Subpage Migration in Heterogeneous Memory Systems

Shashank Adavally
Shashankadavally@my.unt.edu
University of North Texas
Texas, USA

Dr. Krishna Kavi
Krishna.Kavi@unt.edu
University of North Texas
Texas, USA

Dr. Gayatri Mehta
Gayatri.Mehta@unt.edu
University of North Texas
Texas, USA

ABSTRACT

With the increasing demands for very large physical address spaces and the advent of memory technologies that can support large memories, there is a need to reduce the sizes of system tables such as TLBs and page tables. One can use very large (huge) pages instead of traditional 4K byte pages. However, huge pages are likely to lead to internal fragmentation and may make page migration strategies that aim to move heavily used pages to faster memories inefficient. If only a small portion of a huge page is heavily accessed, it may be worth migrating only that portion to a faster memory. This paper proposes two hardware-based page migration techniques (i) subpage migration with Address Reconciliation (that is, updating physical addresses of migrated pages) and (ii) subpage migration with Reverse Migration (whereby no Address Reconciliation is needed). We observed speedup ranging up to 17% over migrating huge pages and up to 55% over the baseline (no migration).

CCS CONCEPTS

• **Computer systems organization** → **Heterogeneous (hybrid) systems**;

KEYWORDS

Heterogeneous memory systems, Flat-address memory, Reverse migration

1 INTRODUCTION

Our research is driven by two emerging trends in computer systems. First, with increasing memory footprints of modern applications in high performance computing, big data analytics, cloud computing and machine learning, there is a demand for large and high bandwidth memory systems [5], [15]. 3D-DRAMs provide high bandwidth compared to traditional DRAMs but at higher cost and lower capacity. On the other end, the advances in non-volatile memories can support high capacities at lower cost, but at higher access latencies. This led researchers to combine these diverse memory technologies into one system as either hierarchical organization (i.e. fast memory is used as a Last Level Cache) [7] or as flat-address memories (i.e. fast and slow memories combine to form as single address space). In flat-address system, different page placement [24] and page migration [13], [28] strategies have been studied for their memory performance. In the case of page placement, applications are profiled and data is placed either in the fast or slower memories based on access behaviors. Once allocated, the data (or pages) are not relocated. In page migration techniques, heavily accessed pages (hot pages) are migrated from slower memories (NVM) to faster memories (3D DRAM); cold pages are moved from faster memories to slower memories to make room for the hot pages. Pages can be migrated (or swapped) at regular intervals (epoch based) or individually (on-the-fly). The migration of pages between the memory systems incur execution and energy overheads. In addition to the cost of actual data movement between memory devices, OS tables (Translation Look-aside Buffers (TLBs) and page tables) must also be updated since physical addresses used by OS are based on the physical location of pages and a migration changes physical addresses. We call this process of changing physical addresses and updating system tables *"Address Reconciliation"* or AR.

The second trend that drives our research is related to increasing applications' virtual address spaces from 256 TB to 128 PB [3], which leads to an increase in the number of bits needed for virtual addresses from 48 bits to 57 bits: 48 bits virtual addresses require 4 levels of page table traversals to find its respective physical page number and 57 bits lead to an increase in the number of page table traversals (or page walks) to 5 adding to access delays [4]. While page walk caches, which caches some intermediate levels of translation table entries can help, however, the ever increasing physical memory sizes will require larger and larger page walk caches. Similarly, the larger physical memory sizes also necessitates larger TLB's (and larger TLBs will necessarily have to be designed with limited set-associativities). One solution to address both TLB and page walk cache issues is to increase the page size from 4KB to 2MB, reducing the number of pages in an address space, and in turn reducing the number of page tables and TLB sizes. But large page sizes bring new issues that must be addressed. To avoid internal fragmentation, new allocation techniques are needed. In addition, since current page migration techniques (either epoch based or on-the-fly) migrate at page granularity, migration of large (or huge) pages can cause excessive overheads in terms of data movement, and these overheads have to be offset with concomitant usage of migrated pages. We observed that in most applications, only small portions of a large page are heavily accessed and migrating the entire page may not lead to performance gains since the page contains both hot and cold portions. We propose to divide a huge page into subpages and explore migrating only the hot subpages. The key contributions of this work are:

- **Subpage Migration with Address Reconciliation (SpAR)**: This technique migrates hot portions of a huge page (or SubPages) to faster memory instead of the entire huge page. Upon the migration of all the subpages (or a significant number

HMEM'21, June 2021,

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/1122445.1122456

of subpages, the remaining subpages are also migrated and Address Reconciliation is performed to update physical addresses (and make the physical addresses visible to OS).

- **Subpage Migration with Reverse Migration (SpRM)**: Similar to the above technique, this SpRM migrates subpages to faster memory and maintains the migration metadata in a table called ReMap table. As the Remap table fills up, instead of migrating remaining subpages of huge pages and and updating physical addresses, we reverse migrate some (cold) subpages back to slower memories to make room for new hot subpages. Reverse migration eliminates the need for Address Reconciliation since the migration is transparent to OS. However it may incur more overheads as subpages are swapped back to their original location.

2 BACKGROUND AND MOTIVATION

2.1 Background

There have been many studies on page migration techniques for flat-address heterogeneous memory systems (HMA), They propose different approaches to solve the general challenges associated with page migration, viz., selecting candidate pages to migrate, determining migration frequency and managing migration metadata. Meswani et al. [18] presented a study where page migration in HMA is accomplished by a hardware/software based approach. When pages are migrated, physical addresses change. This work relies on OS to update physical addresses for all migrated pages. We refer this process of updating physical address Address Reconciliation (this requires updating TLBs and Page Table Entries). Since OS-based address update incurs large overheads, the authors choose a longer epoch to reduce frequent OS interventions. However, it has been observed that page migration at shorter intervals is more beneficial than waiting for longer epoch times [23, 25], since, migrating hot pages sooner result in more beneficial accesses after the migration.

Address Reconciliation can be avoided using very large Remap table [1] that contains the new locations of migrated pages: the Remap table aids in redirecting accesses using current physical addresses to correct physical locations of the migrated pages (making page migration transparent to OS). The size and management of this Remap table present new challenges. A number of different approaches have been proposed to keep this table in memory while using a small on-chip cache for recently accessed entries [7, 17, 23, 25]. Sim et al. used a Transparent Hardware based Management (THM) of flat-address memory [25]. THM restricts where a migrated page can be placed to reduce the ReMap table: a set of slow memory pages compete for a single fast memory page. This can reduce potential benefits since only one of the slow memory pages from a given set can be migrated to fast memory even if all of them are heavily accessed.

Authors of Thermostat [2] proposed a technique to estimate access rates of huge pages by projecting the computed access rate for random sample set of 4KB pages and migrate the huge page to the slower memory. In our work, we maintain hardware-assisted access counts at subpage level granularity and migrate the individual hot subpages instead of the huge page to the faster memory to prevent migrating less useful portions of huge page due to internal fragmentation.

[1]ReMap table is a hardware structure in our proposal that holds the new physical page addresses of the migrated pages as a temporary reference before Address Reconciliation.

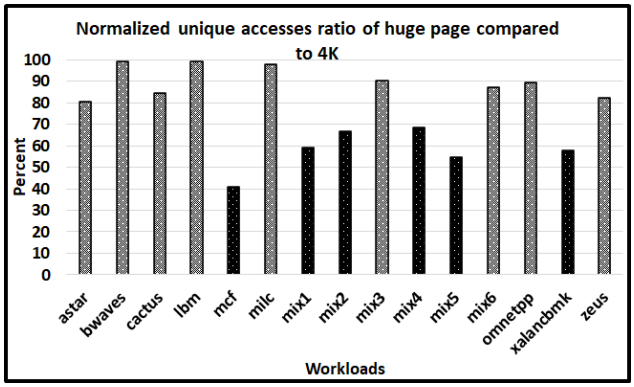

Figure 1: Normalized unique accesses ratio of huge page compared to 4K

2.2 On-The-Fly Migration

In our previous research [1, 13], we migrated a page immediately when it receives sufficient number of memory accesses (i.e., On-The-Fly or OTF migration), unlike any epoch-based schemes described above. Since in OTF migration, a page migration can take place at any time, it is important to ensure that a migration does not halt user program execution [2]. Moreover, OS based address reconciliation on each page migration is prohibitive for OTF migration. To mitigate these issues, we devised a special hardware called MigC [13], placed on the processor chip, which performs actions necessary for our OTF page migration.

Figure 3 (in Appendix) shows a high-level system architecture of MigC. There are hot and cold buffers to temporarily store data from pages as they are being migrated. There is a Wait queue in MigC, which holds read/write requests from Last Level Cache (LLC) for the currently migrating pages; these requests will be serviced from the hot/cold buffers. The memory controllers (MCs) are equipped with a separate Migration Queues (Mig.Q) to service requests from MigC for the migrating pages. There is a small remap (ReMap) table which holds new physical page addresses of the migrated pages. The ReMap table is consulted on every LLC miss (or on a write-back), using the old physical address to find the new location. The size of the ReMap table is kept small (e.g., 1024 entries) so that it can be placed on-chip. Whenever the table is full to a certain level, say 50%, address reconciliation process starts, i.e., entries from ReMap table are deleted and the new physical addresses are made visible to OS as describe in [13]. Some details of the page migration and address reconciliation are included in the Appendix.

2.3 Motivation for Subpage Migration

With the increase in memory sizes, huge pages can be used to reduce the overhead of page table walks. But one of the disadvantages with huge pages is the potential for internal fragmentation. When a page is allocated in the physical memory, some portions of the page

[2]Epoch based approaches stop program execution and migrate several hot pages at the end of an epoch. OS manages the migration as well as updating TLBs and page table entries with new physical addresses.

may not be used by the application due to the way applications are programmed and how address space is divided into different sections (code, static data, heap and stack). Page migration in systems using huge pages can be expensive due to migration overhead of huge pages. Internal fragmentation can make it even worse since some portions of a migrated page may not used. We propose to migrate the pages at a smaller (or subpage) granularity to reduce the overhead of migrating huge pages, as well as migrating only heavily used portion of a huge page. To understand internal fragmentation, we measured the ratio of normalized unique accesses to a huge page compared to the unique accesses to a regular page (4K) as shown in Figure 1. This ratio indicates if the unique accesses are distributed evenly across all subpages of a huge page or not: a ratio that is less than 100% implies that the accesses are not evenly distributed. It can be seen from Figure 1 that workloads like mcf, mix1, mix2, mix4, mix5 which were classified as migration friendly in our previous work[13] (and shown in Table 4 in Appendix) have a lower ratio of unique accesses to a huge page normalized to 4K size, implying internal fragmentation. [3] This also indicates that only useful subpage of huge pages should be migrated. Our focus is to improve migration efficiency in above mentioned list of applications as they suffer the most due to internal fragmentation: an analysis such as the one described here can be used to identify such memory behaviors.

3 SUBPAGE MIGRATION

In our subpage migration technique, the Migration controller logic remains similar to the page migration controller used in our previous work [1, 13]. We maintain entries and access counters in the ReMap table at subpage granularity. For example, if the page size is 512K, and the subpage size is 64K, we track the 8 subpages of a page with access counters and migrate only those subpages that receive sufficient number of accesses. As described in the section 1, we evaluated two subpage migration techniques.

- 1) Subpage migration with Address Reconciliation (SpAR)
- 2) Subpage migration with Reverse Migration (SpRM).

In SpAR, we migrate the subpages as the access counts reach the threshold and mark them as migrated in the ReMap table. Since operating system tracks the details of the pages at huge page level but not at the subpage level, the page can be reconciled only if all the subpages have been migrated from slower to faster memory. When it becomes necessary to reconcile addresses when the ReMap table becomes full to a selected level, we migrate any remaining subpages of some huge pages that are not migrated and then complete address reconciliation. Note that in subpage migration, subpages from a specific page are migrated to a huge physical frame in HBM (High Bandwidth Memory, fast memory), so that Address Reconciliation of the page is easier. A free frame is created by migrating a cold page from the HBM to PCM (Phase Change Memory, slow memory) in case if HBM is out of frames. This process can hinder the performance if all the sub-pages are not beneficial but it is advantageous to reconcile the huge page so that the future accesses will be serviced without indirection through ReMap table. We select pages that have very high percentage of subpages already migrated for Address Reconciliation.

We also experimented with a different technique called subpage migration with Reverse Migration (SpRM). Here, migration is still done at subpage granularity, but when the ReMap table becomes full to a selected level, we select some subpages in the ReMap table whose Migration Benefit Quotient(MBQ)[4] is low and reverse migrate them to their original locations (that is, slower memory) and remove the entries from the remap table. The advantages here are: 1) Address Reconciliation can be eliminated making the page migration completely transparent to the Operating System; 2) eliminate the need to migrate less beneficial subpages to perform Address Reconciliation. Disadvantages with reverse migration are: 1) we pay additional overhead for reverse migration, 2) we may have to migrate very beneficial subpages several times, unlike the case where such hot subpages are among the pages whose addresses are likely to be reconciled.

4 EXPERIMENT SETUP

In this section, we will describe our simulation setup and the benchmarks we used. This setup and workloads are the same that we used in our previous work [13], [1].

4.1 Simulation Infrastructure

We model a 16-core system with a flat-address heterogeneous memory consisting of 1GB of HBM and 16GB of PCM using Ramulator [16]. Ramulator is a trace driven, cycle-level memory simulator with support for a simple multi-core CPU model with cache hierarchies. Each core is 4-wide out-of-order issue with 128 Reorder Buffer (ROB) entries and operates at 3.2GHz. The cores have private L1-D caches (32KB, 4-way, 2-cycles) and shared L2 (16MB, 16-way, 21-cycles) as LLC. Physical address tags, write-back policies are used for all caches and LLC is inclusive. Ramulator does not model L1-I cache, and assumes non-load/store instructions are executed in one cycle. The memory system configuration is shown in Table 1; for timing parameters of HBM we rely on [16] and for PCM timing on [21].

Parameter	HBM	PCM
Channels, capacity	8, 1GB (8 x 128 MB)	2, 16GB (2 x 8 GB)
Memory Controller (MC)	1 per channel	1 per channel
Row buffer	2KB	2KB
Queue size/MC	RD 32, WR 32, Mig. 32 entries	RD 64, WR 256, Mig. 32 entries
Latency	tCAS-tRCD -tRP-tRAS: 14ns-14ns -14ns-34ns	Read 80ns (7.5ns tPRE + 62.5ns tSENSE + 10ns tBUS) Write 250ns tCWL
Bus/channel	128 bit, 1 GHz	64bit, 400MHz

Table 1: Baseline configuration

A basic address mapping function is added to Ramulator to support our flat-address heterogeneous memory system: pages are allocated to frames of different types of memories in a round-robin

[3] Even though xalancbmk has noticeably lower ratio, application itself is categorized as migration unfriendly in our previous work [13], and thus unlikely to benefit either from "full page" migration or subpage migration.

[4] MBQ tracks the number of accesses received after migration to faster memory.

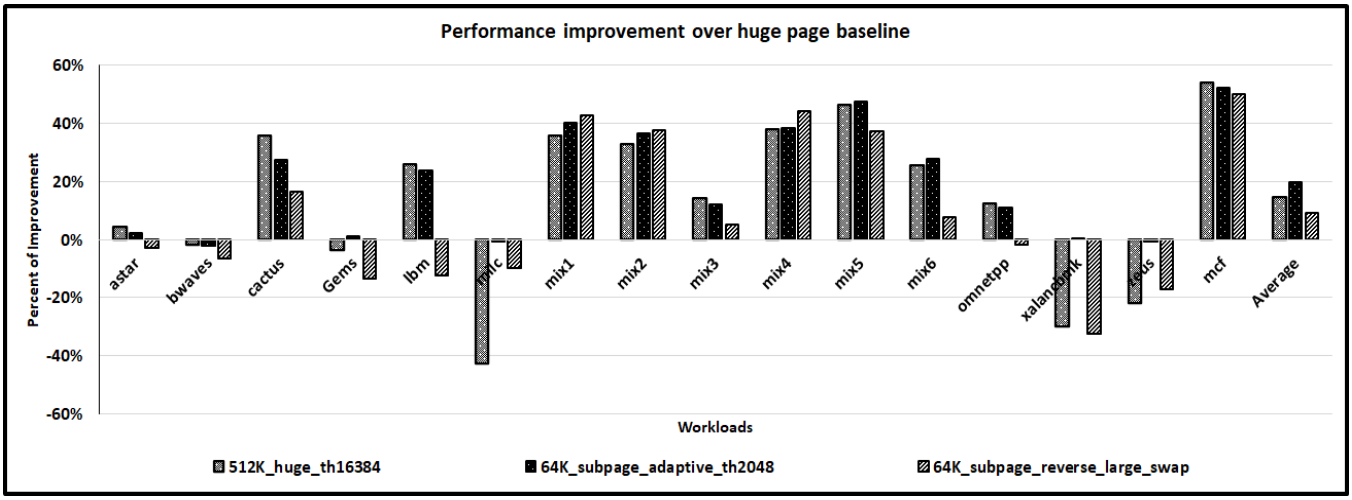

Figure 2: Performance improvement (%) of Subpage migration over Huge page migration (negative y-axis shows degradation)

fashion (viz., 4 pages to faster, 3D DRAM memory, then 4 pages to slower NVM memory), as long as there are free frames in faster memory. When faster memory capacity is exhausted, only slower memory frames are assigned. *This allocation ensures that pages for all application span both memory devices and thus necessitating page migration considerations. We also made sure that the memory footprints of our benchmarks are at least twice as large as the HBM capacity, requiring the use of both HBM and PCM (NVM).* We incorporate our MigC unit in Ramulator with all necessary details to perform functions as described in previous sections (more details can be found in [1, 13]). MigC operates at the same clock rate as the CPU cores.cWe assume the huge page size as 512KB and we selected the subpage size to be 64 KB (We observed 64 KB to be the optimal size for subpage based on our experiments). The ReMap table is implemented as a 1024 entry fully-associative table and each entry contains a bit vector used to distinguish the subpages that have been migrated. We conservatively assumed an access latency of 10 CPU cycles for ReMap table. We included all timing overheads (listed in Table 2 assuming 3.2GHz clock rate) for performing different HW/OS tasks. Finally, we used CACTI [20] to model the power rating of the MigC components. [5]

5 RESULTS

Figure 2 shows the performance improvement of our proposed techniques compared to the baseline (no migration). First bar (labeled 512K_huge_th16384) represents huge page migration with hotness threshold of 16384 (the entire huge page is migrated if number of accesses to the page reaches 16384 or more). We proportionately increased the threshold with page size and selected 16384 as threshold. Similarly, second bar (labeled 64K_subpage_adaptive_th2048) represents subpage migration with Address Reconciliation with hotness threshold of 2048. The third bar (64K_subpage_reverse_large_swap) represents subpage migration with reverse migration. It can be seen that on average (including both migration friendly and unfriendly workloads), subpage migration with Address Reconciliation performs about 20% better than baseline (i.e. no migration), 2% better than the huge page migration and 10% better then subpage migration with reverse migration. Looking at the performance of individual workloads, workloads that were categorized as migration unfriendly [6] were not expected to achieve performance gains from either page or subpage migration. Some of the migration friendly workloads do benefit from subpage migration when compared to migrating entire huge pages, by up to 17%. This is because of huge page migrations were difficult to adapt to changing hotness thresholds as done in [1]. If the hotness threshold is decreased, then too many huge pages will be migrated, defeating the benefits of migration and if threshold is increased, very few huge pages will be migrated, missing some pages that can benefit from migrating to faster memory. With subpage migration, we can maintain a balance between the number of migrations and threshold with relative ease. Thus, we observed more migrations were possible with subpages than huge pages, still maintaining the balance between migration benefits and migration overheads. [7] As shown in Figure 1, we also believe that workloads like mix1, mix2, mix4, mix5 are largely affected by internal fragmentation and are benefiting from subpage migration. With reverse migration, it can be observed that we are losing performance for some workloads like xalancbmk, zeusmp, GemsFDTD. This is because we reserved a fixed amount of swap space in HBM for migrated subpages and this space is not available for permanently placing hot pages in HBM. This is beneficial for workloads that benefit if very few subpages of a page are candidates for migration (like mix1, mix2, mix4, mix5). We will further investigate the memory access behaviors that benefit from subpage migrations and reverse migrations, as well as adaptively adjusting swap space.

[5] Further details of our experimental setup and benchmarks can be found in our previous publications [13], [1]. Some information is included in Appendix.

[6] See Table 4 in Appendix for a list of such workloads

[7] Due to the page count limitation, we couldn't include additional data related to the improved demand request traffic to HBM with subpage migration resulting in improved performance

6 CONCLUSION

In this paper we studied two subpage migration techniques: subpage migration with Address Reconciliation and subpage migration with reverse migration when systems use very large (huge pages) in order to reduce the sizes of TLBs and page tables. The idea with both these techniques is to migrate data at subpage granularity to overcome issues such as internal fragmentation in huge pages, and migrating only useful portions of a huge page. With subpage migration, our hardware migration controller can better adapt to changing memory access behaviors of applications, than with huge page granularity. The hotness threshold for migrating pages (either at subpage or entire page level) is changed based on the number of pages migrated and the benefit accrued by the migrated pages in terms of faster accesses. In our experiments, we observed as much as 55% improvement in performance compared to the baseline (i.e. no migration) and up to 17% compared to huge page migration.

REFERENCES

[1] Shashank Adavally, Mahzabeen Islam, and Krishna Kavi. 2021. Dynamically Adapting Page Migration Policies Based on Applications' Memory Access Behaviors. *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 17, 2 (2021), 1–24.

[2] Neha Agarwal and Thomas F. Wenisch. 2017. Thermostat: Application-Transparent Page Management for Two-Tiered Main Memory. *SIGPLAN Not.* 52, 4 (April 2017), 631–644. https://doi.org/10.1145/3093336.3037706

[3] Article. 2021. *Intel 5-level paging*. Technical Report. https://en.wikipedia.org/wiki/Intel_5-level_paging

[4] Article. 2021. *Page tables*. Technical Report. https://en.wikipedia.org/wiki/Page_table

[5] Chaim Bendelac and Panos Kokkalis. 2017. SAP HANA Memory Usage Explained. https://www.sap.com/documents/2016/08/205c8299-867c-0010-82c7-eda71af511fa.html. [Online; accessed January-20-2019].

[6] Tim Bird. 2009. Measuring function duration with ftrace. In *Proceedings of the Linux Symposium*. Citeseer, 47–54.

[7] Chiachen Chou, Aamer Jaleel, and Moinuddin K Qureshi. 2014. CAMEO: A Two-Level Memory Organization with Capacity of Main Memory and Flexibility of Hardware-Managed Cache. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 1–12.

[8] Standard Performance Evaluation Corporation. 2015. SPEC CPU 2006. https://www.spec.org/cpu2006/.

[9] DOE. 2018. US Department of Energy ECP Proxy Application Suite. https://proxyapps.exascaleproject.org/ecp-proxy-apps-suite/.

[10] Harish Patil. 2018. PinPlay. https://software.intel.com/en-us/articles/program-recordreplay-toolkit.

[11] Mike Heroux and Simon Hammond. 2015. MiniFE: Finite Element solver. https://portal.nersc.gov/project/CAL/designforward.htm#MiniFE.

[12] RD Hornung, JA Keasler, and MB Gokhale. 2011. *Hydrodynamics challenge problem*. Technical Report. Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States).

[13] Mahzabeen Islam, Shashank Adavally, Marko Scrbak, and Krishna Kavi. 2020. On-the-Fly Page Migration and Address Reconciliation for Heterogeneous Memory Systems. *To Appear in ACM Journal on Emerging Technologies in Computing Systems* (2020). http://csrl.cse.unt.edu/kavi/Research/JETC-2019.pdf

[14] Jewillco. 2015. Graph500-v2-spec. https://github.com/graph500/graph500/tree/v2-spec.

[15] Kimberly Keeton. 2017. Memory-Driven Computing. USENIX Association, Santa Clara, CA.

[16] Y. Kim, W. Yang, and O. Mutlu. 2016. Ramulator: A Fast and Extensible DRAM Simulator. *IEEE Computer Architecture Letters* 15, 1 (2016), 45–49.

[17] A. Kokolis, D. Skarlatos, and J. Torrellas. 2019. PageSeer: Using page walks to trigger page swapps in hybrid memory systems. In *Proceedings of the 25th IEEE International Symposium on High Performance Computer Architecture*. IEEE.

[18] Mitesh R Meswani, Sergey Blagodurov, David Roberts, John Slice, Mike Ignatowski, and Gabriel H Loh. 2015. Heterogeneous memory architectures: A hw/sw approach for mixing die-stacked and off-package memories. In *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*. IEEE, 126–136.

[19] Jamaludin Mohd-Yusof, Sriram Swaminarayan, and Timothy C Germann. 2013. Co-design for molecular dynamics: An exascale proxy application, 2013. https://www.lanl.gov/orgs/adtsc/publications/science_highlights_2013/docs/Pg88_89.pdf.

[20] N. Muralimanohar, Rajeev Balasubramonian, and N. Jouppi. 2007. CACTI 6 . 0 : A Tool to Understand Large Caches. https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.147.3834&rep=rep1&type=pdf.

[21] Prashant J Nair, Chiachen Chou, Bipin Rajendran, and Moinuddin K Qureshi. 2015. Reducing read latency of phase change memory via early read and Turbo Read. In *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*. IEEE, 309–319.

[22] Osnat Levi (Intel). 2018. Pin - A Dynamic Binary Instrumentation Tool. https://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool.

[23] Andreas Prodromou, Mitesh Meswani, Nuwan Jayasena, Gabriel Loh, and Dean M Tullsen. 2017. MemPod: A clustered architecture for efficient and scalable migration in flat address space multi-level memories. In *High Performance Computer Architecture (HPCA), 2017 IEEE International Symposium on*. IEEE, 433–444.

[24] Luiz E Ramos, Eugene Gorbatov, and Ricardo Bianchini. 2011. Page placement in hybrid memory systems. In *Proceedings of the international conference on Supercomputing*. ACM, 85–95.

[25] Jaewoong Sim, Alaa R Alameldeen, Zeshan Chishti, Chris Wilkerson, and Hyesoon Kim. 2014. Transparent hardware management of stacked dram as part of memory. In *Microarchitecture (MICRO), 2014 47th Annual IEEE/ACM International Symposium on*. IEEE, 13–24.

[26] John R Tramm, Andrew R Siegel, Tanzima Islam, and Martin Schulz. 2014. XSBench - The Development and Verification of a Performance Abstraction for Monte Carlo Reactor Analysis. In *PHYSOR 2014 - The Role of Reactor Physics toward a Sustainable Future*. Kyoto. https://www.mcs.anl.gov/papers/P5064-0114.pdf

[27] Carlos Villavieja, Vasileios Karakostas, Lluis Vilanova, Yoav Etsion, Alex Ramirez, Avi Mendelson, Nacho Navarro, Adrian Cristal, and Osman S Unsal. 2011. Didi: Mitigating the performance impact of tlb shootdowns using a shared tlb directory. In *Parallel Architectures and Compilation Techniques (PACT), 2011 International Conference on*. IEEE, 340–349.

[28] 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* (Providence, RI, USA) *(ASPLOS '19)*. Association for Computing Machinery, New York, NY, USA, 331–345. https://doi.org/10.1145/3297858.3304024

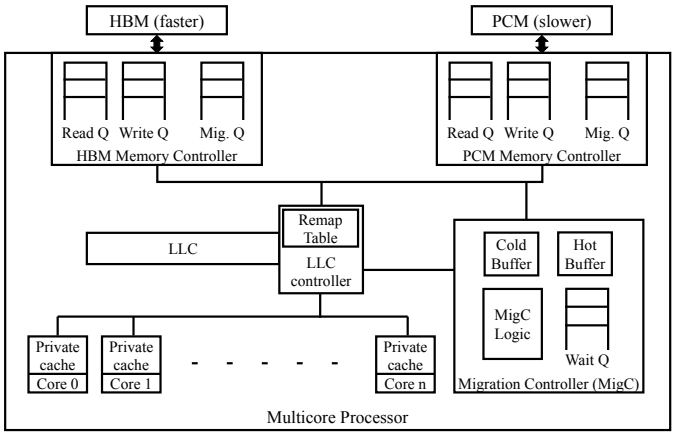

Figure 3: High-level system architecture

Appendices

A ON-THE-FLY MIGRATION

A.1 User Transparent Page Migration

We migrate a hot page from slower PCM into a free frame of HBM if available (one-way migration), or select an HBM page that has not been accessed recently (LRU Cold page) and swap the hot and cold pages (two-way migration). Consider that MigC finds that a PCM page (say page A with physical address, PA 8192) meets the hotness threshold for migration. MigC then finds a cold page from HBM (say page B with physical address, PA 0) to swap with the hot page.

MigC inserts entries for pages A and B in the ReMap table with their current OS visible physical addresses (namely 8192 and 0) and future PA (after migration, namely 0 and 8192). ReMap table is always looked up using OS visible (original) PA. Mig flag is set to 1 when these pages are being migrated and a Pair flag is set to 1 to indicate a two-way migration involving a hot and cold pair (this flag will be set to zero for one-way migration). The Pair flag will be checked during address reconciliation (AR) to update page table entries for both (or one) pages involved in the migration.

MigC waits for any pending read requests to the pages involved in the migration that were already issued to complete. Then, MigC starts reading hot and cold pages into their respective buffers (inside MigC). Any new requests (after the migration is initiated) for these pages from LLC will be held in MigC resident Wait Queue and will be served from these buffers. After completely reading the migrating page contents into the buffers, MigC starts writing contents of buffers to their respective new page frames. When migrations are completed, the Mig flag for these pages will be reset (to indicate completion of migration). All future requests for these pages will be directed to proper new locations based on the ReMap table information.

A.2 Address Reconciliation

Address reconciliation can be eliminated (and make page migration transparent to OS) by using very large ReMap tables, sufficient to track all migrated pages during the lifetime of an application. However, this is not practical for emerging systems with very large memories (several hundred giga bytes to tera bytes). We use a very small ReMap table and periodically evict old entries to make room for new entries for future migrations. Removing entries from the ReMap table requires updates to physical addresses (i.e., address reconciliation) to reflect the new location of the page consistently throughout the system, and making the new physical addresses visible to OS. We reconcile entries from the ReMap table pair-wise if the Pair flag is 1, thus updating the physical addresses of the pages swapped during the migration. When the Pair flag is 0 then we perform AR only for that entry. The following actions must be performed to ensure correct address reconciliation. We use the same example hot and cold page pair, A (PA=8192) and B (PA=0), respectively. First, all cache lines from these pages, which are currently residing in the cache hierarchies and tagged with OS visible (old) physical address, must be invalidated (and dirty lines written back), since the current OS visible PA will be replaced with the new PA. All future accesses to these pages will only have access to the new PA. Next, corresponding page table entries (PTEs) for A and B need to be updated with new PAs. The TLB entries in all cores using the old PA must also be invalidated (known as TLB shootdown).

Task	Time Requirement
ReMap table lookup	10 cycles (after LLC)
Light-weight TLB invalidation at core	300 cycles (round trip latency to off-chip memory [27])
Page walk	150 cycles
OS reverse mapping	4480 cycles (measured using Ftrace [6] on a real machine running Linux)

Table 2: Timing parameters at 3.2GHz clock

B WORKLOADS

We use sixteen multi-programmed SPEC CPU2006 [8] workloads, four multi-threaded benchmarks from the US Department of Energy (DOE) provided ECP Proxy Applications [9] as well as the BFS from Graph500 suite [14]. We selected SPEC benchmarks with large memory footprints, at least twice the capacity of HBM. SPEC benchmarks allow us to compare our work with other studies. We profile benchmarks using PinPlay kit [10] to collect a representative slice of 500M instructions from each of the applications. To make a multi-programmed workload, we run a 16-core Ramulator simulation where each core runs one of the SPEC traces to completion. We either run 16 copies of the same benchmark on 16 cores (each such workload is labeled by the benchmark name in our graphs) or run a random mix of benchmarks on 16 cores (these workloads are labeled as mix1 to mix6 and described in Table 3). The publicly released multi-threaded HPC proxy benchmarks by the US Department of Energy (DOE) that we used are- XSBench [26], LULESH [12], CoMD [19] and miniFE [11]. We ran each HPC benchmark in a 16-thread setup and collected 500M instruction traces for each of the threads using Pin tools [22]. By running traces of the 16 threads of a HPC benchmark in Ramulator we obtain a multi-threaded workload (each such workload is labeled with the name of the benchmark). The

	mix1	mix2	mix3	mix4	mix5	mix6
astar	2x		1x			1x
bzip2		1x	1x	2x		
cactus		2x	2x	1x		
dealII		3x	1x	1x		
gcc	1x		2x	1x		3x
Gems		2x	2x	1x		
lbm	2x	3x		1x	6x	1x
leslie			2x	1x		
libq	2x		1x	3x		4x
mcf	3x	2x		1x	5x	
milc	2x		2x	1x		2x
omnetpp	1x					3x
soplex	2x	3x		3x	5x	
sphinx	1x		2x			3x

Table 3: SPEC multi-programmed mix workloads

Friendliness	**Benchmark**
Very Friendly	mcf, mix1, mix2, mix4, mix5
Moderately Friendly	lbm, omnetpp, astar cactus, mix3, mix6
Least friendly or Unfriendly	milc, Gems, zeusmp xalancbmk,

Table 4: Migration Friendliness of Applications

memory footprint of the workloads range between 2GB to 11GB, ensuring that the workloads fit in physical memory and do not require access to secondary storage. They are large enough to cause migration but not unrealistically small. Table 4 shows the category of each workload. Workloads under very friendly category indicates that they benefit the most from page migration based on their behavior. Similarly, moderately friendly represents that these workloads may be take advantage of page migration minimally and least friendly workloads doesn't benefit from migration [13].