Exploiting superpages in a nonvolatile memory file system

Sheng Qiu
Texas A&M University
herbert1984106@neo.tamu.edu

A. L. Narasimha Reddy
Texas A&M University
reddy@ece.tamu.edu

***Abstract*—Emerging nonvolatile memory technologies (sometimes referred as Storage Class Memory (SCM)), are poised to close the enormous performance gap between persistent storage and main memory. The SCM devices can be attached directly to memory bus and accessed like normal DRAM. It becomes then possible to exploit memory management hardware resources to improve file system performance. However, in this case, SCM may share critical system resources such as the TLB, page table with DRAM which can potentially impact SCM's performance.**

In this paper, we propose to solve this problem by employing superpages to reduce the pressure on memory management resources such as the TLB. As a result, the file system performance is further improved. We also analyze the space utilization efficiency of superpages. We improve space efficiency of the file system by allocating normal pages (4KB) for small files while allocating super pages (2MB on x86) for large files. We show that it is possible to achieve better performance without loss of space utilization efficiency of nonvolatile memory.

I. Introduction

For decades, modern file systems are designed on the assumption that the underlying storage devices are block-based, such as disk or flash-based SSD. The recent development of nonvolatile memory technologies such as phase change memory (PCM) are poised to revolutionize storage in computer systems. These technologies collectively are termed Storage Class Memory (SCM). The SCM devices are attached directly to memory bus and are byte-addressable. SCM can offer comparable access latency to DRAM and are orders of magnitude faster than traditional disks. Processor can access persistent data through memory load/store instructions. Figure 1 shows the potential system hierarchy while building SCM as the persistent storage. As shown in Figure 1, the DRAM and SCM can sit in parallel and be accessed through memory bus. It becomes possible to leverage the memory management module to simplify and accelerate file system operations such as space management and file block addressing. Our previous work – SCMFS [21] has exploited memory management hardware to improve file system performance. However, this approach also added more work on memory hardware resources such as TLB and MMU which caused increased data TLB misses. In this paper, we show that it is possible to obtain better file system performance by reducing pressures on such resources.

In this work, we propose to solve the problem of increased TLB misses by employing superpages. As a result, the performance of our file system is further improved. Compared with normal pages (usually 4K), the super pages (2MB on x86) are able to enlarge the coverage of TLB because a TLB entry for super pages covers more memory than normal 4KB pages. As a result, we can effectively reduce the TLB misses when the size of TLB is limited and fixed.

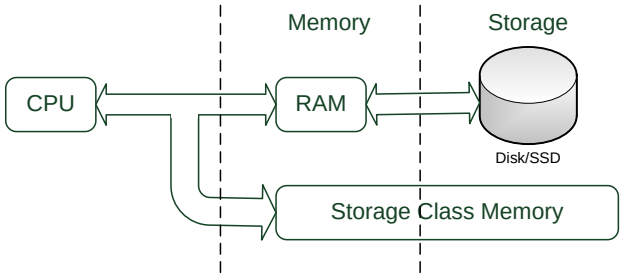

Fig. 1. Storage Class Memory

We also analyze the space utilization efficiency of superpages. We improve space efficiency of the file system by allocating normal pages for small files and metadata while allocating super pages for large files. We show that it is possible to achieve better performance without loss of space utilization efficiency of nonvolatile memory.

The primary contributions of this paper are as follows: (a) This paper analyzes the impact of TLB misses while designing a nonvolatile memory file system. (b) We propose to solve this problem by employing superpages for large files while utilizing normal pages for small files and metadata. (c) We show that it is possible to achieve better performance without loss of space utilization efficiency of nonvolatile memory.

The remainder of the paper is organized as follows: We will discuss the related work in Section II. In Section III we present the motivation for superpage. In Section IV we present the implementation of superpage within our file system which are then evaluated in Section V. Section VI concludes the paper.

II. Related Work

A number of recent works proposed hybrid file systems via byte-addressable NVRAM and HDDs [11], [5]. In [11], Miller et al. proposed using a byte-addressable NVRAM file system which used NVRAM as storage for file system metadata, a write buffer, and storage for the front parts of files. In the Conquest file system [5], the byte-addressable NVRAM layer holds metadata, small files and executable files while the large files reside on HDDs. Hybrid file systems for byte-addressable NVRAM and NAND Flash are proposed to address NAND-Flash file system specific issues using byte-addressable NVRAM [3], [8], [16]. They include mount

latency, recovery overhead against unexpected system failure, and the overhead in accessing page metadata for a NAND Flash device. All of these previous works assume the NVRAM is small and stores only metadata and/or small files. While our file system is designed for purely nonvolatile memory based persistent storage which expects the NVRAM to be large enough to hold the whole file system data.

BPFS [2] file system designed for non-volatile byte-addressable memory, uses shadow paging techniques to provide fast and consistent updates. It also requires architectural enhancements to provide new interfaces for enforcing a flexible level of write ordering. DFS[6] is another file system designed for flash storage. DFS incorporates the functionality of block management in the device driver and firmware to simplify the file system, and also keeps the files contiguous in a huge address space. It is designed for a PCIe based SSD device by FusionIo, and relies on specific features in the hardware.

Solutions have been proposed to speed up memory access operations, to reduce writes, and for wear-leveling on PCM devices. Some of these solutions improve the lifetime or the performance of PCM devices at the hardware level [9], [10]. Some of them use a DRAM device as a cache of PCM in the hierarchy. [17] presents a page placement policy on memory controller to implement PCM-DRAM hybrid memory systems. Several wear-leveling schemes to protect PCM devices from normal applications and even malicious attacks have been proposed [12], [13], [18], [22]. Since our work focuses on the file system layer, all the hardware techniques can be integrated with our file system to provide better performance or stronger protection.

The importance of TLB performance and support for superpages has been described in [15], [20], [4], [19]. Impact on TLB misses on application performance prompted proposals for effective superpage management [15]. The architectural and operating system support required to exploit medium-sized superpages (e.g., 64KB) is presented in [20]. Our approach focuses on employing superpages within a nonvolatile memory file system. We propose to utilize both normal pages and super pages to achieve better performance of file system without loss of space utilization efficiency of the SCM device.

III. Motivation

When building storage class memory for persistent storage, we may have a system hierarchy as shown in Figure 1. The SCM and normal DRAM can be both accessed through memory bus. Then it becomes possible to leverage the memory management module within the operating system to help manage the space of SCM devices. For this consideration, we have built a file system (SCMFS) [21] on virtual address space and largely simplified the complexity and operations of file system. As shown in Figure 2, regular file systems are built on top of the generic block layer, while SCMFS is on top of the modified memory management module. We simplified file block addressing by allocating each file within contiguous virtual address space. To get the location of the request data,

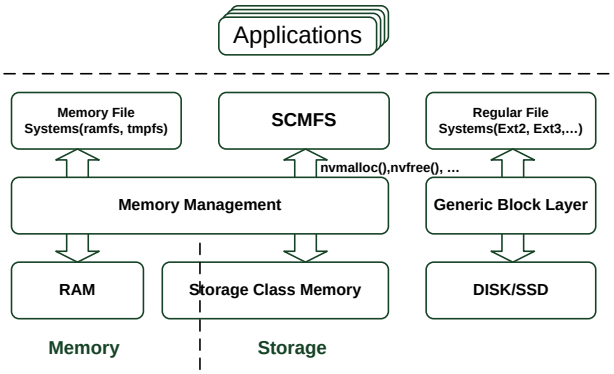

Fig. 2. File systems in operating systems

we just add the offset to the start address of the file. The actual physical location of the data is available through the TLB or page table translation (VA to PA). Although this approach largely reduced the instruction set and the complexity of the file system, it added more pressure on memory management hardware such as TLB, since both normal applications and file system share those resources now.

To relieve the pressure on TLB, operating systems sometimes map the whole memory in the system to a linear address space using a larger page size (e.g., 2MB). However, this approach does not fit SCMFS. As shown in Figure 3, File1 and File2 may have different access permissions, i.e. File1 can be opened for both read and write, while File2 is only readable. In such a case, we can not allocate File1 and File2 on the same superpage. It is difficult to maintain different access permissions within one superpage since its corresponding TLB entry defines one unified access permission for the whole page. If we map each file into a separate superpage, there will be lots of internal fragmentation for small files. We will present how we employ superpages within our file system to solve the space utilization issue in section IV.

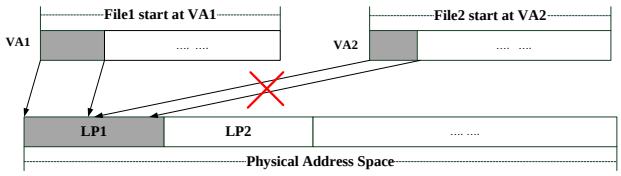

Fig. 3. Address Mapping

IV. Superpage Implementation

To accelerate the memory access speed, modern processors cache the virtual to physical address mappings from the page tables in TLB. Expensive performance penalties are incurred whenever we get TLB misses. To enlarge the coverage of TLB, most hardware and operating systems support superpages. In this section, we describe how to efficiently employ superpages within our file system.

A. Preservation for superpage

A superpage is a memory page of larger size than an ordinary page. To allocate a superpage, we are required to

have a contiguous memory space which is usually multiple sizes of a normal page. Therefore it is not guaranteed to be able to obtain a superpage successfully even though there is still sufficient physical memory. In our implementation, we solve this problem by preserving a contiguous, configurable size of SCM for allocating superpages. We divide the physical space of SCM into two regions, one for normal page allocation and the other for superpages. The boundary between normal and super page region is configurable during file system mounting. Figure 4 shows the layout of the physical space of SCM.

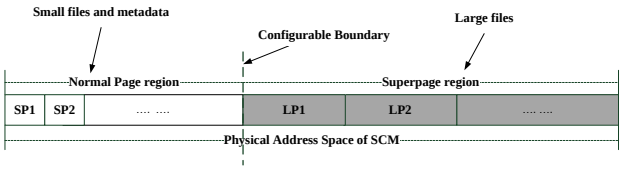

Fig. 4. Physical Space of SCM

B. Space utilization

In a file system, we may have a number of small and large files as well as metadata that need to be stored. The space utilization efficiency is very important especially for SCM devices considering their expensive cost. We want to utilize superpages for storing data which may potentially reduce the pressure on TLB and improve file system performance. However, allocating the whole file system data with superpage will generate lots of internal fragmentation, especially for small files and metadata. In such a case, we may have a low space utilization efficiency on the SCM device.

To achieve better performance without loss of space utilization efficiency, we propose using both normal and super pages within our file system. As shown in Figure 4, small files and metadata are mapped to normal pages while large files are stored within super pages. As a result, we solve the internal fragmentation issue for small size data while TLB misses are reduced effectively whenever accessing large files on super pages.

One potential problem is that it is not easy to decide how the file size will grow during creation. Therefore, initially, we always allocate normal pages for file data. Whenever the file size become larger than a configurable threshold, we begin to migrate this file to super pages. After migration, the original file data (on normal pages) will be freed and the corresponding inode metadata will be updated. To minimize the impact on SCMFS's performance, we use a background kernel thread to handle the migration. This kernel thread will pick up those files that are not being written currently to do migration. It is noted that read request can still be handled by the original file (on normal pages) during migration, while write request has to wait until the migration process finishes. Since most large files in real system are multi-media or read-heavily files, which usually keep a relatively stable size once written. Therefore the migration between normal and super pages will not be frequent.

C. Modifications to kernel

To support superpages within our file system, we made several modifications to the original Linux kernel 2.6.33. We first add a memory zone "ZONE_STORAGE" into the kernel. We put all the address range of DRAM space which we used to emulate SCM into the new zone "ZONE_STORAGE". Then we add a set of memory allocation/deallocation functions for super pages. Generally, there are four main functions used by our file system. The function nvmalloc_superpage() allocates designated number of super pages from "ZONE_STORAGE" while nvfree_superpage() is the corresponding function for deallocation. Another two functions are nvmalloc_expand_superpage() and nvmalloc_shrink_superpage(). The former one is used when the file size increases and the mapped super pages are not enough, while the latter one is used to recycle the allocated but unused super pages.

V. EVALUATION

In this section, we evaluate the performance of enhanced SCMFS with superpage support. We implemented superpages within SCMFS on a linux kernel of version 2.6.33.

A. Methodology

To evaluate superpage performance of SCMFS, we use multiple benchmarks. The first benchmark, IOZONE [1] creates a large file and issues different kinds of read/write requests on this file. Since the file is only opened once in each test, we use IOZONE to evaluate the performance of accessing file data. The second benchmark we use is postmark [7], which creates a lot of files and performs read/write operations on them. The file size can be configured within one specific range. We use this benchmark to evaluate superpage's impact when accessing both small and large files in SCMFS. In all experiments, we track the number of allocated superpages and the actual file data size. We see that our approach keeps the internal fragmentation within 1% on average. In the experimental environment, the test machine is a commodity PC system equipped with a 2.33GHz Intel Core2 Quad Processor Q8200, 8GB of main memory. We configured 4GB of the memory as the type "ZONE_STORAGE", and used it as Storage Class Memory.

In all the benchmarks, we compare the performance of SCMFS with/without superpage supported to that of other existing file systems, including ramfs, tmpfs and ext2. Since ext2 is designed for a traditional storage device, we run it on ramdisk which emulates a disk drive by using normal RAM in main memory. It is noted that tmpfs, ramfs and ramdisk are not designed for persistent memory, and none of them can be directly used on storage class memory.

B. IOZONE Results

We run IOZONE with sequential and random workloads for both read and write. To obtain the performance of TLB, we used the performance counters in the modern processors through the PAPI library [14]. Figure 5 and 6 show the data

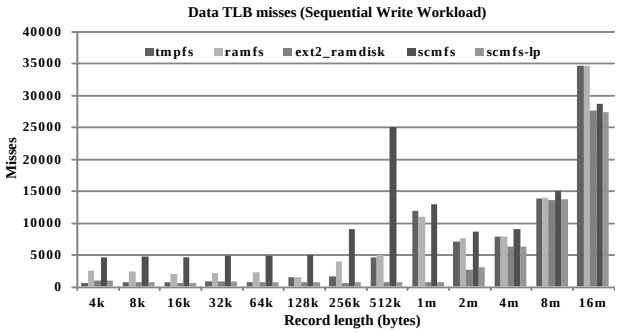

Fig. 5. TLB misses – IOZONE Sequential Write workload

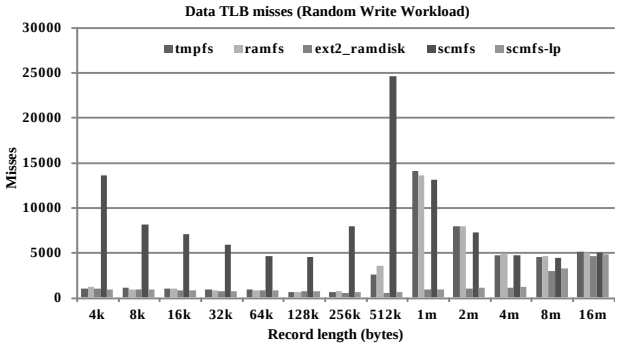

Fig. 6. TLB misses – IOZONE Random Write workload

TLB misses of all file systems while running IOZONE's sequential and random write workloads. We can see SCMFS with superpage support (bar scmfs-lp) effectively reduced the data TLB misses compared with original SCMFS. When the request size become larger (more than 2MB), the variance of data TLB misses tends to be smaller among all file systems. This is because the number of TLB entries for superpages is limited which may not cache all the superpages when request sizes is larger.

Figure 7 and 8 show the corresponding throughput for IOZONE's sequential and random write workloads. We can see that employing superpages within SCMFS improves throughput performance significantly. It is noted that in Figure 7, ext2 on ramdisk performs much better than other file systems when request size is within 128kb–512kb. This is because within that range, ext2 has much lower L2 data cache misses compared to other file systems. The performance on read workloads is consistent with the write workload. We did not show it for lack of space.

C. Postmark Results

In this section, we evaluate the impact of superpages by running postmark benchmark. We use postmark to generate both intensive read and write workloads. In our experiment, postmark created a number of small and large files and performed read, append and delete transactions. We again used the PAPI library to investigate the detailed performance of TLB.

Figure 9 and 10 show the data TLB misses of postmark for all file systems. We can see that utilizing superpages effectively reduces data TLB misses of SCMFS which is

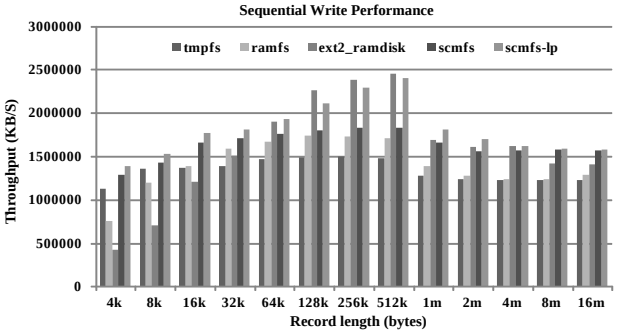

Fig. 7. Throughput – IOZONE Sequential Write workload

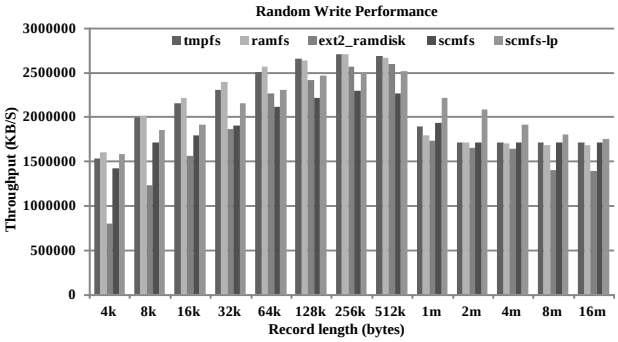

Fig. 8. Throughput – IOZONE Random Write workload

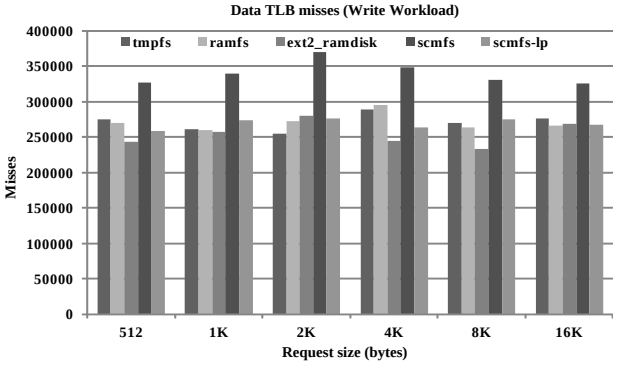

Fig. 9. TLB misses – Postmark's write workload

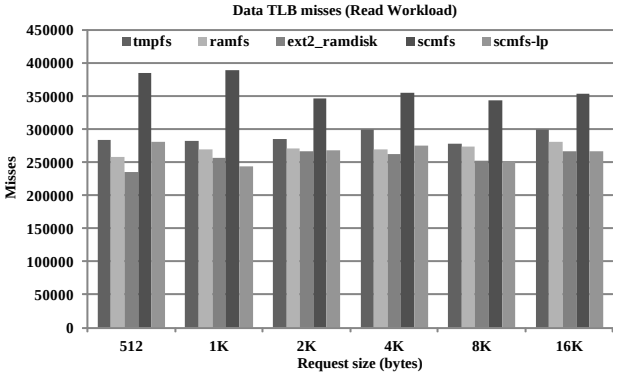

Fig. 10. TLB misses – Postmark's read workload

consistent with IOZONE results. The throughput performance is shown in Figure 11 and 12. We again achieved better performance while employing superpages within SCMFS.

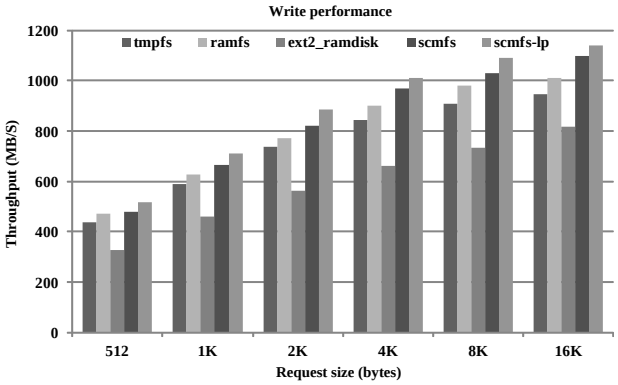

Fig. 11. Throughput of Postmark's write workload

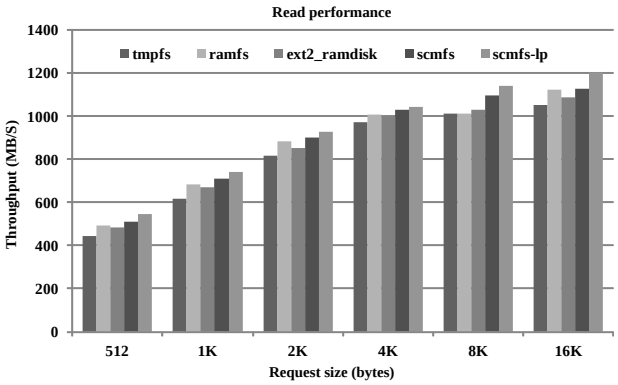

Fig. 12. Throughput of Postmark's read workload

VI. CONCLUSION

In this paper, we explored the approach of utilizing memory management hardware for managing persistent storage of SCM. We analyzed the potential impact of TLB misses on file systems designed specially for SCM. We proposed an effective solution by employing superpages within a nonvolatile file system to relieve the pressure on the TLB. The experimental results show that our approach reduced the data TLB misses by an average of 65% and further improved the throughput by 8.5% compared to the same file system with no superpage support. To maintain space utilization efficiency of nonvolatile memory, we utilized both normal and super pages within our file system.

REFERENCES

[1] "Iozone file system benchmark," Available on July 2011 from http://www.iozone.org/, 2011.

[2] J. Condit, E. B. Nightingale, C. Frost, E. Ipek, B. C. Lee, D. Burger, and D. Coetzee, "Better i/o through byte-addressable, persistent memory," in *Proc. of the Symposium on Operating Systems Principles*, 2009, pp. 133–146.

[3] I. Doh, J. Choi, D. Lee, and S. Noh, "Exploiting non-volatile ram to enhance flash file system performance," in *Proc. of the 7th ACM and IEEE International Conference on Embedded Software*, 2007, pp. 164–173.

[4] N. Ganapathy and C. Schimmel, "General purpose operating system support for multiple page sizes," in *Proc. of the USENIX Annual Technical Conference*, 1998, pp. 91–104.

[5] A. i A. Wang, P. Reiher, G. J. Popek, and G. H. Kuenning, "Conquest: Better performance through a disk/persistent-ram hybrid file system," in *Proc. of the 2002 USENIX Annual Technical Conference*, 2002, pp. 15–28.

[6] W. K. Josephson, L. A. Bongo, D. Flynn, and K. Li, "Dfs: A file system for virtualized flash storage," in *Proc. of the USENIX Conference on File and Storage Technologies*, vol. 6, 2010, pp. 85–100.

[7] J. Katcher, "Postmark: A new file system benchmark," technical Report TR3022. Network Applicance Inc. October 1997.

[8] E. Kim, H. Shin, B. Jeon, S. Han, J. Jung, and Y. Won, "FRASH: Hierarchical file system for FRAM and Flash," *Computational Science and Its Applications*, vol. 4705, pp. 238–251, 2007.

[9] B. Lee, P. Zhou, J. Yang, Y. Zhang, B. Zhao, E. Ipek, O. Mutlu, and D. Burger, "Phase-Change Technology and the Future of Main Memory," *IEEE Micro*, vol. 30, no. 1, pp. 143 –143, Jan 2010.

[10] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting Phase Change Memory as a Scalable Dram Alternative," in *Proc. of the 36th Annual IEEE/ACM International Symposium on Computer Architecture*, 2009, pp. 2–13.

[11] E. Miller, S. Brandt, and D. Long, "Hermes: High-performance reliable mramenabled storage," in *Proc. of the 8th IEEE Workshop on Hot Topics in Operating Systems (HotOS)*, 2011, pp. 83–87.

[12] M.Qureshi, J. Karidis, M. Franceschini, V. Srinivasan, L. Lastras, and B. Abali, "Enhancing lifetime and security of pcm-based main memory with start-gap wear leveling," in *Proc. of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2009, pp. 14–23.

[13] M.Qureshi, A. Seznec, L. Lastras, and M. Franceschini, "Practical and secure pcm systems by online detection of malicious write streams," in *Proc. of the 17th IEEE International Symposium on High Performance Computer Architecture*, 2011, pp. 478–489.

[14] P. J. Mucci, S. Browne, C. Deane, and G. Ho, "Papi: A portable interface to hardware performance counters," in *Proc. of the Department of Defense HPCMP Users Group Conference*, 1999, pp. 7–10.

[15] J. Navarro, S. Iyer, P. Druschel, and A. Cox, "Practical, transparent operating system support for superpages," in *Proc. of the 5th USENIX OSDI*, 2002.

[16] Y. Park, S. Lim, C. Lee, and K. Park, "Pffs: A scalable flash memory file system for the hybrid architecture of phase-change ram and nand flash," in *Proc. of the ACM Symposium on Applied Computing*, 2008, pp. 1498–1503.

[17] L. E. Ramos, E. Gorbatov, and R. Bianchini, "Page placement in hybrid memory systems," in *Proceedings of the international conference on Supercomputing*, ser. ICS '11. New York, NY, USA: ACM, 2011, pp. 85–95. [Online]. Available: http://doi.acm.org/10.1145/1995896.1995911

[18] N. H. Seong, D. H. Woo, and H.-H. S. Lee, "Security refresh: Prevent malicious wear-out and increase durability for phase-change memory with dynamically randomized address mapping," in *Proc. of the 37th Annual IEEE/ACM International Symposium on Computer Architecture*, 2010, pp. 383–394.

[19] I. Subramanian, C. Mather, K. Peterson, and B. Raghunath, "Implementation of multiple page size support in hp-ux," in *Proc. of the USENIX Annual Technical Conference*, 1998, pp. 105–118.

[20] M. Talluri and M. D. Hill, "Surpassing the tlb performance of superpages with less operating system support," in *Proc. of the 6th ASPLOS*, 1994.

[21] X. Wu and A. N. Reddy, "Scmfs : A file system for storage class memory," in *Proc. of the SC11*, 2011, pp. 1498–1503.

[22] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A Durable and Energy Efficient Main Memory Using Phase Change Memory Technology," in *Proc. of the 36th International Symposium on Computer Architecture*, 2009, pp. 14–23.