HMMU: A Hardware-based Hybrid Memory Management Unit

Fei Wen, Mian Qin, Paul V. Gratz, *Senior Member, IEEE,* and A.L. Narasimha Reddy, *Fellow, IEEE*
Department of Electrical and Computer Engineering, Texas A&M University
Email: fei8wen@gmail.com; celery1124@tamu.edu; pgratz@gratz1.com; reddy@tamu.edu

The demand for memory capacity increases dramatically for mobile applications. Mobile device manufacturer rapidly expanded the DRAM size, catering to such demand. For example, the DRAM capacity of the flagship phones from the Samsung Galaxy S series have expanded by 16X over the past ten years. While this approach has been largely successful to date, the size of DRAM is constrained by both cost/economics and energy consumption. Unlike data centers, mobile devices are highly cost-sensitive and have a highly limited energy budget. Moreover, the DRAM technology has a substantial background power, constantly consuming energy even in idle due to its periodic refresh requirement, which scales with DRAM capacity. Therefore, the approach of provisioning more DRAM is not sustainable and hard limits will soon be hit on the scaling of the future mobile memory system. The emergence of several Non-Volatile-Memory (NVM) technologies, such as Intel 3D Xpoint, memristor, Phase-change-memory(PCM), provides a new avenue to address this growing problem. These new memory devices promise an order of magnitude higher density [2] per cost and lower static power consumption than traditional DRAM technologies, however, their access delay is significantly higher, typically also within one order of magnitude of DRAM. Further, these new technologies show significant overheads associated with writes and are non-volatile. Thus, these emerging memory technologies present a unique opportunity to address the problems of growing application workload footprints with hybrid memory systems composed of both DRAM and emerging NVM memories.

Ideally, one would prefer to place data objects with high-locality and access frequency in the DRAM to ensure good performance. Meanwhile objects of larger size shall be placed on the NVM, which has higher capacity and could help reduce or avoid swapping between memory and storage devices. Further, heavily written objects should likely be placed on the DRAM, to avoid the high write overheads of NVM devices in terms of both latency and power consumption (depending on particular technology, this also helps to prolong the NVM device lifetime).

Originally published in "Hardware Memory Management for Future Mobile Hybrid Memory Systems" by F. Wen, M. Qin, P.V. Gratz and A.L.N. Reddy, 2020. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Volume 39, 3627 - 3637, Copyright 2020 by IEEE. [1]

I. Pure Hardware Management without OS Intervention

One must also reconsider the mechanism to execute the data migration. In the era of magnetic disks and flash drives, moving data between memories and storage was managed by operating system. OS-driven page migrations incur significant overheads: the process is usually triggered by a system interrupt, followed by context switch and kernel interrupt handling. A TLB shootdown might also be necessary after the page mapping was updated. The associated delays, which was once hidden behind the access latencies of traditional storage media, now become the dominant factors as the NVM device has much shorter access latencies.

Some existing work has begun to explore system design for emerging hybrid memories. Broadly this prior work falls into one of two categories, first, some advocate using DRAM as a pure hardware managed cache for NVM [3], [4]. This approach implies a high hardware cost for metadata management and imposes significant capacity and bandwidth constraints. Second, some have advocated for a purely software, OS managed approach [5]–[7]. As we discussed previously, this approach implies significant slowdowns due to software overhead of the operating system calls.

II. Flexible Granularity

Our proposed hybrid memory management unit (HMMU) is transparent to the user and as well as the operating system, thus, it does not incur the overheads of management of OS based approaches. The HMMU manages both DRAM and NVM memories in flat address space to leverage the full capacity of both memory classes. Most of the memory space is managed in page size granularity, thus we avoid the tremendous tag store overhead that comes with the cache-line level data management. The HMMU has its own page table that redirects each physical page address to the corresponding memory device frame address. Each table entry also has several metadata bits to count the number of recent read/write accesses. Owing to the hardware efficiency, the HMMU is able to instantly update these profiling counters upon each access, and make decisions on page migration coordinately.

Various applications could have widely different data access patterns: those with high spatial locality may access a large number of adjacent blocks of data; while others may have a larger stride between the requested addresses. For applications with weak or no spatial locality, there is very limited benefit

to moving the whole page of data into fast memory, as most of the non-touched data may not be used at all. Based on this observation, our solution reserves a small portion of the available DRAM space to use as a hardware-managed cache, for sub-page size block management, which manipulates the data placement and migration in finer granularity. Our design supports flexible block size, ranging from the regular cache line size of 64B, up to 1024B. After comparing the results by sweeping all possible block sizes, we found the optimal choice to be 512B. We leverage spacial locality patterns seen in real application workloads to reduce unnecessary data migration, saving energy and NVM lifetime. Upon the first accesses to a slow memory page, instead of moving the whole page into fast memory, we will only move the requested block of that page into the "cache" zone in fast memory. We then keep track of the total number of cached blocks belonging to every page. Only after the count of cached blocks meets a certain threshold will we swap the whole page to fast memory.

Data locality could vary starkly across different phases of application.Therefore we set aside several metrics to evaluate the efficiency of page swap. We calculate the average number of references to the most recently visited pages. When these numbers increase, the threshold is lowered to allow more data moves. The threshold is raised to suppress migration between memories to save time/energy when these numbers decrease. This is commonly seen in streaming applications which traverse a vast range of data only once.

III. FPGA-based Emulation Experiment

Evaluating the proposed system presents several unique challenges because we aim to test the whole system stack, comprising not only the CPU, but also the memory controller, memory devices and the interconnections. Much of the prior work in the processor memory domain relies upon software simulation as the primary evaluation framework with tools such as Champsim [8] and gem5 [9]. However, detailed software simulators that meet our goals impose huge simulation time slow-downs versus real hardware. Moreover, there are often questions of the degree of fidelity of the outcome of arbitrary additions to software simulators [10].

Thus, we elected to emulate the HMMU architecture on an FPGA platform. FPGAs provide flexibility to develop and test sophisticated memory management policies while its hardware-like nature provides near-native simulation speed. The FPGA communicates with the ARM CortexA57 CPU via a high-speed PCI Express link, and manages the two memory modules (DRAM and NVM) directly. The DRAM and NVM memories are mapped to the physical memory space via the PCI BAR (Base Address Register) window. From the perspective of the CPU, they are rendered as available memory resource same as other regions of this unified space. Note that the CPU caching is still enabled on the mapped memory space, due to the PCIe BAR configurations. Our platform emulates the NVM access delays by adding stall cycles to the operations executed in FPGA to access external DRAM. We measured the round trip time in FPGA cycles to access external DRAM DIMM first, and then scaled the number of stall cycles according to the speed ratio between DRAM and 3D Xpoint. Hence the platform is not constrained to any specific type of NVM, but rather allows us to study and compare the behaviors across any arbitrary combinations of hybrid memories. The experiment results show that our project has the following advantages compared to previous work:

- With a ratio of 1/8 DRAM vs 7/8 NVM, we achieved 88% of the performance of an untenable full DRAM configuration, with 39% less energy consumption.
- Compared to inclusive DRAM caches, we preserve the full main memory capacity for the user applications.
- Parallel access to both the DRAM and NVM is supported, rendering a higher effective memory bandwidth. This also helps to suppress the excessive cache insertion/replacements and prevent cache thrashing.
- The data placement and migration are executed by hardware. This eliminates the long latency incurred by the OS managed virtual memory swap process.
- To obtain optimal performance with applications with various localities, we created a combined management policy that addresses data at page and sub-page block granularity, dependent on locality.

References

[1] F. Wen, M. Qin, P. V. Gratz, and A. L. N. Reddy, "Hardware memory management for future mobile hybrid memory systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2020. [Online]. Available: https://doi.org/10.1109/TCAD.2020.3012213

[2] J. Choe, "Intel 3d xpoint memory die removed from intel optane pcm," 2017, https://www.techinsights.com/blog/intel-3d-xpoint-memory-die-removed-intel-optanetm-pcm-phase-change-memory.

[3] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable high performance main memory system using phase-change memory technology," in *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ser. ISCA '09. New York, NY, USA: ACM, 2009, pp. 24–33.

[4] C. C. Chou, A. Jaleel, and M. K. Qureshi, "Cameo: A two-level memory organization with capacity of main memory and flexibility of hardware-managed cache," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, Dec 2014, pp. 1–12.

[5] A. Hassan, H. Vandierendonck, and D. S. Nikolopoulos, "Software-managed energy-efficient hybrid dram/nvm main memory," in *Proceedings of the 12th ACM International Conference on Computing Frontiers*, ser. CF '15. New York, NY, USA: ACM, 2015, pp. 23:1–23:8.

[6] V. Fedorov, J. Kim, M. Qin, P. V. Gratz, and A. L. N. Reddy, "Speculative paging for future nvm storage," in *Proceedings of the International Symposium on Memory Systems*, ser. MEMSYS '17. New York, NY, USA: ACM, 2017, pp. 399–410.

[7] Z. Wang, Z. Gu, and Z. Shao, "Optimizated allocation of data variables to pcm/dram-based hybrid main memory for real-time embedded systems," *IEEE Embedded Systems Letters*, vol. 6, no. 3, pp. 61–64, Sept 2014.

[8] ChampSim, "Champsim," 2016, https://github.com/ChampSim/ChampSim.

[9] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011.

[10] T. Nowatzki, J. Menon, C. Ho, and K. Sankaralingam, "Architectural simulators considered harmful," *IEEE Micro*, vol. 35, no. 6, pp. 4–12, Nov 2015.