DRAM Translation Layer: Software-Transparent DRAM Power Savings for Disaggregated Memory

Wenjing Jin
Seoul National University
Republic of Korea
wenjing.jin@snu.ac.kr

Wonsuk Jang
Seoul National University
Republic of Korea
skylark0827@snu.ac.kr

Haneul Park
Seoul National University
Republic of Korea
skyp0714@snu.ac.kr

Jongsung Lee
Seoul National University
Samsung Electronics
Republic of Korea
leitia@snu.ac.kr

Soosung Kim
Seoul National University
Republic of Korea
soosungkim@snu.ac.kr

Jae W. Lee
Seoul National University
Republic of Korea
jaewlee@snu.ac.kr

ABSTRACT

Memory disaggregation is a promising solution to scale memory capacity and bandwidth shared by multiple server nodes in a flexible and cost-effective manner. DRAM power consumption, which is reported to be around 40% of the total system power in the datacenter server, will become an even more serious concern in this high-capacity environment. Exploiting the low average utilization of DRAM capacity in today's datacenters, it is appealing to put unallocated/cold DRAM ranks into a power-saving mode. However, the conventional DRAM address mapping with fine-grained interleaving to maximize rank-level parallelism is incompatible with such rank-level DRAM power management techniques. Furthermore, existing DRAM power-saving techniques often require intrusive changes to the system stack, including OS, memory controller (MC), or even DRAM devices, to pose additional challenges for deployment. Thus, we propose *DRAM Translation Layer (DTL)* for host software/MC-transparent DRAM power management with commodity DRAM devices. Inspired by Flash Translation Layer (FTL) in modern SSDs, DTL is placed in the CXL memory controller to provide (i) flexible address mappings between host physical address and DRAM device physical address and (ii) host-transparent memory page migration. Leveraging DTL, we propose two DRAM power-saving techniques with different temporal granularities to maximize the number of DRAM ranks that can enter low-power states while provisioning sufficient DRAM bandwidth: *rank-level power-down* and *hotness-aware self-refresh*. The first technique consolidates unallocated memory pages into a subset of ranks at deallocation of a virtual machine (VM) and turns them off transparently to both OS and host MC. Our evaluation with CloudSuite benchmarks demonstrates that this technique saves DRAM power by 31.6% on average at a 1.6% performance cost. The hotness-aware self-refresh scheme further reduces DRAM energy consumption by up to 14.9% with negligible performance loss via opportunistically migrating cold pages into a rank and making it enter self-refresh mode.

CCS CONCEPTS

• **Hardware** → **Memory and dense storage**; **Enterprise level and data centers power issues**.

KEYWORDS

DRAM, Power Management, Datacenters, Disaggregated Memory, CXL, Pooled Memory, Address Translation

ACM Reference Format:
Wenjing Jin, Wonsuk Jang, Haneul Park, Jongsung Lee, Soosung Kim, and Jae W. Lee. 2023. DRAM Translation Layer: Software-Transparent DRAM Power Savings for Disaggregated Memory. In *Proceedings of the 50th Annual International Symposium on Computer Architecture (ISCA '23), June 17–21, 2023, Orlando, FL, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3579371.3589051

ISCA '23, June 17–21, 2023, Orlando, FL, USA

ACM ISBN 979-8-4007-0095-8/23/06.
https://doi.org/10.1145/3579371.3589051

1 INTRODUCTION

Recently, demands for DRAM capacity and bandwidth have increased rapidly to run emerging datacenter applications in memory, such as machine learning and data analytics, with low latency and/or high throughput [43, 44]. However, the conventional multi-socket server imposes strong coupling between compute resources and memory resources to make it challenging to scale DRAM capacity and bandwidth flexibly according to applications' demands [35, 40]. Several proposals attempt to address this issue by augmenting OS to classify hot and cold memory pages and offload some of the cold pages to either persistent storage or unused DRAM space of remote nodes [19, 20, 31]. However, they incur a large latency/throughput penalty for memory due to coarse-grained data transfers with heavyweight disk or network I/Os.

Memory disaggregation is a promising solution to this problem by allowing datacenter providers to scale both DRAM capacity and bandwidth in a flexible and cost-effective manner. Emerging processor interconnects such as Compute Express Link (CXL) [3] catalyze the adoption of this technology. CXL provides high-bandwidth, low-latency, cache-coherent memory load/store semantics through the

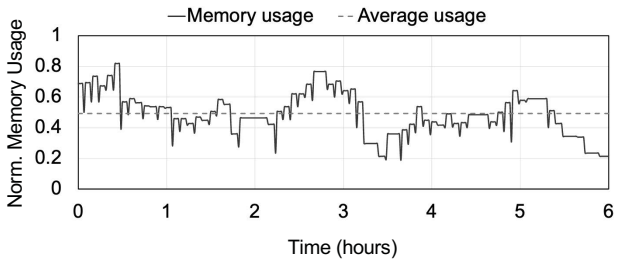

Figure 1: Memory usage profiling of Microsoft Azure VM traces.

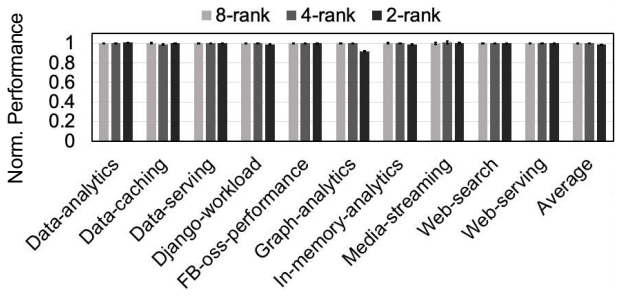

Figure 2: Performance profiling with varying numbers of active ranks.

PCIe-based physical interface, enabling applications to access large-capacity disaggregated memory without OS intervention. State-of-the-art datacenter CPUs, such as AMD's Genoa [8] and Intel's Sapphire Rapids [4], already support CXL for memory expansion.

However, the widespread adoption of CXL-based high-capacity, high internal-bandwidth DRAM devices will pose another challenge in DRAM power management, which is already a serious concern. According to a recent report from Meta [57], DRAM power consumption is expected to reach 38% of total power consumption in their datacenter infrastructure. Deployment of terabyte-scale disaggregated DRAM devices in datacenters is likely to push up this number by lowering the compute-to-memory capacity ratio. Thus, reducing DRAM power consumption in disaggregated memory is critical to save total cost of ownership (TCO) in datacenters [41].

JEDEC standards already specify low-power states to save DRAM energy consumption, such as maximum power saving mode (MPSM) and self-refresh mode (SR) [48]. The former turns off a DRAM device with no data retention for maximal energy savings; the latter maintains data retention with periodic refresh commands. However, it is still challenging to exploit them in the conventional server partly due to the practice of DRAM address interleaving to maximize rank-level parallelism [34]. Existing proposals for managing the power state of a specific rank include memory allocation-time coordination, run-time active migration, or both. However, these proposals require intrusive modifications to OS [24, 26, 30, 34, 59], memory controller (MC) [17, 21, 28, 34, 59], or even DRAM devices [28, 34], to pose deployment challenges in today's datacenters.

To address this challenge, we propose *DRAM Translation Layer (DTL)*, a mechanism for flexible address mapping and data migration *within* CXL-based memory devices. Inspired by Flash Translation Layer (FTL) in modern SSDs, DTL introduces a level of indirection in address translation from host physical address (HPA) to DRAM device physical address (DPA). Unlike FTL, DTL adopts a hardware automated data path to have minor (<2%) impact on remote memory access latency from the host with a low hardware cost. Leveraging DTL, we can effectively consolidate unallocated/cold DRAM pages into a subset of ranks to maximize rank-level power-saving opportunities while provisioning sufficient DRAM bandwidth. In particular, we propose two DRAM power management techniques: *rank-level power-down* and *hotness-aware self-refresh*. The former applies the maximum power saving mode (MPSM) to a subset of ranks by consolidating unallocated pages to them at deallocation of a virtual machine (VM). The latter further saves DRAM power over the remaining active ranks that require data retention by entering self-refresh (SR) mode for a rank to which cold pages are collected. Both techniques are completely transparent to the host, requiring no changes to OS, hypervisor, MC, or DRAM device itself. Our evaluation with CloudSuite benchmarks shows the rank-level power-down technique reduces DRAM energy consumption by 31.6%. For the remaining active ranks, the hotness-aware self-refresh scheme can provide additional DRAM energy savings by up to 14.9%.

Our contributions can be summarized as follows:

- We introduce DRAM Translation Layer (DTL), the first proposal of a mechanism for address (re)mapping from host physical address to DRAM device physical address and memory page migration transparent to both software stack and MC on the host.
- Leveraging DTL, we propose a rank-level power-down technique to maximize the number of ranks with none of their pages allocated to any VMs and hence can be safely put into maximum power saving mode (MPSM).
- We also propose a hotness-aware self-refresh scheme to further save DRAM energy by collecting and migrating cold pages to a rank and putting it into self-refresh mode.
- We present detailed evaluation of the two power-saving techniques to show their effectiveness in terms of DRAM power savings and negligible performance cost via both real-machine measurements and trace-driven simulation.

2 BACKGROUND AND MOTIVATION

Low Utilization of DRAM Capacity in Datacenters. Prior studies show that DRAM power consumption already accounts for a considerable portion of the total system power [9, 22, 57] and an increasing portion of the total cost of ownership (TCO) in datacenters [41]. However, the memory capacity utilization on average is relatively low with only around 40-60% [34, 38, 46, 52]. We reproduce this in Figure 1 using the VM traces from Microsoft Azure public dataset [15]. We randomly pick 400 VMs whose trace contains information about the numbers of vCPU, vMemory size, and the lifetime following the same original distribution and schedule them for six hours on a server machine with 48 vCPUs and 384 GB memory. The results demonstrate that the average memory capacity usage is less than 50%. Note that the memory usage here means a sum of memory capacity reserved for all active VMs. The actual

memory footprint may be smaller as there are free, unallocated pages within each VM.

Low Returns from Rank-level Parallelism. We also find there is only small performance degradation even if we reduce the number of available ranks in a server machine while keeping the number of DRAM channels and banks constant. We run 10 CloudSuite benchmarks [2] on a real machine with four DRAM channels and varying numbers of ranks per channel from eight through two. We allocate 64GB of memory and 28 physical CPU cores in each configuration. Each workload is tuned to enlarge the working set as much as possible. Figure 2 shows an average of 0.7% performance loss for the 2-rank configuration compared to the 8-rank configuration. This is partly because of a sufficient amount of bank/channel-level parallelism giving rank-level parallelism only marginal additional returns. A similar observation is made in a long-latency disaggregated memory environment (Section 3.3). Thus, there are opportunities to take advantage of this, together with low memory utilization, to save DRAM power with little performance loss.

DRAM Low-power States. DRAM modules support several low-power states performed at the rank-level, which consists of a group of DRAM banks selected by the Chip Select (CS) bit and operating in tandem. In the self-refresh mode, the DRAM device retains data without external clocking and thus consumes only about 10-20% of power [30, 34] compared to the standby (active) mode, with an exit penalty of hundreds of nanoseconds [47]. The maximum power saving mode (MPSM) is similar but with no self-refresh. MPSM does not guarantee data retention and does not respond to any external commands other than *MSPM_exit*, thus providing the lowest power state. According to JEDEC, turning off refresh operations for half of the ranks could reduce the DRAM power in the self-refresh mode by 33% [32]. This implies that DRAM in MPSM consumes only 3.4-6.8% of power compared to the standby power. MPSM exit penalty is in an order of hundreds of nanoseconds [47].

Limitations of Existing Work. Existing proposals to leverage the low-power states of DRAM devices for system power savings mostly target the conventional server and hence require modifications to OS, MC, or both. On the software side, virtual-to-physical address mappings are controlled by OS and it is very challenging to consolidate unallocated/cold pages into a specific rank without coordination with/modifications to the OS. This becomes even more challenging with DRAM address interleaving of channel, rank, and bank bits to balance utilization and maximize parallelism in DRAM accesses. To overcome this challenge, there are proposals for hardware support. For example, both RamZzz [59] and Huang et al. [26] propose tracking mechanisms for hot/cold pages and dynamic migration of DRAM pages to consolidate cold pages. GreenDIMM [34] is a recent proposal for sub-array level power management to not interfere with DRAM address interleaving. However, this proposal requires modifications to all of OS, MC, and DRAM devices. In spite of the power-saving potentials, such intrusive changes to the system hardware/software stack require careful multi-party coordination including DRAM vendors, CPU vendors, and datacenter operators, to pose significant deployment challenges.

Our Approach. CXL opens up opportunities for encapsulating power-state management completely *within* the CXL memory device. This makes the DRAM power management transparent to the host, including OS, MC, and user applications. It can be realized by introducing a level of address translation and a mechanism for flexible data migration, which currently lacks in the CXL memory device. To this end, we propose DRAM Translation Layer (DTL) and two device-side power-management schemes for both unallocated and cold pages.

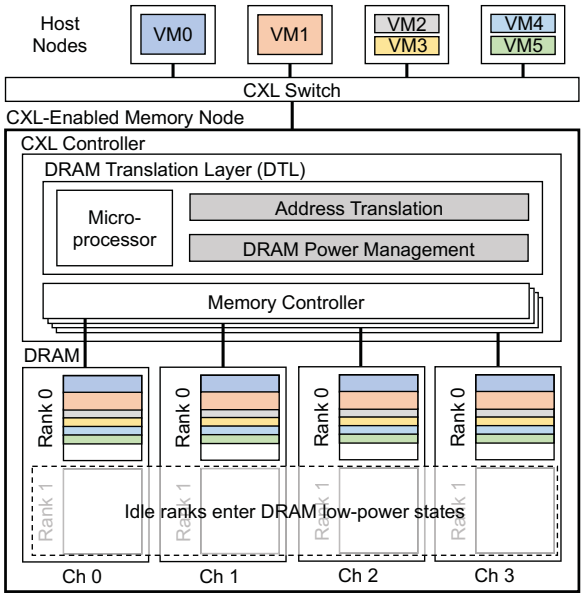

Figure 3: Design overview.

3 DRAM TRANSLATION LAYER

3.1 Overview

Our design goal is to translate unallocated/cold DRAM capacity into DRAM power savings transparently to OS and MC on the host, while minimizing performance degradation caused by address translation or page migration. At the core of our design is DRAM translation layer (DTL), which flexibly controls DRAM address mappings and data migration to maximize the number of DRAM ranks that can be put into low-power states. In particular, DTL serves as an in-device address translation layer between host physical address (HPA) and DRAM device physical address (DPA).

Figure 3 shows an overview of the design. In a multi-tenant VM environment, VMs on multiple compute nodes share a CXL-attached pooled memory node. DTL evenly allocates the VM's data across the memory channels without rank-interleaving. Thus, idle ranks can enter low-power states without compromising available channel bandwidth. We integrate DTL in the CXL controller, which provides the following three functionalities:

- **Address translation** from HPA to DPA to support flexible data migration and address remapping
- **Rank-level power-down** for consolidating unallocated DRAM pages to a subset of ranks at VM deallocation to maximize opportunities for entering maximum power saving mode (MPSM)

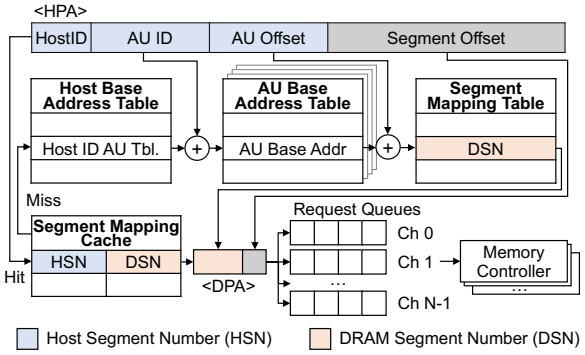

Figure 4: Address translation in DTL.

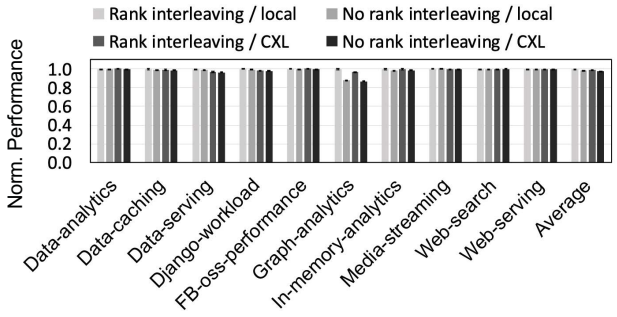

Figure 5: Performance impact of rank-interleaving with different memory access latencies (local and CXL).

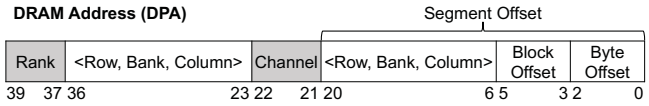

Figure 6: DRAM physical address bit mapping for a 1TB CXL memory device (4 channels with 8 ranks/channel).

- **Hotness-aware self-refresh** for periodically monitoring per-rank accesses to identify a cold (victim) rank and swapping cold-hot pages across ranks to maximize the rank idle time for entering self-refresh mode

3.2 Address Translation

Figure 4 illustrates the HPA-DPA translation process in DTL. DTL adopts address translation at a *segment* granularity. A large segment would reduce the storage overhead of the segment mapping table while increasing the overhead of segment migration and the degree of internal hot/cold fragmentation. We use the segment size of 2MB by default to balance the storage overhead and the segment migration overhead. When a DRAM access request arrives, DTL looks up the *segment mapping cache*, which takes a two-level TLB-like structure for fast HPA-to-DPA translation. Upon a cache hit, the request is sent to the corresponding MC for DRAM access. If it misses, DTL walks through a three-level indirection path to retrieve the HPA-to-DPA mapping.

The address translation process finds the DRAM segment number (DSN) for the given host segment number (HSN). Figure 4 also shows the breakdown of the HPA address bits including HSN and the three tables on the miss path. An HSN consists of host ID, allocation unit (AU) ID, and AU offset, where AU is the minimum memory allocation unit to each VM. We use the AU size of 2GB as it is the minimum vMemory allocation size per instance in the top-three cloud vendors [7, 23, 42]. DTL first retrieves the base address of the AU table for host ID from the host base address table. In the next step, it accesses the corresponding AU table with AU ID as an index to fetch the AU base address. Finally, DTL accesses the segment mapping table using AU offset as an index, which maintains HSN-to-DSN mappings for all segments that belong to the AU in question.

Note that the first two levels of the tables are stored in on-chip SRAM while the last-level table containing all HSN-to-DSN mappings is maintained in DRAM. Section 6.6 discusses the space overhead for these table structures. Thus, the address-translation miss path costs two SRAM accesses followed by one DRAM access. This cache miss path is infrequently walked through as the two-level segment mapping cache filters most of the address translation requests.

3.3 Rank-level Power-down

DRAM Physical Address Bit Mapping. Commodity DRAM devices enter low-power states at a rank granularity. However, the conventional DRAM address interleaving, which evenly distributes data across ranks at a fine granularity, makes it difficult to enter the low-power states. Previous studies [10, 60] suggest that the performance benefits of rank interleaving are relatively small and may become even smaller for CXL devices due to long remote access latency. As shown in Figure 5, our experiments with CloudSuite benchmarks also confirm this point. (Refer to Section 5.1 for the experimental methodology.) Even if we do not exploit fine-grained rank-interleaving (but still exploit channel-interleaving), the average performance loss is only 1.7% for local memory. This cost is further reduced to 1.4% in the long-latency environment for CXL memory. Thus, we decide to utilize channel-interleaving only but no rank-interleaving to uncover opportunities for entering rank-level low-power modes. Figure 6 shows the proposed DRAM address mapping for a 1TB CXL memory device. Rank bits are placed as the most significant bits not to exploit rank-interleaving. Channels are interleaved at segment granularity. Each channel's capacity and bandwidth resources are evenly allocated to every VM instance, thereby exploiting memory-level parallelism across channels and banks.

Rank-level DRAM Power State Transition. DTL performs power state transition at a *rank-group* granularity, where a rank group indicates a set of ranks with the same rank index across all memory channels. DTL keeps track of the memory allocation status of all VMs and the power state of each rank. At every VM deallocation, DTL checks if the unallocated memory capacity among active ranks exceeds the size of a single rank-group. If so, DTL migrates the live segments from the rank-group with the least allocated space to VMs to increase the number of rank-groups in the power-down state (MPSM). Since the capacity utilization of each channel is always the same due to the proposed DRAM address mapping

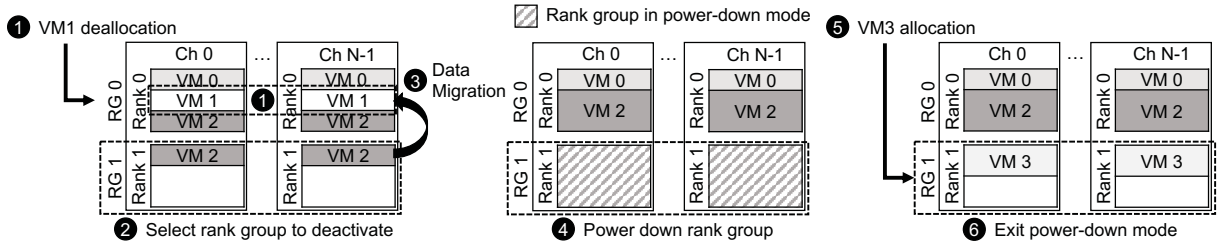

Figure 7: Rank-level Power-down (RG indicates rank group).

scheme, changing the power state at a granularity of individual ranks (instead of rank-groups) would lead to unbalanced usage of DRAM channel bandwidth for some of the live VMs. Therefore, we control the DRAM power state at a rank-group granularity without compromising the available channel bandwidth for each VM instance.

Example Walk-through. Figure 7 illustrates how the proposed power-saving technique works. When the unallocated capacity is large enough to make a power-down rank-group ❶ after deallocation of VM1, ❷ DTL selects rank-group (RG) 1 as a victim rank group due to its low capacity utilization. ❸ The segments of VM2 allocated to rank-group 1 are migrated to rank group 0 for idle-rank expansion. After that, DTL sends a power transition request to the DRAM controller, and ❹ the victim rank-group finally enters the maximum power saving mode to reduce the DRAM background power. When ❺ VM3 asks for memory resource allocation, ❻ rank-group 1 exits power-down mode and is reactivated due to the shortage of available space in the active ranks. There is a latency penalty in order of hundreds of nanoseconds for exiting maximum power saving mode. However, the *MPSM_exit()* command is followed by the initialization stage of the newly allocated VM with memory allocation operations to the DTL reserved memory, rather than regular memory load/store requests from the pre-allocated VMs to the reactivated ranks. Thus, no performance penalty is observed by those existing VMs.

3.4 Hotness-aware Self-refresh

DTL can achieve further power savings via hotness-aware self-refresh. Even if the rank retains only a small amount of live segments, the entire DRAM rank needs to be refreshed periodically for data retention. No further DRAM power savings can be achieved if we apply only rank-level power-down. The basic idea of this optimization is to collect cold segments to a specific victim rank and put that rank into self-refresh mode.

Additional Structures for Hotness-aware Self-refresh. Additional SRAM structures called *migration table*, are introduced to track access to each segment. As shown in Figure 8, every segment has an entry with three fields: access bit, rank number, and segment number. The last two numbers indicate the target rank-segment pair for migration. The DTL enables us to find a segment remapping plan that would maximize idle time for the victim (cold) rank. Besides, DTL provides a per-rank memory access counter to select the least accessed rank as the victim rank to enter self-refresh mode and a timer for gauging the coldness of the segments in the victim rank.

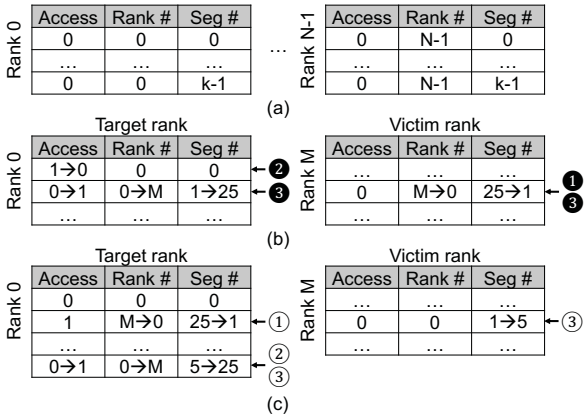

Figure 8: Illustration of migration table updates: (a) initial state, (b) update for the segment access in the victim rank, and (c) update for access to the segment whose corresponding entry has already been swapped.

Two Phases of Hotness-aware Self-refresh. Hotness-aware self-refresh consists of two phases: profiling phase and migration phase. DTL suppresses excessive data migration by postponing entering the migration phase until a sufficiently long idle time is expected after migration. Specifically, DTL transitions to the migration stage only if there is no access to the hypothetical victim rank that would be produced by segment migration according to the migration table for the profiling time threshold (e.g., 50ms). Otherwise, the timer will be reset to stay in the profiling phase.

Phase 1: Profiling. At the beginning of the profiling phase, the rank with the fewest accesses for each channel over a specific time window (i.e., 0.5ms) is selected as the *victim rank*. A *target rank* is chosen in a round-robin manner among the other active ranks. A cold segment in the target rank is progressively swapped with a hot segment in the victim rank to fill the victim rank ideally with cold segments only. The target segment pointer (TSP) is utilized to identify a cold segment in the target rank, whose role is similar to the hand (iterator) in the CLOCK algorithm [14]. CLOCK implements pseudo-LRU for OS page replacement with a 1-bit reference for each page frame to store access history. TSP traverses from the first entry and wraps around once it reaches the bottom of the rank.

Figure 8(b) and (c) illustrate the migration table update algorithm for access to the segment whose corresponding entry is in the hypothetical victim rank (with the rank number set to M). ❶ If the

segment being accessed is in the victim rank, the corresponding entry in the victim rank swaps with the entry pointed by the TSP. Like the CLOCK algorithm, if the access bit is set to 1, ❷ DTL resets its access bit to 0 and then moves TSP to the next entry. ❸ TSP keeps moving to the next entry until an entry whose access bit is 0 or a timeout occurs. Then it swaps the rank and segment numbers between the victim entry and the target entry. Note that actual segment migration does not happen until the migration phase; it only *simulates* a segment migration plan that potentially maximizes the victim rank idle time.

A timeout is enforced for two reasons. The first is to prevent a long search time (i.e., over a single DRAM access latency) for a cold segment in the target rank. Otherwise, there would be a backlog with DRAM requests arriving continuously. The second is to collect cold segments from multiple target ranks. When a timeout occurs, TSP moves to the next target rank in a round-robin fashion. The default timeout threshold is set to 40ns, which is shorter than a single DRAM access latency.

If the corresponding entry of the segment being accessed has already been swapped, which means this segment is not cold, ① DTL restores the entry's rank and segment number. ② Then DTL first retrieves the entry whose access bit is 0 in the target rank and ③ swaps the rank and segment number with the entry in the victim rank.

Phase 2: Migration. In the migration phase, DTL traverses the entire victim rank and finds the hot segments that need to be migrated. DTL performs a swap operation between the hot segment in the victim rank and the corresponding cold segment in the target rank. A new HPA-to-DPA mapping update, including an update of the segment mapping table and an invalidation of the corresponding entry in the segment mapping cache, if any, follows every swap operation. After all migration is done, the rank and segment numbers in the migration table are re-initialized. Finally, DTL puts the victim rank into self-refresh mode.

Exit from and Re-entry into Self-refresh Mode. If a segment of a rank in self-refresh mode is accessed, the rank is transitioned back to the standby mode. DTL restarts the profiling stage and selects a victim rank. In most cases, a reactivated rank requires only a small amount of data migration to re-enter the self-refresh mode. This is because most segments in the victim rank still remain cold.

4 IMPLEMENTATION

4.1 Address Mapping Granularity

Consideration of Memory Access Stride. DTL proposes a new segment-level indirection layer for HPA to DPA translation. As we mentioned in Section 3.3, DTL interleaves channel bits in DRAM address but not rank bits. To exploit channel-level parallelism effectively, a segment size smaller than the memory access stride is preferred to distribute adjacent memory accesses to different channels. Note that our DRAM physical address (DPA) has channel bits at the next higher-order bits to the segment offset (Figure 6).

Figure 9 shows the distribution of post-cache memory access strides using 8 CloudSuite benchmarks for both single-application traces and mixed traces. The details of CloudSuite trace generation are described in Section 5.2. There are a couple of observations. First, even with single-application traces, the memory stride of

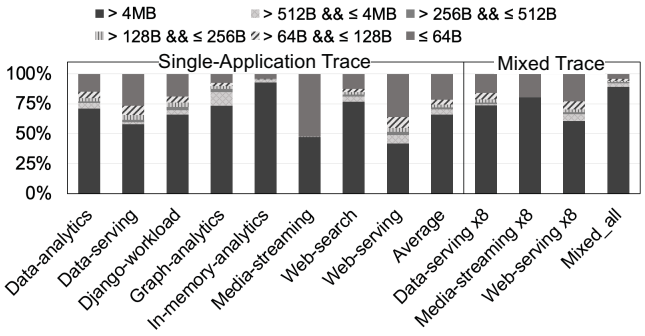

Figure 9: Memory access stride distribution.

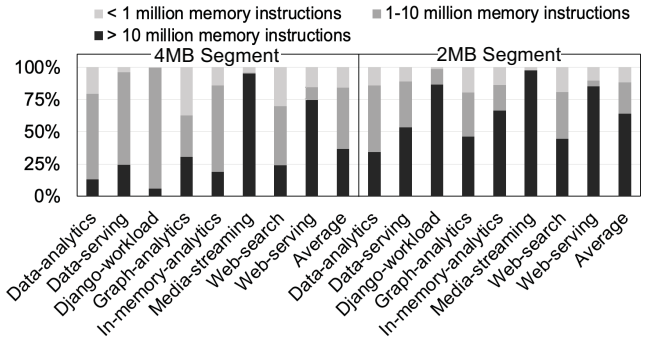

Figure 10: Segment size and segment access distance.

4MB or longer has a dominating portion in the distribution. Second, with multiple copies of applications running, the portion of 4MB or longer becomes even higher. In particular, the portion of 4MB strides or higher becomes dominant even for the three applications with narrow access strides in standalone execution (i.e., Data-serving, Media-streaming, and Web-serving). Moreover, 89.3% of memory accesses have a stride greater than 4MB in the mixed workload of the 8 applications.

Thus, on the one hand, any segment size smaller than or equal to 4MB would be acceptable to exploit channel-level parallelism. On the other hand, we prefer a segment size of 2MB or larger to reduce the storage overhead for the segment mapping table and fit the migration table in on-chip SRAM. Therefore, we consider 2MB and 4MB for candidate segment size.

Segment Reuse Distance and Mapping Granularity. We further analyze the correlations between the remapping granularity and the segment reuse distance of the post-cache memory accesses. As shown in Figure 10, 33.2% and 61.5% of segments can be considered cold with 4MB and 2MB remapping granularity, respectively, since the access distance is more than 10 million *memory* instructions. For hotness-aware self-refresh, a high ratio of cold segments implies a greater chance of collecting enough cold segments to put a rank into self-refresh mode and maintain this low-power state. Given that 2MB granularity shows a much higher ratio of cold segments, we choose 2MB for the address mapping granularity to keep the space overhead manageable while maintaining a large portion of cold segments.

4.2 Data Migration

Two Types of Data Migration. In rank-level power-down, DTL *copies* the live segments in the victim rank to the free space in the remaining ranks. In contrast, in hotness-aware self-refresh, DTL *swaps* cold and hot segments according to the content of the migration table.

There are three hardware structures to support data migration: allocated segment queue, free segment queue, and reverse mapping table. The allocated (free) segment queue consists of multiple queues, each containing allocated (free) memory addresses of the corresponding rank, which is used for memory allocation and searching the source (destination) address of copy operation. The reverse mapping table maintains the mapping information from DSN to HSN, which is used for updating the segment mapping table after data migration. These structures are stored in the reserved DRAM space where the segment mapping table is placed.

Atomic Data Migration. There are two separate queues on the scheduling path of each channel: foreground request queue and migration queue. A single segment migration request is internally broken down into cache line-sized requests to access DRAM, and internal counters track the progress of migration. To ensure foreground performance, the migration queue issues a request only if there is no pending DRAM access request in the foreground request queue of the same channel. After the migration of the entire 2MB segment is complete, the mapping information is updated by invalidating the corresponding entry, if any, in the segment mapping cache, and updating the segment mapping table.

DTL holds the status of outstanding migration requests in registers for each channel, including the old DSN, the new DSN, and the completion bit. The completion bit is set when the migration request is complete, but the mapping table still needs to be updated. When a foreground write request arrives, DTL first checks if the request falls on any of the segment being migrated. If not, it just proceeds normally to issue a DRAM request. If so, DTL checks whether the completion bit is set. If the completion bit is 1, DTL issues a write request to the foreground queue corresponding to the new DSN. If not, depending on whether the requested cache line is already migrated or not, DTL either proceeds with the write with the original DSN (if not migrated yet) or simply aborts the entire request in progress (if already migrated). Then the internal counter is reset and the migration request is retried. If the time of retries for a request exceeds a certain threshold (e.g., three), DTL moves the request to the tail of the migration queue for re-execution. Correctness is guaranteed as foreground requests have higher priority than migration requests.

4.3 Support Functions for Segment-level Operations

Balancing Segment Allocation. When DTL is required to allocate memory for a VM, each channel provides an equal number of free segments from its free segment queue. Moreover, for the rank with the highest capacity utilization in each channel, its free segment queue has the highest priority when providing free segments. This strategy minimizes data migration overhead while provisioning sufficient bandwidth.

Table 1: Experimental setup.

Server configurations	
CPU	Intel Xeon Gold 6258R, 28-core @ 2.7GHz
DIMM configuration	4R x4 128GB DDR4-2933 DIMMs
Channel	6 channels, 2 DIMMs per channel
Total DRAM size	1TB (384GB used)
Memory access latency	
Native DRAM	121ns
CXL memory	210ns

Virtualizing Rank Group. Due to migration at segment granularity, the utilization of each rank in a rank-group may vary due to hotness-aware self-refresh. Thus, when a VM is deallocated and rank-level power-down is applied, the index of the idle rank can be different in each channel. In this case, DTL forms a virtual rank-group by taking idle ranks with different rank IDs and putting it into maximum power saving mode.

5 METHODOLOGY

5.1 Setup for Rank-level Power-down

Experimental Setup. We use a real server machine and emulate CXL memory latency with Quartz [54]. The machine configurations are summarized in Table 1. We put ranks into a low-power state at a granularity of two ranks instead of one since a pair of ranks share a clock enable pin (CKE) [48] in the DRAM DIMM we used. DRAM access latency measured by Intel Memory Latency Checker (MLC) [5] on our server is 121ns. The CXL memory is emulated using Quartz, whose latency is set to a reported value from a recent work [41]. Quartz monitors hardware event counters and emulates different memory latencies by injecting delays into the user application thread. Our evaluation uses 24 CPU cores to offer 48 vCPUs with hyperthreading. We provision 8GB per vCPU, which falls within a typical range of VM configurations (i.e., 4-11GB/vCPU [1]). Thus, we only utilize 384GB in total out of the 1TB DRAM via memory ballooning [55], and the standby power numbers are scaled down proportionally.

Modeling Rank-level Power-down. We apply the VM scheduling scheme in Figure 1 in Section 2. The workload running on each VM is randomly selected from CloudSuite benchmarks and we measure the actual DRAM power consumption for six hours using Intel's Performance Counter Monitor (PCM) [6]. The lifetime of a VM in the VM trace [15] is a multiple of 5 minutes. At every 5-minute interval we re-calculate the number of active ranks required based on the snapshot of VM memory usage. For the rest of the paper, the term *N-rank configuration* refers to a configuration in which N active ranks are utilized to provide enough memory capacity. We use the 8-rank configuration as a baseline by populating two 4-rank DIMMs per channel. To estimate DRAM power savings with rank-level power down, we profile the DRAM power consumption with a varying number of ranks per channel from two through eight. To model the 2-rank configuration, we utilize the rank sparing feature [56] to disable two ranks in the 4-rank DIMM. Using these numbers, we estimate the total DRAM power consumption over the 6-hour VM schedule by combining the power

Table 2: Normalized power consumption in the different DRAM power states.

DRAM Power State	Normalized Power
Standby	1.0
Self-refresh	0.2
MPSM	0.068

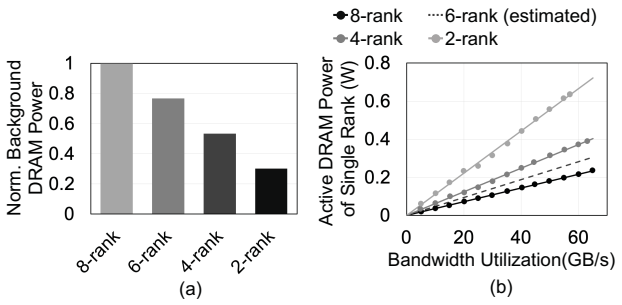

Figure 11: DRAM power consumption with different number of active ranks per channel: (a) normalized DRAM background power and (b) normalized DRAM active power to memory bandwidth utilization ratio for a single rank.

number with a different number of active ranks for every 5-minute interval.

DRAM Power Estimation. The total DRAM power is a sum of background power and active power. Table 2 tabulates the normalized background power in different power states. Figure 11(a) shows the background power (including refresh power) with a varying number of active ranks per channel. Our measurements on the server machine also demonstrate a near-linear scaling of the active power to the bandwidth utilization. The power consumption of the 6-rank configuration is estimated by interpolating 4-rank and 8-rank configurations. We also take into account the impact of data migration on power consumption. Additional active power consumption is reflected in the corresponding time interval, which is a short period at the deallocation of a VM until segment migration is completed. To estimate this interval, we measure the execution time of `memcpy()` with imposed bandwidth limitation by setting the thermal control register (i.e., THRT_PWR_DIMM_[2:0]) as segment migration opportunistically utilizes the unused bandwidth by foreground VMs.

Execution Time Estimation. The overall execution time is adjusted via post-processing. Section 3.3 demonstrates that CloudSuite reports a 1.4% performance loss on average by disabling rank interleaving. We apply this overhead to all rank configurations. The performance overhead of reducing the number of active ranks and DTL address translation (see Section 6.1) is also reflected. Note that data migration is performed in background by utilizing unused DRAM bandwidth and does not affect foreground VM performance.

5.2 Setup for Hotness-aware Self-refresh

Trace-driven Simulation Setup. We extend Linux pagemap for Intel Pintool [39] to collect physical memory address traces for the 8

Table 3: Simulator configurations.

Simulator configurations	
Memory	16GB, 32GB
Simulation length	20 billion instructions/workload
Host-side cache configurations	
L1-d	32KB, 8-way, LRU
L2	1MB, 8-way, LRU
LLC	8MB, 16-way, LRU
DTL cache configurations	
L1 segment mapping cache	64-entry, fully-associative cache
L2 segment mapping cache	1024-entry, 4-way set associative cache

Table 4: Memory accesses per kilo-instructions (MAPKI).

Workloads	MAPKI	Workloads	MAPKI
Data-analytics	1.9	Graph-analytics	6.5
Data-caching	1.5	In-memory-analytics	2.5
Data-serving	4.2	Media-streaming	4.6
Django-workload	0.8	Web-search	0.7
FB-oss-performance	3.6	Web-serving	0.7

CloudSuite benchmarks that run to completion on Pintool. We conservatively collect traces in memory bandwidth-intensive regions to make it more challenging to collect cold segments. We implement a custom trace-driven simulator to model the hotness-aware self-refresh mechanism. The simulator first generates post-cache traces via cache simulation whose parameters are summarized in Table 3. Then it randomly mixes the post-cache traces with allocated memory of 208GB, 224GB, 240GB (6-rank), and 304GB (8-rank). Finally, the simulator quantifies the benefits of the hotness-aware self-refresh mechanism as discussed in Section 3.4.

Table 4 presents the memory accesses per kilo-instructions (MAPKI) of Cloudsuite, which is in line with previous studies on cloud services and datacenter workloads that generally report low LLC MPKIs [12, 33, 37, 49]. However, some emerging datacenter workloads show relatively higher LLC MPKI values [51]. To account for this, we adjust the trace replay rate to achieve a memory bandwidth utilization of >30GB/s, which translates to MAPKI of 15.2. 30GB/s is 32% of the memory bandwidth in our environment, which is higher than the 95th percentile of memory bandwidth utilization in datacenters [29]. The profiling time threshold is 50ms by default. SRAM power is estimated from TSMC 40nm GP standard cell library. Other overhead, such as data migration delay and self-refresh exit penalty, is factored in manually.

6 EVALUATION

6.1 CXL Memory Access Latency

DTL employs a fully hardware-automated datapath for the DRAM access path. Thus, it increases the CXL memory access latency only marginally by 4.2ns on average. The average memory access time (AMAT) is estimated with the following equations (SMC stands for Segment Mapping Cache):

$$AMAT_{CXL} = CXL\_mem_{lat} + Addr\ translation \quad (1)$$

Where:

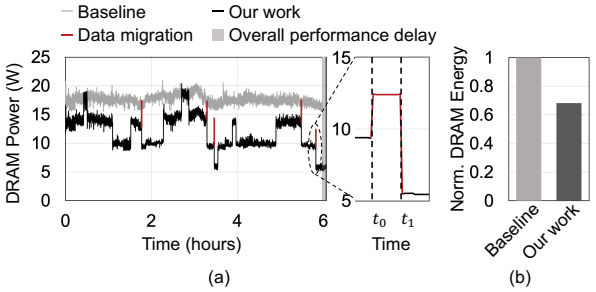

Figure 12: DRAM power and energy consumption of CloudSuite: (a) runtime DRAM power consumption and (b) normalized DRAM energy consumption.

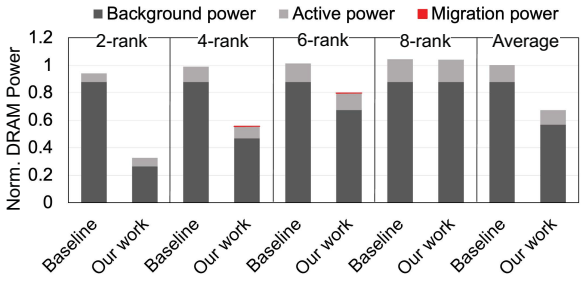

Figure 13: DRAM power breakdown.

$$\begin{aligned} &Addr\ translation = \\ &\quad L1\_SMC\ hit\ time + L1\_SMC\ miss\ ratio \\ &\quad \times (L2\_SMC\ hit\ time + L2\_SMC\ miss\ ratio \\ &\quad \times L2\_SMC\ miss\ penalty) \end{aligned} \tag{2}$$

L1 and L2 SMC hit latencies are 1 and 7 cycles, respectively, at 1.5GHz clock rate. According to our simulation of SMC, L1 and L2 miss ratios are 14.7% and 15.4%, respectively. An SMC miss costs two SRAM accesses (to host base address table and AU base address table) taking one cycle per each, and one DRAM access to the segment mapping table. We obtain SRAM access latencies using CACTI-P [36] (L1 SMC), Intel CPU's L2 TLB latency (L2 SMC), and synthesis results from TSMC 40nm standard cell library (other SRAM structures). Plugging in these numbers yields the AMAT of 214.2ns (i.e., only 4.2ns increase from the vanilla CXL memory on average with max/min increases of 123.7ns and 0.67ns, respectively). This translation overhead increases the execution time of CloudSuite benchmarks only by 0.18%.

6.2 Evaluation of Rank-level Power-down Scheme

Segment Migration Overhead. Figure 12(a) shows the DRAM power consumption of CloudSuite workloads over a 6-hour schedule of VMs. The rank-level power-down scheme reduces the DRAM power consumption significantly. The red line depicts DRAM power consumption during segment migration triggered by a VM deallocation. We further zoom in the window of the last occurrence of data migration. When the deallocation of a VM is finished at time *t0*, DTL determines that it can reduce the number of active ranks from four to two per channel by consolidating unallocated segments. The segments in the least utilized rank start migration at time *t0*, which is finished by time *t1*. Then DTL puts the victim rank into maximum power saving mode to save DRAM power. In this specific case, the migrated segments amount to 24GB in size, and the migration time is 1.3 seconds, much shorter than the interval of VM configuration changes.

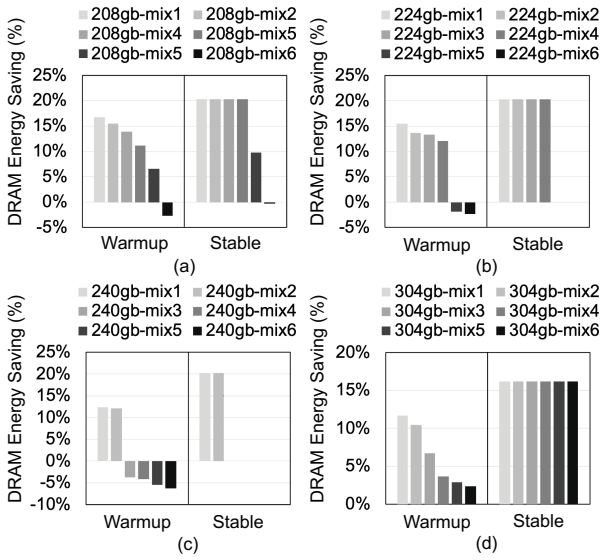

Figure 14: Additional DRAM energy savings from hotness-aware self-refresh with warmup-time of up to 60 seconds.

DRAM Energy Savings. This technique effectively reduces DRAM energy consumption with only marginal performance degradation. The overall DRAM energy consumption decreases by 31.6% compared to the baseline system as shown in Figure 12(b). The overall execution time increases only by 1.6% for using fewer active ranks and disabling rank-interleaving. Figure 13 presents the DRAM power breakdown for the baseline and the rank-level power-down scheme. Compared to the baseline that uses all 8 ranks per channel, the rank-level power-down scheme uses the smallest number of ranks required, thus greatly reducing DRAM background power. In contrast, the DRAM active power does not vary widely due to the foreground activities of live VMs. Since data migration is completed in a very short period, the additional DRAM energy consumption is negligible. With two active ranks per channel, DRAM power consumption is reduced by an average of 65.2% over the baseline. The overall DRAM background power and total DRAM power consumption are reduced by 35.3% and 32.7% on average, respectively.

6.3 Evaluation of Hotness-aware Self-refresh Scheme

DRAM Energy Savings. Figure 14 shows the additional DRAM energy savings from hotness-aware self-refresh *after* rank-level power-down is applied. In the warmup phase, DTL takes an iterative process of entering and exiting from the self-refresh mode to separate hot and cold segments. This phase takes about 10-60 seconds depending on the workload.

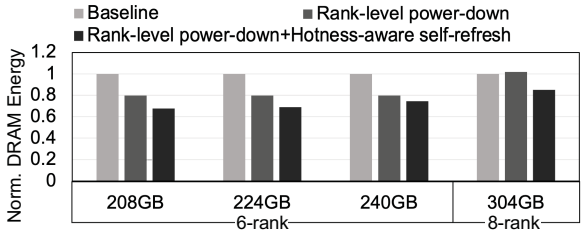

Figure 15: Total energy savings from two power-saving schemes for 6/8 ranks.

Figure 14(a) depicts that four of the six traces with 208GB allocated memory save 20.3% of DRAM energy in the stable phase by staying in the self-refresh mode with the victim rank of each channel. Due to an insufficient number of cold segments, frequent access to the victim ranks in two channels (208gb-mix5) and all victim ranks (208gb-mix6) results in ping-ponging between self-refresh and standby mode, which decreases the overall energy savings. Figure 14(b) and (c) show that several workloads do not enter the self-refresh mode in the stable phase (with missing bars), especially with the 240GB allocated memory of which each channel's unallocated memory in the active ranks is less than 50% of the capacity of a pair of ranks. As the amount of unallocated memory capacity becomes smaller, it is difficult to collect enough cold segments quickly. Too many accesses to victim ranks in the profiling stage keep resetting the timer to prolong the profiling stage, which makes it difficult to re-enter the self-refresh mode. Figure 14(d) shows that additional allocated memory makes collecting enough cold segments easier to enter self-refresh mode. However, due to the higher background DRAM power consumption in the 8-rank configuration, the energy savings are lower than the traces with 208GB allocated memory.

A prior study [41] demonstrates that the data access patterns in the datacenter remain relatively stable for a long period (minutes to hours). This implies opportunities for staying in the stable phase for a long time after the warmup. The portion of the energy consumption of SRAM access for migration table management is less than 0.1% of DRAM energy, and the total self-refresh exit latency is less than 0.001% in the warmup time, which is negligible.

6.4 Put It All Together: Total DRAM Energy Savings

Figure 15 summarizes the total energy savings when the two proposed mechanisms are applied together. One rank-group can be powered down through the first mechanism, saving 20.2% of DRAM energy consumption. When unallocated memory in the active ranks of each channel is 50% of the capacity of a pair of ranks or greater, the second mechanism is effective as we observe in Figure 14. By further applying the second mechanism to such cases, 25.6%-32.3% of the total DRAM energy savings are achieved over the baseline. Since DRAM capacity demands entail the usage of all 8 ranks to disallow entrance into power-down mode, the first mechanism cannot be applied for the 8-rank configuration that requires all ranks to be active. The results show that the second method can save 14.9% of DRAM energy consumption with the allocated memory of 304GB.

Table 5: Size of the data structures to support 16 hosts.

Size of the CXL memory	384GB	4TB
Size of the remapping cache		
L1 segment mapping cache	328B (64 entries)	752B (128 entries)
L2 segment mapping cache (1024 entries)	5.1KB	5.9KB
Size of the SRAM structures		
Host base addr table	138B	
AU base addr table	24.4KB	260KB
Hot-cold migration table	432KB	5MB
Size of the DRAM structures		
Segment mapping table	456KB	5.5MB
Reverse mapping table	552KB	6.5MB
Free segment queues	432KB	5.3MB
Allocated segment queues	432KB	5.3MB
Free AU queue	192B	2.8KB

Table 6: Power and area breakdown.

	Power (mW) (384GB/4TB)	Area (mm^2) (384GB/4TB)
Segment mapping cache	1.7 / 2.1	0.0035 / 0.0037
SRAM structures	2.9 / 13.0	0.1 / 1.1
Microprocessor	21.2	0.0515
Total	25.7 / 36.2	0.165 / 1.1

6.5 Power and Area Estimation of CXL Controller

We estimate the area and power overhead of the proposed CXL controller, composed of a quad-core ARM Cortex-R5 processor, cache and SRAM. Total power dissipation of the controller is 0.8W at 1.5GHz and the total area is $5.4mm^2$. For power and area estimation of the controller, we synthesized the RTL of the ARM IP with the TSMC 40nm GP standard cell library at 625MHz frequency. The power and area estimation can be normalized to 25.7mW and $0.165mm^2$ at 7nm technology, assuming both are proportional to the $(Technology)^2$ following the methodology of Biswas and Chandrakasan [11]. CACTI-P is used for modeling power and area of the segment mapping caches. CPU power is estimated with the switching factor of 0.1 and the power of memory cells are calculated assuming that CXL bandwidth is fully utilized. Linear frequency scaling is applied to obtain the power at 1.5GHz frequency.

6.6 Scaling CXL Memory Device

As the capacity of DRAM devices is expected to scale in the future, we quantify the overhead of SRAM/DRAM structures for a hypothetical 4TB CXL device. Table 5 compares the data structure overhead for 384GB and 4TB CXL memory. We assume the 4TB device has 8 channels with two 8-rank 256GB DIMMs per channel. The SRAM and DRAM structures scale mostly linearly with the capacity. The total size of the on-chip SRAM structure increases from 0.5MB to 5.3MB. The total size of the DRAM structure changed from 1.9MB to 22.6MB. Still, it takes only 0.0005% of the 4TB DRAM size. Table 6 shows the detailed power and area breakdown at 7nm technology [11]. Total power consumption and area estimation of the CXL controller are 36.2mW and $1.1mm^2$, respectively. Thus, we

believe that DTL can be practically deployed for terabyte-scale CXL memory devices as well. A similar degree of DRAM power savings is expected assuming the DRAM capacity utilization remains similar. Since a higher-capacity DRAM device often has more DRAM channels and ranks, the performance loss would become smaller.

7 DISCUSSION

Limitations of OS/Hypervisor-based Approaches. The key innovation of DTL is realizing both rank-level power management and hot-cold consolidation *without* changing the host-side stack, which gives a huge benefit for deployment. The host OS/hypervisor is not an ideal layer to implement these techniques in disaggregated environments. The rank-level power-down technique requires the knowledge of (i) memory allocation of *all* VMs running concurrently across multiple OS/hypervisor images and (ii) DRAM device address mappings. Disaggregated memory is highly virtualized and opaque to the host OS/hypervisor for good reasons (e.g., flexible VM management/migration, minimizing security risks). In this environment, it is extremely challenging for the host OS/hypervisor to realize such techniques. The hotness-aware self-refresh scheme requires continuous monitoring of memory accesses and fast entrance into/exit from self-refresh mode, which cannot be efficiently handled by OS.

Rank-level Parallelism Under Heavy Memory Load. Performance gains from rank-level parallelism in a terabyte-scale disaggregated memory environment are likely marginal due to diminishing returns from additional ranks. Since the rank-bits are commonly located in high-order bits [25], we first leverage bank-/channel-level parallelism, beyond which additional memory access parallelism is left small [16].

8 RELATED WORK

DRAM Power Management. There are proposals to reduce DRAM power consumption for unallocated/cold pages [26–28, 30, 34, 59]. ESKIMO [27], RTC [28] and GreenDIMM [34] identify the unused data and stop refresh operations for them to save background power. ESKIMO turns off the refresh operation of unallocated pages by tracking page allocation/de-allocation. RTC expands the coverage of power-down with partial-array auto refresh (PAAR) and reduces refresh operations even within allocated pages at a row granularity. GreenDIMM proposes a deep power-down state (DRAM power-down state without refresh operations) at a sub-array granularity with additional control circuits. However, these proposals require intrusive hardware changes to realize the functionalities. They also need to modify OS or MC to support them. Other proposals [26, 30, 59] classify hot/cold data to cluster cold data into a rank and put it into self-refresh mode. RAMZzz [59] separates hot and cold ranks via page migration to make the cold rank enter self-refresh mode. Huang et al. [26] classify the data based on the access count and dynamically migrate data to secure the idle time of a cold rank. To gather the access count of each page, they suggest a hardware/software cooperative approach. DimmStore [30] introduces rank-aware data allocation and access rate-based data placement. DimmStore also needs support from the OS to separate and manage the system region and the data region at a rank level. However, all of the aforementioned proposals require modifications to OS, MC, or even DRAM devices, to varying degrees. Unlike them, DTL suggests a software/MC-transparent architecture to make it easier to deploy.

Memory Address Remapping. Hardware-based address translation has been investigated in various contexts for its benefits such as software transparency and low latency [13, 18, 45, 50, 53, 58]. MXT [53] employs a compression translation table (CTT) for real-to-physical address translation, which is further aided by a shared cache to mitigate the latency associated with accessing the compressed memory. In the context of heterogeneous memory systems, Wen et al. [58] utilizes heterogeneous memory management unit (HMMU) to migrate frequently accessed pages into fast memory. Dong et al. [18] and Ramos et al. [45] propose to maintain page remapping information in MC. As the memory capacity scales, so does the translation table size, making them less practical. Unlike these works targeting tightly coupled, non-virtualized DRAM devices, DTL leverages latency-tolerant, asynchronous CXL interface along with unique characteristics of cloud VMs to feature coarser granularity in segment migration and different metadata structures (closer to FTL). CAMEO [13] and PoM [50] maintain a remapping table in fast memory (i.e., on-die memory). CAMEO introduced LLT (Line Location Table) and LLP (Line Location Predictor) and proposed a fine-grained management method at a cache-line granularity. PoM proposed a segment-based management method based on Segment Remapping Cache (SRC) and Segment Remapping Table (SRT). Both proposals have a limited degree of freedom in migrating pages to either a congruence group or a segment. In contrast, DTL has much greater flexibility to consolidate unallocated/cold pages effectively.

9 CONCLUSION

We propose DRAM Translation Layer (DTL) as a new low-cost address translation layer from host physical address to DRAM device physical address, placed within the CXL controller. Leveraging DTL, we introduce two software- and host MC-transparent power-saving techniques: rank-level power-down and hotness-aware self-refresh. Our evaluation with CloudSuite benchmarks demonstrates that the rank-level power-down scheme reduces DRAM energy consumption by 31.6% at a small cost of performance degradation (1.6%) by consolidating unallocated segments into a subset of ranks and putting them into maximum power-saving mode. For the remaining active ranks that require data retention, we apply hotness-aware self-refresh to further reduce the DRAM energy by up to 14.9% with negligible performance loss. DTL opens up interesting research directions by providing means for flexible memory management to improve reliability, availability, as well as security, to name a few.

ACKNOWLEDGMENTS

This work was supported by a research grant from Samsung Electronics (Memory Business) and by an Institute of Information & Communications Technology Planning & Evaluation (IITP) grant (2021-0-00853, Developing Software Platform for PIM Programming) funded by the Korea Government (MSIT). The authors thank Kiseok Oh and Ju Yun Jung for their help with the experimental setup. Jae W. Lee is the corresponding author.

REFERENCES

[1] Azure VM Comparison. https://azureprice.net.
[2] Cloudsuite. https://github.com/parsa-epfl/cloudsuite.
[3] Compute Express Link (CXL). https://www.computeexpresslink.org.
[4] Intel Max Series Brings Breakthrough Memory Bandwidth and Performance to HPC and AI. https://www.intel.com/content/www/us/en/newsroom/news/introducing-intel-max-series-product-family.html.
[5] Intel Memory Latency Checker v3.9a. https://www.intel.com/content/www/us/en/developer/articles/tool/intelr-memory-latency-checker.html.
[6] Intel Performance Counter Monitor. https://github.com/intel/pcm.
[7] Amazon. Amazon Web Services (AWS). https://aws.amazon.com.
[8] AMD. Offering Unmatched Performance, Leadership Energy Efficiency and Next-Generation Architecture, AMD Brings 4th Gen AMD EPYC™ Processors to The Modern Data Center. https://www.amd.com/en/press-releases/2022-11-10-offering-unmatched-performance-leadership-energy-efficiency-and-next.
[9] Raja Appuswamy, Matthaios Olma, and Anastasia Ailamaki. 2015. Scaling the Memory Power Wall with DRAM-aware Data Management. In *Proceedings of the 11th International Workshop on Data Management on New Hardware*. 1–9.
[10] Luis Angel D Bathen, Mark Gottscho, Nikil Dutt, Alex Nicolau, and Puneet Gupta. 2012. ViPZonE: OS-level Memory Variability-driven Physical Address Zoning for Energy Savings. In *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. 33–42.
[11] Avishek Biswas and Anantha P Chandrakasan. 2018. CONV-SRAM: An Energy-efficient SRAM with In-memory Dot-product Computation for Low-power Convolutional Neural Networks. *IEEE Journal of Solid-State Circuits* 54, 1, 217–230.
[12] Shuang Chen, Christina Delimitrou, and José F Martínez. 2019. Parties: Qos-aware Resource Partitioning for Multiple Interactive Services. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. 107–120.
[13] Chia Chen 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 *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 1–12.
[14] FJ Corbato. Festschrift: In Honor of PM Morse, chapter A Paging Experiment with the Multics System.
[15] Eli Cortez, Anand Bonde, Alexandre Muzio, Mark Russinovich, Marcus Fontoura, and Ricardo Bianchini. 2017. Resource Central: Understanding and Predicting Workloads for Improved Resource Management in Large Cloud Platforms. In *Proceedings of the 26th Symposium on Operating Systems Principles*. 153–167.
[16] Frank Denneman. Memory Deep Dive: Optimizing for Performance. https://frankdenneman.nl/2015/02/20/memory-deep-dive/.
[17] Bruno Diniz, Dorgival Guedes, Wagner Meira Jr, and Ricardo Bianchini. 2007. Limiting the Power Consumption of Main Memory. In *Proceedings of the 34th annual international symposium on Computer architecture*. 290–301.
[18] Xiangyu Dong, Yuan Xie, Naveen Muralimanohar, and Norman P Jouppi. 2010. Simple But Effective Heterogeneous Main memory with On-chip Memory Controller Support. In *SC'10: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–11.
[19] Subramanya R Dulloor, Amitabha Roy, Zheguang Zhao, Narayanan Sundaram, Nadathur Satish, Rajesh Sankaran, Jeff Jackson, and Karsten Schwan. 2016. Data Tiering in Heterogeneous Memory Systems. In *Proceedings of the Eleventh European Conference on Computer Systems*. 1–16.
[20] Assaf Eisenman, Darryl Gardner, Islam AbdelRahman, Jens Axboe, Siying Dong, Kim Hazelwood, Chris Petersen, Asaf Cidon, and Sachin Katti. 2018. Reducing DRAM Footprint with NVM in Facebook. In *Proceedings of the Thirteenth EuroSys Conference*. 1–13.
[21] Xiaobo Fan, Carla Ellis, and Alvin Lebeck. 2001. Memory Controller Policies for DRAM Power Management. In *Proceedings of the 2001 international symposium on Low power electronics and design*. 129–134.
[22] Saugata Ghose, Abdullah Giray Yaglikçi, Raghav Gupta, Donghyuk Lee, Kais Kudrolli, William X Liu, Hasan Hassan, Kevin K Chang, Niladrish Chatterjee, Aditya Agrawal, et al. 2018. What Your DRAM Power Models are not Telling You: Lessons from a Detailed Experimental Study. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 2, 3 (2018), 1–41.
[23] Google. Google Cloud Platform. https://cloud.google.com.
[24] Huang Hai, P Padmanabhan, and GS Kang. 2003. Design and Implementation of Power-aware Virtual Memory. In *Proceedings of the USENIX Annual Technical Conference*.
[25] Christian Helm, Soramichi Akiyama, and Kenjiro Taura. 2020. Reliable Reverse Engineering of Intel DRAM Addressing Using Performance Counters. In *2020 28th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, 1–8.
[26] Hai Huang, Kang G Shin, Charles Lefurgy, and Tom Keller. 2005. Improving Energy Efficiency by Making DRAM Less Randomly Accessed. In *Proceedings of the 2005 International Symposium on Low Power Electronics and Design*. 393–398.
[27] Ciji Isen and Lizy John. 2009. ESKIMO-Energy Savings using Semantic Knowledge of Inconsequential Memory Occupancy for DRAM Subsystem. In *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 337–346.
[28] Syed MAH Jafri, Hasan Hassan, Ahmed Hemani, and Onur Mutlu. 2020. Refresh Triggered Computation: Improving the Energy Efficiency of Convolutional Neural Network Accelerators. *ACM Transactions on Architecture and Code Optimization (TACO)* 18, 1 (2020), 1–29.
[29] Svilen Kanev, Juan Pablo Darago, Kim Hazelwood, Parthasarathy Ranganathan, Tipp Moseley, Gu-Yeon Wei, and David Brooks. 2015. Profiling a Warehouse-scale Computer. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture*. 158–169.
[30] Alexey Karyakin and Kenneth Salem. 2019. DimmStore: Memory Power Optimization for Database Systems. *Proceedings of the VLDB Endowment* 12, 11 (2019), 1499–1512.
[31] Hiwot Tadese Kassa, Jason Akers, Mrinmoy Ghosh, Zhichao Cao, Vaibhav Gogte, and Ronald Dreslinski. 2021. Improving Performance of Flash Based {Key-Value} Stores Using Storage Class Memory as a Volatile Memory Extension. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. 821–837.
[32] Taek Woon Kim. DDR5 Performance-Boost and Power-Saving Features, and IDDs. https://www.jedec.org/sites/default/files/TaekWoon_Kim_r5.pdf.
[33] Hyun Ryong Lee and Daniel Sanchez. 2022. Datamime: Generating Representative Benchmarks by Automatically Synthesizing Datasets. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 1144–1159.
[34] Seunghak Lee, Ki-Dong Kang, Hwanjun Lee, Hyungwon Park, Younghoon Son, Nam Sung Kim, and Daehoon Kim. 2021. GreenDIMM: OS-Assisted DRAM Power Management for DRAM with a Sub-Array Granularity Power-Down State. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. 131–142.
[35] Seok-Hee Lee. 2016. Technology Scaling Challenges and Opportunities of Memory Devices. In *2016 IEEE International Electron Devices Meeting (IEDM)*. 1–1.
[36] Sheng Li, Ke Chen, Jung Ho Ahn, Jay B. Brockman, and Norman P. Jouppi. 2011. CACTI-P: Architecture-level Modeling for SRAM-based Structures with Advanced Leakage Reduction Techniques. In *ICCAD: International Conference on Computer-Aided Design*. 694–701.
[37] Qixiao Liu and Zhibin Yu. 2018. The Elasticity and Plasticity in Semi-containerized Co-locating Cloud Workload: A View from Alibaba Trace. In *Proceedings of the ACM Symposium on Cloud Computing*. 347–360.
[38] Chengzhi Lu, Kejiang Ye, Guoyao Xu, Cheng-Zhong Xu, and Tongxin Bai. 2017. Imbalance in the Cloud: An Analysis on Alibaba Cluster Trace. In *2017 IEEE International Conference on Big Data (Big Data)*. IEEE, 2884–2892.
[39] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood. 2005. Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation. *Acm sigplan notices* 40, 6 (2005), 190–200.
[40] Chris A Mack. 2011. Fifty Years of Moore's Law. *IEEE Transactions on semiconductor manufacturing* 24, 2 (2011), 202–207.
[41] Hasan Al Maruf, Hao Wang, Abhishek Dhanotia, Johannes Weiner, Niket Agarwal, Pallab Bhattacharya, Chris Petersen, Mosharaf Chowdhury, Shobhit Kanaujia, and Prakash Chauhan. 2022. TPP: Transparent Page Placement for CXL-Enabled Tiered Memory. *arXiv preprint arXiv:2206.02878* (2022).
[42] Microsoft. Microsoft Azure. https://azure.microsoft.com.
[43] John Ousterhout, Parag Agrawal, David Erickson, Christos Kozyrakis, Jacob Leverich, David Mazières, Subhasish Mitra, Aravind Narayanan, Guru Parulkar, Mendel Rosenblum, et al. 2010. The Case for RAMClouds: Scalable High-performance Storage Entirely in DRAM. *ACM SIGOPS Operating Systems Review* 43, 4 (2010), 92–105.
[44] Andy Patrizio. Facebook and Amazon are Causing a Memory Shortage. https://www.networkworld.com/article/3247775/facebook-and-amazon-are-causing-a-memory-shortage.html.
[45] Luiz E Ramos, Eugene Gorbatov, and Ricardo Bianchini. 2011. Page Placement in Hybrid Memory Systems. In *Proceedings of the international conference on Supercomputing*. 85–95.
[46] Charles Reiss, Alexey Tumanov, Gregory R Ganger, Randy H Katz, and Michael A Kozuch. 2012. Towards Understanding Heterogeneous Clouds at Scale: Google Trace Analysis. *Intel Science and Technology Center for Cloud Computing, Tech. Rep* 84 (2012), 1–21.
[47] Samsung. 288pin Registered DIMM based on 16Gb M-die. https://image.semiconductor.samsung.com/resources/data-sheet/128GB_DDR4_16Gb_M_Die_Registered_DIMM_Rev1.0_Feb.19.pdf.
[48] Samsung. DDR4 SDRAM Specification. https://semiconductor.samsung.com/resources/data-sheet/DDR4_Device_Operations_Rev11_Oct_14-0.pdf.
[49] Rathijit Sen and Karthik Ramachandra. 2018. Characterizing Resource Sensitivity of Database Workloads. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 657–669.
[50] Jaewoong Sim, Alaa R Alameldeen, Zeshan Chishti, Chris Wilkerson, and Hyesoon Kim. 2014. Transparent Hardware Management of Stacked DRAM as Part

of Memory. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 13–24.

[51] Akshitha Sriraman, Abhishek Dhanotia, and Thomas F Wenisch. 2019. SoftSKU: Optimizing Server Architectures for Microservice Diversity @Scale. In *Proceedings of the 46th International Symposium on Computer Architecture*. 513–526.

[52] Muhammad Tirmazi, Adam Barker, Nan Deng, Md E Haque, Zhijing Gene Qin, Steven Hand, Mor Harchol-Balter, and John Wilkes. 2020. Borg: the Next Generation. In *Proceedings of the fifteenth European conference on computer systems*. 1–14.

[53] R Brett Tremaine, Peter A Franaszek, John T Robinson, Charles O Schulz, T Basil Smith, Michael E Wazlowski, and P Maurice Bland. 2001. IBM Memory Expansion Technology (MXT). *IBM Journal of Research and Development* 45, 2 (2001), 271–285.

[54] Haris Volos, Guilherme Magalhaes, Ludmila Cherkasova, and Jun Li. 2015. Quartz: A Lightweight Performance Emulator for Persistent Memory Software. In *Proceedings of the 16th Annual Middleware Conference*. 37–49.

[55] Carl A Waldspurger. 2002. Memory Resource Management in VMware ESX Server. *ACM SIGOPS Operating Systems Review* 36, SI (2002), 181–194.

[56] Lidia Warnes, Michael Bozich Calhoun, Dennis Carr, Teddy Lee, Dan Vu, and Ricardo Ernesto Espinoza-Ibarra. Rank Sparing System and Method. US Patent 8,892,942.

[57] Johannes Weiner, Niket Agarwal, Dan Schatzberg, Leon Yang, Hao Wang, Blaise Sanouillet, Bikash Sharma, Tejun Heo, Mayank Jain, Chunqiang Tang, and Dimitrios Skarlatos. 2022. TMO: Transparent Memory Offloading in Datacenters. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 609–621.

[58] Fei Wen, Mian Qin, Paul V Gratz, and AL Narasimha Reddy. 2020. Hardware Memory Management for Future Mobile Hybrid Memory Systems. *IEEE Transactions on computer-aided design of integrated circuits and systems* 39, 11 (2020), 3627–3637.

[59] Donghong Wu, Bingsheng He, Xueyan Tang, Jianliang Xu, and Minyi Guo. 2012. RAMZzz: Rank-aware DRAM Power Management with Dynamic Migrations and Demotions. In *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–11.

[60] Dongli Zhang, Moussa Ehsan, Michael Ferdman, and Radu Sion. 2014. DIMMer: A Case for Turning off DIMMs in Clouds. In *Proceedings of the ACM Symposium on Cloud Computing*. 1–8.