Supporting Dynamic Translation Granularity for Hybrid Memory Systems

Bokyeong Kim*, Soojin Hwang†, Sanghoon Cha‡, Chang Hyun Park§, Jongse Park†, Jaehyuk Huh†
*Samsung Research, †School of Computing, KAIST, ‡Samsung Advanced Institute of Technology, §Uppsala University
bokyeong.kim@samsung.com, sjhwang@casys.kaist.ac.kr, s.h.cha@samsung.com,
chang.hyun.park@it.uu.se, jspark@casys.kaist.ac.kr, jhhuh@kaist.ac.kr

***Abstract*—Hybrid memory has become a promising new solution for meeting ever growing memory capacity demands in a cost-effective way. In hybrid memory systems, the fast and high bandwidth memory is used to store performance-critical data, while the slow and low bandwidth memory provides capacity backup. In supporting such hybridization, virtual memory is the key mechanism, which can combine different memory components to a single memory view. For efficient translation for virtual memory, page size has been growing. However, the hybrid memory support requires fine-grained migration to quickly move only necessary memory portions to the precious fast memory. To address the challenges posed by the conflicting goals in the hybrid memory support based on virtual memory, this paper investigates decoupling of address translation into a two-step process. With the two-level translation, the critical core-side TLBs perform the translation to an intermediate address space, and the memory-side translation provides the actual physical location in memory devices. As the second-level translation handling page migration across different memory types, is decoupled from the first-level translation, it allows dynamic adjustment of its mapping granularity to improve the efficiency of translation and data reuse in the fast memory. This paper proposes a hardware architecture which identifies the memory access behavior of an application online and selects the best mapping granularity for the second-level translation.**

I. Introduction

To address growing memory capacity challenges while satisfying bandwidth and latency requirements, hybrid memory has become a promising cost-effective alternative to the traditional homogeneous memory [5], [10], [14]. In such hybrid memory, the small fast memory serves critical data, while the backing slow memory provides a large capacity. The advent of capacity-oriented non-volatile memory, new memory extension technologies such as CXL [1], and hardware-based compressed memory [17] has accelerated the hybridization of memory systems.

From the perspective of operating systems, such hybrid memory raises a new problem of page placement. A general approach for supporting the hybrid memory has been relying on the virtual memory and page allocation by the operating system [2], [10]. The operating system detects hot pages with assistance from the processor architecture, and hot pages are migrated to the fast memory by updating the mapping in page tables.

Meanwhile, another challenging problem in high capacity memory systems is efficient translation from virtual to physical addresses. Modern processors rely on caching of page

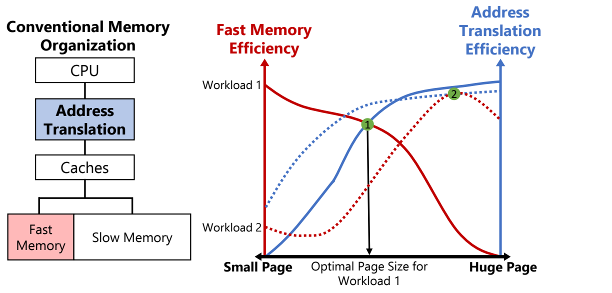

Fig. 1: Performance trade-off by page size in hybrid memory systems. Page sizes have conflicting effects on fast memory efficiency and address translation efficiency

table entries in TLBs for fast translation. The limited TLB capacity in MMU (memory management unit) has been causing significant performance degradations for large memory applications [4], [8]. To mitigate the adverse effect of TLB misses, a common recent solution is to increase the coverage of translation entries with huge pages. In commercial x86 systems, 2MB and 1GB huge page sizes have been added, and the Linux kernel employs 2MB page size whenever contiguous 2MB free pages are available.

In hybrid memory, the address translation in MMU handles the page placement in different types of memory as well as the traditional memory capacity allocation and permission validation. However, for better management of hybrid memory, flexible fine-grained page mapping is preferred to capture a more precise working set in the fast memory, and not to waste the page migration bandwidth.

Therefore, supporting hybrid memory systems incurs a new trade-off in page sizes. For efficient address translation via TLBs, larger page sizes are preferred. For efficient utilization of precious fast memory capacity, finer-grained memory management is necessary. Figure 1 presents the potential trade-off caused by page size in hybrid memory systems. Each workload has a different preferred page size for efficient utilization of fast memory (red curves), which depends on the locality characteristics of the workload. On the other hand, more efficient address translation prefers larger pages (blue curves). Under the trade-off, the support for hybrid memory needs to balance between the conflicting goals.

To mitigate the overheads of supporting hybrid memory, this paper advocates a hardware-assisted two-level translation technique for hybrid memory, called *decoupled translation*. The two-level translation decouples page placement from the capacity allocation and permission checking. It adds an additional layer of translation done after last-level cache (LLC) misses. At each core, a virtual address is translated into a *unified address*. After an LLC miss, the unified address is translated to a real address, which is the location in the physical memory consisting of different types of hybrid memory.

The two-level translation opens a new opportunity to employ a different mapping granularity in each layer. For the critical core-side translation, a huge page can be used to significantly reduce TLB misses, as TLB misses in core-side translation cause significant performance degradation. It relieves the hybrid memory system from the overheads of virtual memory supports. For the new memory-side translation, depending on the memory access patterns of workloads, the best mapping granularity (*frame size*) can be chosen independently from the core-side translation.

To utilize such adjustable frame size, this paper proposes a technique to use the best frame size for each workload. The proposed mechanism chooses the mapping granularity of the memory-side translation, selecting the best size to improve the memory-side TLB performance while supporting effective page mapping for heterogeneity. To find the best frame size for each workload, the paper proposes a dynamic frame size selection mechanism based on workload memory access patterns. Our simulation result shows that a single fixed frame size does not produce the best performance across diverse applications. With the dynamic frame size selection, the performance of the proposed technique with two-level translation is 22% higher than the prior hybrid memory approach with the conventional single-layer virtual memory.

II. Background

A. Hybrid Memory

Emerging memory technologies such as non-volatile memory, hardware compressed memory [17], and disaggregated memory enable co-existence of different memory components with diverse characteristics. Recent advancements of new memory extension technologies such as CXL [1] are accelerating the adoption of hybrid memory. In such hybrid memory systems, data are migrated between fast and slow memory components, to keep more performance-critical data in the fast memory. If locality exists in workloads, such hybrid memory can effectively provide performance close to the fast memory while the overall capacity is greatly increased by the backing slow memory.

There are primarily two approaches to manage the data placement in the hybrid memory: (1) HW-managed mechanism [5], [7], [11], [14], and (2) OS-managed mechanism [10], [16]. The HW-managed mechanism treats the fast memory as either the HW-managed DRAM cache [7], [11], or a subregion of global memory space [5], [14]. The HW-based approach allows fine-grained page management and provides transparency to OS, although the flexibility of mapping and migration is limited due to the HW-based mechanism.

OS Management of Hybrid Memory: In OS-managed approaches, the operating system manages the virtual memory system and page allocation for the entire hybrid memory [2], [6], [10], [16]. The operating system is in charge of identifying the *hotness* of pages and placing *hot* pages in the fast memory while putting *cold* pages to the slow memory. Both of the fast and slow memory are in the physical address space visible to the operating system. With the conventional virtual memory support, in MMU of a core, a virtual address is translated to a physical address, which can be either in the fast or slow memory. The OS-managed approaches have become a dominant mechanism for the current hybrid memory systems.

However, another challenging problem in the high capacity memory system is the efficient translation from virtual to physical addresses. Modern processors rely on the caching of page table entries in TLBs for fast translation. The limited TLB capacity often cannot cover growing memory working sets of large memory applications. To mitigate such performance loss by TLB misses, employing huge pages has become a common technique used in current systems. However, such a trend in virtual memory incurs a critical conflict with the OS-based hybrid memory support. In the hybrid memory systems, a flexible fine-grained page size is preferred to capture precise working sets in the fast memory and not to waste page migration bandwidth for unnecessary part of huge pages.

B. Challenges

Conflicting objectives with page sizes: OS-managed hybrid memory suffers from conflicting objectives with page sizes. For efficient page placement, a ***small page size*** is preferable so that the operating system migrates pages at finer granularity to the right location based on their hotness property. Migrating only the necessary data across the heterogeneous memory components also minimizes bandwidth waste on both fast and slow memory sides. Furthermore, each application has a different preferred migration granularity, depending on its memory access locality characteristics. On the contrary, for minimizing address translation cost, a ***huge page size*** is favorable so that the the limited TLBs can cover large translation footprints. In this work, we aim to achieve the conflicting objectives by introducing a novel address translation scheme, which will be discussed in the following section.

Nimble hot page detection and efficient migration: Another challenge in supporting hybrid memory with the conventional virtual memory is its relatively high costs for detecting hot pages and migrating them. It involves the execution of OS codes to check access status stored in page tables and to identify hot pages with a certain hotness selection policy. Once hot pages are selected, they are migrated by coping data from slow to fast memory. In addition, a slow TLB shootdown step must be taken to update the necessary page tables synchronously. Due to the high cost of page migration, the promotion of hot pages can occur only infrequently. For example, a prior software approach periodically checks hotness status of pages and migrate them [10]. The period is set to 0.1 seconds in the trace-based evaluation [10]. Recent

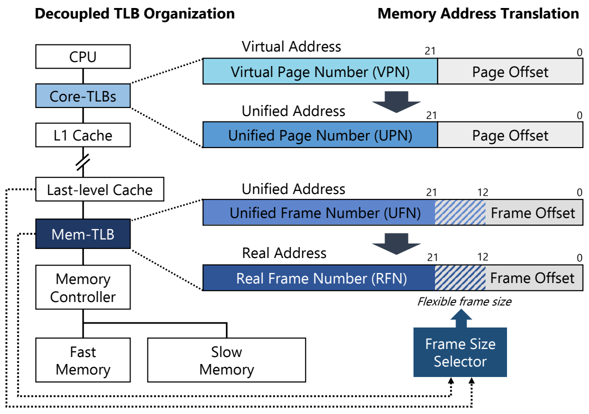

Fig. 2: Two-level translation architecture. The frame size selector is used for the dynamic frame size selection.

studies with real system evaluation used longer periods of 5 seconds [16], and 30 seconds [2]. Such slow promotion is unable to respond to the fast changing behaviors of workloads, losing potential data reuse opportunities.

III. Two-Level Decoupled Translation

A. *Two-level Translation for Hybrid Memory*

This study explores a new HW-SW cooperative architecture for hybrid memory, which embraces the advantages of the HW-oriented approach in the SW approach based on virtual memory. Challenges of supporting hybrid memory with virtual memory stem from the conflicting goals of efficient translation and fine-grained page placement. To address the conflict, this paper employs a two-level translation technique, which decouples the page placement from the conventional capacity allocation and permission checking. Such two-level translation were adopted in the hardware-based compressed memory, to address dynamically determined compressed page sizes [9], [17]. Unlike the compressed memory, we advocate to use the decoupled translation for general hybrid memory as a solution to mitigate the conflicting goals of virtual memory between translation efficiency and fast memory efficiency.

Figure 2 presents the baseline two-level translation architecture. An additional address space is the *unified address (UA)* space, which is an intermediate address space between the virtual address space and real physical address space. The unified address space is a logical-physical space, which hides the actual placements of pages in the real memory. The capacity of the unified address space is bounded by the real address space. If the page allocation in the hybrid memory space is exclusive across different memory types, the unified space size is equal to the sum of all real memory devices. By the conventional translation at the core-side TLB, denoted as *core-TLB* in the rest of the paper, a virtual address is translated to a unified address.

All caches are addressed with the unified address, and they are accessible after the core-TLB translation. If an LLC miss occurs, the unified address must be translated to a real physical address, denoted as *real address* in this paper. For the translation, an extra unified page table (*unified frame table*), and memory-side TLBs called *mem-TLBs* are added. When a mem-TLB miss occurs, a HW page walker access the unified frame table. Coherent DMAs are done by unified addresses too. By using a unified address, DMAed data become consistent with the cached data. To access the real memory, the DMA engine must translate the address to a real address via the mem-TLBs.

Memory-side Translation: The granularity of memory-side translation is a *frame*, and its size can differ from the page size of the core-side translation. To support the memory-side translation, the OS setups a *unified frame table*. The unified frame table is a system-wide translation table, and thus only one instance exists in a system. For fast accesses to the unified frame table, the table is organized as a simple linear table. Note that unlike the virtual address space with 48 bits or more in 64-bit architectures, the unified address space is bounded by the installed physical memory size. Therefore, organizing the frame table as the linear flat table provides a fast single memory access to retrieve the final translation entry, in opposite to four step page walks in the tree-based page table for x86 systems.

HW migration engine: For more efficient migration, we use a HW-based migration engine. Based on the hotness threshold, a frame is migrated between the fast and slow memory. The HW engine also updates the unified frame table for the placement changes. Since a frame migration is done silently by changing the mapping between the unified and real spaces, the migration does not affect the core-TLBs and data in the on-chip caches. Therefore, the migration does not require costly core-side TLB shootdowns and the cached data are still accessible via their unified addresses. The mem-TLBs are organized similar to the core-TLBs. The mem-TLBs are shared by all cores, and to provide translation bandwidth for multiple cores, the TLBs are multi-banked and banks are statically assigned by unified address.

Hotness identification and migration: Frames are promoted to the fast memory based on the hotness selection. In this study, since our focus is on the decoupled translation architecture, we employs a variant of hotness selection proposed by the prior study [10]. The mechanism uses an access counter for each frame. When the access counter is increased beyond a threshold, the frame is promoted to the fast memory. The access counters are cleared periodically. However, with the hardware-assisted migration engine, as soon as the counter of a frame exceeds the threshold, it is migrated to the fast memory without any involvement of OS. Unlike the prior approaches with the conventional virtual memory [2], [10], [16] which use a periodic batch-oriented migration scheme to reduce the costs, once the threshold is reached for a frame, the hardware engine can immediately promote the frame to the fast memory. Such nimble migration improves the hit rates of fast memory significantly.

OS support: The mem-TLBs are based on a HW managed mapping. However, the OS needs to initialize and assign the memory space for the system-wide unified frame table. The

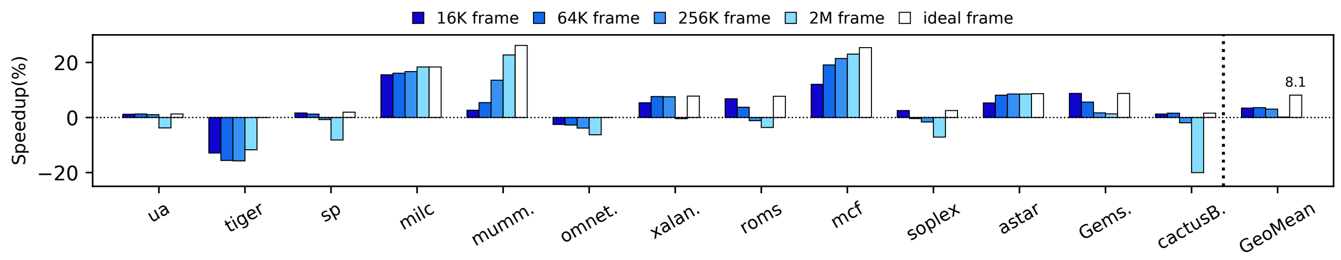

Fig. 3: Speedups with various frame sizes and the best frame size (ideal frame), compared to 4KB frame size.

OS sets aside a contiguous memory region for the unified frame table, and initializes the table as specified by the ISA. Once the unified frame table memory is set up, the pointer to the beginning of the table is provided to the HW. Afterwards, the management of the unified frame table is done by the HW. All mapping modifications are done by the HW for efficiency.

The benefits of decoupled translation are as follows: 1) The performance critical core-TLBs can reduce misses effectively with huge pages or even range-oriented translation [4]. 2) The large on-chip caches can reduce the translation burden on the mem-TLBs, since the mem-TLBs are accessed only after the LLC miss. 3) The migration incurs less overheads without any OS involvement and TLB shootdown for the core-TLBs and cached data are still valid after migration. 4) The frame size can be fine-grained to capture the working set of applications and to use the precious fast memory capacity more efficiently.

B. Analysis

To analyze the effect of decoupled translation, we first show the performance in terms of IPC (instructions-per-cycle) with various frame sizes. The simulation methodology is presented in Section V-A. Figure 3 presents speedups of each workload with five different frame sizes, compared to that with 4KB frame size. The last bar (ideal frame) of each workload is the speedup when the best frame size is applied out of 10 possible frame sizes from 4KB to 2MB. Note that the figure shows only selected five frame sizes out of the 10 frame sizes. It represents the potential benefit when the best frame size can be selected for each workload.

In the figure, the first important message is the need for choosing a right frame size for each workload. For a given workload, if the mem-TLB hit rate can be significantly improved with larger frame sizes, or the workload has a large amount of spatial locality, larger frame sizes are preferred. Otherwise, smaller frame sizes can provide better performance with its more efficient fine-grained migration. For `tiger`, the best frame size is 4KB, providing 15.8% better performance than the worst frame size, 256KB. For `mumm`, `mcf`, and `cactusB`, the differences between the best and worst frame sizes are high with 26.2%, 25.4%, 21.6% respectively. The performance result implies the importance of dynamically applying a right frame size for each application. In addition, when selecting the best frame size, multiple factors, such as translation cost and spatial locality, must be considered.

IV. Dynamic Frame Size Selection

A. Translation for Dynamic Frame Size

Although the performance degradation by mem-TLB misses is lower than that by core-TLB misses, mem-TLB misses still increase the LLC miss handling times for walking the unified frame table. The principle of reducing mem-TLB misses is similar to the conventional TLBs. Increasing the frame size reduces mem-TLB misses, but it can potentially increase the cost of migration and waste the limited fast memory capacity. However, for some workloads, large frame sizes can reduce both fast memory misses and mem-TLB misses, if spatial locality is strong for the workloads. An important observation is that different workloads have their preferred frame sizes.

To address the trade-offs in the frame size, we propose dynamic frame size selection. Depending on the preferred frame size of the current workload, the frame size is adjusted dynamically. It supports a dynamic frame size selected from five candidate frame sizes, 4KB, 16KB, 64KB, 256KB, and 2MB.

B. Frame Size Selection

A key mechanism for supporting the dynamic frame size is to find the best frame size for the current workload. To select the best frame size, we should consider two factors: 1) how many fast memory accesses occur with each frame size (*fast memory hit rates*), and 2) how many mem-TLB hits occur with each frame size (*mem-TLB hit rates*). The biggest challenge in the dynamic frame size selection is the difficulty of predicting the mem-TLB and fast memory hit rates for all candidate frame sizes. During the runtime, the system is running with a single frame size at a time. However, to decide the best frame size, the selection algorithm uses fast memory and mem-TLBs hit rates estimated for 5 candidate frame sizes.

Figure 4 shows the flow of frame size selection in the proposed architecture. In general, the mem-TLBs have low miss rates with larger frame sizes. However, the shapes of miss curves vary by applications, because each workload has a different access pattern. For fast memory hit rates, the frame size is a unit of migration and hotness tracking. Since the fast memory hit rates can change sensitively by the hotness selection of the migration policy, the estimation mechanism should also incorporate the behavior of the migration policy. We propose the two estimation techniques for mem-TLB and

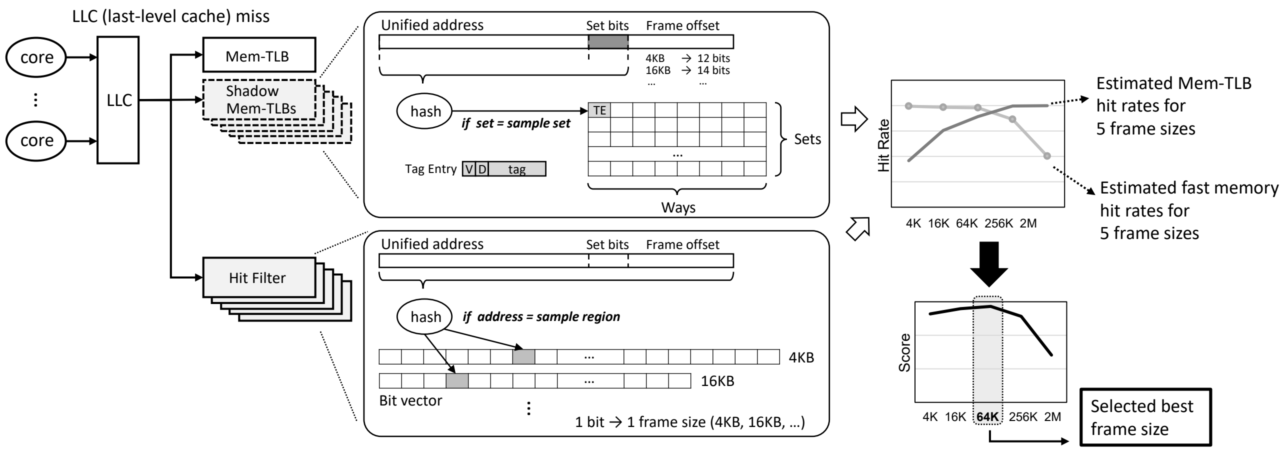

Fig. 4: Dynamic frame size selection architecture. Shadow mem-TLBs and hit filters produce hit rate curves for mem-TLBs and fast memory.

fast memory hit rates, *shadow mem-TLB* and *hit filters*. The two techniques construct estimated hit rate curves for mem-TLBs and fast memory for five possible frame sizes.

Shadow mem-TLBs for estimating mem-TLB misses: Shadow mem-TLBs mimic a small sampled set of mem-TLBs with different frame sizes. Five segments of shadow mem-TLBs estimate the mem-TLB hit rates with the pre-selected five frame sizes. Each segment has 32 entries, organized as 4 sets and 8 ways for each set, which is the same associativity as the original mem-TLBs. Shadow mem-TLBs have only tags without translated target addresses, since they only check whether an access is hit or not with the corresponding hypothetical frame size. The shadow mem-TLBs trace hit/miss status for sampled sets from the mem-TLBs. The fixed sampled sets are different for different frame sizes.

The objective of this shadow lookup is to generate the hit rate curve with a range of frame sizes. On an access for the sampled sets, the address goes through the shadow mem-TLBs for candidate frames. Each shadow mem-TLB segment corresponds to a different frame size, so its index bits are different for the same address. At the end of epoch, the hardware extracts hit rates for all shadow mem-TLB segments to generate the estimated hit rate curve, and uses it for dynamic frame size selection. It requires shadow lookups for every address translation which belongs to the sampled sets. However, it is not on the critical path of L3 miss handling, since it does not return the translated result.

Hit filters for estimating fast memory hit rates: Estimating hit rates of fast memory for hypothetical frame sizes is more difficult than the hit rates of mem-TLBs. Mem-TLBs is filled by a single miss for a page number, but fast memory migration is done only when accesses to a frame exceeds a threshold. The estimation mechanism must also consider the threshold-based promotion policy to reflect the behavior of migration. To mimic the migration decisions, the hit estimation mechanism tracks sampled frames both in the fast and slow memory. The total size of the sampled region for each frame size is 16MB, which are distributed randomly across the unified address space. For example, with 4KB frame size, 4096 frames are sampled, while for 2MB frame size, 8 frames are sampled.

Since the number of sampled frames can be large for small frame sizes, we employ a simple counting bloom filter for each frame size, not to store the tags. The bloom filter reduces the accuracy, but it can reduce the area overhead significantly. In each entry of a bloom filter, one bit is used to mark whether the frame is in the fast or slow memory hypothetically for the frame size. Extra bits are used for each entry to count the number of accesses to the entry. The extra bits differ by the frame size, as the promotion thresholds are different. For example, for 4KB frame size, a hit filter has 4096 entries, and each entry contains 3 bits (one for fast/slow memory status, and two bits for counter).

For every LLC miss, if the miss address (in the unified address space) is one of the sampled addresses, the corresponding entry of a hit filter is accessed. If the miss address is sampled by more than one filter, multiple hit filters are accessed for different frame sizes. In each hit filter, fast memory hits are counted, depending on the status bit. In addition, the access counter of the entry can be incremented.

Final decision making: When selecting the best frame size, the scores of each candidate sizes is calculated based from the fast memory and mem-TLB hit rates estimated for the frame size. Among five frame sizes, the size with the highest score is selected. The score is a weighted sum of the estimated hit rates of mem-TLBs and fast memory. Note that the hit rates of mem-TLBs and the hit rates of fast memory tend to change in opposite directions when the frame size is increased. However, each application has a different curve shapes for the two hit rates. The controller selects the frame size of the point that two curves meet to minimize the miss overhead from two factors.

The effect on the performance from each factor may vary, depending on the configuration of heterogeneous memory, as the cost of mem-TLBs and fast memory misses can change by the configuration. In this paper, we get the best coefficient by experiments, and use the frame size with the maximum score reflecting relative weights between mem-TLB and fast memory hit rates. The score for each frame size `F` is as follows:

$$score[F] = hit\_rate_{memTLB}[F] + (\alpha \times hit\_rate_{fastMem}[F]) \quad (1)$$

hit_rate_{memTLB} and $hit\_rate_{fastMem}$ are the hit rates of mem-TLBs and fast memory for a given frame size. α is tuned for the current fast and slow memory latencies. For the evaluation, α was set to 5.

C. Per-class Frame Size Selection

A limitation of the global frame size is that multiple workloads in a system can be heterogeneous and they prefer different frame sizes. To support such multi-workload scenarios, we extends the global frame size with multiple workload classes. The administrator can set a workload class for a given application, and a frame size is selected for each class. To support per-class frame sizes, the per-class frame size for each 2MB region is encoded in the first unified table entry of the 2MB region. The operating system initializes the frame size information in the unified frame table. Once the sampled period is done, a new frame size for each class is encoded in the page table entries for 2MB-aligned addresses.

The translation step for the dynamic frame size requires two accesses to the mem-TLBs, as it must retrieve the per-class frame size information from 2MB entries before accessing the actual translation entry. First, an LLC miss accesses the mem-TLBs as a 2MB frame size. The retrieved 2MB entry has the frame size information in addition to the real frame number of the entry. If the frame size of the 2MB region is smaller than 2MB, an additional access to the mem-TLB occurs to retrieve the TLB entry for the requested frame. This translation mechanism requires a two-step translation when the frame size, smaller than 2MB. However, it provides the flexibility of providing per-class frame sizes. As the 2MB granularity entry covers a large 2MB region, its hit chance in the mem-TLBs is very high. Therefore, the translation requires two TLB accesses, but the first access at the 2MB entry is mostly a hit in the mem-TLB.

To select the best frame size for each class, hit filters are replicated for each class. Therefore, to provide 8 classes, 8 hit filter sets are necessary. However, as mem-TLBs are shared, the shadow mem-TLBs are also shared by all classes.

D. Area Overheads

Shadow mem-TLBs have 32 entries and operate on the sampled sets, and each entry has only a tag for hit rate measurement. A hybrid memory system has five showow mem-TLBs with 32 entries, correspoding to five candidate frame sizes. For 4KB frame size, an entry requires 52 bits tag area, and the total area for 32 entries is 208B. The total area overhead for five candidate framesizes is about 1KB.

Each hit filter is a bloom filter with multiple bits in each entry. With 16MB sampled region, 4096, 1024, 256, 64, and 8 entries are used for 4KB, 16KB, 64KB, 256KB, and 2MB frame sizes, respectively. For 4KB frame size, a 2-bit counter and a status bit are used for each entry, and the number of counter bits increases as the threshold doubles with doubling frame sizes. The total area for five frame sizes is about 3KB. To support 8 classes, 24KB is necessary.

Component	Configuration
CPU	8 cores, x86-64 ISA, 3.2GHz
L1 cache	32KB private, 8-way set-associative (SA)
L2 cache	256KB private, 8-way SA
L3 cache	8MB shared, 16-way SA
core-TLB	1024/512 entries per core (conv/two-level) 4-way SA, miss latency 50 cycles
mem-TLB	4096 entries, 8-way SA, miss latency 200 cycles
DRAM	256MB, 8 channels, DDR4-1600 tCAS = 11, tRCD=11, tRP= 11, tRAS = 28
PCM	4 channels, read/write latency = 150/300ns

TABLE I: Configuration of the simulated system.

V. Evaluation

A. Methodology

Our evaluation uses an execution-driven simulator that combines ZSim [13] and DRAMSim2 [12]. Table I presents the detailed simulation parameters for cores, cache hierarchy, and hybrid memory configurations. We use core-TLBs with 1024-entries for the baseline system. For the decoupled translation system, we configure 512-entries 4-way set-associative core-TLBs and 4096-entries 8-way set-associative mem-TLB. For fair comparison, the number of total entries of core-TLBs and mem-TLBs in the decoupled translation is identical to that of core-TLBs in the baseline system. We set the minimum frame size to 4KB and maximum to 2MB for dynamic frame size selection. Our fast and slow memory components are based on the DDR4 and PCM technology. We provision 32MB of fast memory per core, and 512MB of fast (DRAM) memory is used for 8 cores.

For workloads, we use memory-intensive benchmarks from the SPEC CPU 2006, SPEC CPU2017, Biobench, and NAS Parallel Benchmark (NPB) [3]. With the 8-core configuration, we execute eight copies of each workload. After fast-forwarding a workload for 50 billion instructions, we collect the results by running 500 million instructions except for two workloads (`mcf` and `tiger`) that have very low IPCs. For these workloads, we run 100 million instructions to limit the simulation time.

Our baseline system uses a software-managed page migration technique, which employs the conventional virtual memory system with a single address translation layer. Its hotness detection is based on the prior HMA system [10]. The two baselines, `conv 4K` and `conv 2M`, periodically measure the access count for each page and classify a page as *hot* when the access count exceeds a specific threshold. `Conv 4K` uses 4KB page size, and `conv 2M` uses 2MB huge page size only.

B. Results

Performance with dynamic frame sizes: Figure 5 presents speedups normalized to the `conv 4K` run. The figure shows

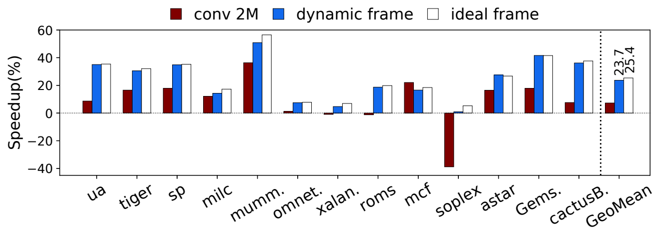

Fig. 5: Speedups normalized to the baseline with 4KB page size (`conv 4K`).

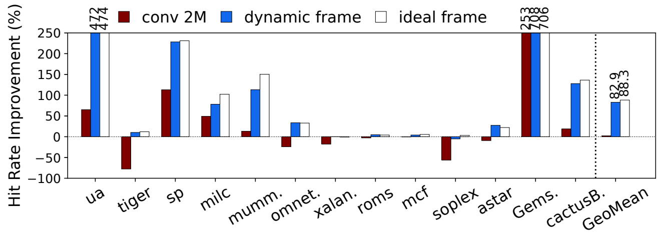

Fig. 6: Improvements of fast memory hit rates compared to the baseline with 4KB page size (`conv 4K`).

three bars. `Conv 2M` uses 2MB page instead of 4KB with the conventional single translation. `Dynamic frame` uses our dynamic frame size mechanism with five candidate frame sizes. `Ideal frame` represents an ideal selection of the best frame size for each workload. For the results with `dynamic frame` and `ideal frame`, the core-TLBs use 2MB page size.

`Dynamic frame` provides performance close to `ideal frame`, showing that our estimation mechanism performs effectively. Its average performance is lower only by 1.7% compared to the ideal frame size. On average, `dynamic frame` achieves 23.7% speedup compared to `conv 4K`. Compared to the two baselines, `dynamic frame` consistently outperforms both `conv 4K` and `conv 2M`. Several workloads have much higher speedups with `conv 2M` than with `conv 4K`. There are two reasons they perform better on `conv 2M`: (1) Performance degradation from core-TLB misses is greatly reduced by using huge pages, and (2) fast memory hit rates are also increased by high spatial locality. For `tiger` and `omnet`, even if the fast memory hit rates of `conv 4K` are higher than `conv 2M`, `conv 2M` has better performance, since the gain from the high fast memory hit rates is offset by the high core-TLB misses in `conv 4K`. Note that decoupled translation mitigates such a conflict as the core-TLBs use 2MB pages, when the mem-TLBs chooses to use a small page size.

Figure 6 presents the improvements of fast memory hit rates compared to `conv 4K`. On average, `dynamic frame` improves hit rates by 83% from `conv 4K`, and it is close to `ideal frame` which has 88% improvement over `conv 4K`. It shows that using larger frame sizes than 4KB not only improves translation efficiency, but also has prefetching effects when applications have strong spatial locality. However, for

	Workloads
mix1	milc, tiger
mix2	cactusBSSN, sp
mix3	mummer, ua
mix4	xalancbmk, roms
mix5	omnetpp, xalancbmk, soplex, mcf
mix6	tiger, ua, sp, mcf

TABLE II: Mix scenarios for multi-class evaluation.

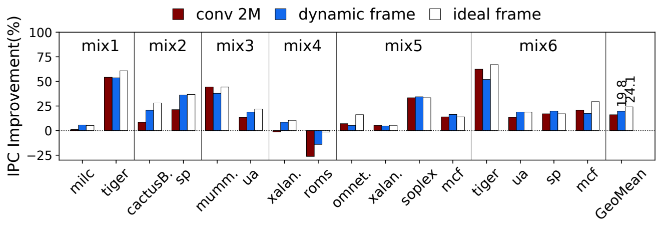

Fig. 7: IPC improvements compared to `conv 4K`.

applications with low spatial locality, such as `tiger`, large page sizes can reduce fast memory hit rates, but our dynamic frame size selection chooses a small frame size for such workloads.

Performance with multi-class dynamic frames: For multi-class evaluation, we run different applications in a system with multiple classes. Table II shows the workload selection for six mix scenarios. For each scenario, an equal number of instances of each application run on 8 cores. Figure 7 presents IPC (instructions per second) improvements compared to `conv 4K`. In the mix scenarios, `dynamic frame` perform effectively, close to `ideal frame` within 4.3% difference. In some workloads, inaccurate selections occur during execution, since mem-TLB hit rate estimation can be affected by co-running workloads. Nevertheless, `dynamic frame` achieves 19.8% performance improvement compared to `conv 4K` on average. Compared to `conv 2M`, the performance gain is smaller, since the mix selection contains many 2MB-favoring applications.

Fairness: In addition, we investigate how fairness among workloads is affected by different hybrid memory schemes. Strict fairness ($StrictF$) is defined as follows [15]:

$$StrictF = \frac{min_i rIPC_i}{max_i rIPC_i}, \text{ where } rIPC = \frac{IPC^{MT}}{IPC^{ST}}$$

$rIPC$ is a relative IPC, and MT denotes for multi-programmed runs while ST for single runs. Figure 8 shows the fairness result. In the figure, `conv 4K` shows a good fairness since it can provide fine-grained assignments of fast memory across applications. However, `conv 2M` exhibits a much worse fairness result, as it can often cause unbalanced assignments due to the coarse-grained allocation. Although `conv 2M` can improve the throughput, it can potentially degrade fairness. However, `dynamic frame` provides the best of the two conventional ways, and it allows similar fairness to `conv 4K` and better performance than `conv 2M`.

TLB misses and fast memory hit rates: Table III presents TLB Misses Per Million Instructions (MPMI) and fast memory hit rates of the conventional single-level translation and the

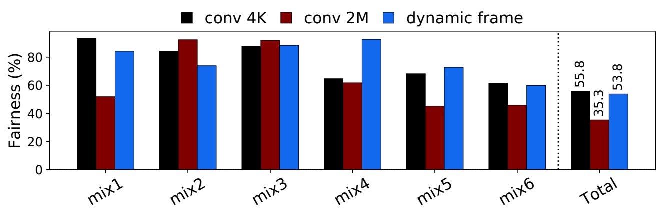

Fig. 8: Fairness of mixed workloads (higher is better.)

	conv. (Single-level)				Two-level Trans.		
	core-TLB MPMI		fast mem. hit rates		mem-TLB MPMI	fast mem. hit rates	ideal frame size(B)
Workloads	4K	2M	4K	2M	dyn.	dyn.	
ua	227.6	0.4	0.06	0.09	11.7	0.33	256K
tiger	194607.2	3.0	0.24	0.05	179979.9	0.27	4K
sp	324.9	0.5	0.13	0.27	83.9	0.42	16K
milc	1289.8	0.6	0.05	0.07	78.5	0.09	2M
mumm.	16278.2	0.4	0.13	0.15	0.4	0.28	1M
omnet.	5197.0	0.2	0.27	0.21	10173.0	0.37	4K
xalan.	1786.0	0.2	0.91	0.75	803.5	0.91	128K
roms	686.9	0.09	0.82	0.80	262.7	0.86	32K
mcf	28604.9	84.1	0.69	0.68	1188.2	0.71	1M
soplex	1110.9	0.2	0.85	0.37	1.8	0.81	16K
astar	18928.3	0.3	0.22	0.20	937.3	0.28	1M
Gems.	1274.9	0.01	0.04	0.14	782.5	0.32	16K
cactusB.	54.1	0.2	0.34	0.40	53.5	0.77	32K

TABLE III: TLB MPMI and fast memory hit rates of conventional single-level systems and two-level translation with dynamic frame size.

two-level translation. For the two-level translation, since the core-TLB MPMI is almost identical to that of the conventional 2MB page size, only the mem-TLB MPMI is shown. The last column shows the ideal frame size. The table shows that the the ideal frame size depends on the combined impact of the mem-TLB and fast memory hit rates. For example, for `tiger`, 4KB frame incurs many mem-TLB misses, but it can provide the best fast memory hit rate. As the improved fast memory hit rate is more important for `tiger`, the ideal frame size is 4KB. Note that the core-TLB uses 2MB page size, hiding the overhead of core-side address translation. For `mcf`, the ideal one is 1MB, if all 10 frame sizes are considered. In that case, the reduced mem-TLB miss rate is the most critical factor.

VI. Conclusions

This paper proposed a technique to dynamically find the best frame size for a workload to reduce the overhead of memory-side address translation for hybrid memory. Although two-level translation can reduce the overheads of page migration for hybrid memory, the paper showed that it is important to select an appropriate frame size. The proposed dynamic frame size selection mechanism can almost match the performance of an ideal frame size for each workload.

VII. Acknowledgement

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grants funded by the Ministry of Science and ICT, Korea (IITP2017-0-00466, ITRC support program IITP-2022-2021-0-01817) and National Research Foundation of Korea (NRF-2022R1A2B5B01002133).

References

[1] *Compute Express Link (CXL)*. [Online]. Available: https://www.computeexpresslink.org/

[2] N. Agarwal and T. F. Wenisch, "Thermostat: Application-transparent Page Management for Two-tiered Main Memory," in *Proceedings of the 2017 22nd Annual IEEE/ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2017.

[3] D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagum, R. Fatoohi, P. Frederickson, T. Lasinski, R. Schreiber, H. Simon, V. Venkatakrishnan, and S. Weeratunga, "The Nas Parallel Benchmarks," *Int. J. High Perform. Comput. Appl.*, vol. 5, no. 3, pp. 63–73, 1991.

[4] A. Basu, J. Gandhi, J. Chang, M. D. Hill, and M. M. Swift, "Efficient Virtual Memory for Big Memory Servers," in *Proceedings of the 2013 40th Annual IEEE/ACM International Symposium on Computer Architecture (ISCA)*, 2013.

[5] 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 *Proceedings of the 2014 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2014.

[6] T. Heo, Y. Wang, W. Cui, J. Huh, and L. Zhang, "Adaptive page migration policy with huge pages in tiered memory systems," *IEEE Transactions on Computers*, vol. 71, no. 1, pp. 53–68, 2022.

[7] D. Jevdjic, G. H. Loh, C. Kaynak, and B. Falsafi, "Unison Cache: A Scalable and Effective Die-Stacked DRAM Cache," in *Proceedings of the 2014 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2014.

[8] V. Karakostas, J. Gandhi, F. Ayar, A. Cristal, M. D. Hill, K. S. McKinley, M. Nemirovsky, M. M. Swift, and O. Ünsal, "Redundant Memory Mappings for Fast Access to Large Memories," in *Proceedings of the 2015 42Nd Annual IEEE/ACM International Symposium on Computer Architecture (ISCA)*, 2015.

[9] S. Kim, S. Lee, T. Kim, and J. Huh, "Transparent dual memory compression architecture," in *Proceedings of the 2017 26th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2017.

[10] M. R. Meswani, S. Blagodurov, D. Roberts, J. Slice, M. Ignatowski, and G. H. Loh, "Heterogeneous memory architectures: A HW/SW approach for mixing die-stacked and off-package memories," in *Proceedings of the 2015 21st Annual IEEE/ACM International Symposium on High Performance Computer Architecture (ISCA)*, 2015.

[11] M. K. Qureshi and G. H. Loh, "Fundamental Latency Trade-off in Architecting DRAM Caches: Outperforming Impractical SRAM-Tags with a Simple and Practical Design," in *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture (ISCA)*, 2012.

[12] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "DRAMSim2: A Cycle Accurate Memory System Simulator," *IEEE Computer Architecture Letters*, vol. 10, no. 1, pp. 16–19, 2011.

[13] D. Sanchez and C. Kozyrakis, "ZSim: Fast and Accurate Microarchitectural Simulation of Thousand-core Systems," in *Proceedings of the 2013 40th Annual IEEE/ACM International Symposium on Computer Architecture (ISCA)*, 2013.

[14] J. Sim, A. R. Alameldeen, Z. Chishti, C. Wilkerson, and H. Kim, "Transparent Hardware Management of Stacked DRAM As Part of Memory," in *Proceedings of the 2014 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2014.

[15] H. Vandierendonck and A. Seznec, "Fairness Metrics for Multi-Threaded Processors," *IEEE Computer Architecture Letters*, vol. 10, no. 1, pp. 4–7, 2011.

[16] Z. Yan, D. Lustig, D. Nellans, and A. Bhattacharjee, "Nimble Page Management for Tiered Memory Systems," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2019.

[17] L. Zhang, E. Speight, R. Rajamony, and J. Lin, "Enigma: architectural and operating system support for reducing the impact of address translation," in *Proceedings of the 24th ACM International Conference on Supercomputing (ICS)*, 2010.