New Memory Organizations For 3D DRAM and PCMs

Ademola Fawibe[1], Jared Sherman[1], Krishna Kavi[1] Mike Ignatowski[2], and David Mayhew[2]

[1] University of North Texas,
AemolaFawibe@my.unt.edu, JaredSherman@my.unt.edu, Krishna.Kavi@unt.edu,
[2] Advanced Micro Devices,
Mike.Ignatowski@amd.com, David.Mayhew@amd.com

Abstract. The memory wall (the gap between processing and storage speeds) remains a concern to computer systems designers. Caches have played a key role in hiding the performance gap by keeping recently accessed information in fast memories closer to the processor. Multi and many core systems are placing severe demands on caches, exacerbating the performance disparity between memory and processors. New memory technologies including 3D stacked DRAMs, solid state disks (SSDs) such as those built using flash technologies and phase change memories (PCM) may alleviate the problem: 3D DRAMs and SSDs present lower latencies than conventional, off-chip DRAMs and magnetic disk drives. However these technologies force us to rethink how address spaces should be organized into pages and how virtual addresses should be translated into physical pages. In this paper, we present some preliminary ideas in this connection, and evaluate these new organizations using SPEC CPU2006 benchmarks.

Keywords: 3D-Stacked DRAM, PCM, Flash Memory, Victim Cache

1 Introduction

A typical computer system uses multiple levels of memory hierarchy to hide the speed gap between processor and storage subsystems. A multicore based system may consist of private L1 (data and instruction), L2 caches and shared L3 caches with DRAM based main memory connected to one or more multi-core sockets via high speed network and magnetic disk drives as secondary storage systems. Current memory organization is based on the access latency of these technologies. Newer technologies and implementations for main memories (e.g., 3D stacked DRAM) and secondary memories (e.g., Flash or PCM based solid state disks) offer substantially different latencies forcing us to rethink memory organizations. We use the term Flash Replacement Technology Drive (FRTD) to refer to future non-volatile memories that may include PCM or other flash-like technologies. As an illustration, Figure 1 shows the range of access times to different memories in typical memory hierarchies[10].

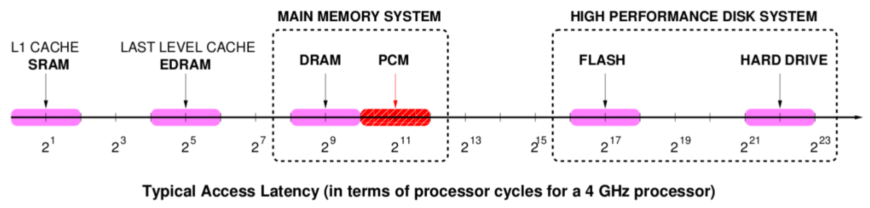

Fig. 1. Access Latencies for Different Memory Technologies

These latencies are very conservative: we expect 3D stacked DRAMs with 30ns latencies and PCM with 100ns read and 1000ns write latencies. These short latencies, which are several orders of magnitude smaller than those for disk drives, offer new opportunities and challenges to computer architects. For example, should we consider transferring only small amounts of data between secondary and primary memories on each request since each request incurs only a few hundred nanoseconds in latency? This also implies the need to eliminate OS intervention or context switching processes on page faults. Since we anticipate very large DRAM memories (as much as 16-32 GBytes), should we still use 4KB or 8KB pages, or use very large pages, say 64KB, 128KB or even 512KB page, so that the sizes of page tables and TLBs can be smaller? In other words, should we consider pages with subpages and only transfer subpages between primary and secondary memories? In addition to benefiting from low latencies of these new technologies, it is necessary to remember that SSDs can only be written a finite number of times. As an example, PCMs have an endurance in the range of 10^6 to 10^8 writes[10]. Thus new memory organizations should explore the use of write buffers to minimize the amount and frequency of data written to these devices. In this paper we will explore some of these design issues.

The rest of the paper is organized as follows: Section 2 details our approach, our experimental framework and methodology is outlined in Section 3, results and analysis are presented in Section 4, related research is covered in Section 5 and we present our conclusion in Section 6.

2 Our Approach

We present a design which utilizes the previously discussed technologies to produce a memory hierarchy for future multicore processors. Figure 2 displays an overview of our proposed memory hierarchy. Each component is addressed in detail in the following subsections.

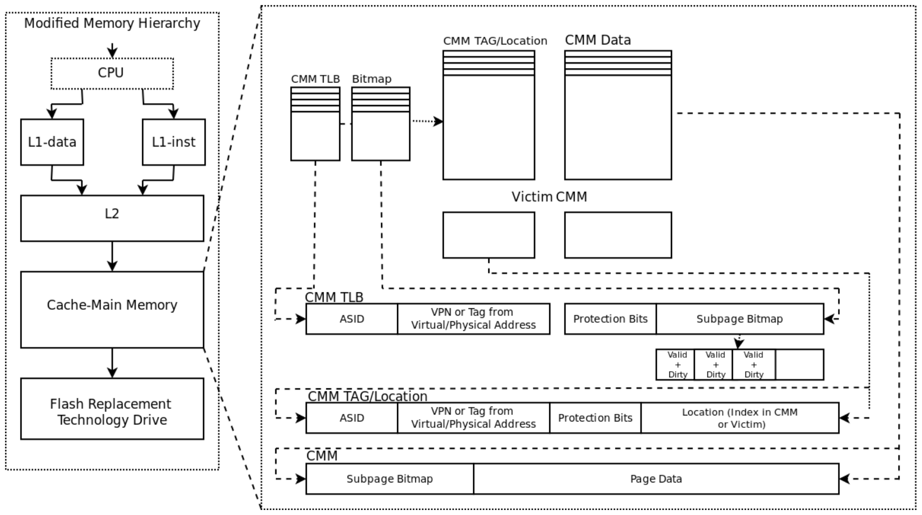

Fig. 2. New Memory Hierarchy with CMM and FRTD

2.1 Caches & Addressing

Our design utilizes virtually indexed caches. Note that one can envision virtual address space as being divided into segments and segments into pages. It is then possible to allocate a fixed number of physical pages to a segment and utilize cache like set-associative mapping to translate virtual page addresses into physical pages similar to *page coloring* [5]. Page coloring allows the use of virtual addresses to index large L2 or L3 caches. While conventional virtual memory systems can still be used within our studies, in this paper we will only use virtual addresses.

2.2 CMM

Given the physical proximity to the processor, as well as the low access latencies, the 3D stacked DRAM memory may be viewed both as a cache and a primary memory of a computing system. We will utilize cache-like addressing using set-associative indexing schemes with our main memory. For this reason, we will call this main memory a CMM (Cache-Main Memory). Figure 2 illustrates the components of the CMM.

Page Structure. The memory structure consists of large pages which are divided into subpages. A subpage is the smallest unit of data that is transferred between the CMM and the backing store. It will be necessary to keep track of valid and dirty subpages within a page. To accomplish this, a subpage bitmap tracking the valid and dirty status for each subpage within a page is kept.

Page Lookup. The virtual address from the processor, in combination with an address space identifier (ASID) is used to locate a CMM entry (page). While cache-like indexing limits the possible locations for a page, the associativity of the CMM is expected to be greater than those found in caches. To accelerate this process, we will use a CMM TLB structure. The 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. It should be noted, however, that only the tag portion of a TLB entry needs to be designed as Content Addressable Memory (CAM). The subpage bitmap can be stored in SRAM and indexed directly once an entry in the TLB is located.

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). However, when a CMM page is evicted and written back to Flash Replacement memory, the subpage bit map is meaningless. As such, subpage data is not copied back, and is invalidated once a page has been written back. Only the actual CMM pages reside in the 3D stacked DRAM layers. The other structures can reside in the memory controller, possibly on the same layer as the processing cores.

Prefetching. Prefetching has been shown to be extremely effective in reducing cache miss rates, however they can also substantially increase memory traffic[9], [1]. At the CMM level, we expect less contention than at the L1 or L2 cache level. Therefore, we may direct the memory controller to use idle CMM cycles to prefetch additional subpages on a memory request. Note that prefetch may occasionally delay requests for new pages (or subpages), since it is necessary to complete the transfer of current subpage being prefetched before initiating a new request.

2.3 Victim CMM

Victim caches[4] have traditionally been used to decrease conflict misses in low associativity caches. We adapt this concept to alleviate the conflict misses in CMM. The Victim CMM is a fully-associative structure. However, only the tag structure needs to be designed as content-addressable memory, while the page data can be stored in DRAM cells.

Data Copyback. In order to hide delays from writing back dirty subpages (and pages) to FRTD, we can copy back subpages in the background, when the channel to the FRTD is idle. However, since our goal is to reduce the frequency of writing to FRTD, we need to explore the trade-offs between background copy back and delaying copy back. For this purpose we will view the victim CMM as a write buffer, where a portion will be designated as write-through (with background copy back) and the remainder will be designated as write-back (with delayed copy back). The sizes of write-through and write-back portions can be varied based on application memory access behaviors, trading-off execution times with frequency of writes to FRTD.

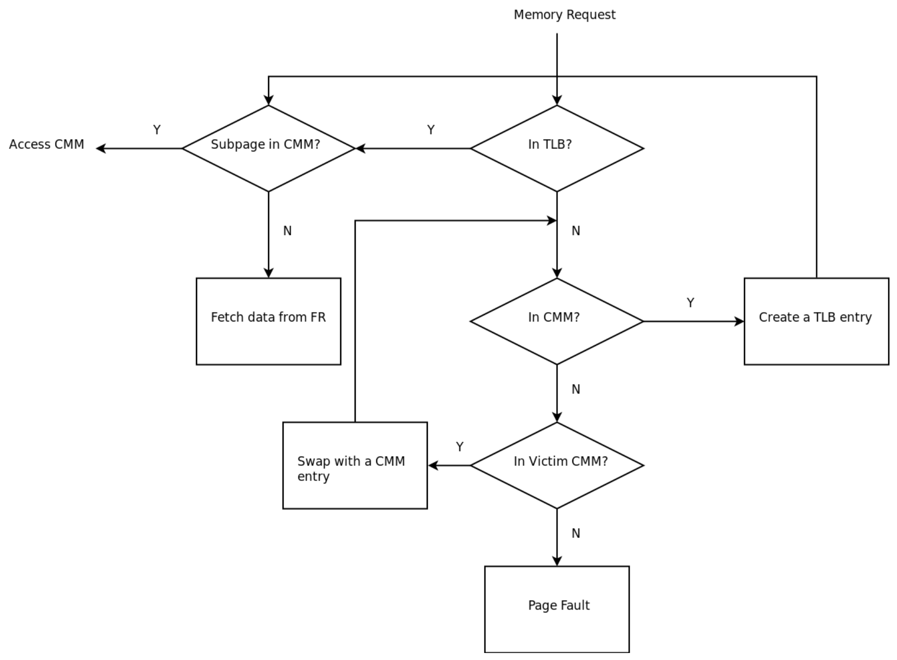

Fig. 3. Memory Request Flowchart

Figure 3 illustrates how a page entry within the CMM is located. It should be noted that the TLB, Tag structure and Victim can be accessed in parallel.

3 Experimental Framework

3.1 Simulation Environment

Simics[7] is a full system simulator, capable of running complete operating systems. Simics may be extended with user-created modules to implement custom memory hierarchies. Custom modules were created for our CMM and FRTD devices, and used to simulate our memory hierarchy, along with the g-cache module provided by Simics (for cache simulation). Our target platform consisted of a SPARC V9 architecture with a single UltraSPARC III Cu processor at 3 GHz, running Solaris 10. We utilize a subset of the SPEC CPU2006 benchmark suite as our test applications.

Although 3D DRAMs can be very large, since in this paper we are simulating single core system running CPU2006 benchmarks, we are using DRAM that is between 10 to 30 percent of the memory footprint of the benchmarks, so that our memory organizations are stressed significantly.

3.2 Methodology

The caches and CMM are warmed for 200 million user-level transactions, after which simulation data is gathered for the subsequent 500 million user-level transactions. Table 1 reflects our configuration parameters.

Table 1. Configuration Parameters. Numbers in cycles unless otherwise stated

Parameter	L1	L2	CMM	FRTD
Cache Access Latency	1	10	N/A	N/A
Read - Initial Latency	N/A	N/A	90	300
Read - Transfer Time	N/A	N/A	3	3
Write - Initial Latency	N/A	N/A	90	3000
Write - Transfer Time	N/A	N/A	3	3
Bus Width	N/A	N/A	128 bits	128 bits
TLB Hit Penalty	N/A	N/A	1	N/A
TLB Miss Penalty	N/A	N/A	4	N/A
Victim Hit Penalty	N/A	N/A	3	N/A
Total Size	32KB (each)	512 KB	Variable	N/A
Line Size	128	128	Variable	N/A
Associativity	4	8	Variable	N/A

Although we collect statistics for all memory structures, we will focus mostly on CMM statistics for this paper. We investigate various parameters with regards to the CMM TLB size, CMM Victim size and write-back / write-through threshold and number of subpages prefetched. We measure overall execution times, victim hit rates, CMM-FRTD channel utilization in the form of total data copied back to and requested from the FRTD.

4 Results and Analysis

During the course of our study for this paper, we explored several different design parameters for achieving best execution performance and minimize writes to FRTD. In this section we present our results.

4.1 TLB Size

Since the TLB is fully-associative, its size is often small. We experimented with various sizes for TLB (number of entries) from very small to very large and results are shown in Figure 4. For 6 out of 10 benchmarks, we were able to achieve 98% hit rates with a TLB size of 64 (which is 3.125% of the number of pages in CMM). A few benchmarks (e.g. Sjeng) show poor TLB hits regardless of the size of the TLB. We anticipate that when CMMs become large, TLB size

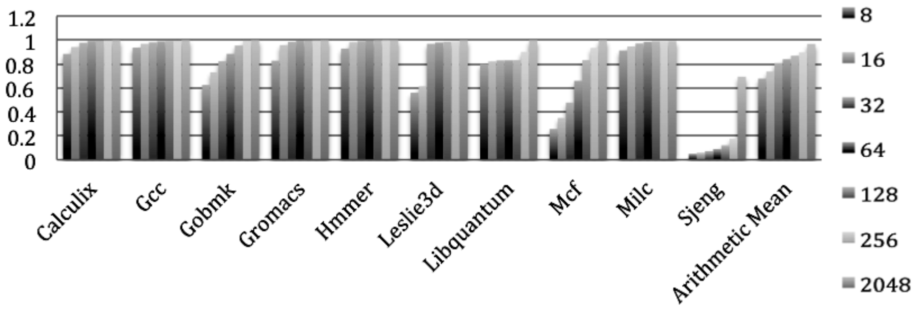

Fig. 4. TLB Size: Hit Rates

should be increased proportionally to maintain high hit rates. Note that for our experiments we used 64MB CMM.

4.2 Page Size

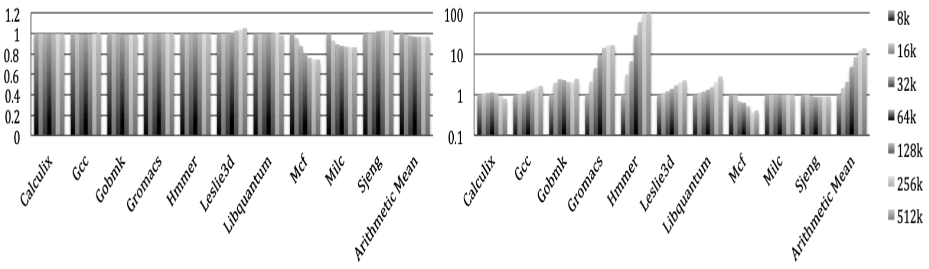

Fig. 5. Page Size: (a) Execution Times and (b) Data Write-back (Log Scale)

In our CMM design we use large pages that are divided into subpages, and only transfer subpages when requested. Large pages allow us to use smaller page tables and TLBs, and transferring smaller amounts (subpages) on each request minimizes CPU stalls on page misses. While the size of bitmaps per page (for tracking valid and dirty subpages) increases for large pages and small subpages, the overall bitmap size is independent of page size since the total number of bits required for the overall CMM bitmap remains constant for a given CMM size. However the size of bitmaps increase with smaller subpages. Another issue to remember is that when pages are very large, there will be very few pages in CMM, and this may cause more conflict misses. In some cases, we may even see internal fragmentation since some applications may not need the large pages.

Figure 5 shows execution times and amount of data written back to FRTD for different page sizes. It should be noted that the data is shown as relative values to the first page size (i.e, 8K pages). We observe that execution time reduces (in most cases, not significantly) with larger pages since the latency of accesses are amortized across large data sizes. However, large pages also cause more data to be copied back, and since our goal is to minimize data written back, we feel 16K or 32K are good page sizes for CMM. Using 8K pages would require overly large TLB and page tables.

4.3 Subpage Size

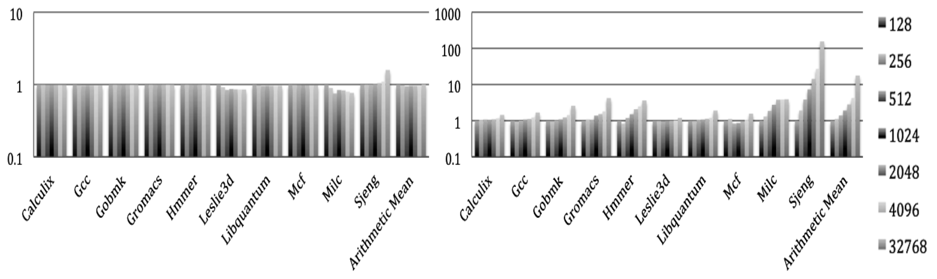

Fig. 6. Subpage Size: (a) Execution Times (Log Scale) and (b) Data Write-back (Log Scale)

Current systems transfer 4K or 8K pages on each request to overcome latencies of magnetic disk drives. However since FRTD latencies are smaller, we should consider transferring smaller amounts, but sufficient to cover the latencies. Subpage size determines the minimum amount of data transferred between the CMM and FRTD. We experimented with different subpage sizes for 32K pages and the results are shown in Figure 6; the data is normalized against the smallest subpage size (i.e., 128 bytes). As with page size, we observe a decrease in execution times with larger subpages. Larger subpages also reduce the size of of bitmaps, but they cause more data to be written back to FRTD. Our analysis shows that 512 or 1024 byte subpages balance execution times and amount of data written back to FRTD.

4.4 Pre-Fetch

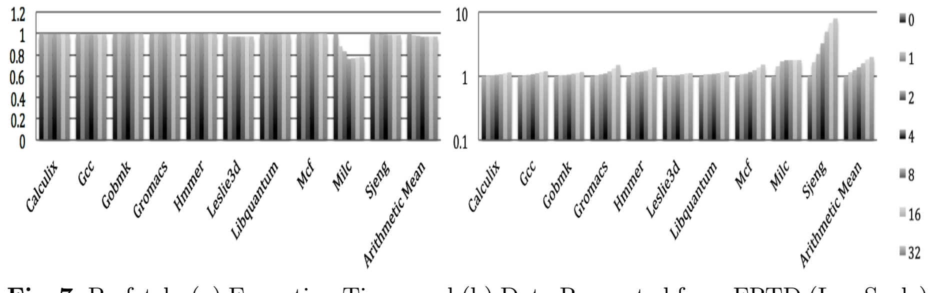

Fig. 7. Prefetch: (a) Execution Times and (b) Data Requested from FRTD (Log Scale)

The execution times when using smaller subpages can be improved by prefetching additional subpages on each request. Once a subpage prefetch is initiated, it cannot be interrupted. This can delay handling new requests. Thus the number of subpages that can be prefetched is limited by the spatial localities exhibited by applications. Figure 7 shows results for different number of subpages being prefetched. Here we use 512 byte subpages (and 32KB pages). The data shown is normalized against no prefetch. We do not prefetch beyond page boundaries. For some applications we notice a reduction in execution times as more subpages are prefetched, indicating higher spatial localities. As shown in the figure, the amount of data transferred increases with increased prefetches. We observe that for most applications, prefetching one additional subpage is sufficient to improve execution times and reduce data transfers between CMM and FRTD.

4.5 Victim Size

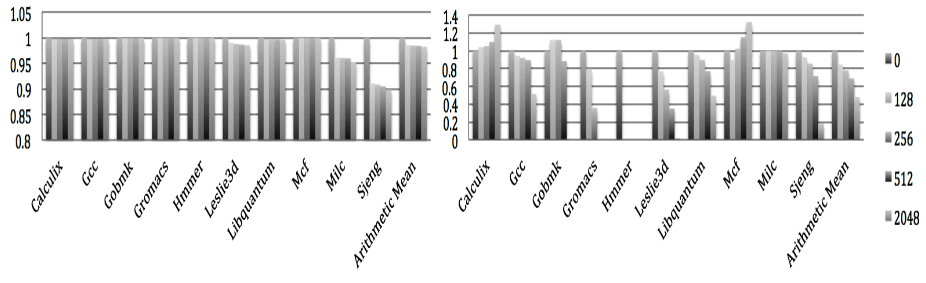

Fig. 8. Victim Size: (a) Execution Times and (b) Data Write-back

Since victim CMM is a fully-associative structure, its size should be limited. However, a very small victim CMM may not help either in reducing execution times or data copied back to FRTD. From Figure 8, it can be seen that for most applications victim CMM with up to 128 entries provide most benefit with respect to data copied back and execution times. The data is normalized against no victim CMM. Additional entries beyond 128 provide somewhat diminishing returns and therefore may be less cost-effective. For some applications, however, a victim CMM shows no benefit. It should be noted that the simulations in the figure used a static write-back / write-through threshold of 75%/25%. Our analyses show that either 128 or 256 entry victim CMM is a good choice for reducing execution times and the amount of data written back to FRTD.

4.6 Associativity

As mentioned previously, the CMM uses a set-associative design for page placement. Selecting a specific associativity to use in the design becomes a balance between escalating hardware costs (and longer access paths) with increased associativity against reduced conflicts for CMM pages. Figure 9 shows execution

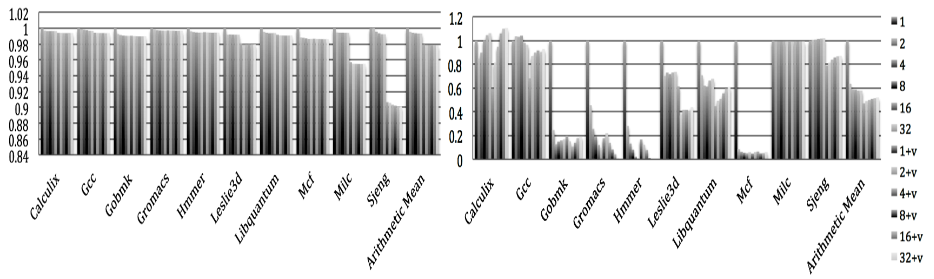

Fig. 9. Associativity: (a) Execution Times and (b) Data Write-back

times and data written back to FRTD for different associativities. We show data for CMM with (+v) and without a 256 entry victim CMM. The data is normalized against a direct mapped CMM. As expected, higher associativities reduce execution times and the amount of data written back since CMM page conflicts are reduced. However we also notice that the use of a victim CMM is more beneficial than increasing CMM associtivity. For example a 2-way set-associative CMM with 256 entry victim CMM shows the same performance in terms of the amount of data written back as a 32-way CMM. The conclusion to be drawn here is that associativity can be significantly reduced with the inclusion of a victim CMM, thus reducing hardware complexity of a very large CMM.

5 Related Work

Given our focus on the memory system organization and architecture, we describe related works in this area, and do not review existing research on the design or fabrication of either 3D stacked DRAMs or Flash Replacement Solid State Disks.

The ideas presented in [10] are closely related to our research. In that work, the researchers have explored the use of Phase Change Memories (PCM) as primary memory in place of DRAM. They compare the use of conventional organizations using very large DRAM systems (with 4GB to 32GB DRAM), PCM in place of DRAM (32GB) and a hybrid configuration that uses a 1GB DRAM as a cache to 32GB PCM. The purpose of the DRAM cache is to minimize writes to PCM, since PCM has limited write endurance. The simulations show that the hybrid DRAM cache plus PCM as primary memory is better suited, although using 32GB DRAM achieves better execution times. We use DRAM as primary memory and PCM as secondary memory, replacing magnetic disk drives. In addition, we investigate new virtual memory architectures.

There has been other research aimed at reducing writes to SSD devices (Flash or PCM). For example, [2] proposes changes to NAND flash organization with multiple layers of banks and sectors in order to delay in-place updates. In a sense, some NAND cells (banks and sectors) behave like caches. Our focus is in reducing write-backs to SSDs with innovative memory architecture and possibly with support from runtime systems.

Loh[6] investigated different organization of 3D DRAMs, in terms of how the bit cells are arranged in different layers, and published memory access performance comparisons of the various organizations.

Sun et. al[11] proposed using Magnetic Random Access Memories (MRAM) for shared caches in multicore processors. The MRAM is stacked over processors. This study is limited to using MRAM as non-uniformly accessed caches and do not utilize 3D layers for primary memory.

Page coloring[5] is similar to our page indexing schemes in CMM. Page coloring speeds up the virtual to physical address translations such that there are negligible differences between the virtual and physical addresses as far as (large) L2 caches are concerned. We will explore how the page placement techniques proposed in this paper can be adapted to our CMMs so that different program segments can be mapped in such a way the pages within the segment are indexed using cache line indexing schemes.

Cache memories and cache indexing schemes to improve how addresses map to different cache sets have been studied extensively [3]. There are too many variations and too many publications to summarize here. However, a recent study compared different cache indexing techniques in [8]. In this paper, the authors report techniques that aim to spread access to cache memories more uniformly across all cache sets. We are planning to evaluate some of these indexing methods within the context of CMM, particularly when multiple cores share the CMM.

6 Conclusion

The emerging memory technologies such as 3D stacked DRAM and solid state storage present new opportunities and challenges to computer architects. In this paper we presented a memory organization for large 3D stacked DRAM and large SSDs based on PCM technologies. We use cache like set-associative mapping for placing pages in main memory. We call our memory CMM (Cache-Main Memory). We view CMM with very large pages containing several subpages. Only a subpage is transferred on a request. We explored a number of design parameters with our CMM design including page sizes, subpage sizes, prefetching additional subpages, use of victim CMM and associativites. Our experiments indicate that 32K pages with 512 byte subpages, prefetching one additional subpage and using 128 entry victim CMM results in a good trade-off between reducing execution times and reducing the amount of data copied back to PCM.

Our study in this paper is limited to a single processor, and as such does not reflect the expected behavior of multi-core systems. We plan to extend our studies to multicore systems running both high performance (multithreaded) and throughput (multiprogrammed) benchmarks. We will explore using known techniques for reducing cache conflicts[8] to reduce CMM page conflicts.

References

1. Callahan, D., Kennedy, K., Porterfield, A.: Software prefetching. In: Proceedings of the fourth international conference on Architectural support for programming

languages and operating systems. pp. 40–52. ASPLOS-IV, ACM, New York, NY, USA (1991), http://doi.acm.org/10.1145/106972.106979

2. Debnath, B., Mokbel, M., Lilja, D., Du, D.: Deferred updates for flash-based storage. In: Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on. pp. 1 –6 (may 2010)
3. Hennessy, J.L., Patterson, D.A.: Computer Architecture - A Quantitative Approach (4. ed.). Morgan Kaufmann (2007)
4. 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. pp. 364–373. ISCA '90, ACM, New York, NY, USA (1990), http://doi.acm.org/10.1145/325164.325162
5. Kessler, R.E., Hill, M.D.: Page placement algorithms for large real-indexed caches. ACM Trans. Comput. Syst. 10, 338–359 (November 1992), http://doi.acm.org/10.1145/138873.138876
6. Loh, G.: 3d-stacked memory architectures for multi-core processors. In: Computer Architecture, 2008. ISCA '08. 35th International Symposium on. pp. 453 –464 (june 2008)
7. 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, 50–58 (February 2002), http://dl.acm.org/citation.cfm?id=619072.621909
8. Nwachukwu, I., Kavi, K., Fawibe, A., Yan, C.: Evaluation of techniques to improve cache access uniformities. In: Proceedings of the 40th Annual Conference on Parallel Processing (ICPP) (September 2011)
9. Porterfield, A.K.: Software methods for improvement of cache performance on supercomputer applications. Ph.D. thesis, Rice University, Houston, TX, USA (1989), aAI9012855
10. 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. pp. 24–33. ISCA '09, ACM, New York, NY, USA (2009), http://doi.acm.org/10.1145/1555754.1555760
11. Sun, G., Dong, X., Xie, Y., Li, J., Chen, Y.: A novel architecture of the 3d stacked mram l2 cache for cmps. In: High Performance Computer Architecture, 2009. HPCA 2009. IEEE 15th International Symposium on. pp. 239 –249 (feb 2009)