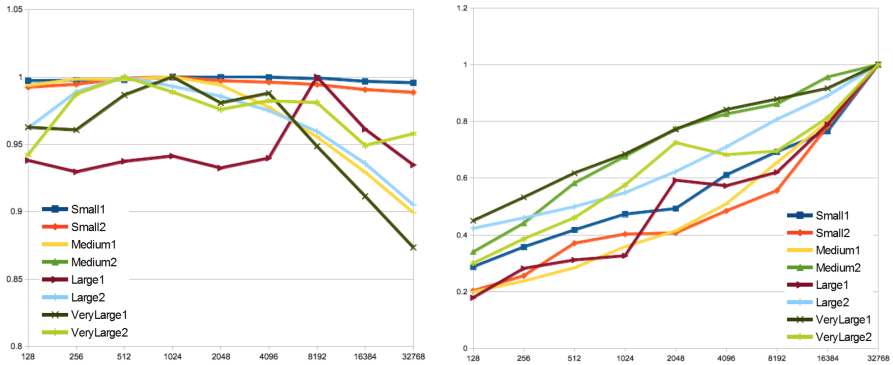
Subpage Size. When using larger pages (32KB), since the new technologies present very low data transfer latencies, we explore the use of subpages within a page as the unit of data transferred between DRAM and backing store (i.e., PCM). Subpage size is an important consideration in our design in both hardware cost and performance benefits. Hardware cost will increase as subpages get smaller because subpages must be tracked by a bitmap. Here we fix the page size at 32KB.

Figures 3(a) and 3(b) display the relative values for IPC and amount of data written back to PCM respectively for different subpage sizes. In most cases, IPC tends to peak at 512 or 1024 byte subpages, with the very large working set benchmark mixes displaying the most volatility when moving outside that range. The small working set mixes appear somewhat impervious to subpage size



(a) Page Size: Relative IPC (b) Page Size: Write-backs

Fig. 2. Page Size: subfigures (a) and (b) use a 0 to 1 relative scale. 4KB to 2MB pages are evaluated. Subpage size is set at 512 bytes.



(a) Subpage Size: Relative IPC (b) Subpage Size: Write-backs

Fig. 3. Subpage Size: subfigures (a) and (b) use a 0 to 1 relative scale. Subpage sizes from 128B-32KB. Simulated page size is 32KB, therefore 32KB subpage effectively represents no subpages.

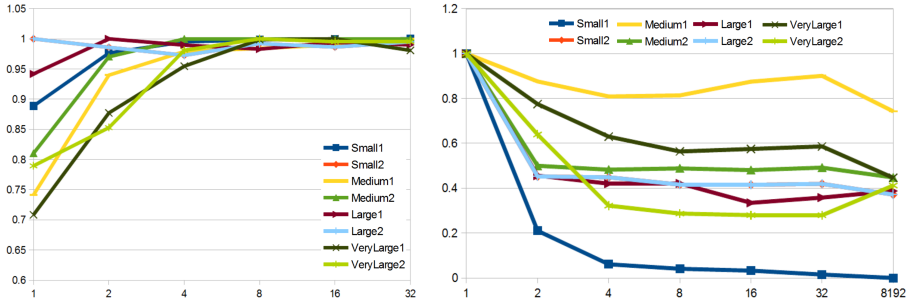
variations due to very few page faults and underutilization of CMM as discussed previously.

For write-backs, subpage size is easily the most important parameter to consider. As can be seen from Figure 3(b), all workloads display nearly a linear relationship between subpage size and amount of data copied back to PCM. On average, a 32KB page with 32KB subpages writes back 3 times more data to PCM as a 32KB page with 128 byte subpages. This is expected since we only write back dirty subpages to PCM. Taking all things into consideration, we would conservatively recommend a 512 byte subpage size.

Associativity. As described in Section 2, we use both page tables and set-associative style indexing to address the CMM. In this section we describe the effect of associativity within our CMM. Figure 4(a) shows that increasing asso-

ciativity appears to improve IPC. Performance gains are due to reduced conflict misses as associativity increases. Beyond 8-way, however, performance gains are negligible. This phenomenon has been reported for very large caches, and our findings here corroborate previous studies [8].

Figure 4(b) shows that conflict reduction in CMM due to increased associativity can significantly reduce the amount of data copied back; but insignificant savings are achieved beyond 8-way.



(a) Associativity: Relative IPC

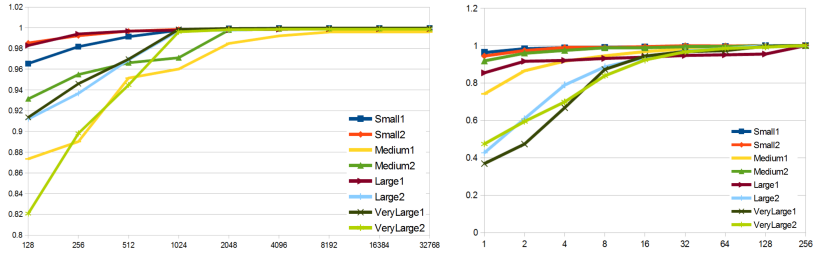
(b) Associativity: Write-backs

Fig. 4. Associativity: subfigures (a) and (b) use a 0 to 1 relative scale. Associativities from 1 to 32 are evaluated.

Translation Look-aside Buffers (TLB). As described in Section 2, we use TLBs to speedup address translation and verify if the requested subpage is currently available in CMM using subpage bitmaps stored in TLB. Our TLB is a fully-associative cache of CMM page addresses and bitmaps for current pages tracked in TLB. We evaluated the performance impacts of using different sizes for TLBs from 128 entries to 32,768 (which represents one entry in TLB for each of CMM page).

We felt that TLB hit rate was the best basis for comparison. It can be clearly seen from Figure 5(a) that hit rate appears to max out at 1024 to 2048 entries, depending on the benchmark. Since the TLB contains a mapping between virtual-physical addresses, the number of TLB entries can be considered in terms of total coverage of CMM, representing the percentage of total CMM pages for which the physical address and bit map can be stored in TLB at any given time. Our experiments indicate that a 3% to 6% coverage is sufficient for TLB sizes. Even at these coverages, a TLB for 32GB CMM will be very large, and it may be necessary to use set-associative TLBs instead of fully associative TLBs. It should be noted, however, TLB's will be even larger when using 4K or 8K pages.

Pre-fetch. Pre-fetching has been shown to be very effective in reducing cache miss rates, however it can also substantially increase memory traffic [9][10]. At the CMM to PCM level, we expect less contention than at the L1 or L2 cache



(a) TLB: Hit Rate (b) Banks: Relative IPC

Fig. 5. TLB: (a) shows actual hit rates for varying TLB sizes. Sizes from 128 to 32,768 entries are explored. In these simulations CMM has a fixed size of 256MB and 32KB pages, which give 32,768 total pages that may reside in CMM, and therefore that many pages may be addressed by the TLB.
Banks: (b) shows relative IPC for varying bank sizes. Comparison on a 0 to 1 relative scale. 3-D DRAM with 1 to 256 banks are evaluated.

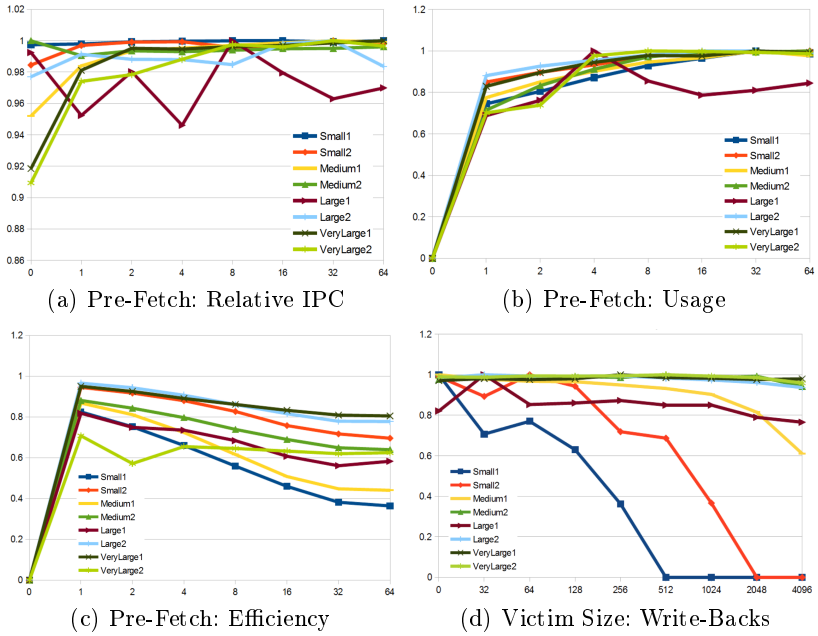
level. Therefore, we may direct the memory controller to use idle CMM cycles to pre-fetch additional subpages from backing store on a memory request. Pre-fetch may occasionally delay requests for new pages or subpages, since the current pre-fetched subpage must complete before a new request can be issued.

We explored pre-fetch policies of zero (no pre-fetch), up to 64 which is the total number of subpages in a page (with 32KB pages and 512 byte subpages). Figure 6(a) shows that the reduction in miss rates as a result of the spatial locality of pre-fetched pages account for a greater performance increase than the performance loss attributable to delaying demand fetches. However for large working set benchmark mix Large1, the applications exhibit poorer spatial localities and thus pre-fetching can actually be detrimental to performance.

Figure 6(b) and Figure 6(c) display subpage usage and efficiency respectively. Usage is a generally increasing trend because as more subpages are pre-fetched, more of those subpages are used, generating fewer demand requests. Efficiency is the percentage of prefetched subpages that were actually used. It declines as more subpages are prefetched. While we did not explicitly perform energy consumption calculations here, this figure captures the essence of energy efficiency of pre-fetching as unused subpage contribute to wasted power. Our results indicate that pre-fetching just one subpage provides the best tradeoff. We plan to extend these studies by using adaptive pre-fetching techniques similar to those proposed in [11].

Victim Buffer. Victim caches [12] have traditionally been used to decrease conflict misses in low associativity caches. We adapt this concept to alleviate the conflict misses in CMM, causing write-back to PCM. Our victim buffer can also be viewed similar to DRAM caches or write buffers used in [13][14].

Figure 6(d) shows that data copied back becomes zero for smaller benchmarks as victim size is increased. Here the CMM, along with the victim buffer, is able



(a) Pre-Fetch: Relative IPC (b) Pre-Fetch: Usage

(c) Pre-Fetch: Efficiency (d) Victim Size: Write-Backs

Fig. 6. Pre-Fetch: (a) and (b) use a 0 to 1 relative scale. For (a), the X-axis represents the maximum number of subsequent subpages to be pre-fetched. Data simulates a CMM using 32KB pages with 512 byte subpages. For (b), a 1 would represent the pre-fetch policy had the most used subpages among all policies for a given benchmark. (c) shows the percentage of used subpages for a given prefetch policy where a 1 would mean every pre-fetched subpage was used.
Victim Cache: (d) displays write-back data for varying victim cache sizes. Comparison on a 0 to 1 relative scale. Victim cache sizes from 32 to 4,096 entries are explored along with having no victim cache at all (0). Page size is fixed at 32KB.

to contain the entire address space of the benchmarks. However, data copied back for larger workloads is largely unaffected by victim buffer size. Much larger victim buffer sizes may have reduced write-backs even for large benchmarks, but this will add to the cost of the hardware.

Banks. Large 3-D DRAM are designed using multiple banks, each with a dedicated connection to the common bus. We assign addresses to different banks statically and simulate contention to banks.

Demand requests may interrupt pre-fetches and copybacks to the same bank, after the currently transferring subpage has finished, but may not interfere with copybacks or pre-fetches to different banks. Thus multiple prefetches and copybacks may occur simultaneously. A new demand request to a bank must wait for prior demand requests to finish. We explore the impact on performance by varying the number of banks.

Figure 5(b) shows that increasing the number of banks provides an IPC boost. However, the figure also shows diminishing returns beyond 16 or 32 banks. We feel that this behavior is due to our use of (mostly) virtual addresses to DRAM and our static assignment of addresses to banks. This means that most applications's addresses map to very few banks. However, with larger benchmarks, more cores, the virtual to physical address translation, and more dynamic distribution of addresses to banks would lead to better utilization of banks and scalable performance improvements with more banks. We will explore these issues in our future work.

5 Future Work

We are currently exploring in detail the implications of our ideas in terms of changes needed with OS virtual memory management. We will evaluate the advantages of our virtual memory management in the context of benchmarks with very large working sets, in order to fully exercise memory systems with 8GB-32GB. For this purpose, we will use server class and cloud application suites. We are currently working on models to evaluate energy consumed by applications when using our designs.

In the interest of further reducing the amount of data copied back, we will explore ideas (described in Section 6) such as eliminating dead writes, partial or delayed writes, flip-writes and other techniques that rely on compiler analysis.

6 Related Work

The ideas presented in Qureshi, et. al [14] are closely related. In that work, they explored the use of PCM as primary memory in place of DRAM. They compare the use of conventional organizations using very large DRAM systems (4GB-32GB), consider the merits of PCM vs DRAM, and subsequently examine a hybrid configuration that uses a 1GB DRAM as a cache to 32GB PCM. The purpose of their DRAM cache is to minimize writes to PCM. Their studies show that a hybrid configuration is a fair trade-off between scalability and latency. Our approach differs from this study as we use DRAM as primary memory and PCM as secondary memory, replacing magnetic disk drives.

The study by Lee, et. al [13] is similar to that of Qureshi [14], in that they also use PCM as a replacement to DRAM as the main memory system. Like Qureshi, Lee, et. al use DRAM based buffers between LLC and PCM, however Lee studies the use of multiple DRAM buffers instead of a single DRAM based cache. As an example, the authors show improved write endurance using four 512-byte buffers instead of using a single 2048-byte buffer. For our own purposes, we view the use of multiple buffers as an associative DRAM cache. Lee also investigates another technique for reducing writes, called partial writes or delayed writes. This compares data evicted from caches with the contents of PCM row buffers and writes only the data that is actually modified (it is possible for modified data to return to its original value). The study explores different granularities

for these comparisons and, as expected, finer granularity leads to fewer bytes being written back.

Other studies have explored techniques to minimize write-backs. Cho, et al [15], writes either a modified, i.e. dirty, value or its complement, depending on the number of bits that will be written back. More recently, Bock, et al [16], explore how to eliminate writing modified data that is no longer useful. Useless data stems from freed objects, such as heap deallocations, and from popping stack frames which make previously stacked data inaccessible. We refer to these as dead writes.

Techniques such as these, including delayed writes [13], flip writes [15], and dead write elimination [16] are complimentary to our efforts. We explore them as a part of our on-going research.

Page coloring, Kessler, et. al [4], is similar to our page indexing schemes. Page coloring speeds up the virtual-physical address translations such that there are negligible differences between the virtual and physical addresses for large caches.

7 Conclusion

In this paper we investigated the use of 3-D DRAM as main memory and phase-change memory (PCM) as secondary memory. These technologies hold significant promise in reducing memory access latencies. PCM devices can also reduce the amount of energy consumed. However, data stored in non-volatile devices such as PCM can only be modified a limited number of times. Thus, it becomes critical to minimize the amount of data modified or copied back to such devices.

Our goal in this work is to study how to benefit from the faster accesses to 3-D DRAM and PCM devices. For this purpose, we describe a new memory organization that views DRAM both as a conventional main memory and a cache: the cache view allows us to use set-associative addresses to locate data in DRAM, while the memory view allows us to permit the use of traditional virtual memory management using page tables, eliminating aliases resulting from virtual addresses and enforcing protection at page level. Our approach can significantly eliminate the need for page table walks and context switches on page faults. We investigated the use of large pages with subpages, where the unit of transfer is a subpage between 3-D DRAM and PCM devices.

We reported the results of our experimental evaluation of a wide range of design parameters for page sizes, subpage sizes, TLB sizes, set-associativities, victim cache sizes, number of subpages to pre-fetch, in a multi-core environment. We reported our analyses, indicating preferred values for configuration parameters. With the right parameter choices, we show that a Cache-Main Memory organization, implemented in future technologies, can outperform contemporary DRAM memory configurations.

Acknowledgements. This project is supported in part by the NSF Net-Centric Industry/University Cooperative Research Center and a unrestricted research grant from the Advanced Micro Devices.

References

1. Loh, G.H., Hill, M.D.: Efficiently enabling conventional block sizes for very large die-stacked dram caches. In: MICRO. (2011) 454–464
2. Barr, T.W., Cox, A.L., Rixner, S.: Translation caching: skip, don't walk (the page table). SIGARCH Comput. Archit. News **38**(3) (June 2010) 48–59
3. Fawibe, A., Sherman, J., Kavi, K., Ignatowski, M., Mayhew, D.: New memory organizations for 3d dram and pcms. In: ARCS. (2012) 200–211
4. Kessler, R.E., Hill, M.D.: Page placement algorithms for large real-indexed caches. ACM Trans. Comput. Syst. **10** (November 1992) 338–359
5. Magnusson, P.S., Christensson, M., Eskilson, J., Forsgren, D., Hållberg, G., Högberg, J., Larsson, F., Moestedt, A., Werner, B.: Simics: A full system simulation platform. Computer **35**(2) (February 2002) 50–58
6. Loh, G.: 3d-stacked memory architectures for multi-core processors. In: Computer Architecture, 2008. ISCA '08. 35th International Symposium on. (june 2008) 453 –464
7. Gove, D.: Cpu2006 working set size. SIGARCH Comput. Archit. News **35**(1) (March 2007) 90–96
8. Dube, P., Zhang, L., Daly, D., Bivens, A.: Performance of large low-associativity caches. SIGMETRICS Perform. Eval. Rev. **37**(4) (March 2010) 11–18
9. Callahan, D., Kennedy, K., Porterfield, A.: Software prefetching. In: Proceedings of the fourth international conference on Architectural support for programming languages and operating systems. ASPLOS-IV, New York, NY, USA, ACM (1991) 40–52
10. Porterfield, A.K.: Software methods for improvement of cache performance on supercomputer applications. PhD thesis, Rice University, Houston, TX, USA (1989) AAI9012855.
11. Ebrahimi, E., Lee, C.J., Mutlu, O., Patt, Y.N.: Prefetch-aware shared resource management for multi-core systems. SIGARCH Comput. Archit. News **39**(3) (June 2011) 141–152
12. Jouppi, N.P.: Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. In: Proceedings of the 17th annual international symposium on Computer Architecture. ISCA '90, New York, NY, USA, ACM (1990) 364–373
13. Lee, B.C., Ipek, E., Mutlu, O., Burger, D.: Architecting phase change memory as a scalable dram alternative. SIGARCH Comput. Archit. News **37**(3) (June 2009) 2–13
14. Qureshi, M.K., Srinivasan, V., Rivers, J.A.: Scalable high performance main memory system using phase-change memory technology. In: Proceedings of the 36th annual international symposium on Computer architecture. ISCA '09, New York, NY, USA, ACM (2009) 24–33
15. Cho, S., Lee, H.: Flip-n-write: A simple deterministic technique to improve pram write performance, energy and endurance. In: Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on. (dec. 2009) 347 –357
16. Bock, S., Childers, B., Melhem, R., MossÃĪ and, D., Zhang, Y.: Analyzing the impact of useless write-backs on the endurance and energy consumption of pcm main memory. In: Performance Analysis of Systems and Software (ISPASS), 2011 IEEE International Symposium on. (april 2011) 56 –65