International Journal of Control and Automation
Vol. 9, No. 11 (2016), pp.353-362
http//dx.doi.org/10.14257/ijca.2016.9.11.30

A Task Scheduling Scheme Considering Hybrid Main Memory and Interactive Tasks

Jeongmi Shin and Yeonseung Ryu[1]

Department of Security and Management Engineering, Myongji University
116 Myongji-ro, Cheoin-gu, Yongin, Gyeonggi-do, Korea
shinblly@gmail.com, ysryu@mju.ac.kr

Abstract

The latest mobile devices usually use multi-core technologies such as dual-core, quad-core or octa-core for high performance. Further, as the capacity of main memory in the mobile devices is growing, a lot of tasks are capable of running concurrently. With these trends, however, mobile devices run out of battery power faster than before. In recent years, next generation non-volatile memory technology (NVRAM) have developed significantly and considered as low-power main memory architecture. Traditionally, the bandwidth-aware multi-core task scheduling schemes have been studied in order to address the bandwidth saturation problem of shared main memory. In this paper, we propose a multi-core scheduling scheme considering DRAM/NVRAM hybrid main memory. The goal of the proposed scheme is to reduce the execution time of tasks by avoiding the memory bandwidth saturation as well as to improve the response time of interactive tasks. We have showed through trace-driven simulation that the proposed scheme outperforms the legacy scheduling schemes.

***Keywords**: Non-volatile memory, Memory bandwidth bottleneck, Task scheduling, Interactive task*

1. Introduction

Today there are more mobile devices which employ multi-core processor technology. Interactive mobile devices such as smart phones are usually used in an interactive way for calling, texting, internet browsing and Social Networking Service. However, it is general that interactive mobile devices should handle lots of background tasks. For example, devices need to sense data from the environment and analyze them in real-time in the background. Also, there are background tasks for instant communication like chatting and mail which are waiting for incoming request from network. Even though lots of background tasks are running, users of interactive mobile devices expect fast response time of their interactive tasks [1, 2].

Multi-core processors offer a higher performance over single-core processors but the interconnect bandwidth between processors and shared main memory has become a major bottleneck. In order to mitigate the memory bandwidth bottleneck there have been a lot of bandwidth-aware multi-core task scheduling schemes [3-11]. They try to select and schedule job segments that could maximize their total bandwidth requirement to as close to the peak memory bandwidth as possible. Another issue regarding the main memory is the power consumption. Recent works showed that DRAM-based main memory spends a significant portion of the total system power [12]. Since the capacitors used in DRAM lose their charge over time, DRAM must refresh all the cells approximately 20 times a second, reading each one and re-writing its contents. Such endless refresh operations become a major factor for power depletion of computer systems.

[1] Corresponding Author

Recently, the next generation Non-volatile Random Access Memory (NVRAM) technology such as PRAM, STT-MRAM and ReRAM have developed significantly [13-16]. In June 2016, for example, IBM and Samsung demonstrated MRAM cells for devices with diameters ranging from 50 down to 11 nanometers in only on nanoseconds [17]. NVRAM is random-access memory that retains data when power is turned off. Further, NVRAM is becoming promising because of their high density, comparable read access speed and low power consumption. For several past years, many researchers have studied the next-generation computer system architecture using the next generation NVRAM technology. In particular, studies on NVRAM-based main memory organization have been very active [13, 18-19]. However, cost per byte of new NVRAMs is higher than that of DRAM now. As a result, hybrid main memory using DRAM and NVRAM seems to be practicable instead of pure NVRAM-based main memory in the near future. Some recent studies have introduced NVRAM-based hybrid main memory organization [20-29].

In this paper, we study a multi-core task scheduling scheme considering DRAM/NVRAM hybrid main memory. Previous multi-core scheduling schemes considering memory bandwidth bottleneck problem have dealt with DRAM-only main memory [3-11]. With DRAM-only main memory, access latencies are independent of access request type (i.e., read and write). In the hybrid main memory with DRAM and NVRAM, on the contrary, access latency highly depends on the target memory [13-16]. In general, NVRAM read latency is longer than DRAM access latency and NVRAM write latency is much longer than NVRAM read latency. For example, write access time of PRAM is about 3 times slower than that of DRAM. Hence, legacy DRAM-based bandwidth-aware scheduling schemes cannot be used for hybrid main memory.

The goal of the proposed scheme is to reduce the execution time of tasks by avoiding the memory bandwidth saturation as well as to improve the response time of interactive tasks. The proposed scheduling scheme classifies the tasks according to the task characteristics and gives the higher priority to the interactive tasks in order to improve the response time of interactive tasks. Further, when selecting tasks to run, the proposed scheduling scheme considers different access latency of hybrid memory medium and tries to reduce the total execution time of tasks by avoiding the memory bandwidth saturation. The proposed scheduling scheme estimates the total bandwidth requirement of tasks by considering different access latency of DRAM and NVRAM, and selects tasks to avoid the memory bandwidth saturation. We show, through trace-driven simulation, that the proposed scheme outperforms the legacy scheduling schemes.

The rest of this paper is organized as follows. In Section 2, we introduce background knowledge of memory bandwidth-aware multi-core scheduling scheme. Section 3, we present a novel multi-core scheduling scheme in detail. Section 4 presents the evaluation results. Finally, Section 5 concludes the paper.

2. Related Works

2.1. Memory Bandwidth Aware Multi-Core Scheduling

Since the early 2000s, multi-core processor technology has developed by Intel, AMD and others significantly to improve performance. In multi-core computers, main memory is usually treated as a shared resource among all cores. Tasks running concurrently on different cores compete for access of main memory, thereby increasing their execution times. As the trend is to put dozens of cores on one chip, the available bandwidth to access memory will be a critical problem. For a simple example, let us assume that the peak bandwidth of the memory bus is 6.4GB/s. If the scheduling time quantum is 100ms and the amount of data requested by all running tasks exceeds 640MB during one time quantum, then memory saturation occurs and some tasks cannot finish their works within one time quantum.

In order to mitigate the memory bandwidth bottleneck there have been a lot of bandwidth-aware multi-core task scheduling schemes. Most of them attempt to predict the bandwidth requirement of tasks, and select synergistic concurrent tasks to avoid bandwidth saturation. In doing so, they usually define job segments as segments of code executed during a scheduling time quantum. And they try to select and schedule job segments that could maximize their total bandwidth requirement to as close to the peak memory bandwidth as possible. [4] profiled bandwidth usage information of each task and calculate average available bandwidth for new task. [5] showed the impact of memory bandwidth fluctuation on overall performance for tasks on multi-core system. They proposed a new scheme that maintains the total bandwidth requirement at a steady level instead of maximizing the bandwidth utilization. [8] introduced a quantitative profiling method for analyzing the impact of contention for memory bandwidth. Profiling data is presented a bandwidth graph which helps to determine how much the application suffers when its available bandwidth is reduced. [11] also presented a memory trace tool which uses hardware performance counters in order to predict bandwidth usage of multi-threaded programs with low overhead.

2.2. NVRAM Technology

Non-volatile random-access memory (NVRAM) is random-access memory that retains data when power is turned off. Among the NVVRAMs, PRAM(Phase change RAM) and STT-MRAM(Spin-Torque Transfer Magnetic RAM) are becoming promising candidates for main memory because of their high density, comparable read access speed and low power consumption. Table 1 shows the comparison of PRAM, STT-MRAM and DRAM. A PRAM (also known as Phase Change Memory) cell uses a special material, called phase change material, to represent a bit. STT-MRAM uses a Magnetic Tunnel Junction (MTJ), the fundamental building block, as a binary storage. An MTJ comprises a three-layered stack: two ferromagnetic layers and an MgO tunnel barrier in the middle. The cost per byte of new NVRAMs is higher than that of DRAM now. As a result, shown in Figure 1, hybrid main memory using DRAM and NVRAM seems to be practicable instead of pure NVRAM-based main memory in the near future. Some recent studies have introduced NVRAM-based main memory organization [13-29].

Table 1. Comparison of Memories

	PRAM	STT-MRAM	DRAM
Volatility	No	No	Yes
Cost/TB	High	High	Low
Read Latency	50 ~ 100 ns	30 ns	15 ~ 50 ns
Write Latency	150 ns	30 ~ 100 ns	15 ~ 50 ns
Endurance	10^8 for write	-	-
Idle Power	~ 0.05 W	~ 0.05 W	~1.3 W/GB
Read Energy	~ 0.1 nJ/b	~ 0.1 nJ/b	~ 0.1 nJ/b
Write Energy	~ 0.5 nJ/b	~ 0.5 nJ/b	~ 0.1 nJ/b

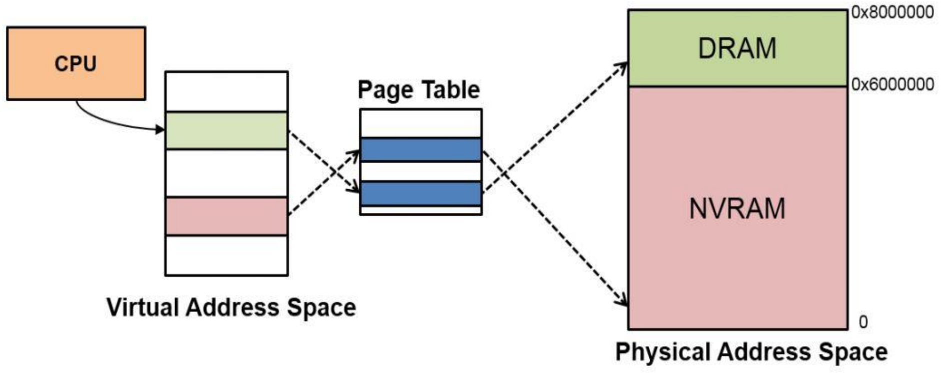

Figure 1. Address Space of Hybrid Main Memory Architecture [21]

However, legacy bandwidth-aware multi-core scheduling methods can not applicable with the DRAM/NVRAM hybrid main memory. This is because the bandwidth requirement cannot be represented by a unique memory transaction type. With hybrid main memory of DRAM and NVRAM, memory transactions should be classified into three categories based on access latency: DRAM access, NVRAM read, and NVRAM write. [21] studied a hybrid main memory-aware task scheduling scheme. They collected effective memory bandwidth usage of each task using the memory access monitoring and classified tasks (i.e., latency-sensitive and bandwidth-sensitive) according to how much tasks access memory. Then they selected next tasks according to the task characteristics. However, they consider only memory bandwidth usage when selecting the next task, but do not consider the task characteristic such as interactive or non-interactive tasks. Hence, they are not applicable to the interactive mobile devices which need to provide fast response time to interactive tasks, and thus novel scheduling schemes must be developed.

3. Proposed Scheme

3.1. System Model

In this section, we propose a multi-core task scheduling scheme called MQ-HAMC (Multilevel feedback Queue Hybrid memory-Aware Multi-Core) which reduces the execution time of tasks by avoiding the memory bandwidth saturation and improves the response time of interactive tasks. An interactive task is one that spends more of its time

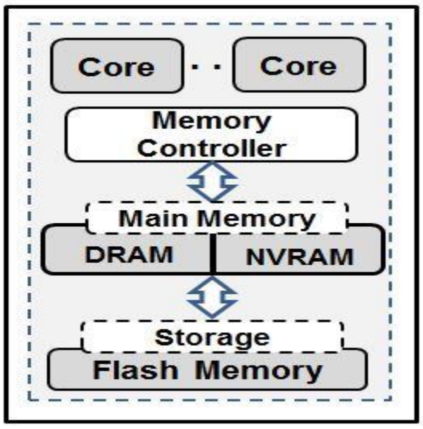

Figure 2. System Architecture Considered In This Paper

doing I/O than it spends doing computations (i.e., I/O intensive). A non-interactive task, in contrast, generates I/O requests infrequently, using more of its time doing computations (i.e., CPU intensive). Typically interactive tasks require fast response time in comparison with non-interactive tasks. Figure 2 shows the system architecture of mobile devices considered in this work. We assume that mobile devices are equipped with multi-core processors and DRAM/NVRAM hybrid main memory.

We assume that there are *N* cores in our mobile device and there are multiple ready queues. Figure 3 illustrates our multi-level queue model. Basically our scheduling is very similar to multilevel feedback queues scheduling. The basic operation of multilevel feedback queues is as follows:

1. When a new task is created, it is inserted at the end of the top-level queue.
2. If the task is completed within the time quantum of the given queue, it leaves the system.
3. If the task voluntarily relinquishes control of the core and later it becomes ready again, it will be inserted at the tail of the same queue which it relinquished earlier.
4. If the task uses all the time quantum of the given queue, it is pre-empted and inserted at the end of the next level queue.
5. This process will continue until the task completes or it reaches the bottom level queue.
6. At the bottom level queue the tasks circulate in round robin fashion until they complete.

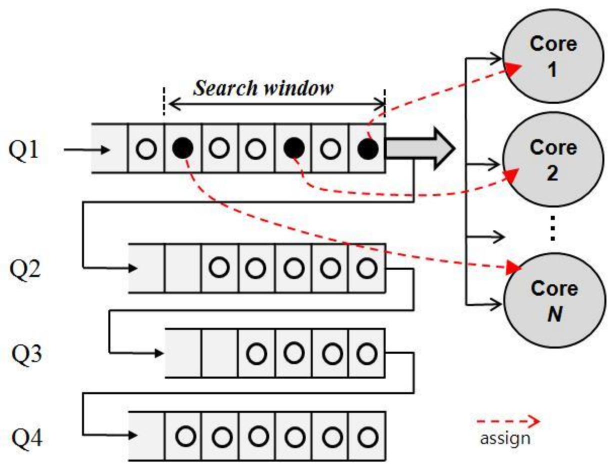

Figure 3. Multi-Level Queue Model for Interactive Tasks [28]

The time quanta are assigned to each queue, but the lower level queue has a time quantum which is greater than (i.e. 2 times) that of the previous higher level queue. For scheduling, the scheduler always starts fetching tasks from the head of the highest level queue. If the highest level queue has become empty, then the scheduler fetch a task from the next lower level queue. The same procedure is implemented for picking up in the subsequent lower level queues. Meanwhile, if a task comes into any of the higher level queue, it will preempt a task in the lower level queue. Whenever a core becomes idle, the scheduler is activated to assign a new task to the idle core.

3.2. Memory Bandwidth Calculation

The proposed scheduler maintains a system variable which stores the sum of memory bandwidth requirements of currently running tasks:

$$B_{tot} = \sum_{i=1}^{K} BW(Task_i) \quad (1)$$

where *K* is the number of running tasks in the system and *BW*(*Task$_i$*) is the bandwidth requirement of *Task$_i$*. Precisely speaking, the bandwidth requirements are not real but predicted value. Whenever the scheduler determines the next tasks to run, it finds a task that could maximize the total bandwidth requirement. That is, the scheduler finds a $Task_n$ satisfying the following:

$$B_{tot} + BW(Task_n) < PMB \tag{2}$$

where *PMB* is the peak memory bandwidth on the system.

Since search operation that finds the next task satisfying Equation (2) could be time consuming, MQ-HAMC defines a search window in order to limit search overhead. If the scheduler cannot find the next task at a given level queue, it tries to search from the next lower level queue.

We assume that there is a monitor module in the system as presented in [21]. The monitor collects memory access statistics for each task based on the access types (i.e., read or write) and the target memory type (i.e., DRAM or NVRAM). For the prediction of memory bandwidth requirement of tasks, we adopt a bandwidth prediction scheme from our previous work [26]. We define the *effective memory bandwidth* (EMB) which can be represented by the number of memory transactions, as shown in Equation (3).

$$EMB = PMB \times number\ of\ memory\ transactions \times access\ latency \tag{3}$$

where *PMB* is the peak memory bandwidth on the system.

We define $Task_{i,j}$ as the *j*-th execution instance of a *Task i*. We measure $EMB_{i,j}$, the effective memory bandwidth for the $Task_{i,j}$, using Equation (4).

$$EMB_{i,j} = EB_{D,i,j} + EB_{NV,i,j} \tag{4}$$
$$EB_{D,i,j} = PMB \times TC_{D,i,j} \times L_D$$
$$EB_{NV,i,j} = PMB \times (L_{NVR} \times TC_{R,i,j} + L_{NVW} \times TC_{W,i,j})$$

where

$EB_{D,i,j}$ = effective DRAM bandwidth used by the $Task_{i,j}$
$EB_{NV,i,j}$= effective NVRAM bandwidth used by the $Task_{i,j}$
$TC_{D,i,j}$ = number of DRAM access transactions of the $Task_{i,j}$
$TC_{Ri,j}$ = number of NVRAM read access transactions of the $Task_{i,j}$
$TC_{W,i,j}$= number of NVRAM write access transactions of the $Task_{i,j}$
L_D = memory access latency of DRAM
L_{NVR} = memory access latency of NVRAM read
L_{NVW} = memory access latency of NVRAM write

In order to predict the bandwidth requirement of $Task_{i,j}$, we use the following Equation:

$$BW_{i,j} = \propto \cdot EBW_{i,j-1} + (1-\propto) \cdot BW_{i,j-1} \tag{5}$$

where

$BW_{i,j}$= predicted bandwidth value for the *j*-th instance of *Task i*
$EBW_{i,j}$= effective bandwidth value for the *j*-th instance of *Task i*
$\propto$ = a constant weighting factor ($0 < \propto < 1$)

4. Experimental Results

In order to evaluate the proposed scheme, we have developed a trace-driven simulator. For the workload, we generated synthetic workload which is a mix of interactive tasks (I/O intensive tasks) and non-interactive tasks (CPU intensive tasks). The workload of the *j*-th instance of $Task_i$ consists of the following parameters:

$$Task_{i,j} = (ET_{i,j}, TC_{D,i,j}, TC_{R,i,j}, TC_{W,i,j}) \tag{6}$$

where

$ET_{i,j}$ = execution time of $Task_{i,j}$
$TC_{D,i,j}$ = number of DRAM access transactions of $Task_{i,j}$
$TC_{R,i,j}$ = number of NVRAM read access transactions of $Task_{i,j}$

$TC_{W,i,j}$ = number of NVRAM write access transactions of $Task_{i,j}$

In case of the interactive tasks, their instances have short execution time. All the workload parameters are generated from the exponential distribution. Table 2 shows parameter values used in our simulation.

Table 2. Parameters for Our Experiment

Parameter	Value
The number of cores	2, 4, 8
The number of queues	4
Quantum unit of each queue (Q1 ~ Q4)	Q1=100, Q2=200, Q3=400, Q4=800
Peak memory transaction rate	35 transactions/us
DRAM-to-NVRAM ratio	1:1
DRAM read/write latency (ns)	25
NVRAM read latency (ns)	67.5
NVRAM write latency (ns)	215
Window size	1/3 of queue length

For each task *i*, the first instance, $Task_{i,1}$, is inserted at the end of the top-level queue. If the task instance is completed within the time quantum of the given queue, it leaves the system. And the next instance is inserted at the tail of the same queue. However, if an instance uses all the time quantum of the given queue, it is pre-empted and inserted at the end of the next level queue. The following instance is inserted into the same queue of the preceding instance.

We compared the proposed scheduling scheme with traditional Round Robin (RR) and Multi-level Feedback Queue (MQ) scheduling algorithms. Figure 4 shows comparison of normalized average execution time by RR scheme. We can see that the MQ-HAMC outperforms RR and MQ about 5~10%. This is because the proposed MQ-HAMC tries not to delay the execution of tasks by avoiding the memory saturation situation. Figure 5 shows comparison of normalized average response time of interactive tasks by RR scheme. We can see that the MQ-HAMC outperforms RR and MQ about 12~19%. This is because MQ-HAMC gives higher priority to the interactive tasks. MQ scheme also provides faster response time than RR. But, its performance is worse than MQ-HAMC because it does not avoid the memory saturation situation.

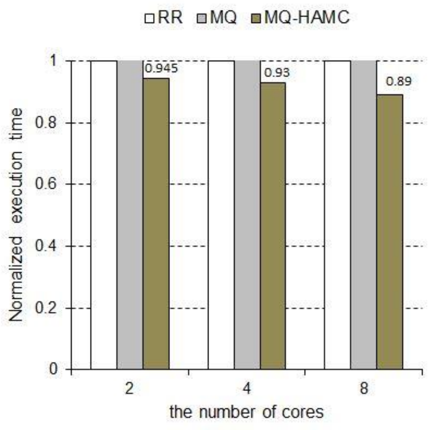

Figure 4. Comparison of Execution Time

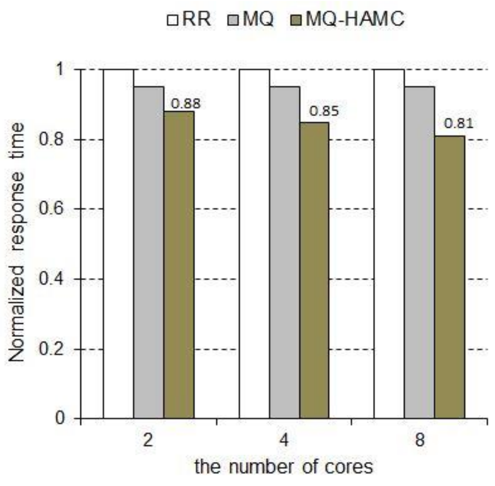

Figure 5. Comparison of Response Time

4. Conclusion

In this paper, we study a memory bandwidth-aware multi-core scheduling scheme considering next generation hybrid main memory and interactive tasks. Though there have been a lot of bandwidth-aware multi-core task scheduling schemes, these methods are not applicable with the DRAM/NVRAM hybrid main memory. This is because the bandwidth requirement cannot be represented by a unique memory transaction type. The proposed scheduling scheme considers different access latency of hybrid memory medium and tries to reduce the total execution time of tasks by avoiding the memory bandwidth saturation. Further, the proposed scheme considers interactive tasks for providing fast response time when scheduling tasks to execute. We showed through trace-driven simulation that proposed multi-core scheduling scheme outperforms legacy scheduling schemes such as round-robin and multi-level feedback queue scheme.

Acknowledgments

This work was supported by 2016 Research Fund of Myongji University. This paper is a revised and expanded version of a paper entitled “Issues on Multi-core Scheduling and NVRAM-based Hybrid Memory” presented at “The 10th 2016 International Interdisciplinary Workshop Series, Jeju, Korea, August 16-19”.

References

[1] G. Lim, C. Min and Y. Eom, “Load-balancing for Improving User Responsiveness on Multicore Embedded Systems”, Proceedings of the Linux Symposium, **(2012)**, pp. 25-33.
[2] J. Dai, W. Liu, X. Lin, Y. Ye, C. Xiao, K. Wu, Q. Zhuge and E. Sha, “User Experience Enhanced Task Scheduling and Processor Frequency Scaling for Energy-Sensitive Mobile Devices”, Proceedings of High Performance Computing and Communications, **(2015)** August 24-26.
[3] C. Antonopoulos, D. Nikolopoulos and T. Papatheodorou, “Scheduling Algorithms with Bus Bandwidth Considerations for SMPs”, Proceedings of International Conference on Parallel Processing, **(2003)** October 6-9.
[4] Merkel and F. Bellosa, “Memory-aware Scheduling for Energy Efficiency on Multicore Processors”, Proceedings of Conference on Power aware Computing and Systems, **(2008)**, pp. 1-5.
[5] E. Koukis and N. Koziris, “Memory and Network Bandwidth aware Scheduling of Multiprogrammed Workloads on Clusters of SMPs”, Proceedings of International Conference on Parallel and Distributed Systems, **(2006)**, pp. 345–354.
[6] D. Xu, C. Wu and P. Yew, “On Mitigating Memory Bandwidth Contention through Bandwidth-aware Scheduling”, Proceedings of International Conference on Parallel Architectures and Compilation Techniques, **(2010)**, pp. 237–248.

[7] H. Kim, D. Niz, B. Andersson, M. Klein, O. Mutlu and R. Rajkumar, R, "Bounding Memory Interference Delay in COTS-based Multi-core Systems", Proceedings of IEEE Real-time and Embedded Technology and Applications Symposium, **(2014)** April 15-17.

[8] D. Eklov, N. Nikoleris, D. Black-Schaffer and E. Hagersten, "Bandwidth Bandit: Quantitative Characterization of Memory Contention", Proceedings of International Symposium on Code Generation and Optimization, **(2013)**, pp. 1-10.

[9] J. Zhao, H. Cui, J. Xue, X. Feng, Y. Yan and W. Yang, "An Empirical Model for Predicting Cross-core Performance Interference on Multicore Processors", Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques, **(2013)**, pp. 201-212.

[10] X. Fan, Y. Sui and J. Xue, "Contention-Aware Scheduling for Asymmetric Multicore Processors", Proceedings of 21st International Conference on Parallel and Distributed Systems, **(2015)** December 14-17.

[11] W. Wang, T. Dey, J. Davidson and M. Soffa, "DraMon: Predicting Memory Bandwidth Usage of Multi-threaded Programs with High Accuracy and Low Overhead", Proceedings of International Symposium on High Performance Computer Architecture, **(2014)**, pp. 380-391.

[12] L. Barroso and U. Holzle, "The Case for Energy-proportional Computing", IEEE Computer, vol. 40, no. 12, **(2007)**.

[13] M. Qureshi, V. Srinivasan and J. Rivers, "Scalable High Performance Main Memory System using Phase-change Memory Technology", Proceedings of the 36th International Symposium on Computer Architecture, **(2009)**, pp. 24-33.

[14] C. Augustine, N. Mojumder, X. Fong, S. Choday, S. Park and K. Roy, "Spin-Transfer Torque MRAMs for Low Power Memories: Perspective and Prospective", IEEE Sensors, vol. 12, no. 4, **(2012)**, pp. 756-766.

[15] S. Senni, L. Torres, G. Sassatelli, A. Gamatie and B. Mussard, "Exploring MRAM Technologies for Energy Efficient Systems-on-chip", IEEE Journal on Emerging and Selected Topics in Circuits and Systems, vol. 6, no 3, **(2016)**, pp. 279-292.

[16] M. Ueki, K. Akeuchi, T. Yamamoto, A. Tanabe, N. Ikarashi, M. Saitoh, T. Nagumo, H. Sunamura, M. Narihiro, K. Uejima, K. Masuzaki, N. Furutake, S. Saito, Y. Yabe, A. Mitsuiki, K. Takeda, T. Hase and Y. Hayashi, "Low-power Embedded ReRAM Technology for IoT Applications", Proceedings of Symposium on VLSI Circuits, **(2015)** June 17-19.

[17] http://www.mram-info.com/ibm-demonstrated-11nm-stt-mram-junction-says-time-stt-mram-now

[18] Y. Zhang, J. Yang, A. Memaripour and S. Swanson, "Mojim: A Reliable and Highly-Available Non-Volatile Memory System", ACM SIGPLAN Notices, vol. 50, no. 4, **(2015)**, pp. 3-18.

[19] E. Kultursay, M. Kandemir, A. Sivasubramaniam and O. Mutlu, "Evaluating STT-RAM as an Energy-efficient Main Memory Alternative", Proceedings of IEEE International Symposium on Performance Analysis of Systems and Software, **(2013)**.

[20] H. Park, S. Yoo and S. Lee, "Power Management of Hybrid DRAM/PRAM-based Main Memory", Proceedings of the 48th Design Automation Conference, **(2011)**, pp. 59-64.

[21] W. Hwang and K. Park, "HMMSched: Hybrid Main Memory-aware Task Scheduling on Multicore System", Proceedings of International Conference on Future Computational Technologies and Applications, **(2013)**.

[22] H. Khouzani, C. Yang and J. Hu, "Improving Performance and Lifetime of DRAM-PCM Hybrid Main Memory through a Proactive Page Allocation Strategy", Proceedings of Design Automation Conference, **(2015)**, pp. 508-513.

[23] M. Qiu, Z. Chen, Z. Me.ging and X. Qin, "Energy-aware Data Allocation with Hybrid Memory for Mobile Cloud Systems", IEEE Systems Journal, **(2014)**, pp. 1-10.

[24] L. Ramos, E. Gorbatov and R. Bianchini, "Page Placement in Hybrid Memory Systems", Proceedings of International Conference on Supercomputing, **(2011)**, pp. 85-95.

[25] G. Dhiman, R. Ayoub and T. Rosing, "PDRAM: A Hybrid PRAM and DRAM Main Memory System", Proceedings of 46th Annual Design Automation Conference, **(2009),** pp. 664-469.

[26] S. Oh and Y. Ryu, "Multi-core Scheduling Scheme for Wireless Sensor Nodes with NVRAM-based Hybrid memory", Lecture Notes in Electrical Engineering, vol. 331, **(2014)**, pp.45-52.

[27] J. Hu, M. Xe, C. Pan, C. Xue, Q. Zhuge and E. Sha, "Low Overhead Software Wear Leveling for Hybrid PCM + DRAM Main Memory on Embedded Systems", IEEE Transactions on Very Large Scale Integration Systems, vol. 23, no. 4, **(2015)**, pp. 654-663.

[28] J. Shin and Y. Ryu, "Issues on Multi-core Scheduling and NVRAM-based Hybrid Memory", Asia-pacific Proceedings of Applied Science and Engineering for Better Human Life, vol. 4, no. 1, **(2016)**, pp. 5-8.

[29] R. Salhordeh and H. Asadi, "An Operating System level Data Migration Scheme in Hybrid DRAM-NVM Memory Architecture", Proceedings of Design, Automation & Test in Europe Conference & Exhibition, **(2016)** March 14-18.

Authors

Jeongmi Shin, she received her BS degree in Computer Engineering from Myongji University, Korea, in 2015, and she is currently studying a master’s degree in Security and Management Engineering from Myongji University since 2015. Her research interests include operating systems, information security and convergence security.

Yeonseung Ryu, he received his BS degree in Computer Science and Statistics from Seoul National University, Korea, in 1990, and his MS and PhD degrees in Computer Science from Seoul National University in 1992 and 1996, respectively. In 1996 he joined Samsung Electronics, Co. as a senior researcher. Since 2003, he has been with Myongji University, Korea, where he is currently a full professor in the Computer Engineering Department and Security and Management Engineering Department. From March 2009 to February 2010, he was a visiting scholar at the University of Minnesota, USA. His research interests include operating systems and convergence security.