GreenDIMM: OS-assisted DRAM Power Management for DRAM with a Sub-array Granularity Power-Down State

Seunghak Lee, Ki-Dong Kang, Hwanjun Lee, Hyungwon Park, Younghoon Son[†], Nam Sung Kim[‡], Daehoon Kim
DGIST, [†]Samsung Electronics, [‡]University of Illinois at Urbana-Champaign
{slee, kd_kang, lee.hwanjun, hwpark, dkim}@dgist.ac.kr, yh77.son@samsung.com, nskim@illinois.edu

ABSTRACT

Power and energy consumed by DRAM comprising main memory of data-center servers have increased substantially as the capacity and bandwidth of memory increase. Especially, the fraction of DRAM background power in DRAM total power is already high, and it will continue to increase with the decelerating DRAM technology scaling as we will have to plug more DRAM modules in servers or stack more DRAM dies in a DRAM package to provide necessary DRAM capacity in the future. To reduce the background power, we may exploit low average utilization of the DRAM capacity in data-center servers (i.e., 40–60%) for DRAM power management. Nonetheless, the current DRAM power management supports low-power states only at the rank granularity, which becomes ineffective with memory interleaving techniques devised to disperse memory requests across ranks. That is, ranks need to be frequently woken up from low-power states with aggressive power management, which can significantly degrade system performance, or they do not get a chance to enter low-power states with conservative power management.

To tackle such limitations of the current DRAM power management, we propose `GreenDIMM`, OS-assisted DRAM power management. Specifically, `GreenDIMM` first takes a memory block in physical address space mapped to a group of DRAM sub-arrays across every channel, rank, and bank as a unit of DRAM power management. This facilitates fine-grained DRAM power management while keeping the benefit of memory interleaving techniques. Second, `GreenDIMM` exploits memory on-/off-lining operations of the modern OS to dynamically remove/add memory blocks from/to the physical address space, depending on the utilization of memory capacity at run-time. Third, `GreenDIMM` implements a deep power-down state at the sub-array granularity to reduce the background power of the off-lined memory blocks. As the off-lined memory blocks are removed from the physical address space, the sub-arrays will not receive any memory request and stay in the power-down state until the memory blocks are explicitly on-lined by the OS. Our evaluation with a commercial server running diverse workloads shows that `GreenDIMM` can reduce DRAM and system power by 36% and 20%, respectively, with ~1% performance degradation.

MICRO '21, October 18–22, 2021, Virtual Event, Greece

ACM ISBN 978-1-4503-8557-2/21/10…$15.00
https://doi.org/10.1145/3466752.3480089

CCS CONCEPTS

• **Hardware** → *Enterprise level and data centers power issues*; *Dynamic memory*; • **Software and its engineering** → *Memory management*.

KEYWORDS

DRAM power management, memory off-lining

ACM Reference Format:
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 *Proceedings of The 54th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'21)).* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3466752.3480089

1 INTRODUCTION

In data-center servers, power and energy consumed by main memory, which often consists of many DRAM modules, have grown considerably as the capacity and bandwidth of DRAM has increased [33]. The past studies show that the fraction of main memory power in total data-center server power is ~40% of the total energy in data-center servers and report that it will steadily increase as applications demand larger main memory capacity with more cores per server and memory-based caching technology [20, 28, 32]. Especially, as the capacity of main memory increases, the DRAM background power contributes to a large fraction of the total DRAM power. The background power includes refresh power to retain memory states and static power of DRAM peripheral and I/O components. Especially, the refresh power accounts for more than 35% of the total DRAM power [24] while the average utilization of main memory capacity is 40%–60% in data-center servers [25, 30].

To reduce the background power, current DRAM architectures such as DDR4 support low-power states: power-down and self-refresh that turns off some of the DRAM components in a rank and DRAM does refresh itself with minimum power consumption, respectively [33]. When no memory request is sent to a rank for a certain amount of time, the memory controller can make the rank enter a low-power state. However, such low-power states at the rank granularity become ineffective as memory interleaving techniques are often adopted to improve memory-level parallelism (MLP). The memory interleaving techniques disperse data in the contiguous physical address space across different memory channels, ranks, and banks [22]. Consequently, memory requests of applications even with small memory footprint are sent to multiple ranks, preventing the ranks from entering a low-power state; we observe that an application with 64MB of memory footprint

(i.e., SPECCPU2006 `libquantum` [14]) completely eliminates an opportunity to enter a low-power state. Although some prior work proposes DRAM power management at the sub-rank granularity, it does not consider memory interleaving or considerably increases hardware complexity [16, 21]. Furthermore, a rank in such a low-power state needs to be woken up with an extra latency (e.g., 18 *ns* and 768 *ns* for power-down and self-refresh, respectively [7]) whenever it receives a memory request from a memory controller. Consequently, aggressive power management, which makes ranks frequently enter/exit a low-power state, inevitably hurts the system performance.

To tackle the aforementioned limitations, we propose `GreenDIMM`, OS-assisted power management. Specifically, `GreenDIMM` first proposes to take a memory block in contiguous physical address space mapped to a group of sub-arrays across every channel, rank, and bank as a unit of DRAM power management. This facilitates fine-grained DRAM power management (e.g., 1GB granularity in 64GB main memory or 1.5625% of the total main memory capacity) while preserving the performance benefit of the channel, rank, and bank interleaving techniques. Second, `GreenDIMM` proposes to exploit memory on-/off-lining operations of the modern OS[1] and dynamically remove/add memory blocks from/to the physical address space, depending on the utilization of memory capacity at run-time. Third, `GreenDIMM` proposes to implement a deep power-down state at the sub-array granularity to reduce the background power of the off-lined memory blocks, building on the same circuit components and controlling mechanisms as bank-granularity partial array self-refresh (PASR) supported by LPDDR and DDR DRAM [10]. The deep power-down state can practically eliminate the refresh and static power of sub-arrays associated with off-lined memory blocks by stopping refresh and power-gating peripheral and I/O components of the sub-arrays. After `GreenDIMM` off-lines the memory blocks and makes the corresponding sub-arrays enter the deep power-down state, the OS or applications do not send any memory request to the sub-arrays. That is, `GreenDIMM` does not need to wake up the sub-arrays for sudden memory requests to the off-lined memory blocks in the power-down state and pay a penalty for waking up the sub-arrays from the power-down state until it explicitly on-lines the memory blocks. Lastly, `GreenDIMM` proposes to increase the fraction of memory capacity in the deep power-down state while decreasing the performance overhead of on-/off-lining memory blocks at run-time with optimizations. To the best of our knowledge, `GreenDIMM` is the first effective DRAM power management technique exploiting the memory on-/off-lining operations supported by the modern OS.

We evaluate `GreenDIMM` with a commercial server running Microsoft Azure VM [6], SPECCPU2006 [14], SPECCPU2017 [4], HiBench [15], and cloudsuite [12]. The summary of our evaluations is as follows. `GreenDIMM` can reduce DRAM energy consumption by 32%–36% (i.e., system energy consumption by 9%–20%) for DRAM capacity of 256GB to 1TB. `GreenDIMM` with the proposed optimization can reduce DRAM energy consumption by 55% (i.e., the energy consumption of the server by 30%) with more sub-arrays in the deep power-down state for 1TB capacity at the cost of ~1% performance degradation.

The rest of this paper is organized as follows. Section 2 and Section 3 explains the background and motivation of `GreenDIMM`, respectively. Section 4 describes the main idea and overall architecture of `GreenDIMM`. Section 5 discusses implementation issues of `GreenDIMM`. Section 6 shows experimental results of `GreenDIMM` with a real machine setup. Section 7 describes the related work. Section 8 concludes this paper.

2 BACKGROUND

2.1 DRAM Architecture

A DRAM-based main memory is organized in a hierarchical manner, consisting of channels, Dual In-line Memory Modules (DIMMs), ranks, and banks. Each DRAM device provides 4-, 8-, or 16-bit I/O (i.e., ×4, ×8, and ×16 DRAM devices), and thus a rank comprises 16, 8, or 4 DRAM devices to provide 64-bit I/O. To provide a large capacity for servers, we typically use DIMMs with ×4 or ×8 DRAM devices, each with 4Gb or 8Gb. Then, suppose a rank consisting of eight ×8 DRAM devices. From a memory controller perspective, a (logical) bank in the rank in fact consists of 8 (physical) banks from 8 DRAM devices accessed with the same bank, row, and column addresses in a lock-step manner. A physical bank consists of multiple sub-arrays [29, 34], each consists of multiple MATs. For example, a bank of DDR4 ×8 4Gb DRAM devices has 64 sub-arrays and a sub-array has 16 MATs, each of which typically comprises 512 rows and 512 columns.

2.2 DRAM Power Management with Low-Power States

For DRAM power management, the memory controller can dynamically set DRAM power states [11]. The current DRAM, such as DDR4, supports multiple power states such as power-down and self-refresh that can reduce the background power considerably at the cost of performance. For example, the power-down state disables clock and turns off I/O circuits, consuming only 40%–70% of the power consumed by active state. The self-refresh state additionally turns off power-hungry delay-locked loop (DLL), consuming down to 10% of the power consumed by active state. However, such low-power states require a significant amount of wake-up time (e.g., 18 *ns* and 768 *ns* for power-down and self-refresh, respectively [7]) to turn on the DRAM components and then serve memory requests, considerably increasing the latency of memory accesses.

2.3 Memory On/Off-lining

The memory on/off-lining is a part of the memory hot-plug technique that allows administrators to install/remove DIMMs physically for the purpose of upgrade/replacement without system downtime [17, 23]. For the memory off-lining, the kernel changes the state of a region of the physical address space to the off-line state by updating the `sysfs` file. Before the OS off-lines the region, it first clears the page table entries by removing memory pages associated with the region from the list of available memory, the `mem_map` list, and the buddy lists. Subsequently, the OS migrates the pages in the regions to other on-lined regions. Since the OS does not allocate pages in the off-lined regions, the OS-level memory off-lining provides an opportunity to reduce power and energy consumption

[1] They are originally developed to support memory hot-plug that allows a (failing) DRAM module to be unplugged from a server and replaced with a another DRAM module at run-time.

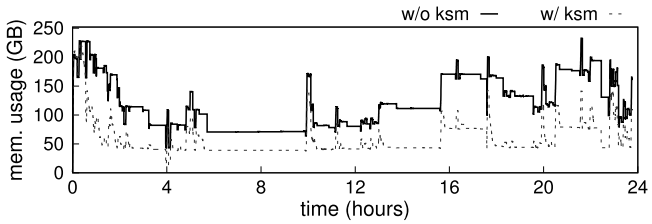


Figure 1: Memory capacity used by a server for 24 hours.

Table 1: DRAM power vs. the utilization of memory capacity.

Capacity	Utilization of Memory Capacity				
	10%	25%	50%	75%	100%
256GB	25.8W	25.8W	25.9W	26.0W	26.0W

of the memory systems. For example, the memory controller can set the power state of DRAM associated with off-lined regions to a deep power-down state that turns off the most of components and does not refresh states without worrying about paying a steep performance penalty to wake up the DRAM from the deep power-down state until the OS explicitly on-lines the regions. However, the OS can on/off-line only a region of contiguous physical address space. This limits the effectiveness of memory on/off-lining for current DRAM power management at the rank granularity, as memory interleaving techniques disperse the region across multiple ranks.

2.4 Kernel Samepage Merging

The Linux OS provides a memory de-duplication feature, Kernel Samepage Merging (KSM). KSM was originally developed to be used with kernel-based virtual machine (KVM). It allows us to share the same data at the page granularity between virtual machines (VMs) and thus fit more VMs into limited physical memory space of a server [2]. It is also useful for any application that generates many instances of the same data. We can enable KSM by compiling the kernel with `config_ksm` option and configure parameters such as the number of scanned pages in a single pass and the time between the passes using the `sysfs` interface. The KSM daemon (`ksmd`) periodically scans those areas of user memory which have been registered with it, looking for pages of identical content which can be replaced by a single write-protected page (which is automatically copied if a process later wants to update its content). Note that KSM only operates on those areas of address space which an application has advised to be likely candidates for merging, by using a system call, `madvise()`. When an application runs, `madvise()` is invoked and sets a flag (i.e., `MADV_MERGEABLE` flag) for mergeable memory regions. Then, `ksmd` periodically merges the mergeable pages by exploiting two trees, `stable tree` and `unstable tree`, for shared pages and non-shared pages, respectively. Specifically, `ksmd` traverses `stable tree` first to find a page with the same contents with the target page, and then traverses `unstable tree` when it fails to find the same page at `stable tree` and the checksum is equal to the old checksum in the previous scanning phase.

3 OPPORTUNITY AND CHALLENGE IN DRAM POWER MANAGEMENT

3.1 Utilization of Memory Capacity in Servers

In prior studies [25, 30], the data-center servers show 40%–60% utilization of the memory capacity on average. That is, the kernel may off-line approximately 50% of memory blocks on average, and thus considerably reduce the power and energy consumed by the main memory if it can place every off-lined memory block into power-down or self-refresh state.

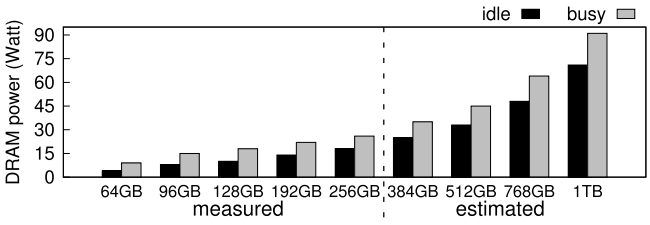


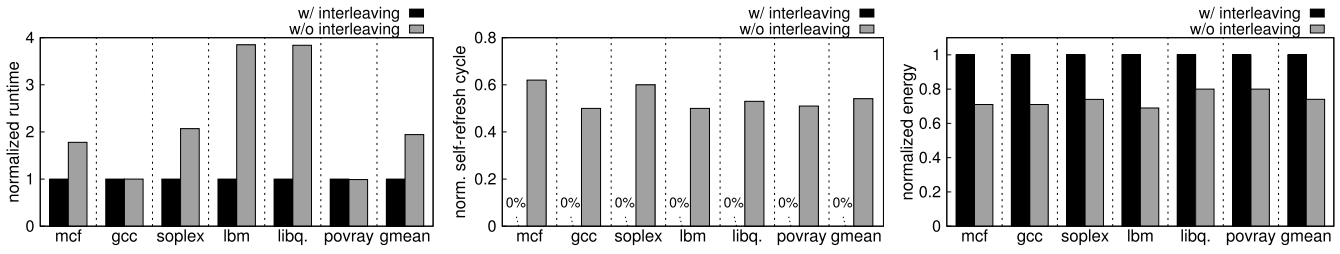
Figure 2: DRAM idle and busy power.

To investigate the utilization of memory capacity in data-center server environments, we measure the utilization of memory capacity for 24 hours after reproducing the execution setup of a server based on the Microsoft Azure VM trace [6]. Specifically, we first establish a virtualized system with KVM hypervisor on a server with a 16-core Intel Xeon processor and 256GB of memory consisting of eight 2R×4 32GB DDR4 DIMMs. Second, we randomly choose 100 different types of VMs from the Microsoft Azure VM trace, each of which has a different number of vCPUs, memory size, and lifetime. Lastly, we schedule/consolidate VMs on the server every five minutes while ensuring that the consolidation ratio of vCPUs is less than or equal to two and the total memory capacity used by VMs does not exceed the maximum memory capacity (i.e., 256GB), as virtualized clouds typically do.

Figure 1 shows the percentage of memory capacity used by VMs running on the server for 24 hours. It shows that the average memory capacity used by the scheduled VMs running on the server is 48% of the total memory capacity, while varying from 7% to 92% for 24 hours. This agrees with prior studies [25, 30] observing 40%–60% utilization of memory capacity on average and 90% usage at peak time [25]. As discussed in Section 2.4, KSM can further reduce the memory capacity used by VMs. Thus, we also measure the memory capacity used by the VMs after we enable KSM; the KVM hypervisor supports KSM for VMs without requiring any modification of source code. Figure 1 ('w/ ksm') shows that KSM can reduce the memory capacity used by the VMs by 4%–90% (24% on average). Consequently, KSM can allow the kernel to off-line 76% of the memory capacity on average.

3.2 DRAM Power in Server Environments

To obtain the breakdown of DRAM and server power, we first measure the power of the server (16-core processor with 64GB–256GB memory) using a HPM-100A power meter. We also measure the power consumed by the DRAM and processor using Running Average Power Limit (RAPL) [8]. In Figure 2, we show DRAM busy and idle power consumption as we increase the memory capacity. For the DRAM busy power consumption, we run 16 copies of a memory-intensive application (i.e., `mcf`) on the processor. This shows that the DRAM consumes a considerable amount of power even when it is idle. For example, 256GB DRAM consumes 18 and 26 Watts for idle and busy power, respectively. That is, the refresh and static power account for 70% of the total DRAM power. Moreover, as the memory capacity increases, both the total DRAM power and the fraction of DRAM background and static power steadily increase.



(a) Performance. (b) Percentage of cycles in self-refresh state. (c) Server Energy.

Figure 3: The impact of memory interleaving on server performance and energy; we run benchmarks with high misses per kilo-instructions (MPKI) from SPECCPU2006 on a commodity server with 64GB DDR4 DRAM (four channels, each with four ranks) with four-way channel and rank interleaving.

For example, while the DRAM power increases from 9 to 91 Watts as the DRAM capacity increases from 64GB to 1TB, the percentage of background DRAM power increases from 44% to 78%. Meanwhile, in Table 1, we show that the DRAM power consumption is relatively constant, although we change the utilization of the memory capacity. This is because not only sub-array peripheral and I/O components consume static power, but also sub-arrays associated with the unused capacity still need to be refreshed periodically without any power management for unused capacity.

3.3 Limitation of Current DRAM Power Management

A memory interleaving technique is often adopted by most commercial processors because it can significantly improve the performance of memory-intensive applications. For example, Figure 3a shows that it improves the performance of `lbm` by 3.8×. Nonetheless, it makes the current DRAM power management ineffective. This is because a memory interleaving technique disperses data in contiguous physical address space across ranks, while the DRAM power management supports the low-power states at the rank granularity [31]. That is, no rank can stay in idle state long enough to enter a low-power state. This is confirmed by Figure 3b ("w/ interleaving") where we measure the number of cycles that each rank spends in self-refresh state and plot the average of cycles from all the ranks. Although the memory capacity used by these memory-intensive benchmark programs is as small as 1.2GB out of 64GB (i.e., ∼2% of the total memory capacity), we observe that no rank ever enters self-refresh state[2]. Another DRAM power management technique is exploiting PASR which stops refresh of idle banks. Our experiment using `Ramulator`[19] running the same memory-intensive benchmarks also shows that no bank has a chance to enter a low-power state. That is, the current DRAM power management techniques relying on a low-power state at the rank or bank granularity (e.g., `RAMZzz` [33]) are not effective when a memory interleaving technique is deployed. In contrast, disabling the memory interleaving technique gives opportunities for the ranks to stay in a low-power state (e.g., 54% of execution cycles on average in Figure 3b ("w/o interleaving")), which allows the server to reduce energy consumption by 26% on average in Figure 3c.

[2]RAPL does not support the measurement of cycles that ranks spend in power-down state but our indirect measurement suggests that ranks do not enter the power-down state, either.

4 OS-ASSISTED DRAM POWER MANAGEMENT

To tackle the limitation of current rank-granularity DRAM power management, we propose an OS-assisted DRAM power management, `GreenDIMM`. Figure 4 describes the overall architecture of `GreenDIMM`. More specifically, `GreenDIMM` consists of three components. `GreenDIMM` takes a memory block in contiguous physical address space mapped to a group of DRAM sub-arrays across every channel, rank, and bank (❶ in Figure 4) with the same sub-array address (❷ in Figure 4). As the most significant bits of a physical address are used to index a group of sub-arrays, `GreenDIMM` not only provides fine-grained DRAM power management but also keeps the performance benefit of memory interleaving techniques (Section 4.1). `GreenDIMM` exploits memory on/off-lining operations of the modern OS to dynamically remove/add memory blocks from/to the physical address space (❸ in Figure 4), depending on the utilization of memory capacity at run-time (Section 4.2). `GreenDIMM` implements a deep power-down state at the sub-array granularity (❹ in Figure 4) to reduce the background power of the off-lined memory blocks (Section 4.3). The deep power-down state turns off the most of the peripheral and I/O components and does not refresh memory cells in sub-arrays mapped to off-lined memory blocks. This practically eliminates the background power consumed by the off-lined memory blocks. Meanwhile, as the off-lined memory blocks are removed from the physical address space, the sub-arrays will not receive any memory requests and stay in the power-down state until the memory blocks are explicitly on-lined by the OS. Thus, `GreenDIMM` can considerably reduce power and energy consumed by unused memory space, which is significant even in highly consolidated data-center environments, with a negligible performance penalty. The rest of this section describes each component in detail.

4.1 Interleaving-Agnostic DRAM Power Management

Modern processors typically use memory interleaving techniques that disperse data in contiguous physical address space across channels, ranks, and banks, hashing a part of physical address bits to choose a channel, rank, and bank addresses. As discussed in Section 3.3, memory interleaving techniques better exploit memory-level parallelism and thus improve the performance of memory-intensive applications. As it is important to preserve the performance benefit of the memory interleaving techniques, `GreenDIMM`

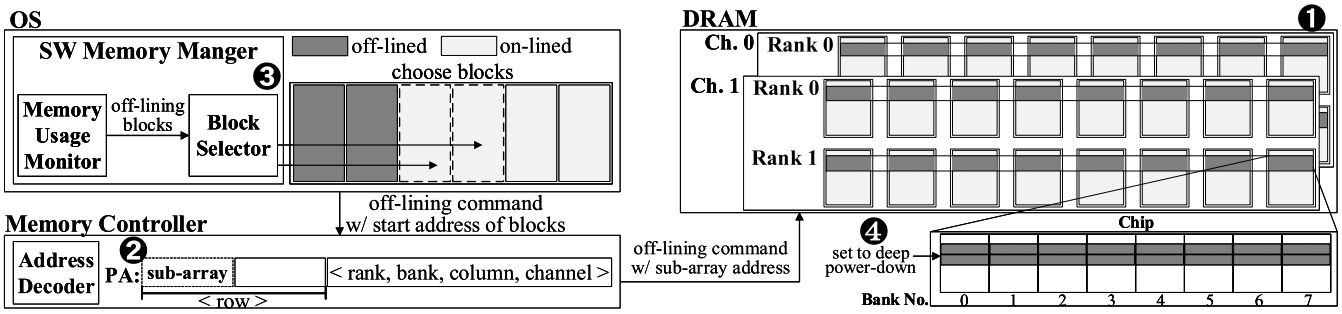


Figure 4: The overall architecture of GreenDIMM.

proposes an interleaving-agnostic DRAM power management technique.

Specifically, GreenDIMM exploits the fact that the most significant N bits of the physical address are used to index a row in a bank and the most significant M bits of the row address are taken to index a sub-array in the bank (i.e., a group of sub-arrays across DRAM devices in a rank). For example, consider 64GB main memory consisting of 4 channels, each with a 2-rank×8 DIMM, which leads to memory interleaving depicted in Figure 5. A rank comprises eight 4Gb DRAM devices and thus provides 4GB with 16 256MB (logical) banks. In Figure 5, each (physical) bank has two types of row address decoders, a global and local row decoders. When decoding the row address, the global decoder chooses a sub-array, and the local decoder chooses a row in the sub-array. In an 4Gb×8 DRAM device, the number of bits for the row address is 15 (= N) and the number of sub-arrays is 64 in a bank as illustrated in Figure 5. That is, the global decoder uses 6 bits (= M) to choose a 4Mb sub-array (i.e., 4MB across 8 DRAM devices in a rank), and uses the remaining 9 bits to choose a particular row among 512 rows in the sub-array.

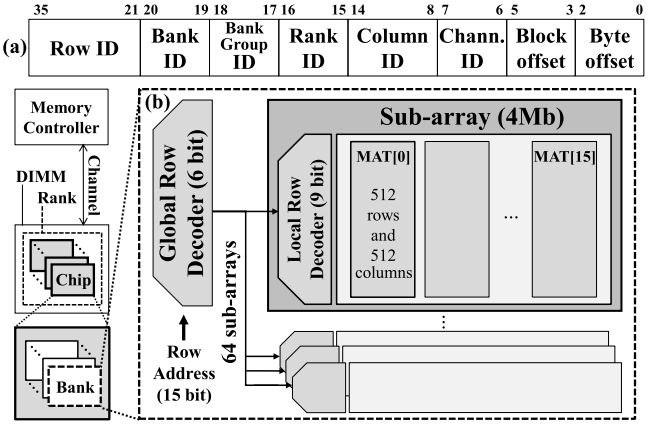


Figure 5: Address mapping (a) for a 64GB main memory consisting of 4 channels, each with a 4-rank ×8 DIMM adn (b) within a DDR4 ×8 4Gb DRAM device. The sub-array in the grey-color box is the minimum unit of DRAM power management.

Exploiting the fact described in previous paragraph, GreenDIMM proposes a group of sub-arrays across every channel, rank, and bank ((❶ in Figure 4)) with the same row address range (❷ in Figure 4) as a minimum unit of DRAM power management. In Figure 5, as the 64GB main memory has 16 ranks, each with 16 banks, the minimum unit becomes 1024MB (= 4MB × 16 banks × 16 ranks), 1.5625% of the total capacity; the percentage does not change with smaller or larger total capacity. This granularity of DRAM power management is fine enough to maximize the chance of reducing the background power of unused memory space while keeping the benefit of interleaving across channels, ranks and banks. The unit of DRAM power management can vary with grouping of more sub-arrays (e.g., two or four adjacent sub-arrays in a bank), and we will further discuss its impact on power/energy reduction and performance in Section 5.1.

4.2 DRAM Power Management Based on Memory On/Off-lining

GreenDIMM implements DRAM power management daemon that leverages the memory on/off-lining operations supported by Linux (❸ in Figure 4). Specifically, `memory_usage_monitor()` of GreenDIMM periodically obtains the current utilization of the memory capacity from `/proc/meminfo` [13] in Linux and then records the memory utilization (e.g., every 1 second); we observe that the period less than 1s increases the monitoring overhead while not helping off-line more memory blocks. If the amount of on-lined but unused memory (or free memory) is greater than a threshold, `off_thr` (e.g., 10% + α of the total memory capacity), `memory_usage_monitor()` sends a request to `block_selector()` of GreenDIMM to choose free memory blocks to be off-lined. We observe that the system performance dramatically degrades when the threshold is less than 10% because pages are frequently swapped between the main memory and the storage. `block_selector()`, which we implemented, searches the list of memory blocks and picks *movable* memory blocks. Among the *movable* blocks, GreenDIMM chooses to off-line blocks only with unused pages. Consequently, off-lining does not migrate data and their physical addresses are unlikely to be cached in TLB. Note that there are *unmovable* memory blocks that are used by the kernel or devices (cf. Section 5.2).

Then, `block_selector()` calls `offline_pages()` of Linux with the starting physical page number (PPN) and the number of pages to off-line the selected memory blocks from the physical address space. After GreenDIMM completes the memory off-lining, it updates

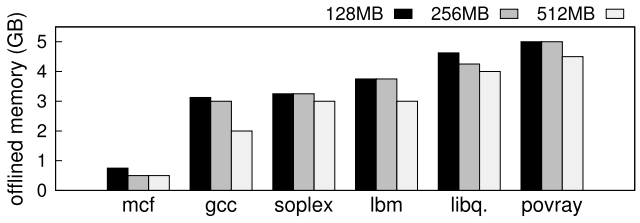


Figure 6: The off-lined memory capacity as the size of a memory block changes; a memory block maps to one (128MB), two (256MB), and four (512MB) sub-array groups.

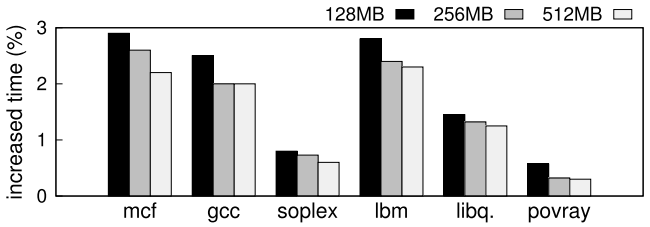


Figure 7: Execution time as the size of a memory block change; a memory block maps to one (128MB), two (256MB), and four (512MB) sub-array groups.

control registers in the memory controller, based on the starting physical address and the number of pages in the off-lined memory block. This makes sub-arrays associated with the off-lined memory blocks enter the deep power-down state. We describe the mechanism and overhead in Section 4.3.

When `memory_usage_monitor()` observes that the amount of free memory is lower than a threshold, `on_thr`, it begins to on-line memory blocks. On-lining an off-lined memory block goes through the off-lining steps in the reverse order, except that `GreenDIMM` should wait until the sub-arrays completely exit from the deep power-down state before it calls `online_pages() of Linux`. `GreenDIMM` can determine whether the sub-arrays are ready by polling a memory controller register bit. We provide more information on the exit latency of the deep power-downstate in Section 4.3.

4.3 Sub-array Deep Power-down State

`GreenDIMM`'s deep power-down state at the sub-array granularity requires two steps. First, the memory controller stops refresh of sub-arrays corresponding to the physical address space of the off-lined memory block. Its implementation basically builds on the current memory controller's support for PASR and partial array auto-refresh (PAAR), stopping refresh of one or more banks [5]. A memory controller supporting PASR has a memory-mapped a bit vector register that enables/disables the refresh of a bank, it will need 16 bits per rank and 128 bits for 4 channels, each with 2 ranks (i.e., memory system configuration in our setup). By contrast, `GreenDIMM` requires fewer bits as it controls a group of sub-arrays across channels, ranks, and banks with a single bit. That is, regardless of the number of channels and ranks, `GreenDIMM` needs only 64 bits as there are always 64 groups of sub-arrays.

Second, the memory controller sets the DRAM mode register [1] such that the peripheral and I/O circuits of sub-arrays are turned off. Its implementation builds on the current memory controller's support for the power-down state at the rank granularity described in Section 2.2 together with the bit vector register to enable/disable the refresh of a sub-array. After the DRAM mode register of all every DRAM device in a rank is concurrently updated, each DRAM device turns off the power gates for the peripheral and I/O circuits of the target sub-array.

Note that, a power gate is a switch transistor that connects the external power/ground with the internal power/ground of DRAM circuits, and the power-gating granularity can be as small as a circuit block (e.g., row decoder of a sub-array). Especially, PMOS switch transistors are mainly used for pull-up (between external and internal power) while NMOS switch transistors are used for pull-down (between internal and external ground). As the use of an excessively large switch transistors can increase both the area overhead and the switch's on/off latency, the size of switch transistors are carefully optimized. In general, a long length transistor of 1.2 times or more is used to prevent leakage current and the turn-on resistance of the power switch is designed to be less than 0.1Ω. Our analysis based on a commercial 1xnm 8Gb DRAM design shows that the size of the switch transistors for each subarray is $1500\mu m^2$, accounting for 0.64% of the total DRAM chip size. Even if the control logic is added in each sub-array unit, the area overhead is negligible as it is less than 1% of the total DRAM chip size.

GreenDIMM control circuit in DRAM is almost identical to ones for PASR/PAAR. The only difference is how we group sub-arrays for preventing refresh. PASR/PAAR can be applied to every group of 16 sub-arrays in each bank; 16 instances of the control circuit are required as there are 16 banks in each chip, while GreenDIMM is applied to every group of 16 sub-arrays across 16 banks in a lock-step manner. As such, we expect DRAM cost increase is almost the same as PASR/PAAR, i.e., 0.1% of total DRAM die area.

The long exit latency of the traditional self-refresh state (e.g., 768ns) is primarily contributed by turning on the DLL circuit of DRAM devices. In contrast, `GreenDIMM`'s deep power-down state does not turn off the DLL circuit, as it places only a part of DRAM devices. Thus, the exit latency of `GreenDIMM`'s deep power-down state is no longer than that of the power-down state which takes only 18ns to disable the clock and turn off the I/O circuits of the entire DRAM device.

Table 2: The number of on-lined and off-lined memory blocks as the memory block size changes.

Application	# of on/off 128MB	# of on/off 256MB	# of on/off 512MB
mcf	6	2	1
gcc	47	24	12
soplex	36	18	8
lbm	30	15	6
libquantum	37	17	8
povray	40	20	9

Table 3: The average latency of off-lining, on-lining, and off-lining failure while running `mcf`.

Event	Avg. Latency (ms)
off-lining	1.58 ms
on-lining	3.44 ms
failure (EAGAIN)	4.37 ms
failure (EBUSY)	6 μs

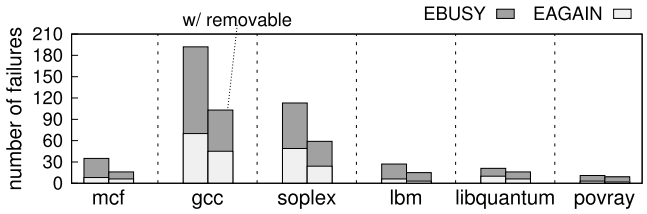


Figure 8: The number of off-lining failures with a `removable` value.

5 IMPLEMENTATION CHALLENGES

5.1 Size of On/Off-lined Memory Blocks

As described in Section 4, `GreenDIMM` monitors the current utilization of memory capacity and determines on-line/off-line memory blocks periodically. The size of a memory block, which is the unit of memory on/off-lining, affects the off-lined capacity of `GreenDIMM` and the applications' performance. The default size of a memory block for on/off-lining is 128MB in Linux and the size of a memory block is configurable through `/sys/devices/system/memory/block_size_bytes`. For example, as the size of a memory block increases, `GreenDIMM` can off-line less capacity since the amount of free memory except for reserved free memory with `off_thr` (10% of total capacity) must be larger than the block size to off-line a block. In addition, the size of a memory block determines the number of on/off-lining including page table updates and page migrations, affecting performance.

Figure 6 plots the off-lined capacity by `GreenDIMM` as the size of a memory block change. We use 128MB, 256MB, and 512MB as the size of a memory block, each of which is the size when a memory block is mapped to one, two, and four sub-array groups in our experimental setup; note that we can manipulate the size of a memory block based on the capacity of a sub-array group. For example, with 512MB memory blocks, four sub-array groups enter to deep power-down state when the kernel off-lines a memory block. Not surprisingly, `GreenDIMM` off-lines more blocks with the smaller block size. For example, while running `gcc`, `GreenDIMM` with 128MB memory blocks off-lines 3.125GB while off-lining 2GB with 512MB memory blocks. Since `GreenDIMM` can off-line the memory block only when the amount of free memory that can be turned off is greater than the single block size, `GreenDIMM` is likely to off-line less capacity with the larger block size. Furthermore, as the size of a memory block increases, the off-lining failure occurs more frequently since all pages in the block must be removable; we will discuss the off-lining failure in Section 5.2. To observe the performance overhead as the memory block size changes, we plot the increased execution time in Figure 7. While all cases show performance degradation less than 3%, the performance overheads slightly increase as the size of a memory block decreases. For example, `GreenDIMM` degrades the performance of `mcf` by 2.9% with 128MB blocks while degrading the performance of `mcf` by 2.2% with 512MB blocks.

The reason for more performance overheads with the smaller size of the memory block is that the number of on-lining/off-lining increases as the size of the block decreases as represented in Table 2. For example, with `gcc`, the number of off-lining/on-lining is reduced from 47 to 12, as the size of memory block increases (i.e., from 128MB to 512MB). Since on-lining and off-lining require extra operations for enabling/disabling memory blocks in the physical address space, more on-linings/off-linings lead to more performance degradation.

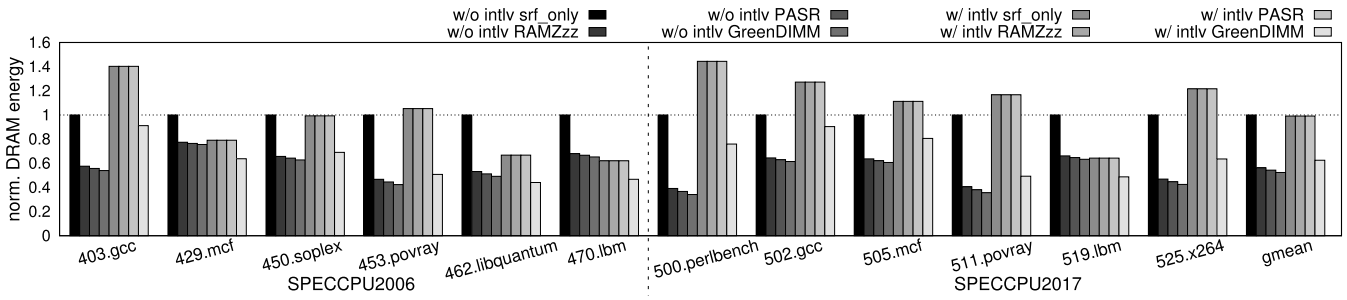
Although `GreenDIMM` shows the different performance overheads and off-lined capacity with varying sizes of the block, we observe that the difference in the performance overheads is not much while the difference in the off-lined capacity is considerable for some applications. We use the size of a memory block that fits in a sub-array group (i.e., 128MB block size) in our evaluation for more power reduction. However, since the block size is configurable and `GreenDIMM` does not require extra features as the size of block changes, `GreenDIMM` can decrease/increase the block size for more power reduction or less performance overheads.

5.2 Identifying Removable Memory Blocks for Off-lining

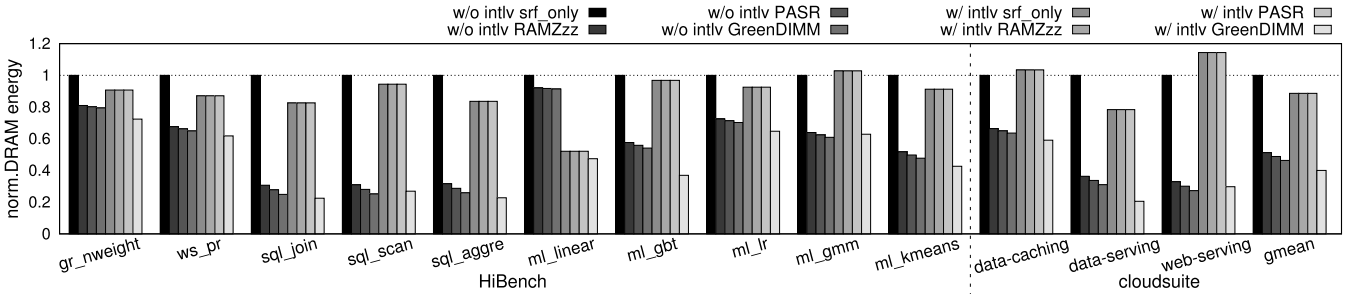
`GreenDIMM` fails to off-line a memory block when any page in the block is used by the kernel or devices since the page is not migratable (i.e., unmovable). Furthermore, repetitive failures can also occur, degrading the performance notably. Although the Linux OS can configure the range of movable memory regions through a booting parameter (e.g., movablecore=8G), the reserved movable regions can also have unmovable pages.

We observe two cases of the off-lining failure, `EBUSY` and `EAGAIN`. `EBUSY` occurs when the kernel fails to isolate the target memory block from the physical address space since some pages in the memory block are unmovable. `EAGAIN` occurs when the kernel cannot use the necessary resource temporarily even though all pages in the block are movable. For example, while off-lining memory blocks, `EAGAIN` occurs when the kernel fails to find other memory regions to migrate pages from the off-lined blocks.

To observe the effects of off-lining failures on the performance, we represent the average latency when on-lining, off-lining, off-lining failure with `EAGAIN`, and off-lining failure with `EBUSY` occur while running `mcf` in Table 3; we map a memory block to a sub-array group (i.e., 128MB block size). As shown, the off-lining failure with `EBUSY` only requires 6μs latency. However, the latency of off-lining failure with `EAGAIN` is even longer than the latency of off-lining success. For example, the failure latency with `EAGAIN` is 4.37ms while the off-lining latency is 1.58ms; in our experiments, the off-lining success occurs only when we attempt to off-line a memory block only with unused pages, requiring no page migration. Since `EAGAIN` occurs after three attempts to migrate pages fail if we choose a memory block that has used pages, it shows about 3x longer latency than the success; note that recent kernel attempts page migration infinitely until it succeeds, which can degrade the performance substantially.



(a) SPECCPU2006 and SPECCPU2017.



(b) HiBench and cloudsuite.

Figure 9: DRAM energy consumption.

To reduce the number of off-lining failures, GreenDIMM uses a variable (i.e., removable) offered by the Linux kernel through sysfs, which represents whether the memory block contains unmovable pages or not; removable is true when all pages in the block are movable. Since the value of removable becomes zero when the memory block has unmovable pages, if GreenDIMM chooses a block to off-line among blocks with removable set to one, the number of off-lining failures will be alleviated.

Figure 8 shows the comparison of the number of off-lining failures between when GreenDIMM chooses blocks to off-line randomly and when GreenDIMM chooses blocks with removable set to 1 first. As plotted, applications showing frequent changes in the memory footprint, such as gcc and soplex, show more off-lining failures than applications showing smaller changes in the memory footprint, such as mcf, lbm, libquantum, and povray. Furthermore, we observe that the portion of EAGAIN increases as the number of off-lining failures increases. Consequently, as the number of off-lining failures increases, the performance can be degraded substantially; note that failures with EAGAIN show much longer latency than failures with EBUSY. However, if GreenDIMM checks the removable for each block when it chooses blocks to off-line, GreenDIMM reduces the number of off-lining failures by about 50%, mitigating delays due to the off-lining failures.

5.3 Memory Off-lining with Kernel Samepage Merging

As discussed in Section 3.1, KSM, which reduces the memory footprint of applications, helps GreenDIMM to reduce the energy consumption by allowing GreenDIMM to off-line further memory blocks. To quickly react to the changes in the utilization of the memory capacity by the KSM, GreenDIMM monitors the current utilization of the memory capacity and attempts to off-line memory blocks as soon as the KSM daemon completes merging pages regardless of GreenDIMM's monitoring period. We will discuss the effects of such simple optimization for the KSM in Section 6.3.

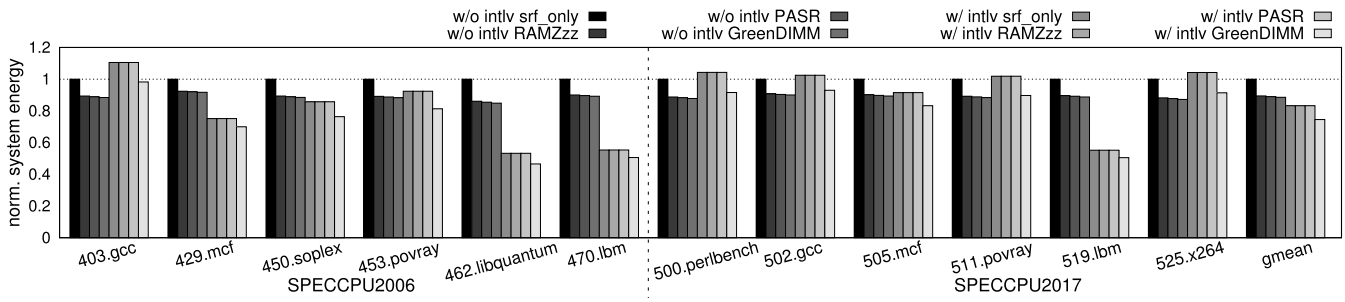
Although the KSM daemon reduces the memory footprint further, it incurs extra performance overheads while consuming computing resources for scanning and merging pages. Furthermore, copy-on-write operations occur when shared pages are modified. However, KSM's parameters related to performance overheads are configurable, such as the number of pages to be scanned and the scanning period. In this work, we use 1000 and 50ms as the number of pages to be scanned and the scanning period, respectively. With the configuration, the KSM daemon only consumes 10% of a core to scan, merge, and manage the pages while showing the considerable reduction in the memory footprint; we can also change the parameter values to reduce the memory footprint further at the cost of performance.

6 EVALUATION

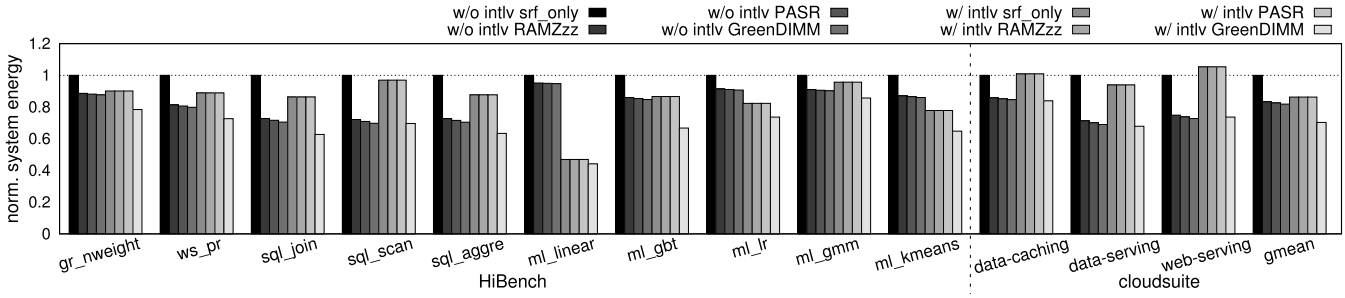
6.1 Experimental Methodology

To evaluate GreenDIMM, we run applications from SPECCPU2006 [14] and SPECCPU2017 [4]. We evaluate GreenDIMM with data-center workloads from HiBench [15], cloudsuite [12]. Furthermore, we also evaluate GreenDIMM in the virtualized server environments using Microsoft Azure VM trace [6]; we establish a hypervisor-based virtualized system on the real machine using KVM hypervisor.

We implement the software manager of GreenDIMM that on/off-lines memory blocks based on the current utilization of the memory capacity and estimate the effects of the on-lining/off-lining on DRAM power reduction using CACTI [3]. We measure the power of the real machine using hardware performance counters and power



(a) SPECCPU2006 and SPECCPU2017.



(b) HiBench and cloudsuite.

Figure 10: System energy consumption

meters as we do in Section 3.1. We compare GreenDIMM with the conventional techniques and prior studies that manage the background power by the channel, rank, and bank-granularity, such as self-refresh, RAMZzz [33], and PASR [10], respectively. Lastly, to observe the synergy when GreenDIMM with KSM as mentioned in Section 3.1, we conduct experiments after enabling KSM with the Azure VM trace.

We use eight 4Gb 2R x8 DDR4-2133 8GB DIMMs (total 64GB) with four channels, each of which has two DIMM slots for running SPECCPU applications and data-center workloads. For running the Azure VM trace, we use eight 8Gb 2R x4 DDR4-2133 32GB DIMMs (total 256GB). We configure the size of a memory block to fit in a group of sub-arrays as discussed in Section 5.1. We off-line a sub-array only when neighboring sub-arrays are off-lined, assuming that two consecutive sub-arrays share a sense amplifier in the middle [26]. We also assume spare rows from separate repair arrays [18], which typically occupy less than 2% of total DRAM rows, are always turned on.

6.2 Energy Reduction

We measure energy consumption with interleaving and without interleaving. We denote them as `w/ intlv` and `w/o intlv`. We compare GreenDIMM with RAMZzz and PASR rank-granularity and bank-granularity power management, respectively. RAMZzz monitors the memory access pattern, and migrates data from cold rank to hot rank to reserve the more idle ranks, and promotes/demotes the power state of the cold ranks for the background power. PASR allows each idle bank to enter the deep power-down state in mobile DRAM for the background power reduction. For comparison with RAMZzz and PASR, we model power reduction by them based on the number of idle ranks/banks.

Figure 9 shows DRAM energy consumption. We normalize all results to the results of `w/o intlv srf_only`, which means that only self-refresh is used for background power management without memory interleaving. As plotted, the memory interleaving increases DRAM energy considerably for CPU-intensive, but not memory-intensive workloads even with rank-/bank-granularity power management, such as RAMZzz and PASR. For example, memory interleaving increases DRAM energy by 40% and 44% for `403.gcc` and `500.perlbench`, respectively, when only self-refresh is used. This is because the memory interleaving prevents ranks/banks from entering low power states even with applications showing a small memory footprint.

However, GreenDIMM reduces DRAM energy consumption effectively for all workloads even though memory interleaving is enabled. GreenDIMM reduces the energy consumption for `403.gcc` that even shows the minimum reduction by 9% compared with `w/ intlv srf_only`. Compared with the results of RAMZzz and PASR, GreenDIMM shows more reduction by 49%p (percent point) when the interleaving is enabled. On the other hand, for memory-intensive workloads, the memory interleaving reduces the DRAM energy considerably. For example, for `470.lbm`, the memory interleaving reduces the DRAM energy consumption by 38% for self-refresh only cases. This is because the memory interleaving reduces the execution time of memory-intensive applications (e.g., `462.libqauntum`, `470.lbm`, and `ml_linear`) substantially even though it prevents ranks/banks from entering low power states while running applications. However, GreenDIMM reduces the background power regardless of memory interleaving even while the memory-intensive application is running, leading to energy reduction further. GreenDIMM reduces DRAM power for SPECCPU applications and data-center workloads by 38% and 60% on average.

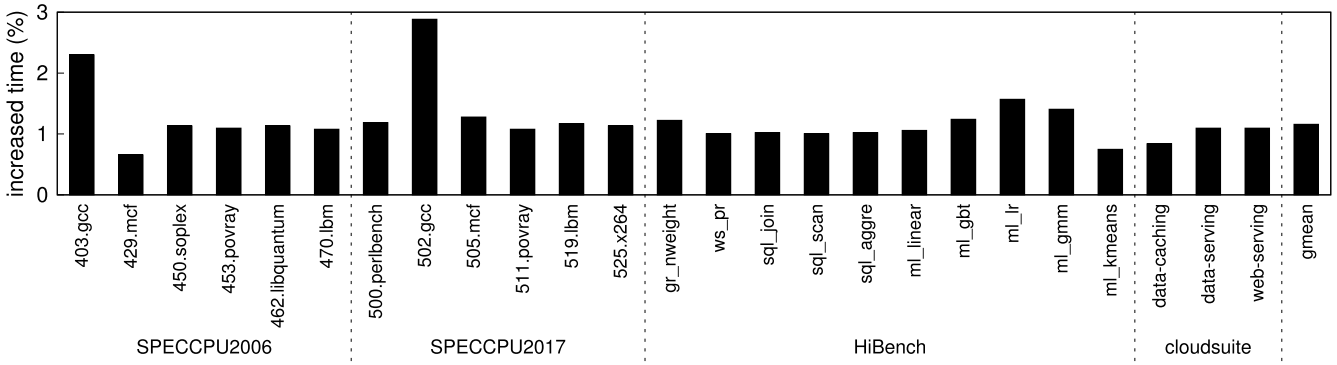


Figure 11: Increased execution time by GreenDIMM.

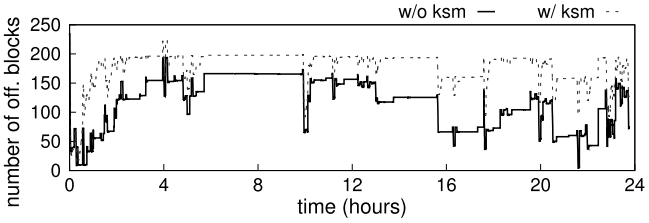


Figure 12: The number of off-lined blocks with VM trace.

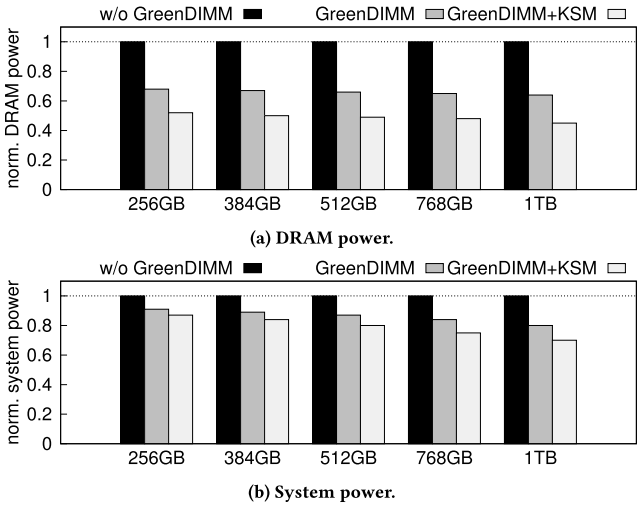


Figure 13: DRAM and system power consumption as the memory capacity increases.

Figure 10 shows system energy consumption. We also normalize all results to results of `w/o intlv srf_only`. As plotted, `RAMZzz`, `PASR`, and `GreenDIMM` reduce the system energy effectively w/o memory interleaving. However, only `GreenDIMM` reduces the system energy consumption with memory interleaving, since channel/rank/bank memory interleaving disturbs the channel-, rank-, and bank- granularity power management as aforementioned before.

The memory interleaving also increases the system energy consumption for non-memory-intensive workloads. For example, in the case of `403.gcc`, memory interleaving increases the system energy consumption by 10% while increasing DRAM energy consumption by 1.4× as plotted in Figure 9a. For memory-intensive workloads, `462.libquantum`, `470.lbm`, `519.lbm`, and `ml_linear`, memory interleaving reduces the system energy by 47%, 45%, 45%, and 54%, respectively, even though the interleaving prevents ranks/banks to enter low power states. However, `GreenDIMM` reduces system energy reduction regardless of memory interleaving. For SPECCPU applications and data-center workloads, `GreenDIMM` reduces system energy by 26% and 30% on average.

Lastly, we investigate the performance overheads in terms of the execution time and tail response latency (i.e., 95^{th} and 99^{th} percentile latency) by on-/off-lining memory blocks of `GreenDIMM` for power management. We observe that `GreenDIMM` consumes 0.34%/0.16% of a single core's cycles for on-lining/off-lining every 1s only when on-lining/off-lining occurs `GreenDIMM` on-lines/off-lines 0.05/0.47 blocks every 1s on average. Consequently, `GreenDIMM` does not incur considerable performance degradation as plotted in Figure 11. For example, `403.gcc` and `502.gcc` show the most performance degradation by less than 3% while other workloads show the degradation by less than 2%. Furthermore, we do not observe notable degradation in tail response latency of latency-critical applications by `GreenDIMM`, such as `data-caching`, `data-serving`, and `web-serving`. For those applications, `GreenDIMM` shows much less number of on/off-lining as the applications show a constant memory footprint.

6.3 Power Reduction in VM Server Environment

As discussed in Section 3.1, the server environments have many opportunities to reduce DRAM power with low average memory utilization. Although rank-granularity low power states fail to reduce the DRAM power, sub-array granularity power management by `GreenDIMM` reduces the DRAM power considerably with the low memory utilization.

Figure 12 shows the number of off-lined blocks. With the VM trace, `GreenDIMM` off-lines 116 blocks (45% of total capacity) on average out of 256 blocks; `GreenDIMM` off-lines 230 (90% of total capacity) and 4 (1.5% of total capacity) memory blocks when the utilization of

memory capacity is minimum and maximum, respectively; we use 1GB as the size of memory block for 256GB memory size. Assuming the sub-array groups mapped to the off-lined memory blocks enter deep power-down state, GreenDIMM reduces the DRAM background power by 46%. Furthermore, with KSM, GreenDIMM off-lines 61 more memory blocks and reduces the DRAM background power by 70%.

Figure 13 shows the estimated DRAM power and system power consumption with GreenDIMM. Based on the measured power with 256GB memory, we estimate the DRAM power and system power as the memory capacity increases using a simple linear model. With 256GB memory capacity, GreenDIMM reduces the DRAM power and system power by 32% and 9% on average, respectively. With KSM, GreenDIMM reduces the DRAM power and system power by 48% and 13%, respectively. GreenDIMM reduces more power as the memory capacity increases since the memory with larger capacity consumes more background power. As plotted in Figure 13, GreenDIMM reduces 36% and 20% of DRAM power and system power, respectively, with 1TB memory capacity. With KSM, GreenDIMM reduces 55% and 30% of DRAM power and system power, respectively, with 1TB memory capacity. These results demonstrate that the background power of DRAM is one of the major contributors to the total system power, and GreenDIMM can reduce the system power considerably by reducing DRAM background power, especially with large capacity.

Note that GreenDIMM will not reduce power consumption of memory in servers running storage and in-memory database applications that consume most of or all the memory space (e.g., page cache for storage servers).

7 RELATED WORK

There have been many hardware/software studies for DRAM power management [9, 10, 16, 21, 27, 33, 35]. RAMZzz [33] proposes rank-aware power management that groups pages showing analogous locality and place them into the same rank. RAMZzz classifies ranks into the hot rank and cold rank, each of which means the frequently accessed rank and rarely accessed rank, respectively. Then, RAMZzz migrates pages in the cold ranks to the hot ranks, and makes the cold ranks enter the low power state (e.g., self-refresh). RAMZzz requires monitoring accesses of all pages, increasing performance overhead considerably, while not considering the memory interleaving. PASR [10] is the refresh policy to reduce the background power consumption for Mobile DRAM. PASR allows some banks in a rank to enter self-refresh state and the other banks to enter deep power-down state, reducing the background power considerably. However, PASR also cannot be adopted to modern systems employing memory interleaving.

ESKIMO [16] proposes a DRAM power management that selectively refreshes allocated pages by tracking allocation/de-allocation of pages. ESKIMO also reduces power for unused pages as GreenDIMM, but ESKIMO focuses on the refresh power reduction while GreenDIMM mostly eliminates the background power by implementing deep power-down state at the sub-array granularity without tracking allocation/de-allocation of pages. PADRAM [21] proposes power-aware page allocation that allows chips to enter low power states more often. When allocating pages, PADRAM first allocates the pages mapped to chips that do not stay at the low power states. However, PADRAM also does not consider the memory interleaving. Zhou et al. [35] propose a utility-based memory allocation policy to reduce the DRAM power. They track Miss Ratio Curve (MRC) for applications at run-time, and identify idle regions and allow them to enter low power states. Malladi et al. [27] propose a mechanism that reduces the background power by eliminating or mitigating long-latency DLL wake-ups by shifting circuitry from the DRAM to the controller. After improving the wake-up latency, they reduce the power consumption by aggressively entering power-down states even during the short idle period. Delaluz et al. [9] propose a data migration policy that reduces energy by dynamically placing arrays with temporal affinity into the same set of banks. While assuming bank granularity low power states, they reduce energy by reserving more idle banks.

8 CONCLUSION

We propose GreenDIMM that is OS-assisted DRAM power management with a sub-array granularity power-down state. GreenDIMM uses a DRAM sub-array group across every channel, rank, and bank as a power management unit, and maps a memory block in the physical address space to the sub-array group. Then, GreenDIMM exploits memory on-/off-lining to off-line unused capacity of memory, allowing the sub-array group mapped to the off-lined blocks to enter deep power-down state. For the sub-arrays mapped to the off-lined blocks, GreenDIMM implements deep power-down state at the sub-array granularity. Consequently, GreenDIMM reduces the background power with the software memory on-lining/off-lining policy based on the current utilization of the memory capacity. Our evaluation with a commercial server running diverse workloads including data-center workloads shows that GreenDIMM reduces DRAM and system power by 36% and 20%, respectively. Furthermore, GreenDIMM reduces DRAM and system power by 36% and 20%, respectively, in VM server environments. We also investigate synergistic potentials in power reduction with KSM. With the KSM, our experimental results show that GreenDIMM can reduce DRAM and system power by 55% and 30%, respectively.

ACKNOWLEDGMENTS

This work was partly supported by Samsung Electronics, National Research Foundation of Korea (NRF) grants funded by the Korean government (MSIT) (NRF-2020R1C1C1013315, NRF-2018R1A5A1060031), Samsung Research Funding Incubation Center of Samsung Electronics under Project Number SRFC-IT1902-03, Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No.2014-3-00065, Resilient Cyber-Physical Systems Research), and National Science Foundation grant (CNS-1705047). Daehoon Kim is the corresponding author.

REFERENCES

[1] Sep 2012. (Accessed: April 2021). Main Memory: DDR4 & DDR5 SDRAM. https://www.jedec.org/category/technology-focus-area/main-memory-ddr3-ddr4-sdram.
[2] Andrea Arcangeli, Izik Eidus, and Chris Wright. 2009. Increasing memory density by using KSM. In *Proceedings of the Ottawa Linux Symposium (OLS)*. 19–28.
[3] Rajeev Balasubramonian, Andrew B Kahng, Naveen Muralimanohar, Ali Shafiee, and Vaishnav Srinivas. 2017. CACTI 7: New tools for interconnect exploration

in innovative off-chip memories. *ACM Transactions on Architecture and Code Optimization (TACO)* 14, 2 (2017), 1–25.

[4] James Bucek, Klaus-Dieter Lange, and Jóakim v. Kistowski. 2018. SPEC CPU2017: Next-generation compute benchmark. In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering (ICPE)*. 41–42.

[5] Maxime Coquelin. Jan 2012. (Accessed: April 2021). PASR: Partial Array Self-Refresh Framework. LWN. https://lwn.net/Articles/478049/.

[6] 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 (SOSP)*. 153–167.

[7] Howard David, Chris Fallin, Eugene Gorbatov, Ulf R Hanebutte, and Onur Mutlu. 2011. Memory power management via dynamic voltage/frequency scaling. In *Proceedings of the 8th ACM international conference on Autonomic computing (ICAC)*. 31–40.

[8] Howard David, Eugene Gorbatov, Ulf R Hanebutte, Rahul Khanna, and Christian Le. 2010. RAPL: Memory power estimation and capping. In *2010 ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED)*. 189–194.

[9] Victor De La Luz, Mahmut Kandemir, and Ibrahim Kolcu. 2002. Automatic data migration for reducing energy consumption in multi-bank memory systems. In *IEEE Design Automation Conference (DAC)*. 213–218.

[10] Elpida Elpida. 2005. Partial Array Self Refresh (PASR). https://media-www.micron.com/-/media/client/global/documents/products/technical-note/dram/e0597e10.pdf?rev=07992f36c55f4e7e8b0c9aaafcda90dd.

[11] Xiaobo Fan, Carla Ellis, and Alvin Lebeck. 2001. Memory controller policies for DRAM power management. In *International Symposium on Low Power Electronics and Design (ISLPED)*. 129–134.

[12] Michael Ferdman, Almutaz Adileh, Onur Kocberber, Stavros Volos, Mohammad Alisafaee, Djordje Jevdjic, Cansu Kaynak, Adrian Daniel Popescu, Anastasia Ailamaki, and Babak Falsafi. 2012. Clearing the clouds: a study of emerging scale-out workloads on modern hardware. In *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 37–48.

[13] Dave Hansen. Apr 2003. (Accessed: April 2021). meminfo documentation. LWN. https://lwn.net/Articles/28309/.

[14] John L Henning. 2006. SPEC CPU2006 benchmark descriptions. *ACM SIGARCH Computer Architecture News* (2006), 1–17.

[15] Shengsheng Huang, Jie Huang, Jinquan Dai, Tao Xie, and Bo Huang. 2010. The HiBench benchmark suite: Characterization of the MapReduce-based data analysis. In *2010 IEEE 26th International Conference on Data Engineering Workshops (ICDEW)*. IEEE, 41–51.

[16] Ciji Isen and Lizy John. 2009. ESKIMO - Energy Savings using Semantic Knowledge of Inconsequential Memory Occupancy for DRAM subsystem. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 337–346.

[17] Yasuaki Ishimatsu. May 29th, 2013. Memory Hotplug. https://www.fujitsu.com/jp/documents/products/software/os/linux/catalog/LinuxConJapan2013-Ishimatsu.pdf.

[18] Brent Keeth, R Jacob Baker, Brian Johnson, and Feng Lin. 2007. *DRAM circuit design: fundamental and high-speed topics*. Vol. 13. John Wiley & Sons.

[19] Yoongu Kim, Weikun Yang, and Onur Mutlu. 2015. Ramulator: A fast and extensible DRAM simulator. *IEEE Computer architecture letters* 15, 1 (2015), 45–49.

[20] Karthik Kumar, Kshitij Doshi, Martin Dimitrov, and Yung-Hsiang Lu. 2011. Memory energy management for an enterprise decision support system. In *IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. 277–282.

[21] Alvin R Lebeck, Xiaobo Fan, Heng Zeng, and Carla Ellis. 2012. Power aware page allocation. In *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 105–116.

[22] Wei-Fen Lin, Steven K Reinhardt, and Doug Burger. 2001. Reducing DRAM latencies with an integrated memory hierarchy design. In *Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture (HPCA)*. 301–312.

[23] Haikun Liu, Hai Jin, Xiaofei Liao, Wei Deng, Bingsheng He, and Cheng-zhong Xu. 2014. Hotplug or ballooning: A comparative study on dynamic memory management techniques for virtual machines. *IEEE Transactions on parallel and distributed systems (TPDS)* (2014), 1350–1363.

[24] Jamie Liu, Ben Jaiyen, Richard Veras, and Onur Mutlu. 2012. RAIDR: Retention-aware intelligent DRAM refresh. In *ACM/IEEE International Symposium on Computer Architecture (ISCA)*. 1–12.

[25] 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)*. 2884–2892.

[26] Haocong Luo, Taha Shahroodi, Hasan Hassan, Minesh Patel, Abdullah Giray Yaglikci, Lois Orosa, Jisung Park, and Onur Mutlu. 2020. CLR-DRAM: A Low-Cost DRAM Architecture Enabling Dynamic Capacity-Latency Trade-Off. (2020), 666–679.

[27] Krishna T Malladi, Ian Shaeffer, Liji Gopalakrishnan, David Lo, Benjamin C Lee, and Mark Horowitz. 2012. Rethinking DRAM power modes for energy proportionality. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 131–142.

[28] David Meisner, Brian T Gold, and Thomas F Wenisch. 2009. PowerNap: eliminating server idle power. In *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 205–216.

[29] Mike O'Connor, Niladrish Chatterjee, Donghyuk Lee, John Wilson, Aditya Agrawal, Stephen W Keckler, and William J Dally. 2017. Fine-grained DRAM: energy-efficient DRAM for extreme bandwidth systems. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 41–54.

[30] 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 (EuroSys)*. 1–14.

[31] Konstantinos Tovletoglou, Lev Mukhanov, Dimitrios S Nikolopoulos, and Georgios Karakonstantis. 2020. HaRMony: Heterogeneous-Reliability Memory and QoS-Aware Energy Management on Virtualized Servers. In *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 575–590.

[32] Malcolm Ware, Karthick Rajamani, Michael Floyd, Bishop Brock, Juan C Rubio, Freeman Rawson, and John B Carter. 2010. Architecting for power management: The IBM® POWER7™ approach. In *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 1–11.

[33] Donghong Wu, Bingsheng He, Xueyan Tang, Jianliang Xu, and Minyi Guo. 2012. RAMZzz: Rank-aware DRAM power management with dynamic migrations and demotions. In *International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*. 32:1–32:11.

[34] Tao Zhang, Ke Chen, Cong Xu, Guangyu Sun, Tao Wang, and Yuan Xie. 2014. Half-DRAM: A high-bandwidth and low-power DRAM architecture from the rethinking of fine-grained activation. In *ACM/IEEE International Symposium on Computer Architecture (ISCA)*. 349–360.

[35] Pin Zhou, Vivek Pandey, Jagadeesan Sundaresan, Anand Raghuraman, Yuanyuan Zhou, and Sanjeev Kumar. 2004. Dynamic tracking of page miss ratio curve for memory management. In *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 177–188.