Memos: A Full Hierarchy Hybrid Memory Management Framework

Lei Liu[1,2], Hao Yang[1,2], Yong Li[3], Mengyao Xie[1,2], Lian Li[2] and Chenggang Wu[2]
[1]Sys-Inventor Group, [2]State Key Lab. of Computer Architecture, ICT, CAS. [3]VMware, CA, US
Email: {liulei2010, yanghao2014, xiemengyao, lianli, wucg}@ict.ac.cn; yongl@vmware.com

***Abstract*—In this paper, we introduce memos, which integrates suitable memory management policies and schedules resources over the entire memory hierarchy in hybrid memory system. Powered by an OS kernel level monitoring tool, memos captures memory patterns online, and then leverages them to guide the memory page placement and data mapping. Experimental results show, on average, memos can benefit memory utilization, contributing to system throughput and QoS by 19.1% and 23.6%. Moreover, memos can reduce the NVM side memory latency by 3~83.3%, energy consumption by 25.1~99%, and benefit the NVM lifetime significantly (40X improvement on average).**

I. INTRODUCTION

IN the era of big data and cloud computing, modern computer systems are facing numerous new challenges, such as fast growing memory footprint, rapidly increasing energy consumption, high demand for throughput, etc. In a typical cloud-computing environment, it is critical to have a large capacity memory system that can provide fast data retrieval and operate at a low energy cost. Recently, the emerging Non-Volatile Memory (NVM) technologies bring an opportunity to build such hybrid memory systems [1,3,10,11,12]. However, efficiently managing such hybrid memory systems requires considerations of a variety of factors including the distinct memory characteristics, diverse workload behaviors, and architecture features, which poses new challenges for the memory management mechanism in modern Operating System (OS) [2,4,5,9,13,14,15,16,17].

Many studies have discussed memory management for "horizontal" hybrid memory system, which organizes DRAM and NVM side by side at the same level in memory hierarchy [1,11] through different memory channels. Based on this design, our study has the following insights: (1) Managing all levels in the entire memory hierarchy can bring enormous benefits, especially for hybrid memory systems. In particular, besides cache, memory channel is an interesting dimension to consider as balancing, scheduling and isolating behaviors [7], which can bring distinct impacts on DRAM and NVM. (2) For a hybrid DRAM-NVM main memory system, an effective memory management scheme should be aware of memory access characteristics, since the performance of NVM and DRAM are sensitive to certain behaviors such as memory footprint, access hotness/hotspot, read/write activities, reuse time, etc. (3) An ideal mechanism for managing hybrid memories should balance memory utilization across memory banks/channels, and efficiently migrate data between NVM and DRAM for dynamically changing access patterns.

Based on the above considerations, we introduce memos, a uniform memory management framework that can flexibly and efficiently manage hybrid memory system. Firstly, memos explores the synergy of all memory levels in the entire memory hierarchy including the last level cache (LLC), channels and memory banks, which collectively have a predominant impact on hybrid memory system performance. Secondly, powered by SysMon, an OS-level application behavior monitoring tool, memos can obtain and leverage a rich source of memory accessing information including memory utilization, hot/cold pages, read/write patterns, and page reuse time, etc. Finally, memos provides high memory bank-level parallelism by rebalancing memory utilization across hot and underutilized banks, thus facilitating data movements between DRAM and NVM, and improving the overall bandwidth utilization. We list our contributions:

(1) Full Hierarchy Hybrid Memory Management. For the first time, memos achieves a full hierarchy, application demand driven memory management covering cache, channels, and DRAM/ NVM banks, greatly benefiting the utilization of hybrid memory system and avoiding sub-optimal solutions caused by previous approaches. Further, we design a write-interval based migration scheme that moves pages across channels when memory behavior changes.

(2) Dynamic Memory Bank/Channel Rebalancing. We propose a memory bank rebalancing approach to provide higher bank parallelism and boost data transfer rates between NVM and DRAM by reducing bank-level interferences. Moreover, we also balance memory requests across channels horizontally to achieve higher memory utilization.

(3) Open Source SysMon. We open sourced SysMon, a kernel tool used to collect memory footprint, page hotness, reuse patterns, etc., for any running process. Specifically, SysMon can detect page level reads/writes and bank balance behaviors, which are critical to consider in hybrid memory environment.

(4) Real Implementation. We implement memos in Linux kernel 2.6.32.15. Moreover, we further deploy a new emulation platform for hybrid memory system on a real multi-core machine by using the channel-partitioning approach [7].

II. MEMOS AND MCHA

A. Design Overview

Fig.1 illustrates the overall design for our Multi-Channel Horizontal memory Architecture (MCHA), which combines NVM and DRAM horizontally. Channel index bits in PFN determine which channel, hence which type of memory (NVM or DRAM), a memory access is serviced. Based on MCHA, we design *memos* in the OS kernel. The core of *memos* is a Decision Policy Mechanism (DPM), which includes memory management policies, and provides the interfaces that interact with other kernel modules. Leveraging runtime information captured by SysMon (i.e. hot/cold pages, read/write features and reuse distance, etc.), DPM firstly determines memory affinity mapping. DPM also makes decision regarding when and how to migrate the physical pages to another sub-memory system. Moreover, memos includes effective state-of-the-art memory policies into its core mechanism. DPM enables different cache policies for different memory channels by reserving segments in LLC for NVM and DRAM channel respectively, and optimizes cache and

This work is supported by NSF of China under grants No. 61502452, 61402445, 61303051, 61303052 and 61303053.

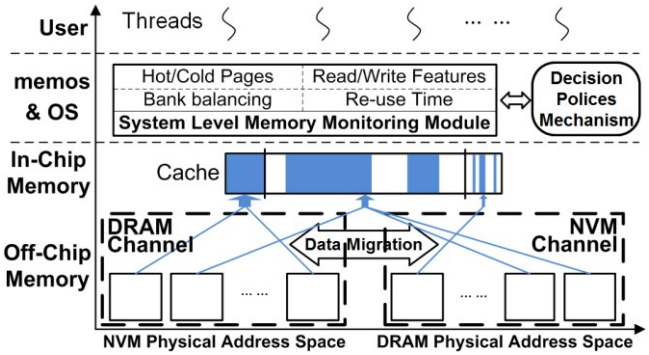

Fig.1: The overview of our design (memos and MCHA).

main memory utilization vertically when mapping pages. Finally, for a specific sub-memory system, DPM employs the proper memory policies (partitioning and rebalancing) based on the characteristics of incoming memory requests.

B. SysMon: Inner-OS Application Profiling Tool

We design SysMon[1], an application profiling tool, to collect application characteristics online. To determine page hotness, SysMon clears and checks (i.e., sampling) page access bit in PTE in continuous sampling windows (passes). In each pass, a given number of samplings (i.e., 200 in default in Fig.4) is performed. Using access bit can obtain the page-level hotness and the reuse time. To capture WD/RD[2] patterns, SysMon examines dirty (also in PTE) and access bits in a sampling window, and then calculates the weighted ratio of reads and writes. By examining the bank indexing bits in the physical address and counting hot pages assigned to each memory bank, SysMon can also calculate variation and bank imbalance factor. Besides, using CPU performance-monitoring unit, SysMon can obtain per-channel bandwidth information with low overhead.

C. Managing Hybrid Memory System

Channel Allocation: Channel resource in memos is primarily used as a way to designate different types of memory resources and is scheduled based on an application page's preference for DRAM or NVM. As such, our scheme attempts to direct hot pages into DRAM channel, especially for those with WD features. RD intensive pages can be directed through NVM channel onto NVM without hurting performance. Cold pages are kept in NVM to save energy and reserve DRAM capacity for hot and WD pages.

To improve the channel utilization, our design also balances the bandwidth across different channels. The core idea is, when a significant channel imbalance [7] is detected due to an excessive number of pages being assigned to DRAM, our scheme will trigger page migration to utilize the underutilized NVM channel to achieve a better overall performance. Memos will stop migrating pages from DRAM to NVM when the DRAM channel bandwidth utilization begins to drop. As such, the DRAM channel bandwidth utilization is always maximized, and the overall bandwidth is guaranteed to be improved.

Cache Allocation Process: In our system, LLC resource is partitioned into dedicated slices to serve the memory requests directed onto DRAM and NVM channel respectively. As memory intensive pages with thrashing behaviors are mapped into DRAM for performance, isolating them in LLC can help to avoid thrashing other data, such as the data from NVM channel.

[1] SysMon with beta version is now open sourced. The core idea is now adapted in VMware VCenter Lab., and Huawei. And, all of the information in section 3 is obtained by SysMon.

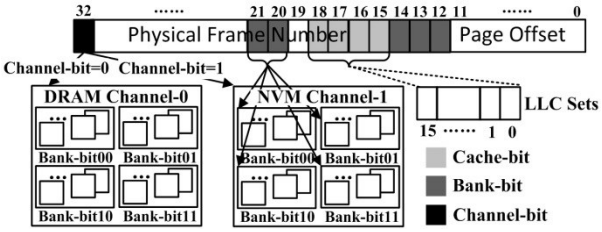

Fig.2: Address mapping across cache-channel-bank of full hierarchy vertical management for hybrid memory system. Based on Intel i7-860.

Moreover, a larger LLC quota to filter NVM accesses can hide the longer NVM latency. In practice, memos is monitoring memory patterns online, and adjusts the cache assignment to each channel dynamically. Taking NVM channel as an example, when WD pages are migrated onto it, memos will enlarge the cache capacity of NVM channel at the same time to filter the expensive write operations. The allocation process works like an expending "balloon". Besides, in some cases, the data (especially these WD pages) from both NVM and DRAM channels, will be merged together in a larger LLC quota, while leaving two reserved smaller amount of LLC quotas (showed in Fig.1) for pages with streaming and rarely-touched behaviors.

Bank Partitioning and Balancing: Memos uses bank partitioning to reduce memory interferences across threads. And, hot pages are migrated from highly utilized banks to lower ones to balance the overall bank utilization. Doing so is particularly effective in NVM due to its longer latency raised by memory interferences on bank among threads and hotspots. Even for DRAM, a balanced bank parallelism can hide memory latency and is crucial to achieve desirable performance for both memory-limited and intensive cases [6,7,8].

Full Hierarchy Management for Hybrid Memory: By combining all of the above, we construct a full hierarchy memory management strategy that manages resources from the upper levels (i.e., cache), through the middle levels (i.e., channels), to the lower levels of the memory banks. Application's pages can be serviced by any flexible combination of a memory channel, banks, and LLC sets. Resources can be adjusted based on applications' needs, or page level memory access patterns. Enabling such a full hierarchy approach provides a large design space and optimization opportunity, particularly for system with hybrid memories.

Leveraging Page-Coloring [6,7,8], we construct a framework that manages resources across the entire memory hierarchy. As illustrated in Fig.2, for a 4K size page (0~11 bits denotes the offset within the page) on a typical 64-bit architecture, bit 32 is used to dictate which channel to use to service a memory request. Therefore, by selecting a physical page with a specified value (0 or 1) at bit 32, we can control which channel, and consequently which memory segment (DRAM or NVM) to accommodate the page. For cache resource, each unique combination of cache index bit values (bits 15, 16, 17, 18 shown in Fig.2), or cache-set color, dictates a slice (1/16) of LLC resource. Thus, we can adjust cache resource allocation and utilization by leveraging these bits. Fig.3 gives an example of scheduling 1/16 cache to NVM channel by mapping data vertically to pages, whose 15~18 bits are with the value of 1 in blue color. Moreover, for the memory bank resource in both NVM and DRAM channel, memos monitors the bank utilization and enables the bank

[2] Read-Domain (RD) and Write-Domain (WD) indicate whether read or write operations are predominant.

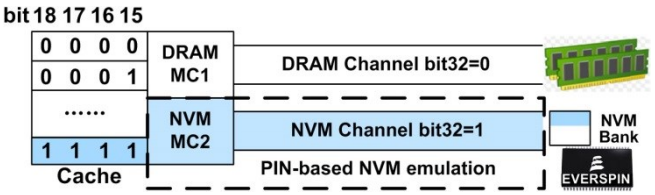

Fig.3: Illustration of the MCHA emulation platform.

TABLE 1: The Parameters of memory hierarchy and NVM system.

L1 cache	32KB instruction cache, 32KB data cache, 64B cache block
L2 cache	256KB data cache, 64B cache block
L3 cache	8MB data cache, 64B cache block
NVM memory system	4GB, 1 rank(8 banks), 4KB page size, t_{CLK}=1.5ns,t_{CL}=10ns, t_{RCD}=20ns,t_{RP}=23ns,t_{WR}=160ns, read(write) energy= 23.293 (57.686)nJ, standby power=584.764mW, endurance=10^6

scheduling using the bank index bits (bit 20,21,12,13 and 14 in Fig.2). Usually, bit 20, 21 are used as a combination to uniquely dictate a group of 8 banks (called a bank-group color) for one application, and memos can assign additional banks groups by using more than one bank-group colors. In general, given L Bank-bits, M Cache-bits and N Channel-bits, and we can use i Bank-bits, j Cache-bits and k Channel-bits to generate a resource allocation policy denoted as (i,j,k), where 0 < i <= L, 0 < j <= M and 0 < k <= N.

Deployment of MCHA: We emulate MCHA (in Fig.3) by deploying a channel-partitioning mechanism [7] to divide the memory address space into two segments, on a typical multi-core server with Intel i7 series CPU and dual-channel with 8GB DDR3 main memory. We mark the channels as DRAM and NVM channel respectively, and set up a PIN-based simulation for NVM channel side by employing modified DRAMSim2 (with NVM parameters) to simulate the NVM performance (i.e. energy, and latency). As PIN tool gets the memory traces with cache behaviors, we further use a cache simulator, DineroIV, to filter the memory accesses on cache hierarchy, and input them to NVM simulator. Table 1 shows the parameters in detail.

D. Data Migration between NVM and DRAM

Migration Mechanism: Fig.4 highlights memos's migration mechanism. In summary, memos tries to segregate hot pages with WD label to DRAM, and keep RD pages as well as cold pages in NVM. At run time, memos visits each application's physical page by using SysMon module, and migrates them when necessary. In Fig.4, the step 1 marks all the pages for a specific application with WD or RD labels. Begins from step 2, memos checks each page, and decides whether to migrate it or not according to the two parameters, NPass and WD_interval. Npass is the number of sampling passes that have been performed, and WD_interval is the distance to the last sampling pass when the page is labeled as WD. After, memos marks pages as "will-migrate" or not in step 3. Note that real migration will start at step 4 after NPass finishes. This design is useful in practice, because, during the following sampling period, the migration label may change due to the change of memory patterns. The entire mechanism works as a loop periodically.

Avoiding Bank Conflicts for Efficient Data Migration: In our mechanism, the bank-level utilization information is always monitored, and such information is used by page placement routine in memos to avoid banks with hot spots. When moving hot pages from NVM to DRAM, these pages will be mapped to

[3] We tune the WD_interval (=2 and 3) parameters in this experiment. Memcached's WD_interval is larger than other applications. Memos provides interface to users, and we can tune the parameters accordingly when environment changes.

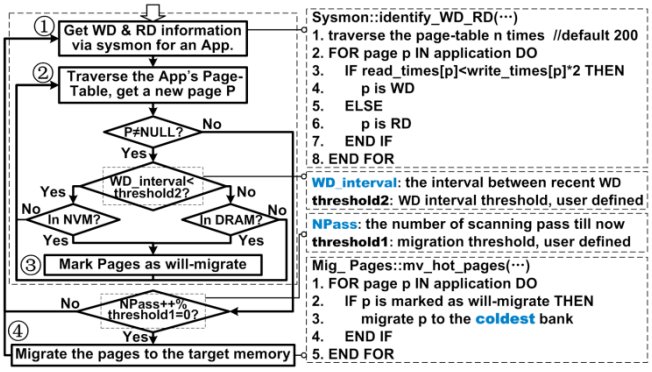

Fig.4: Page migration mechanism based on WD history.

underutilized banks (coldest) for better balance, and vice versa. Doing this, memos avoids bank conflicts, and the memory migration will be benefited from better bank parallelism. Even for NVM, ideal bank parallelism can hide the expensive access latency, as NVM systems often provide a large amount of banks.

III. EVALUATION

A. Effectiveness of Managing Hybrid Memory

Fig.5 shows the effectiveness of memos based on HOT/COLD rate that denotes the ratio of hot pages to cold pages, and WD/RD rate that shows the ratio of write operations to read operations (at page level) on DRAM and NVM, respectively. Taking hmmer as an example, DRAM HOT/COLD rate increases stably over time, indicating that hot pages are migrated to DRAM continuously. Meanwhile, DRAM WD/RD rate shows a similar but a sharper increasing trend. By contrast, HOT/COLD and WD/RD rates in NVM exhibit consistently declining trend, illustrating that pages with RD features and relative low memory access frequency are moved to and kept in NVM. The bottom-right subfigure shows the metrics for a multi-programmed workload that includes several SPEC applications. We further test Memcached[3], whose active working footprint is small but changes frequently. Fig.5 shows memos can handle these changes by migrating hotspots into DRAM to benefit the overall performance. Moreover, memos also tries to distinguish and segregate WD and RD pages across different channels, and thus the WD/RD rate is always higher in DRAM channel, though Memcached often exhibits very unstable write/read features. On average, the overall HOT/COLD (85.4%), and WD/RD (83.2%) rate in DRAM channel is larger than these in NVM channel.

Energy, Latency and Lifetime: We evaluate the energy savings and memory latency reduction using the emulation system in Fig.3. For mcf, the dynamic energy in NVM channel is significantly reduced from 2.13 mWatt to 0.001 mW, while the

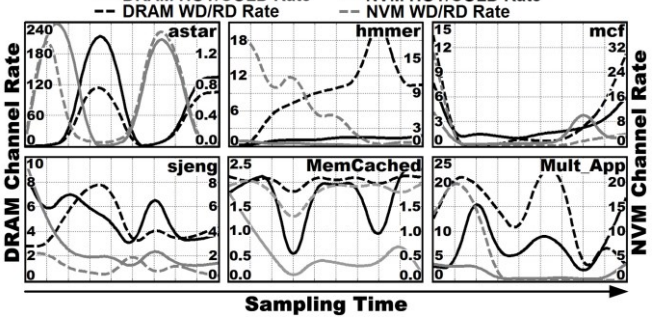

Fig.5: Effectiveness of memos in resource scheduling.

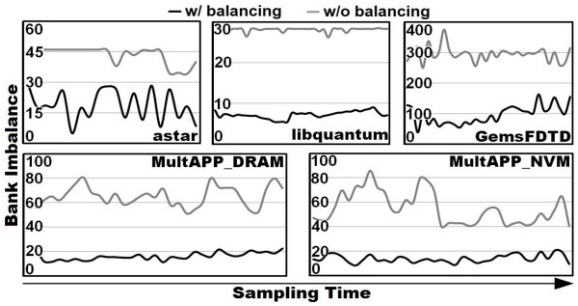

Fig.6: Bank-level parallelism improvement (lower is better).

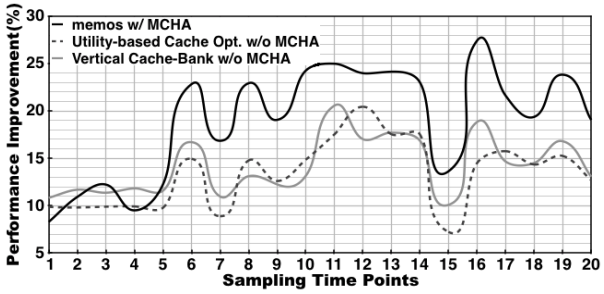

Fig.7: Overall performance throughput improvement.

average memory access latency is reduced from 102 ns to 55.66 ns. For xalan, the energy reduction is from 1.216 mW to 0.304 mW, and the memory latency reduces from 78 ns to 61.21 ns. For the widely used Memcached, the energy consumption is reduced from 0.211 mW to 0.046 mW, while the memory latency is also improved to 60 ns from 62 ns, on average. For lifetime calculation, we model the NVM memory with cell endurance of Endurance_X (10^6 in Table 1). The NVM memory is operated at memory block granularity (i.e., 64 bytes). The overall NVM manages to achieve an overall lifetime, which is 95% of the average NVM cell lifetime. Experimental results show that memos on MCHA can improve the NVM life by 40x (up to 500x) against random memory channel interleaving scheme[4] on platform w/o MCHA and memos, on average.

B. Rebalancing and Caching Effects

Mentioned before, memos rebalances hot pages across memory banks for NVM and DRAM channel by combining both of the rebalancing hot pages across banks for a specific application, and partitioning memory resource across different threads. In single thread cases, seen from Fig.6, by rebalancing hotness across banks, the imbalance (measured by standard deviation of the number of active pages between hottest and coldest banks) is significantly reduced by around 60~70%, indicating that the bank parallelism is significantly improved. In the cases where several applications are running on MCHA, memos attempts to map migrated pages to the coldest bank across channels, contributing significantly to reducing bank conflicts and balancing. From the Multapp_DRAM/NVM sub-figures, the standard deviation of bank imbalance drops to around 20 stably in both DRAM and NVM channel, indicating memos work well in multi-programmed cases for bank balancing.

C. Overall Performance of Memos on MCHA

We compare memos on MCHA with some typical resource scheduling approaches, i.e. cache-bank vertical management [8,9] and utility-based cache partitioning without channel partitioning on our platform. Fig.7 illustrates the experimental results[5]. Memos with MCHA achieves an average of 19.1% performance gain, and outperforms the state-of-the-art approach by 7.3% (up to 11%). Memos performs well on most of the sampling points, especially points 6, 8, 10, 11, and 16. We further find that at these points pages with streaming accesses from thrashing applications (e.g. libquntum) are well confined into a dictated channel with a small segment of LLC, thus minimizing interferences among conflicting applications. At point 16, memos achieves the highest performance gain (around 28%), due to streaming and rarely touched pages are serviced by different channels with a small LLC quota, while frequently accessed pages from all channels are allocated to use a larger amount of LLC, and channel-level bandwidth utilization are nearly balanced. Moreover, memos improves QoS (indicated by Max Slowdown) [6,8,11] significantly (23.6% on average).

Take the case in point 11 as an example, memos on MCHA can bring 34.1% benefits for QoS, while the utility-based cache, and vertical cache-bank partitioning with dual-channel interleaving scheme improve QoS by13.2%, 19.4%. Our new mechanism exhibits obvious advantages, and outperforms them by 20.9% and 14.7%, respectively. In general, memos performs well in memory intensive and high interference cases by isolating interfering memory patterns into dedicated memory channels, appropriate amount of banks and cache segments. Memos with MCHA achieves an even better performance, because the full hierarchy mechanism reduces memory interferences at each level of the memory hierarchy, outperforming the vertical cache-bank and other single level cache optimizations.

[4] We use channel-level interleaved mapping scheme, which maps pages evenly across memory channels, as baseline in lifetime, energy, and performance experiments. This scheme is widely use on multi-core systems with multi-channel configuration.

[5] The baseline in original Buddy System in Linux kernel, running on normal platform with channel-level interleaving page mapping scheme (without channel partitioning).

REFERENCES

[1] G. Dhiman, R. Ayoub, T. Rosing, "PDRAM: A Hybrid PRAM and DRAM Main Memory System," In DAC, 2009.

[2] Y. Hu et al, "Towards Efficient Server Architecture for Virtualized Network Function Deployment: Implications and Implementations," In MICRO 2016

[3] E. Kultursay, M. Kandemir, A. Sivasubramaniam et al, "Evaluating STT-RAM as an Energy-Efficient Main Memory Alternative," In ISPASS, 2013.

[4] S. Li, T. Hoefler, M. Snir, "NUMA-aware shared-memory collective communication for MPI," In HPDC, 2013.

[5] J. Liu, B. Jaiyen, R. Veras, O. Mutlu, "RAIDR: Retention-Aware Intelligent DRAM Refresh," In ISCA, 2012.

[6] L. Liu, Z. Cui, M. Xing et al, "A Software Memory Partition Approach for Eliminating Bank-level Interference in Multicore Systems," In PACT, 2012.

[7] L. Liu, Z. Cui, Y. Li et al, "BPM/BPM+: Software-based Dynamic Memory Partitioning Mechanisms for Mitigating DRAM Bank-/Channel-level Interferences in Multicore Systems," In ACM TACO, 2014.

[8] L. Liu, Y. Li, Z. Cui, et al, "Going Vertical in Memory Management: Handling Multiplicity by Multi-policy," In ISCA, 2014.

[9] L. Liu et al, "Rethinking Memory Management in Modern Operating System: Horizontal, Vertical or Random?" In IEEE Trans. on Computers (TC), 2016.

[10] S. Lee, et al, "CLOCK-DWF: A Write-History-Aware Page Replacement Algorithm for Hybrid PCM and DRAM Memory Architectures," In IEEE Trans. on Computers (TC), 2014.

[11] O. Mutlu, "Main Memory Scaling: Challenges and Solution Directions," In More than Moore Technologies for Next Generation Computer Design, 2015.

[12] M. K. Qureshi, S. Gurumurthi, B. Rajendran, "Phase change memory: From devices to system," In Synthesis Lectures on Computer Architecture, 2011.

[13] L. E. Ramos, E. Gorbatov, R. Bianchini, "Page placement in hybrid memory systems," In ICS, 2011.

[14] H. Seok, Y. Park, K. H. Park, "Migration Based Page Caching Algorithm for a Hybrid Main Memory of DRAM and PRAM," In SAC, 2011.

[15] L. Wang et al, "Articulation points guided redundancy elimination for betweenness centrality," In PPoPP, 2016.

[16] H. B. Yoon et al, "Row Buffer Locality Aware Caching Policies for Hybrid Memories," In ICCD, 2012.

[17] F. Lv, L. Liu et al, "WiseThrottling: a new asynchronous task scheduler for mitigating I/O bottleneck in large-scale datacenter servers," J. of Supercomputing, 2015.