Linux kernel support to exploit phase change memory

Youngwoo Park, Sung Kyu Park and Kyu Ho Park
Korea Advanced Institute of Science and Technology (KAIST)
(ywpark,skpark)@core.kaist.ac.kr and kpark@ee.kaist.ac.kr

Abstract

Recently, phase change memory (PRAM) has been developed as a next generation memory technology. Because PRAM can be accessed as word-level using memory interface of DRAM and offer more density compared to DRAM, PRAM is expected as an alternative main memory device. Moreover, it can be used as additional storage of system because of its non-volatility. However, PRAM has several problems. First, the access latency of PRAM is still not comparable to DRAM. It is several times slower than that of DRAM. Second, PRAM can endure hundreds of millions of writes per cell. Therefore, if PRAM does not be managed properly, it has negative impact on the system performance and consistency. In order to solve these problems, we consider the Linux kernel level support to exploit PRAM in memory and storage system. We use PRAM with a small size DRAM and both PRAM and DRAM are mapped into single physical memory address space in Linux. Then, the physical memory pages, which are used by process, are selectively allocated based on the access characteristics. Frequently updated hot segment pages are stored in DRAM. PRAM is used for read only and infrequently updated pages. Consequently, we minimize the performance degradation caused by PRAM while reducing 50% energy consumption of main memory. In addition, the non-volatile characteristic of PRAM is used to support file system. We propose the virtual storage that is a block device interface to share the non-volatile memory pages of PRAM as a storage alternative. By using 256MB PRAM for virtual storage, we can decrease more than 40% of access time of disk.

1 Introduction

For several decades, DRAM has been the main memory of computer systems. Since the memory requirement is growing to support the increasing number of cores and concurrent applications, DRAM based main memory significantly increases the power and cost budget of a computer system. Recent studies [8, 7] have shown that 30-40% of modern server system energy is consumed by the DRAM memory. Moreover, it is expected that DRAM scaling will be clamped by the limitation in cell-bitline capacitance ratio [4, 10]. Therefore, new memory technologies such as Phase-change RAM (PRAM), Ferroelectric RAM (FRAM), and Magnetic RAM (MRAM) have been proposed to overcome the limitation of DRAM.

Among these memories, PRAM is the most promising technology for future memory. Figure 1 shows the basic structure of PRAM cell. PRAM uses phase change material (GST: Ge2Sb2Te5). It has two phases;an amorphous or a crystalline phase. Since the amorphous and the crystalline phase have a large variance on their resistance, the data is read by measuring the current of PRAM. The phase of GST can be changed by heating the material. The moderate and long current pulse crystallizes GST. On the other hand, short current pulse melts and quenches GST quickly and makes it amorphous.

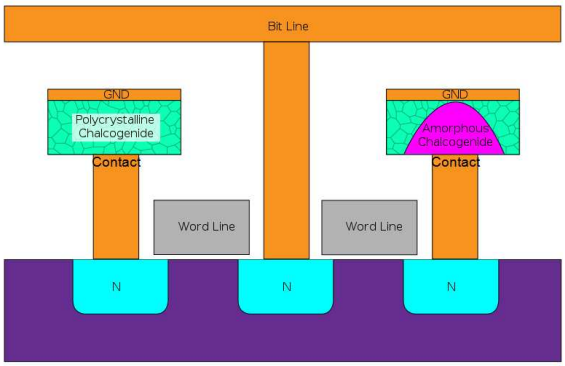

Figure 1: PRAM cell structure [2]

Basically, PRAM is byte-addressable like DRAM. The

great advantages of PRAM are the scalability and low energy consumption. PRAM does not require implementing capacitor for memory cell. PRAM provides superior density relative to DRAM. Because the phase of PRAM is maintained persistently, PRAM is non-volatile memory and has negligible leakage energy. Therefore, PRAM can be used to provide the memory that has much higher capacity and lower power consumption than DRAM.

However, the long current pulse for crystallizing increases the latency of PRAM writes. Although PRAM access latency is tens of nanoseconds, it is still not comparable to DRAM access latency. The frequent access of PRAM can impact on the overall system performance. Also, the PRAM write energy consumption and endurance are limitations of PRAM. The high pulse for phase change increases the dynamic energy consumption. PRAM writing makes the thermal expansion and contraction of material. It degrades the electrode-storage contact and reduces the reliability of programming current. This degrades the write endurance of PRAM cells. PRAM can sustains 10^8 rewrite per cell [11].

In this paper, we consider the Linux level support to exploit PRAM in current computer system. First of all, we decide to use PRAM with a small size DRAM to overcome the limitation of PRAM. PRAM and DRAM are mapped into single physical memory address space. Then, the physical main memory of Linux consists of one small fast region (DRAM) and one large slow region (PRAM). Therefore, the PRAM is used to increase the size of main memory and eliminate a lot of page faults. At the same time, we can use DRAM to reduce the overall main memory access latency. Based on this main memory architecture, we propose a new Linux physical page management mechanism. The physical memory pages, which are used by process, are selectively allocated based on the segment type. Consequently, we minimize the performance degradation and endurance problems caused by PRAM while achieving a large scale and low power main memory.

In addition, we also discuss the block device interface to share the non-volatile main memory pages of PRAM as a storage alternative. It gives a lot of advantage for file system to access metadata and small size file because it can read or write the data as single word level and avoid unnecessary seek latency of disk.

2 Hybrid main memory architecture

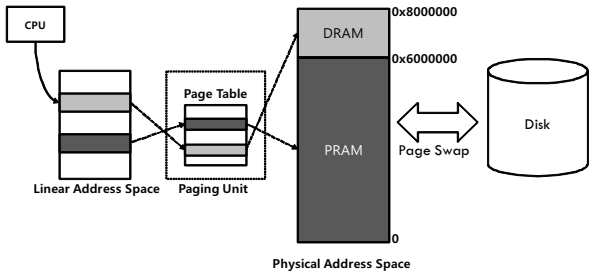

Figure 2: Hybrid main memory architecture

Figure 2 shows the proposed hybrid main memory architecture. Both PRAM and DRAM are used as a main memory. This architecture reduces the cost and power budget because large portion of main memory is replaced by PRAM. Using small size of DRAM, it minimizes the performance degradation. Similar to the traditional main memory architecture, there is memory page swap between hybrid main memory and the second level storage. However, the number of page swapping can be reduced because the main memory capacity is increased by PRAM.

In hybrid main memory architecture, PRAM and DRAM are assigned to single physical address space. All memory pages of PRAM and DRAM can be directly managed by the Linux kernel. However, it is necessary for kernel to distinct the PRAM and DRAM region. We assume that DRAM always has lower physical address than PRAM and the size of each memory is provided by the kernel option. The, Linux kernel has information of the exact physical address range of PRAM and DRAM as shown in Figure 2. Physical pages of PRAM and DRAM can be distinguished by the physical address.

3 Hybrid main memory management

3.1 Free page management

For the hybrid main memory architecture, Linux kernel needs to manage DRAM and PRAM region separately. In current Linux kernel, the physical memory of the system can be partitioned and managed in several nodes. Also, physical memory of each node is divided into several zones. If we can map DRAM and PRAM physical pages into separate memory nodes, memory management facilities can be used in proposed hybrid main

memory architecture. However, Linux node is only proposed for specific NUMA hardware and Linux zone is proposed to cope with the hardware constraints. Instead of making new node or zone for DRAM and PRAM, we use additional *free area* for each zone as shown in Figure 3.

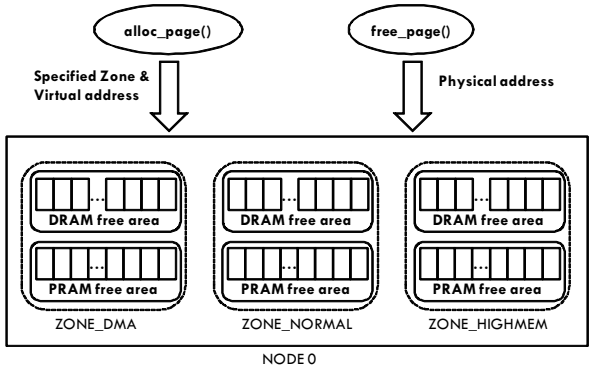

Figure 3: Free page management for hybrid main memory list

Proposed Linux kernel has two *DRAM free area* and *PRAM free area* which contain only the free pages of DRAM and PRAM, respectively. Although both *free areas* are used by the same *buddy system* page allocator, the contiguous DRAM and PRAM block is independently handled. In addition, When the kernel invokes a memory allocation function, the page frames are selectively allocated to one of the *free area* of specified zone. The *free area* selection considers the characteristics of allocated data. We will describe this allocation policy in the section 3.2

After we divide the *free areas*, we also need to consider the page reclamation method. Basically, Linux reclaims free pages when there is not enough number of pages in zones. Although we separately manage the free page list of DRAM and PRAM, the page reclamation occurs when the total number of pages in both DRAM and PRAM *free areas* is lower than the threshold. Therefore, we can fully utilize the main memory region before swapping. Even though one of the memory devices is fully utilized, the free pages of another memory device are contributed as main memory. The large size PRAM main memory region reduces a lot of page swapping.

3.2 Selective allocation

In hybrid main memory architecture, there are two different memory devices. Particularly, PRAM is very different from the conventional DRAM. The physical memory management of Linux kernel, which is only developed for the uniform memory devices, should be changed. The first thing which needs to be addressed is the page allocation.

The goal of the page allocation of Linux is to serve the memory allocation request from the Linux kernel and the user processes. Conventional Linux kernel allocates physical memory pages when kernel functions like *alloc_page()* and *__get_free_page()* are called. These page allocation functions are successfully returned when the free pages of requested size are found and allocated. The conventional *buddy system* allocates groups of contiguous page frames to solve the external fragmentation.

Previously, it is not important where the memory pages are located in main memory because the characteristics of memory pages are always same in uniform memory device. However, in our hybrid main memory architecture, the location of pages can have significant effect on the performance and power consumption of main memory. As we mentioned in section 1, PRAM write operation is much slower than read and require high energy. If memory pages that are frequently updated are allocated in PRAM, it can increase the dynamic power consumption and decrease the overall access latency of main memory. Moreover, it reduces the lifetime of main memory because PRAM has limited write endurance. Therefore, the key of our page allocation is to assign frequently updated data into DRAM instead of PRAM.

The question is how to find the write intensive data before page allocation. Although it is very difficult to predict the future access pattern of each page, we can use the general characteristics of pages for page allocation. Traditionally, the process address space of Linux is partitioned in several linear address intervals called segments. It is well known that the access pattern of data in same segment is almost similar. For example, text segment includes the executable code which is read-only data. Stack segment contains the return address, parameters and local variables which are frequently read write. Table 1 summarizes the general access patterns of Linux segments.

Figure 4 shows the design of selective allocation. The memory segments are identified by the variables which are included in the *mm_struct* memory descriptor. For example, *start_code* and *end_code* store the initial and

Segment type	Access pattern
Text segment	Read only
Initialized data segment	Read centric
Uninitialized data segment	Infrequent read/write
Stack segment	Frequent read/write
Heap segment	Frequent read/write

Table 1: Access pattern of Linux segments

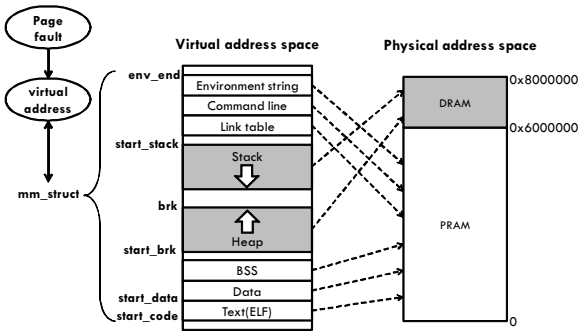

Figure 4: Design of selective allocation

final virtual address of the segments. *start_stack* store the start virtual address of stack segments. Also, we can get the virtual address, where physical page should be mapped. It is delivered by the page fault handler before page allocation. Thus, only if the variable of the *mm_struct* memory descriptor is compared to the virtual address that cause the page fault, we can decide the segment type of page before allocation. After finding the segment type, we selectively allocate a physical page between DRAM and PRAM.

In current design of selective allocation, the page allocation policy is fixed. The pages of heap and stack segments are allocated in DRAM. All other memory pages are allocated into PRAM as shown in Figure 4. However, if we implement new system call *salloc_policy()*, the selective allocation policy is changed by each application. For example, a user can allocate stack and heap pages in PRAM. Also, this system call can be used to decide the allocation policy of file cache, mmap and library pages.

4 Hybrid main memory for virtual storage

4.1 Virtual stroage

Many previous researches prove that non-volatile memory is very effective to reduce the overhead of disk based storage because it is free from seek latency and favorable for small size random accesses data [3, 6, 12]. In proposed hybrid main memory, a part of main memory (PRAM) can be used for the byte-addressable non-volatile storage. In order to exploit PRAM memory as storage, we propose the virtual storage which is a block level interface to use PRAM region of hybrid main memory for storage as shown in Figure 5.

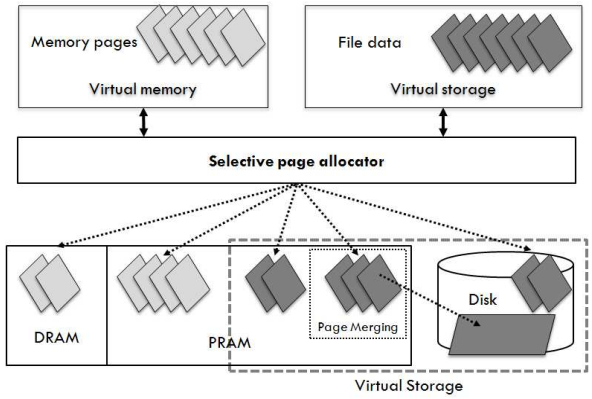

Figure 5: Virtual storage architecture

Similar to the virtual memory, the virtual storage is an abstraction of single and contiguous storage. Physically, it uses PRAM main memory region and disk as a storage device. If the file system writes data to the virtual storage interface, it selectively allocates data into PRAM memory page or disk. However, we do not reserve PRAM pages for storage use. All free pages in PRAM are dynamically allocated for memory page and file page. The mapping from virtual page to physical page in PRAM or disk is maintained by the storage page table which contains the mapping of allocated virtual pages as tree. In order to preserve the mapping information after power-off, it should be stored in PRAM and the root of table is managed in a fixed location of PRAM.

4.2 Selective allocation for virtual storage

In section 3.2, we describes that the memory pages are allocated based on the type of segments. On the other hand, the basic metric for file page allocation algorithm is data size. It is generally known that a small size file is frequently and randomly accessed. Because PRAM has fast access time and much smaller capacity than disk, it is better to keep only data of a small size file in PRAM. If the requested data size to virtual storage is over sev-

eral tens of KB, contiguous physical pages in disk are selected to store the data.

Also, the virtual storage is designed to support write request merging. Although the data size of a request is small, if it is a sequential request that has a nearby virtual address of previous one, the selective allocator decides that those pages are in the same file and move them into disk later. It can allocate a large file, whose size is continuously increased, to disk. Consequently, the write request merging reduces the waste of PRAM space and number of disk access.

5 Evaluation

5.1 Hybrid main memory

Currently, PRAM is not available as a main memory. Instead of hardware, we evaluate proposed hybrid main memory and its management schemes using the M5 simulator [9]. In order to implement the hybrid main memory architecture, we use additional memory modules in a physical main memory space. Two memory modules are used to simulate DRAM and PRAM memory, respectively. The PRAM and DRAM memories are mapped into same physical address space.

In addition, we add the memory access monitoring module. It monitors all memory access of DRAM and PRAM. Because we separately use two memory modules, the memory monitoring module can monitors both DRAM and PRAM at the same time. Although we monitors the total read/write access counter and size of memory, the memory monitors can be extended to get an any information that is related with memory access. Finally, the monitored read/write access count and size is used to calculate the access latency and energy consumption of overall main memory.

Then, the selective allocation for main memory is implemented and evaluated on Linux 2.6.27 which is operated on the M5 simulator. We execute benchmarks on M5 using a simple execution ALPHA processor running at 1GHz. We assume that the processor does not have caches to focus on main memory evaluation. The total main memory size is 256MB. For hybrid main memory, 64MB DRAM and 198MB PRAM is used. The access latency and energy consumption of DRAM and PRAM is calculated by the parameters of Numonyx [11]. Table 2 summarizes the parameters used for DRAM and

Parameter	DRAM	PRAM
Read latency	50ns	50ns
Write latency	50ns	1us
Read Energy	0.1nJ/b	0.05nJ/b
Write Energy	0.1nJ/b	0.5nJ/b
Idle Energy	1W/GB	0.005W

Table 2: DRAM and PRAM characteristics [11]

PRAM. For our workloads, we use MiBench benchmark suite.

First of all, we compare the energy consumption of proposed hybrid main memory with DRAM. For this evaluation, we should assume the idle time of main memory. It has been well-established that the average utilization of server is even below 30% [5]. Therefore, in all evaluations of energy consumption, we assumes that memory is only accessed 40% time.

Figure 6 and Figure 7 shows the reduction in memory energy consumption. The hybrid main memory achieves around 30% energy savings. Moreover, if we use the selective allocation, we can reduce the PRAM write operation which consumes more energy than DRAM read/write and PRAM read operation. The total energy saving ratio is increased by 50%. The selective allocation exploits the read-friendliness of PRAM as well as writes friendliness of DRAM, and hence achieves better overall energy efficiency.

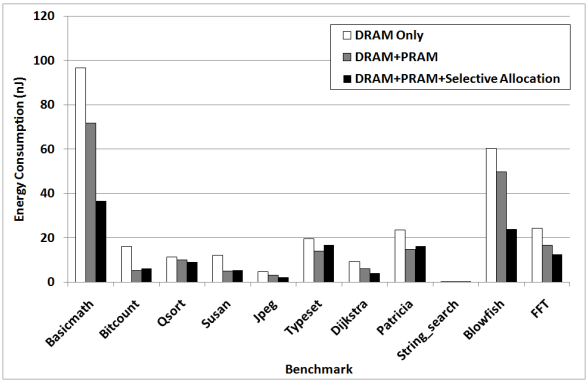

Figure 6: Total energy consumption

Although PRAM is good for scalable main memory, it can increase the overall latency of main memory. Figure 8 shows total access latency and Figure 9 shows the latency overhead against the DRAM main memory. In our experiments, the hybrid main memories that use typical Linux allocation algorithm have more than 100%

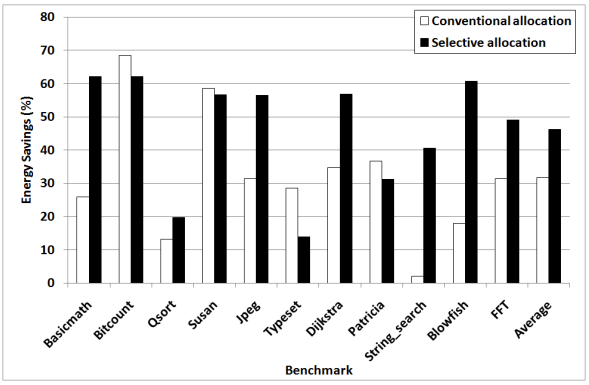

Figure 7: Energy savings against DRAM main memory

latency overhead. However, if we use the selective allocation, we can allocate the infrequently accessed page in PRAM and reduce the 50% latency overhead. Moreover, the latency of 6 applications is only increased under 20%, while reduce much more than 50% energy consumption.

However, some benchmarks use much global variables and update the variables frequently. It causes a lot of write in data segment. The latency of these benchmarks is much larger than DRAM main memory. For Typeset application, selective allocation rather increases the latency and reduces energy consumption.

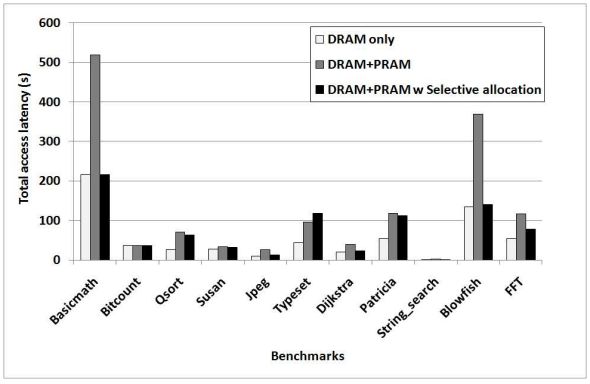

Figure 8: Total access latency

5.2 Virtual storage

In order to evaluate the performance of virtual storage, we estimate the total access time when executing OLTP trace [1]. In virtual storage, each OLTP request can be allocated to PRAM or disk. We use PRAM access latency of Table 2 and assume 5ms disk access latency for evaluation. Then, three allocation policies (random, selective, and selective merging) are compared in this

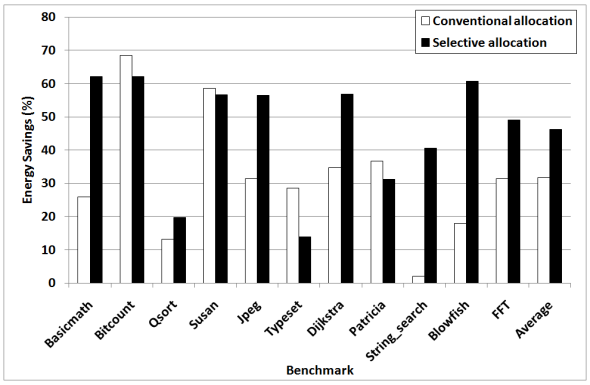

Figure 9: Latency overhead against DRAM main memory

evaluation. The random allocation randomly assigns a request to PRAM or disk. The selective allocation uses PRAM only for the request whose size is under 64KB. The selective merging uses both selective allocation and write request merging which is proposed in section 4.2. Figure 10 shows the evaluation result.

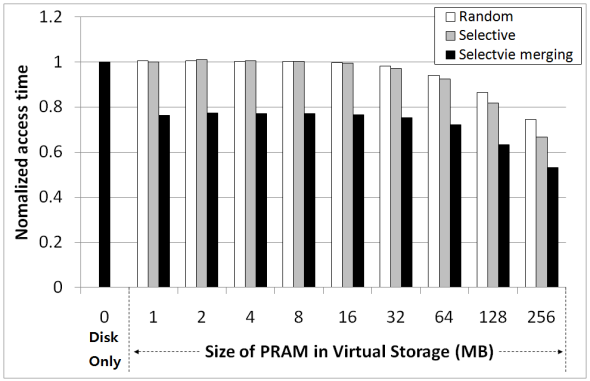

Figure 10: Total storage access time of virtual storage

Figure 10 presents that the use of PRAM decrease the total access time of disk. It is because PRAM is free from seek latency and favorable for small size random access. However, random allocation cannot fully use the PRAM because it dissipates PRAM for large size sequential data. Although the selective allocation reduces the access time of virtual storage, it does not effective when the size of PRAM is small. If the size of PRAM is small, PRAM space is filled with fragmented sequential requests soon. Many of small size random requests do not have the benefits of PRAM.

On the other hand, selective allocation with write request merging is very effective because request merging help to find more sequential data. The virtual storage

can allocate the more random requests to PRAM. It reduces the number of disk accesses and more increase the performance of virtual storage. If 256MB PRAM is used as storage in virtual storage, we decrease more than 40% of access time of disk.

6 Further work

Although the selective allocation statically allocates memory pages by using the general characteristics of segments, it cannot fully reflect the dynamic access pattern of memory page. In order to minimize the access latency of hybrid main memory, it is necessary to dynamically balance and move pages between DRAM and PRAM. For example, if some memory pages are frequently updated in a moment, it is better to be migrated to DRAM at that time. If a page in DRAM is occasionally read, it needs to be migrated to PRAM. The page migration also increases the endurance of PRAM because the frequently updated page will be migrated to DRAM.

In order to implement the memory migration, we may use the LRU lists of OS kernel to manage. There are *active_list* and *inactive_list* of pages in Linux kernel. If we always select the page of *inactive_list* to migrate into PRAM, recently accessed page is stored in DRAM. However, the LRU list of OS kernel does not fully monitor the memory access because memory access is occurred without interruption of kernel. Therefore, we need to think about hardware and software design of kernel LRU list to reflect the memory access pattern.

Also, we described that PRAM is good for storage alternatives. However, all previous file systems statically assign non-volatile RAM only for storage although PRAM can be used both for main memory and storage system. In order to maximize the advantage of PRAM, we need to develop unified management of PRAM for memory and storage devices. All PRAM pages need to be freely allocated for main memory and storage. Linux kernel should be implemented to control the use of PRAM according to overall system status.

7 Conclusion

PRAM will be widely used in the future computing system as memory or storage alternatives. In this paper, we consider the Linux kernel level support to exploit PRAM. We use PRAM for hybrid main memory and propose new page allocation algorithm. Consequently, we minimize the performance degradation and endurance problems caused by PRAM while reducing 50% energy consumption of main memory. Also, we propose the virtual storage which is a new block level interface using PRAM for storage. It allocates small size random access data into PRAM and reduces the number of disk access. We can decrease more than 40% of access time of disk.

References

[1] OLTP Application I/O and Search Engine I/O. http://traces.cs.umass.edu/index.php/Storage/Storage.

[2] Phase-change memory. http://en.wikipedia.org/wiki/Phase-change_memory.

[3] An-I A. Wang, et al. Conquest: Better Performance through a Disk/Persistent-RAM Hybrid File System. In *Proceedings of 2002 USENIX Annual Technical Conference*, 2002.

[4] Benjamin C. Lee, E. Ipek, O. Mutlu, and D. Burger. Architecting Phase Change Memory as a Scalable DRAM Alternative. In *Proceedings of the 36th annual International Symposium on Computer Architecture*, 2009.

[5] David Meisner, David Meisner, Thomas F. Wenisch. PowerNap: eliminating server idle power. In *roceeding of the 14th international conference on Architectural support for programming languages and operating systems*, 2009.

[6] Ethan L. Miller, Scott A. Brandt, and Darrell D. E. long. HeRMES: High-performance reliable MRAM-enabled storage. In *Proceedings of the 8th IEEE Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, 2001.

[7] G. Dhiman, R. Ayoub, and T. Rosing. PDRAM:A Hybrid PRAM and DRAM Main Memory System. In *Proceedings of the 46th Annual Design Automation Conference*, 2009.

[8] Moinuddin K. Qureshi, V. Srinivassan, and Jude A. Rivers. Scalable High Performance Main Memory System Using Phase-Change Memory Technology. In *Proceedings of the 36th annual*

International Symposium on Computer Architecture, 2009.

[9] Nathan L. Binkert, Ronald G. Dreslinski, Lisa R. Hsu, Kevin T. Lim, Ali G. Saidi, and Steven K. Reinhardt. The M5 Simulator: Modeling Networked Systems. *IEEE Micro*, 26(4):52–60, 2006.

[10] P. Zhou, B. Zhao, J. Yang, and Y. Zhang. A Durable and Energy Efficient Main Memory Using Phase Change Memory Technology. In *Proceedings of the 36th annual International Symposium on Computer Architecture*, 2009.

[11] S. Eilert, M. Leinwander, and G. Crisenza. Phase Change Memory: A new memory enables new memory usage models. *2009 IEEE International Memory Workshop*, pages 1–2, 2009.

[12] Y. Park, S. H. Lim, C. Lee, K. H. Park, et. al. PFFS:A Scalable Flash Memory File System for the Hybrid Architecture of Phase change RAM and NAND Flash. In *Proceedings of the 2008 ACM symposium on Applied computing*, 2008.

Proceedings of the Linux Symposium

July 13th–16th, 2010
Ottawa, Ontario
Canada

Conference Organizers

Programme Committee

Proceedings Committee