A comparison of modern memory management schemes in HPC

Sina Karimi
Boston University
United States
sikarimi@bu.edu

Kurt Keville
MIT
United States
klk@mit.edu

ACM Reference Format:
Sina Karimi and Kurt Keville. 2024. A comparison of modern memory management schemes in HPC. In *The International Symposium on Memory Systems (MEMSYS '24), September 30–October 03, 2024, Washington, DC, USA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3695794.3695814

Data-intensive applications are gaining more attention, and the requirements for main memory increase at least linearly. Unfortunately, DRAM scaling is at its limit. To solve this problem, researchers have suggested new types of memory, such as high-bandwidth memories (HBM), non-volatile memories (NVM), compute express link-based memory (CXL), and other emerging technologies. Each type of memory has its own distinct characteristics and advantages. Therefore, future computer systems are anticipated to have multiple types of memory.

Using the two memory technologies at the same time allow for more flexibility of the applications while allowing us to take advantage of the performance of the new memory. Unfortunately, new emerging memory technologies, while providing higher density, show less performance compared to DRAM. DRAM shows less latency while providing more bandwidth. In order to maintain DRAM's performance while taking advantage of the higher density of the new emerging memories, researchers have proposed heterogeneous memory architectures. These architectures employ two memory technologies simultaneously to increase the system's memory capacity while providing performance close to DRAM memories. However, naively using these systems can result in performance degradation for the applications due to the performance of the slower memory.

Memory management is an important part of heterogeneous memory systems. A heterogeneous aware memory management system can help programs take advantage of heterogeneous memory systems transparently without changing the applications. This can be especially helpful to data centers where customers may not be willing to modify their applications. The important tasks for the memory management system can be considered page placement and page migration. Due to the difference in the performance of different memories in the system, it is important to decide where each piece of data should be placed. Pages that are being accessed frequently (hot pages) should be placed in the faster memory to take advantage of the bandwidth there while pages that are rarely accessed (cold pages) should be placed in slower memory which provides more capacity.

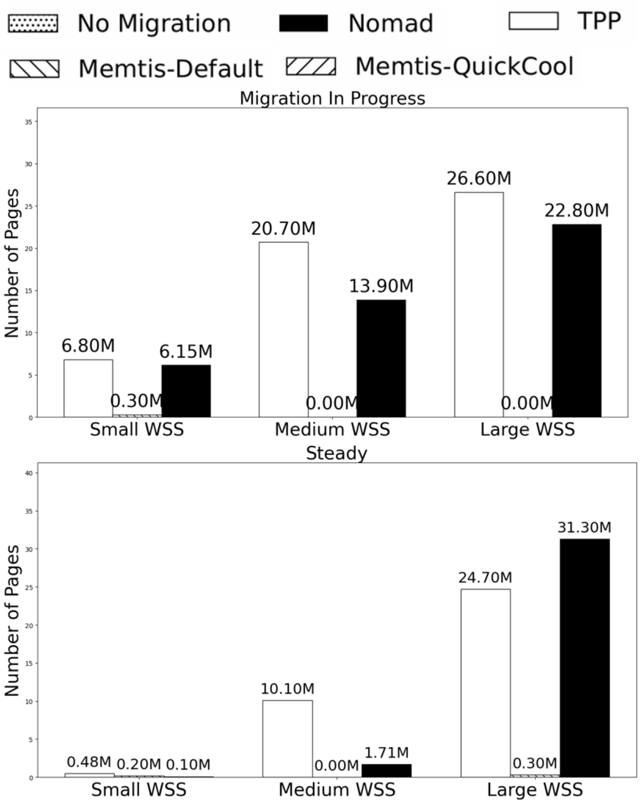

Figure 1: Number of pages being migrated in different phases of migration for Memtis, TPP, and Nomad according to data from [5] for a benchmark with different working set sizes (WSS)

Page migration is an important aspect of heterogeneous memory systems. Due to the dynamic behavior of applications, the behavior of pages in memory can also change. Cold pages can become hot and vice versa. This dynamic behavior requires runtime management of the memory pages to identify behavior changes in memory and migrate cold pages to slow memory and hot pages to fast memory. This progress requires three stages: memory page access monitoring, migration policy, and migration mechanism.

Page access monitoring can be challenging in software. There are three methods for memory access monitoring that have been used in the literature. Page table scanning periodically scans the accessed bit of the page table entries. The accessed bit will be set by hardware whenever a page is being accessed and can only be

Table 1: Comparison of Memory Access Methods

Method	Advantage	Drawback
PTE Scanning	Implemented in kernel	As the size of memory increases, the delay also increases. High accuracy requires high overhead
Page Fault	Very precise. Captures recency	Significant increase in page access latency
Hardware Sampling	Does not affect critical path for memory access	Miss very hot pages. High accuracy requires high overhead

reset by software. By periodically scanning page table entries, we can identify pages that are being frequently accessed. However, higher accuracy requires more frequent scanning which in turn causes more overhead for the system. It's been reported that as the size of the memory increases, the scanning process can take a significant amount of time [4]. Another method is to cause page faults when accessing a page. By capturing these page faults and recording accesses, we can accurately find the pages that are being accessed. This technique has been used in many papers with different implementations. [1, 3] Although we can get very accurate results about the frequency and recency of the access with page faults, this method increases the critical path for accessing pages which increases the latency of page access. Finally, hardware sampling is the latest method, which can sample from instructions that experienced LLC or TLB misses and record their virtual address to identify hot and cold pages.[2] Although this method does not increase the critical path of the memory access, it can cause overhead in the system. It's been reported that this method cannot perform well for the pages that are very frequently accessed.[5]

Many researchers have proposed different page migration mechanisms and policies, such as TPP[3], Nimble[6], Memtis[2], and NOMAD[5]. These methods usually consist of a mechanism to detect the temperature of memory pages and decide whether to migrate the page to another memory. The proposed solutions for page migration assume that the size of hot data is always smaller than the fast-tier memory. When the size of hot data exceeds the capacity of fast-tier memory, these proposed methods perform worse than not migrating data due to memory thrashing. Nomad has addressed this issue using non-exclusive memory tiering and transactional page migration. However, they still suffer from the same problems as previous approaches. Applications that access memory in uniform patterns can also cause the same problem in migration policies as these applications do not have pages that are hot compared to other pages so there is no need for migration.

We propose a different approach for page temperature detection. We suggest finding regions of interest using the probability distribution of access frequency for pages of memory. Our proposed method profiles the initial distribution of access frequency in an application for ground truth. After that, we sample pages from memory to detect the temperature of pages and update the distribution model using the Bayesian inference technique. By doing this, we can find regions of interest in memory that make us more likely to find the information that we want. As an example, for thermal management problems, we can find hot regions in memory space. As for tiered memory systems, we can find regions that are more likely to contain hot pages that recently became cold. By doing this, we decrease the search space for temperature detection techniques. By doing so, we can reduce memory thrashing and migrate data that are not being accessed frequently to the capacity-tier memory.

We are planning to test our method with the same microbenchmarks that are used in NOMAD[5]. The microbenchmark tests the system with different sizes of hot data. By increasing the size of hot data over the size of fast-tier memory, we can test whether our proposed method is able to find regions of interest in the memory. Also, we are planning on doing a head-to-head comparison with other works such as TPP, Memtis, and NOMAD over graph benchmarks and machine learning inference applications.

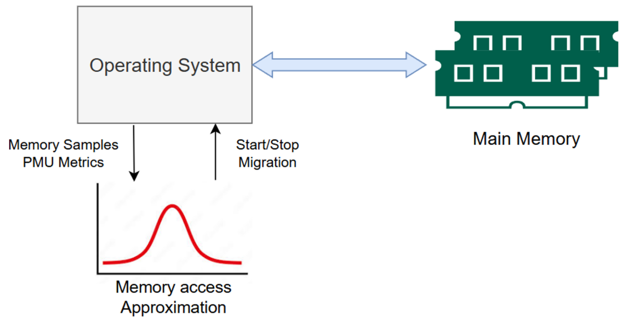

Figure 2: Our proposed solution to prevent overhead caused by migration policies. In this method, we are planning to avoid the overhead caused by traditional memory access methods by approximating the memory access using memory samples and performance metrics like TLB miss.

References

[1] Neha Agarwal and Thomas F Wenisch. 2017. Thermostat: Application-transparent page management for two-tiered main memory. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems.* 631–644.

[2] Taehyung Lee, Sumit Kumar Monga, Changwoo Min, and Young Ik Eom. 2023. Memtis: Efficient memory tiering with dynamic page classification and page size determination. In *Proceedings of the 29th Symposium on Operating Systems Principles.* 17–34.

[3] Hasan Al Maruf, Hao Wang, Abhishek Dhanotia, Johannes Weiner, Niket Agarwal, Pallab Bhattacharya, Chris Petersen, Mosharaf Chowdhury, Shobhit Kanaujia, and Prakash Chauhan. 2023. Tpp: Transparent page placement for cxl-enabled tiered-memory. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3.* 742–755.

[4] Amanda Raybuck, Tim Stamler, Wei Zhang, Mattan Erez, and Simon Peter. 2021. Hemem: Scalable tiered memory management for big data applications and real nvm. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles.* 392–407.

[5] Lingfeng Xiang, Zhen Lin, Weishu Deng, Hui Lu, Jia Rao, Yifan Yuan, and Ren Wang. 2024. Nomad:{Non-Exclusive} Memory Tiering via Transactional Page Migration. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24).* 19–35.

[6] Zi Yan, Daniel Lustig, David Nellans, and Abhishek Bhattacharjee. 2019. Nimble page management for tiered memory systems. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems.* 331–345.