Footprint-Based DIMM Hotplug

doi: 10.1109/TC.2019.2945562

Footprint-Based DIMM Hotplug

Shinobu Miwa, *Member, IEEE,* Masaya Ishihara, *No Member,* Hayato Yamaki, *Member, IEEE,* Hiroki Honda, *Member, IEEE,* Martin Schulz, *Member, IEEE*

Abstract—Power-efficiency has become one of the most critical concerns for HPC as we continue to scale computational capabilities. A significant fraction of system power is spent on large main memories, mainly caused by the substantial amount of DIMM standby power needed. However, while necessary for some workloads, for many workloads large memory configurations are too rich, i.e., these workloads only make use of a fraction of the available memory, causing unnecessary power usage. This observation opens new opportunities for power reduction by powering DIMMs on and off depending on the current workload. In this paper, we propose footprint-based DIMM hotplug that enables a compute node to adjust the number of DIMMs that are powered on depending on the memory footprint of a running job. Our technique relies on two main subcomponents—memory footprint monitoring and DIMM management—which we both implement as part of an optimized page management system with small control overhead. Using Linux's memory hotplug capabilities, we implement our approach on a real system, and our results show that our proposed technique can save 50.6–52.1% of the DIMM standby energy and the CPU+DRAM energy of up to 1.50 Wh for various small-memory-footprint applications without loss of performance.

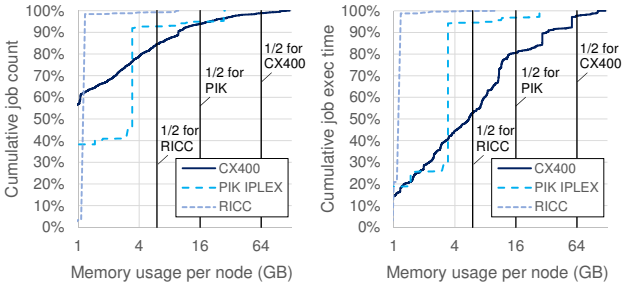
Index Terms—High performance computing, energy saving, DIMM, hotplug.

1 Introduction

The power consumed by High Performance Computing (HPC) systems has become a major constraint in their design and consequently power efficiency is a critical metric for their operation [1]. As we scale towards exascale systems, i.e., systems that can execute 10^{18} operations per second, 20-30 MW is generally seen as the practical limit for their deployment [2], [3], yet, already today the fastest system, the Summit system at the Oak Ridge National Laboratory, consumes 8.8 MW, while only delivering less than a fifth of an exaflop peak [4] and is even further away in terms of sustained performance. Closing this apparent gap requires continued efforts in both hardware and software techniques to increase power efficiency.

While most work focuses on hardware improvements [5], [6], [7] or software techniques optimizing computation [8], [9], [10], [11], [12], software techniques to reduce standby power consumed by main memory have not been focused on in the HPC community, yet, main memory consumes a significant fraction of the overall system power (up to 25% of system power in some cases [13]) and its power efficiency is seen by leading HPC architects as key challenge going forward [14]. Modern supercomputers have large amounts of main memory in the form of DIMMs using DRAM technology, often in the order of tens or even hundreds of GB per compute node. These DIMMs are a constant consumer of power—once a node is powered on—since all DRAM cells must be kept refreshed, whether they are used or not.

While such large memory configurations are driven by an important subset of HPC workloads, there are also equally many workloads for which they are too rich and hence underutilized. One example is Molecular Dynamics, which requires significant computation; for these codes problem sizes that would fill all memory would be intractable. As a result, many jobs on real HPC systems exhibit small memory footprints and hence leave memory unutilized, yet have to spend power also on the unused memory cells. For example, 99.8%, 94.5% and 99.2% of jobs use less than half of the node memory in the CX400, iDataPlex and RICC clusters, respectively (Figure 1 (a)). In addition, the total execution time of the above jobs accounts for 97.8%, 96.8% and 99.9% of the total time on the CX400, iDataPlex and RICC clusters, respectively (Figure 1 (b))[1].



(a) Job count (b) Execution time

Fig. 1. Memory usage of all jobs on a 1,476-node CX400 cluster at Kyushu University from Jan. 1 to Jun. 10, 2013; a 320-node IBM iDataPlex cluster at PIK (Potsdam Institute for Climate Impact Research) from Apr. 2009 to Jul. 2012 [15]; and the RICC cluster at RIKEN from May to Sep. 2010 [16].

In this paper, we propose an operating-system-level solution to the above problem. Our technique, called *footprint-based DIMM hotplug*, dynamically updates the DIMM config-

- *S. Miwa is with the Department of Computer and Network Engineering, the University of Electro-Communications, Tokyo, Japan, 182-8585. E-mail: miwa@hpc.is.uec.ac.jp*
- *M. Ishihara was with the University of Electro-Communications.*
- *H. Yamaki and H. Honda are with the University of Electro-Communications.*
- *M. Schulz is with Technical University of Munich*

Manuscript received September 14, 2018; revised XX XX, 201X.

1. We computed the memory usage of RICC with the memory request at job submission as actual measurements are not available.

uration during operation to achieve power efficiency using the existing DIMM hotplug functionality available in Linux. For this, each compute node monitors the memory footprint of the running job and then dynamically powers on DIMMs at runtime as they are needed. This optimized approach, transparently integrated into the OS's page management system, enables footprint-based DIMM hotplug to achieve large DIMM power savings at little to no performance loss. Footprint-based DIMM hotplug is built upon a Linux technology that allows system software to power down DRAM with firmware and hardware support. The idea of powering partial or entire DIMMs on/off has been proposed at various levels (e.g., application [17], architecture [18] and OS level [19]), but to our knowledge this paper is the first work to offer a real implementation of OS-level DIMM on/off with small control overhead, show the impact on performance and energy of a small-scale system, and estimate performance impact on supercomputing systems.

The main contributions of this paper are summarized below:

- **We perform a memory footprint analysis** of workloads on several production supercomputers and *found that memory is overprovisioned for many jobs*. Our analysis of the workloads on the CX400, PIK IPLEX and RICC systems unveiled that about 95% of jobs use less than half of the node memory.
- **We discuss a novel DIMM hotplug algorithm** called footprint-based DIMM hotplug, which enables compute nodes to adjust the number of powered-on DIMMs depending on the memory footprint of jobs.
- **We provide a lightweight implementation of our technique** based on the DIMM hotplug mechanisms available in Linux kernels 4.1.34 and newer.
- **We emulate power savings**, running our approach on a small-scale system with HPC workloads. To the best of our knowledge, there is no study that shows impacts of OS-level DIMM power management on real systems. Our experiments reveal that our technique can save half of DIMM standby power for various jobs without performance loss.
- **We estimate performance impact of our technique on supercomputing systems** based on real workload data. Our analysis shows that our technique has no degradation in the overall system performance.

The remainder of this paper is organized as follows. Section 2 gives a more detailed description of the DIMM standby power, followed by background on DIMM hotplug. Section 3 describes our footprint-based DIMM hotplug and Section 4 details its implementation. Sections 5 and 6 show the experimental setup and results, respectively. Section 7 lists related work and Section 8 presents our conclusions.

2 Power Saving with DIMM Hotplug

2.1 DIMM Standby Power

Despite advances in memory technologies, including recent advances in 3D-stacking memory, DIMMs are still the most common memory architecture in current computer systems. Figure 2 illustrates the memory architecture of a typical system, as it is also used as a node building block in large scale

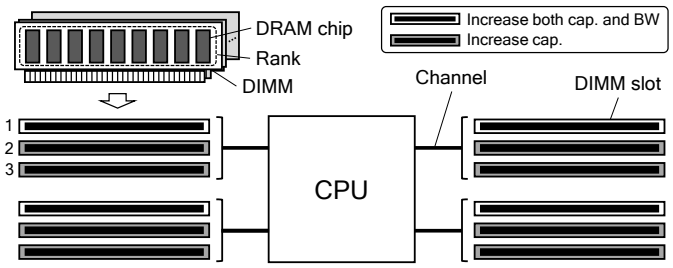


Fig. 2. Memory architecture.

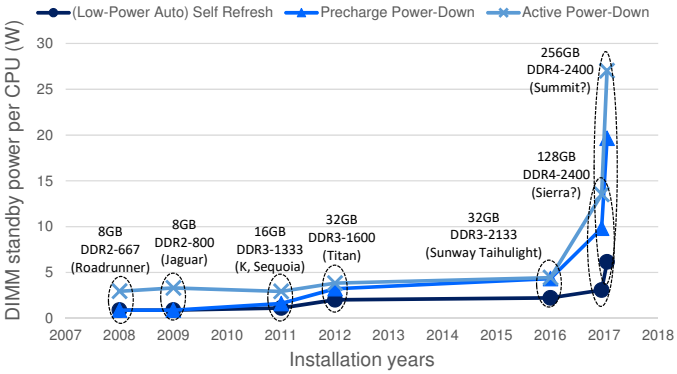


Fig. 3. Trend of DIMM standby power in supercomputing systems. We assume that both Sierra and Summit employs DDR4-2400 memories because the details of the memory configurations (e.g., clock frequency) are not available.

HPC systems. The CPU is surrounded by several DIMM slots, which can hold one DIMM each. Several DIMM slots form one group and each group is connected to the CPU using a separate memory channel. DIMM slots within a group must be populated in fixed order starting with slot 1.

A set of the DRAM chips integrated into one DIMM form a memory rank, i.e., they are wired to the same chip select signal and hence always accessed simultaneously. Multiple memory ranks can simultaneously serve data for multiple memory requests, even though they share external and/or internal memory buses (known as rank-level parallelism); consequently, installing multiple DIMMs in one DIMM group generally also increases the memory throughput as it increases the rank count, though its main benefit is the increase in memory capacity. However, it has been shown that this impact can be ignored on supercomputing systems, as an increase in rank count only has marginal impact due to the typically inefficient physical memory mappings [20], [21]. In summary, channel count is more important than rank count, as supercomputing systems offer sufficient memory bandwidth for running applications.

Although commercial DIMMs employ several techniques to lower power consumption, like self-refresh to reduce standby power [13], even idle DIMMs still consume a noticeable amount of power. For example, David et al. report that a 4 GB DDR3 RDIMM consumes 0.56 W or 0.13 W/GB even in low power modes and with self-refresh + register-off [22]. This is mainly caused by the required refresh operations needed to keep the data stored in the DRAM chips. To make matters worse, real servers are unlikely to use deeper power-down modes due to their large

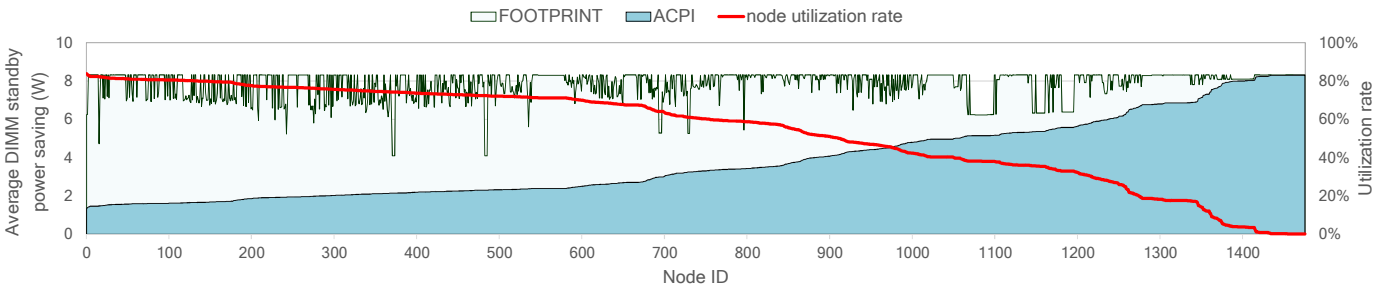


Fig. 4. Potential power savings using DIMM hotplug techniques in the CX400 cluster based on workload data from Jan. 1 to Jun. 10, 2013. We assume that each node has 16x 8 GB DDR3 memory, with each consuming a standby power of 1.04 W (= 8 GB × 0.13 W/GB), and that there was no time and energy overhead of powering DIMMs on/off.

exit latency, so that they are often faced with significant DIMM standby power [23]. In fact, our experiment with a power meter shows that an 8 GB DDR3L-1600 RDIMM used in our server consumes 0.91 W, which is almost equivalent to the power of the precharge power-down mode of 8 GB DDR3L-1600 DRAMs [24]. Thus, in this paper, we focus on power of DIMMs in precharge or active power-down mode, assuming real use cases of DRAM low power mode.

Figure 3 shows the trend of DIMM standby power in HPC systems. We computed the DIMM standby power of each system with its memory specification and Micron's DRAM datasheets [25], [26], [27], [28]. The data shows that, although the improvement in both DRAM architectures and manufacturing processes slightly reduces DIMM standby power, DIMM standby power per CPU in supercomputers keeps increasing.[2] This is caused by the huge memory capacities built into modern supercomputers as the core count per CPU increases in order to suit the class of memory hungry applications. In particular, DIMM standby power in the Sierra and Summit systems, which deploy 128 GB and 256 GB DDR4 memories per CPU, respectively, will consume 13.5 W and 27.0 W, respectively, with current operating approaches. This amounts to 10–20% of the maximum power of a modern server CPU.

2.2 DIMM Hotplug and Its Use for Power Savings

Memory or DIMM hotplug, which is supported in Linux starting at kernel version 2.6, is a technique to enable and disable DIMMs dynamically at runtime [29], [30]. When disabling a DIMM, all data on that DIMM is lost and hence all active pages allocated to that DIMM must be relocated to other DIMMs prior to disabling it. Linux's memory hotplug functionality supports this transparently [30]. Once disabled, if supported by both hardware and firmware, a disabled DIMM can be powered down, thereby eliminating its stand-by power.

To our knowledge, only two studies utilize memory hotplug to implement some form of power awareness: Chen et al. use it to save DIMM standby power for idle nodes [29], and a poster by Miwa et al. provides a preliminary study on saving the DIMM standby power during job execution [31].

TABLE 1
Performance of on-demand memory hot-add for ∗FFT.

Problem size	Baseline	On-demand memory hotadd	
	Exec. time (s)	Exec. time (s)	Slowdown (%)
20,000	4.66	5.61	20.4
40,000	19.2	23.2	20.8

Chen et al. base their approach on memory hotplug provided through ACPI in order to save DIMM standby power [29]. ACPI-based memory hotplug, which is partially supported in Linux starting in kernel version 3.11, shuts down a complete CPU with its connected DIMMs when the CPU becomes idle. This approach provides lower potential for saving DIMM standby power, especially in highly utilized systems, since the entire CPU has to be idle. Figure 4 shows the potential power savings of this ACPI-based memory hotplug in the CX400 cluster. As the figure shows, such ACPI-based memory hotplug can only achieve half of the DIMM standby power savings compared to our footprint-based DIMM hotplug approach.

Miwa et al. propose on-demand memory "hot-add" to control DIMM standby power based on observed memory usage at runtime [31]. Memory hot-add means enabling DIMMs within a running node. The disadvantage of this work is twofold: first, on-demand memory hot-add drastically degrades application performance due to increased I/O accesses. Table 1 exemplifies this. The on-demand memory hot-add temporarily swaps out pages to storage when memory usage exceeds available memory capacity, so that applications show the performance degradation of up to 20.8%.[3] Second, the authors do not show the impact of on-demand memory hot-add on DIMM and node power.

3 Footprint-Based DIMM Hotplug

We propose *footprint-based DIMM hotplug*, a novel technique to reduce DIMM standby power in HPC systems, which provides a more dynamic and finer grained solution compared to previous techniques, and we demonstrate it using

2. DIMM standby power per CPU is appropriate metrics to roughly estimate the impact of DIMM standby power on power of compute nodes where CPU jobs are executed. Some supercomputers shown in Figure 3 employ GPU, but we can ignore its power in such compute nodes because it is relatively small due to the low power state.

3. Our experimental result of an input of 20,000 is slightly different from that shown in the poster paper [31] because our daemon program halts the application processes for 200 ms to measure performance overhead of physically powering on DIMMs, which was ignored in the prior work.

an implementation on a real system. We monitor an application's memory footprint and adjust the system's DIMM configuration to reduce its power needs by only powering on needed DIMMs. This does not require any modifications to applications and therefore offers a practical solution that can easily be added into existing production systems.

3.1 Design

Both components, the monitoring and the adjustment of the DIMM configuration, are executed transparently through modifications to the memory management in the operating system. Monitoring is implemented by extending the page manager and using it to trace which pages are accessed. This provides us with a fast and low overhead mechanism to track an application's memory footprint. Based on the information gathered using this step, which is executed continuously throughout the runtime of the application, we determine an application's memory needs and with that how much memory is actually used and hence must be turned on. The latter we accomplish by using the existing DIMM hotplug APIs in Linux—with the matching firmware and hardware support—to, at startup, first "logically" remove all DIMMs and then only turn on the needed DIMMs in the system as the application executes.

3.2 Performance Considerations

The central design challenge for footprint-based DIMM hotplug is to minimize or eliminate any impact on application performance, since overheads would both increase overall energy usage caused by longer runtimes, thereby canceling out the benefits of footprint-based DIMM hotplug, and reduce the acceptance of the proposed method by end users, who ultimately only care about the "amount of science" achieved.

In particular, we must consider the following main factors that impact application performance: (i) the overhead caused by monitoring the memory footprint at runtime, (ii) the overhead caused by dynamically changing DIMM configurations, (iii) how often we change DIMM configurations, and (iv) the impact on available memory bandwidth by reducing DIMMs.

Overhead from (i) and (ii) directly stems from monitoring and adjusting the application's exception behavior. (i) is implemented through some minimal bookkeeping in the OS's page fault handler, which is insignificant compared to the large fault latencies on today's systems. Enabling DIMMs (ii) is slightly costly, but we can minimize its overhead by executing it very infrequently (iii), especially for large production applications. Low overheads due to (iii) is further ensured by our decision to keep DIMMs active until the end of a job's execution once they are powered up, even if a job deallocates a part of the allocated memory region and a large number of memory cells become unused. While this design decision reduces some of the opportunities for DIMM power saving, in reality it has only very limited practical impact as most HPC applications allocate their working set upfront (e.g., in the form of large arrays or graph data structures), but rarely release them before the end of the execution[4].

To prevent (iv), footprint-based DIMM hotplug only shuts down DIMMs without impact on memory bandwidth. Specifically, we limit our approach to DIMMs plugged into the second or subsequent slots as this doesn't impact memory bandwidth, as described in Section 2.1. Consequently, the available memory bandwidth can be maintained even if the power-on DIMM count changes. Despite the limited positions of DIMMs available to be powered down, our technique still has enough flexibility to save DIMM standby power as standby power of second or subsequent DIMMs accounts for more than half of the entire DIMM standby power.

4 Implementation

We begin to explain our implementation using one particular memory configuration, and then generalize it.

4.1 Initial Assumptions and Constraints

To explain the concepts behind our approach, we first assume the memory system configuration shown in Figure 5: each node consists of one CPU socket with 4 memory channels and two DIMMs installed in each memory channel. Note that only four DIMMs are plugged into the second DIMM slots and hence are candidates to be powered off.

Our technique changes the power state of the second DIMM in all groups together, as partial shutdown and restart of individual DIMMs in the second DIMM slots is typically unsupported, including on our hardware. Most memory controllers are optimized for and only allow a balanced DIMM configuration both for simplicity and since unbalanced for memory channels often lead to reduced memory throughput [32], [33].

4.2 Power Modes

Conceptually, in our approach the memory system has two power modes, as shown in Figure 5. In the *active* mode all DIMMs are powered up and available for the page management system to use, while in the *sleep* mode all DIMMs in the second slots are powered down and unavailable. In the latter case, the total memory capacity of a node is therefore cut in half (assuming all DIMMs have the same memory capacity). Since all nodes keep one powered-up DIMM for each channel, the proposed technique does not impact the maximum memory bandwidth of a node, even during the sleep mode[5].

4. Disabling DIMMs during job execution is costly as it requires page relocation as described in Section 2.2. For example, our experimental result shows that copying 4 GB to a DIMM installed in another channel via the DDR3-1333 interface takes 0.90 second with additional power of 15.9 W. Production supercomputers have fewer opportunities to benefit from this dynamic memory reconfiguration as jobs that both use more than half of the node memory and dynamically release their working sets make up less than 5.5% of the jobs executed.

5. Memory interleaving, which is widely used in production systems, is available for systems that employ our footprint-based DIMM hotplug. In fact, our experimental system shown in Section 5 enables memory interleaving. A page is interleaved to multiple DIMMs connected to different channels and the DIMMs therefore can serve the sequential data of the page in the peak memory bandwidth even during the sleep mode.

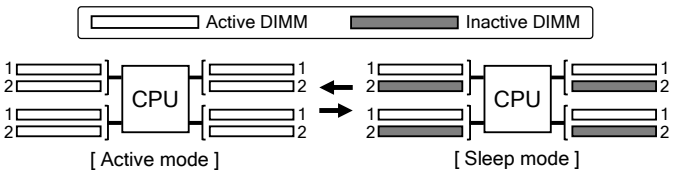


Fig. 5. Power mode in a memory system that has 4 memory channels, with each having 2 DIMMs.

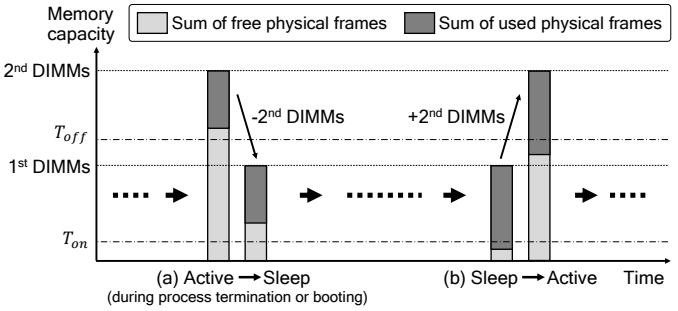


Fig. 6. Mode transition based on free physical frame count.

4.3 Power Mode Transitions

We dynamically switch between the two power modes according to the number of free physical page frames left in a node. A physical page frame is a contiguous piece of physical memory that can be mapped to a virtual page in order to make it accessible to the application. The number of physical page frames that are used or unused, i.e., page frames mapped or not mapped to virtual memory, is tracked by the virtual memory management system. If a node has a large number of free physical frames left, it means that the applications running on that node only uses a small amount of memory; in this case, our technique turns off all second DIMMs. On the other hand, we turn on all of the second DIMMs when there are few free physical frames left in a node.

We use two thresholds, called *power-on* and *power-off*, to determine the right time for the transition described above. As shown in Figure 6, (a) the memory system switches from the active mode to the sleep mode when the number of free physical frames (N_{free}) exceeds the power-off threshold (T_{off}), while (b) the memory system returns to the active mode when N_{free} is less than or equal to the power-on threshold (T_{on}). In order to reduce overhead, though, N_{free} is compared with T_{on} when the page allocation happens, but not with T_{off} at every page deallocation. Instead, we perform the comparison of N_{free} with T_{off} at process termination to reset the number of used DIMMs and minimize the used standby power between jobs and for new jobs. This is generally sufficient for HPC applications, as described in Section 3. We do, however, execute the above comparison at page deallocation when processes are not running, yet (i.e., during boot operations), as this cannot impact application performance. The optimal values of T_{on} and T_{off} highly depend on both system configuration and usage and should be set by system administrators based on results from operational tests conducted during the acceptance testing of a production system.

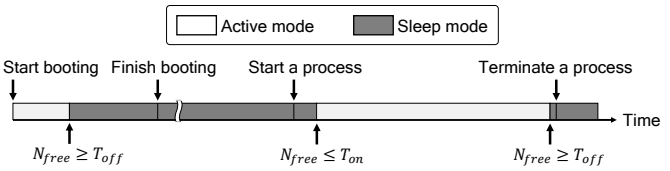


Fig. 7. Time line of power mode in the memory system.

Figure 7 shows a sample timeline of the memory system and its mode transitions: all DIMMs are powered up by the firmware at boot time, and the OS kernel then conducts boot operations in the active mode. Next, the kernel frees some physical frames after the boot completes, resulting in the increase of N_{free}. If $N_{free} \geq T_{off}$, the system enters the sleep mode and stays there until sufficient memory allocations are performed. After a job with a large memory footprint is launched on the node, the kernel starts to allocate the virtual pages of the job using the free physical frames in the DIMMs that are powered on. N_{free} is gradually reduced while the job is running, and the memory system then turns to the active mode when $N_{free} \leq T_{on}$. The kernel frees all physical frames used for the job at the process termination, which causes N_{free} to increase. Once the threshold is reached, the memory system goes back to the sleep mode during the termination process.

4.4 Implementation Details

We implement our technique in the page management system of a Linux Kernel version 4.1.34. Since Linux is the most common operating system in HPC systems (100% of the top500 supercomputers run on Linux as of November 2018 [4]), we believe that many systems will benefit from our implementation, assuming they support and enable memory hotplug.

In Linux terminology, disabling a DIMM within a running node is called *memory hot-remove*, while enabling the DIMM is called *memory hot-add*. This section discusses the implementation of these operations in detail.

4.4.1 Hot-Remove at Process Termination

When a process terminates, the page management system deallocates all physical frames from the virtual pages of the process. The above operation is conducted in the function `exit_mm()`. Since `exit_mm()` is always called from the function `do_exit()`, we append the code needed for monitoring N_{free} and possibly power off DIMMs to that function.

Algorithm 1 shows the pseudocode of memory hot-remove executed in `do_exit()`. First, the current power mode of the memory system is checked after `exit_mm()` finishes (Lines 1–2). If the power mode is active, our algorithm computes N_{free} by invoking the function `global_page_state()` and then compares it with T_{off} (Lines 3–4). When $N_{free} \geq T_{off}$, the algorithm calls the primitive function of memory hot-remove, `offline_pages()`, with the inputs of the start physical frame number and the physical frame count of second DIMMs (Lines 5–7). Finally, the power mode changes into the sleep mode (Line 8). These procedures are executed in

Algorithm 1 Pseudocode of memory hot-remove

```
1: PowerMode ← current power mode
2: if PowerMode = active then
3:    N_free ← global_page_state(NR_FREE_PAGES)
4:    if N_free ≥ T_off then
5:       StartFN ← start frame number of 2nd DIMMs
6:       FrameCount ← frame count of 2nd DIMMs
7:       offline_pages(StartFN, FrameCount)
8:       PowerMode ← sleep
9:    end if
10: end if
```

Algorithm 2 Pseudocode of memory hot-add

```
1: PowerMode ← current power mode
2: if PowerMode = sleep then
3:    N_free ← global_page_state(NR_FREE_PAGES)
4:    if N_free ≤ T_on then
5:       StartFN ← start frame number of 2nd DIMMs
6:       FrameCount ← frame count of 2nd DIMMs
7:       online_pages(StartFN, FrameCount, ...)
8:       PowerMode ← active
9:    end if
10: end if
```

an atomic manner to avoid the redundant memory removes caused by multiple processes terminating simultaneously.

4.4.2 Hot-Remove at Boot Time

During boot operations, some memories are allocated to the kernel via the functions `kmalloc()` and `vmalloc()`, and such memories are freed by the functions `kfree()` and `vfree()`. The functions `vmalloc()` and `vfree()` use a paging system to manage the memory, but `kmalloc()` and `kfree()` do not. In order to gain control to implement our technique, we add the memory hot-remove function to `vfree()`. The procedure for memory hot-remove in `vfree()` is similar to that in `do_exit()`. More specifically, we insert the code listed in Algorithm 1 at the end of `vfree()`.

4.4.3 Memory Hot-Add

Many free physical frames are allocated to the virtual pages of a process when the process calls `_alloc_pages_nodemask()` via `alloc_pages()`. We therefore added the code of memory hot-add into `alloc_pages()`.

Algorithm 2 shows the pseudocode of memory hot-add executed in `alloc_pages()`. Similar to Algorithm 1, Algorithm 2 first checks the current power mode of the memory system (Line 1). If the power mode is sleep (Line 2), the algorithm calculates N_{free} (Line 3). When $N_{free} \leq T_{on}$ (Line 4), the algorithm calls the primitive function for memory hot-add, `online_pages()`, with some information for the second DIMMs (Lines 5–7). Finally, the power mode is changed to the active mode (Line 8), and `alloc_pages_current()` is invoked. In contrast to Algorithm 1, the above procedures do not have to be executed atomically since mutual exclusion has already been implemented within `online_pages()`. Consequently, the redundant memory hot-add never occurs if multiple processes simultaneously call `online_pages()`.

4.5 Extensions to the Other Configurations

So far we have only discussed our proposed technique under the assumption of a specific configuration (i.e., 1 CPU socket, 4 channels and 2 DIMMs per channel). In the following we show how we can generalize our approach to other configurations.

4.5.1 Multiple CPU Sockets

Our technique can also be used on nodes that consist of multiple CPU sockets. The page management system controls all physical frames of the DIMMs connected to any CPU in the entire node, so that the proposed technique can deal with the DIMMs attached to the local CPU as well as the DIMMs attached to other CPU sockets. That is, second DIMMs connected to all CPU sockets are simultaneously powered off if the total number of free physical frames (N_{free}) exceeds T_{off}, while these DIMMs are simultaneously powered on if $N_{free} \leq T_{on}$. Our technique always balances the number of available DIMMs on all CPU sockets as this configuration has been shown to be the most efficient in most cases [32], [33].

The only concern is that DIMM hotplug could potentially interfere with NUMA-aware page allocation implemented in Linux. Some modifications (e.g., counting the number of free physical frames per socket) may be needed for our switching algorithm to benefit from the NUMA-aware page allocation. We will further investigate this issue in the future.

4.5.2 Different Numbers of Channels

Our approach naturally expands to nodes with different numbers of channels. If each channel has multiple DIMMs, the DIMMs in the second or subsequent slots can be controlled by the proposed technique.

4.5.3 Different Numbers of DIMMs per Channel

We can easily extend our proposed technique to nodes that employ 3 DIMMs per channel. Such nodes can define three power modes (e.g., active, sleep and deep sleep), which correspond to the cases of 3, 2 and 1 active DIMMs per channel, respectively. In this scenario, we need at least three thresholds (e.g., T_{on}, T_{off1} and T_{off2}) to switch between the three power modes. If N_{free} exceeds T_{off2} in the active mode, the memory system first enters the sleep mode. The memory system in the sleep mode then turns into the deep sleep mode if $N_{free} \geq T_{off1}$. On reverse, the memory system in a deep sleep mode goes back to active modes in sequence if $N_{free} \leq T_{on}$.

On the downside, our technique does not work for nodes with only one DIMM per channel, since we aim to save DIMM standby power while maintaining memory bandwidth. Memory bandwidth is a key factor in the performance of many HPC applications and hence we purposely took an approach that leaves one active DIMM in each channel when powering DIMMs off, maintaining constant bandwidth. Even, though, some present supercomputers only have one DIMM per channel, our approach opens new opportunities for installing multiple DIMMs per channel to save the operational cost while maintaining the performance of memory-hungry applications.

Nevertheless, powering off first DIMMs remains a promising option, especially for workloads tolerable to decreased memory bandwidth. We found ∗DGEMM as one such example, which shows the performance loss of only 0.35% by removing 6 DIMMs from full DIMMs in our experimental system shown in Section 5 (i.e., disabling 2 out of 4 channels). We will explore this option in future work.

4.5.4 New Memory Technologies

Some state-of-the-art computers adopt HBM (High Bandwidth Memory) [34] or HMC (Hybrid Memory Cube) [35] for increased memory bandwidth. Unfortunately memory hotplug is not available for HBM and HMC at present, but it is possible that the power of DRAM stacks in HBM and HMC will be controlled with memory hotplug in the future. In addition, memory hotplug may possibly have the capability to manage the power of HMC stacks connected in a daisy-chain manner. Our technique will be available in HBM and HMC once these architectures and the corresponding Linux kernels support memory hotplug.

Many researchers expect that the future supercomputing systems will use NVMs (Non-Volatile Memories), such as PCM (Phase Change Memory), in the memory systems [36], [37], [38]. NVMs exhibit very low standby power due to their non-volatility, and hence powering off NVM modules has only a marginal benefit in terms of power savings. However, even with NVMs added, DRAM will remain an important part of system, since—for the foreseeable future—a complete replacement of DRAMs by NVMs is unrealistic, mainly caused by the long write latencies of NVMs. For example, PCMs need 12x time and 43x energy for write accesses when compared to DRAMs [37], [38]. The standby power of the remaining DRAMs can then again be controlled by our technique.

5 Experimental Setup

We installed both the original and our modified version of the Linux Kernel 4.1.34 in a 4-node Xeon server (PowerEdge r620). Each node has one Xeon E5-2630L CPU and 32 GB DDR3L memory (if all DIMMs are powered up). The core count, the clock frequency and the TDP of this CPU are six, 2.0 GHz and 60 W, respectively. These values are similar to those of a SPARC64 VIIIfx CPU used in the K computer [39]. The memory system has eight 4 GB DDR3L-1333 RDIMMs and 4 memory channels (two DIMMs per channel). Thus, the number of memory-system power modes is two (active and sleep). The transition from active to sleep mode reduces the memory capacity to 16 GB and also cuts the needed DIMM standby power in half. Each node is connected to a network switch (PowerConnect 5548) using Gigabit Ethernet.

As discussed in Section 4, our proposed technique is based on two thresholds (T_{on} and T_{off}) to switch between the two power modes. We tested various pairs of thresholds, but we were not able to see any difference in behavior when varying T_{off}, and hence we only show the experimental results of $T_{off} = 24$ GB in the next section. For T_{on}, on the other hand, we vary the threshold from 1 GB to 8 GB. The complete experimental parameters are summarized in Table 3.

We note that we have to take impact of a *kswapd* daemon into account when configuring values of these parameters. In order to prevent *kswapd* from freeing pages additionally, we need to set a value larger than a low watermark (3.39 MB in our experimental system) to Ton. In contrast to this, we do not need to pay special attention to a value of Toff. This is because if Ton is appropriately configured, *kswapd* is in sleep state whenever our technique starts to power DIMMs off, and the running application is never affected by *kswapd*.

Since the firmware/hardware combination of our server currently does not allow an actual hot plug operation (i.e., our server can conduct only logical hot plugging including page relocation), we emulate the effects caused by physical hot plugging. In order to truthfully capture the overhead caused by physically powering DIMMs on/off, we insert the function `udelay()` at the point where we call the respective primitive functions for memory hotplug operations, and then test our benchmark programs with various delays. We choose the inputs for `udelay()` (denoted as D) as follows: the datasheet [24] shows that DDR3L SDRAM takes up to 200 ms for the initialization sequence (i.e, stabilizing power supply), which means that powering up a DIMM takes 200 ms or less. For this reason, D is varied between 0.2 and 200 ms as shown in Table 3, and we conservatively estimate that the delay of 200 ms is the most realistic.

Further, we estimate DIMM standby power. Powered up DIMMs consume standby power depending on their power mode (i.e., part of powered up circuit within a DIMM), while fully powered down DIMMs do not consume standby power. We estimate the standby power of a DIMM as being equal to the power of the active power-down mode of a 4 GB DDR3L-1333 DRAM (i.e., 0.63 W [24]), which accounts for 38.8% of the power of DRAM that shows the CX400-like utilization rate, according to the DDR3L DRAM power calculator provided by Micron [40]. Therefore, we can estimate that power of the active power-down mode is about 9.7% of the system power, if DRAM consumes 25% of the system power [13]. However, this number highly depends on node architecture, DRAM technology and utilization rate. In fact, we can save 5.9% of the overall node power for ∗FFT by halving the number of DIMMs installed in our experimental node, when compared to the execution on full DIMMs. We also assume that the standby power of a DIMM during mode transition is the same in the full case.

For all experiments, we run our benchmarks on our Linux server node and compute DIMM standby energy by using the above assumptions combined with memory usage traces acquired during the run, which we achieve by executing the Linux command `free` per second as a background process.

We use ten diverse MPI-parallel applications (4 computation-intensive and 6 memory-intensive), see Table 2. We test these programs with various inputs to model jobs with different memory footprints, as the effectiveness of the proposed technique can mainly be characterized by memory footprints. Four of the ten programs (`PTRANS`, `*DGEMM` (i.e., star DGEMM), `*FFT` (i.e., star FFT) and `*STREAM` (i.e., star STREAM)), which are selected from the HPC Challenge Benchmark suite [41], are simple and stress particular properties, while the rest (`HACC`, `NAMD`, `QBOX`, `UMT2013`, `BDAS` and `GRAPH`), which are selected from the

TABLE 2
Benchmark programs.

Name	Code description	Type	Inputs (single)	Inputs (multi)	Footprints
PTRANS	Array transpose	memory	Ns: 10-50K	Ns: 20-60K	S to L
∗DGEMM	Matrix-matrix multiplication	computation	Ns: 10-50K	Ns: 20-60K	S to L
∗FFT	Fast Fourier transform	computation	Ns: 10-50K	Ns: 20-60K	S to L
∗STREAM	Stream memory access	memory	Ns: 10-50K	Ns: 20-60K	S to L
HACC	Hardware accelerated cosmology code	computation	np: 20, 40	np: 40, 60	XS
NAMD	Classic molecular dynamics	computation	apoa1, kv1.2	apoa1, kv1.2	S
QBOX	Quantum molecular dynamics	memory	128-384 atoms	256-768 atoms	S to L
UMT2013	Unstructured mesh deterministic radiation transport	memory	-	Zone: 12x12x24, 12x24x24	M to L
BDAS	Big data analysis with machine learning	memory	0.8–3.2M rows	0.8–3.2M rows	S to L
GRAPH	Graph500 benchmarks	memory	2^{23}–2^{25} vertices	2^{23}–2^{25} vertices	S to L

TABLE 3
Hotplug parameters.

Name	Remarks
Power-on threshold (T_{on})	1, 2, 4, 8 GB
Power-off threshold (T_{off})	24 GB
FW and HW overhead (D)	0.2, 2, 20, 200 ms

CORAL, CORAL-2 and Graph500 Benchmark suites [42], [43], [44], represent more realistic workloads. These applications use the following inputs that affect memory footprint: matrix size (Ns); number of particles per dimension (np); names of proteins (`apoa1` and `kv1.2`); number of atoms; number of zones per MPI ranks; number of rows; and number of vertices. The `Inputs` columns present the values of the parameters used. The rightmost column in Table 2 represents the memory footprint size of each application using the above values. All programs are compiled for mpich2-1.4.1p1 using gcc-4.8.0. We launch 4 MPI processes per node in each run and the number of OMP threads was 1. Intel Hyper Threading and Turbo Boost technologies were disabled following common practice in many HPC centers to ensure repeatable program behavior. Further, we run all experiments on dedicated sets of nodes, as usual for current HPC systems.

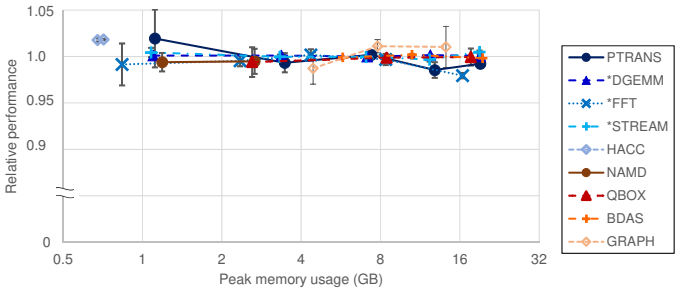


Fig. 8. Performance of footprint-based DIMM hotplug on a single node.

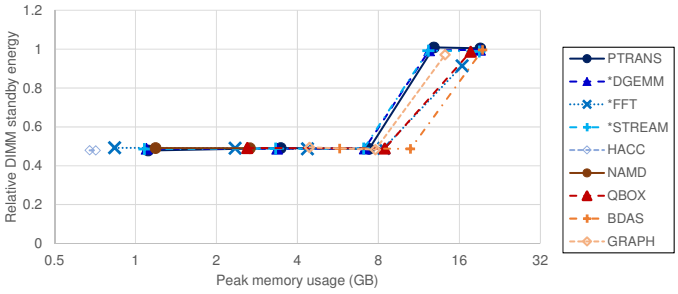


Fig. 9. DIMM standby energy of footprint-based DIMM hotplug on a single node.

6 Experimental Result

First we provide results from a single node, followed by a comprehensive sensitivity study, and experiments that show the performance and energy of footprint-based DIMM hotplug on multiple nodes. Finally, we show the impact of footprint-based DIMM hotplug on real workloads.

6.1 Performance and Energy

Figures 8 and 9 show the performance and DIMM standby energy of our approach on a single node. The vertical axes represent the relative performance or DIMM standby energy running with the modified kernel compared to running the original kernel, while the horizontal axes represent the peak memory usage of the tested applications. Each of the nine dotted lines represents the result from one of the nine tested programs with varying problem sizes. More specifically, we use Ns =10,000–50,000, np =20–40, 128–384 atoms, 0.8–3.2M rows and 2^{23}–2^{25} vertices for the HPCC applications, `HACC`, `QBOX`, `BDAS` and `GRAPH`, respectively. In addition, we use the `apoa1` and `kv1.2` inputs in the `small` dataset for `NAMD`. We did not use `UMT2013` for the single-node experiment as it requires more MPI processes than we had cores available. We set T_{on} to 4 GB and D to 200 milliseconds. We run each program five times and show average results.

Figure 8 shows that footprint-based DIMM hotplug only experiences a small slowdown caused by the control overhead; for many applications we are within 2% of the original kernel. However, we also observed a performance difference of 2% comparing the five trials with one kernel, as shown in the error bars that represents standard deviation in the five trials. This graph indicates that the control overhead of our proposed technique is typically hidden within OS jitter.

Figure 9 illustrates that our approach saves DIMM standby power for jobs with small memory footprints, but (as expected) does not impact the power usage for jobs with large ones. For example, the proposed technique can save 50.6–52.1% of the DIMM standby energy for all programs

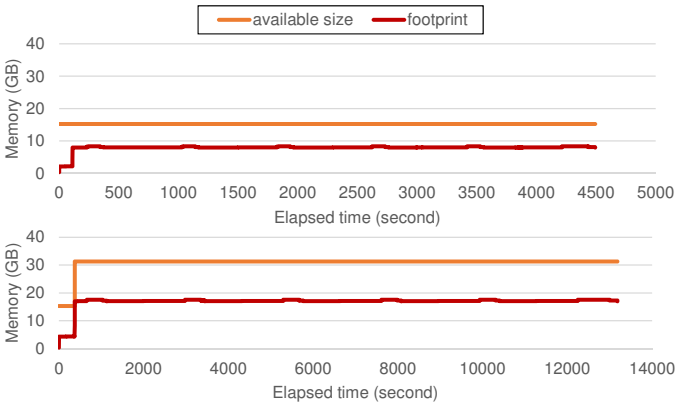


Fig. 10. Memory usage over time (top: QBOX with 256 atoms, bottom QBOX with 384 atoms).

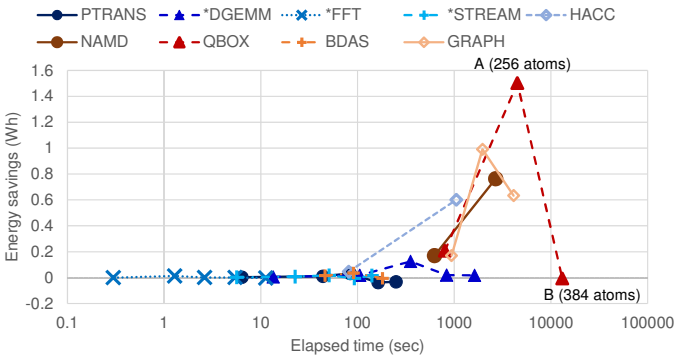


Fig. 11. Energy saving of a single node.

with a peak memory usage of less than 10.5 GB. On the other hand, our technique shows little energy saving if the peak memory usage of a program exceeds 16.4 GB.

Our technique prevents some applications with moderate memory footprints (e.g, ∗DGEMM at the peak memory usage of 12.4 GB) from saving the DIMM standby energy, as it is designed to avoid page relocation during the job execution. Recall that when an application is likely to run out of the memory available in the sleep mode (i.e., $N_{free} \leq T_{on}$), our technique increases the available memory and then keeps the increased memory at the end of the application. However, we believe that our algorithm is acceptable to production systems, because jobs with moderate memory footprints are rarely executed in real supercomputers (e.g., only 0.46% of jobs use a node memory amount of 12–16 GB in the PIK iDataPlex cluster).

Figure 10 shows the memory usage over the runtime of QBOX. The top figure represents a run with 256 atoms, while the bottom one represents a run with 384 atoms. The memory footprint of QBOX with 256 atoms never exceeds 12 GB ($= 16$ GB $- T_{on}$), so that the available memory size stays around 16 GB during the execution. On the other hand, the memory footprint of QBOX with 384 atoms hits the power-on threshold at around 400 seconds. Our page management system immediately doubles the available memory size at that time and then keeps the increased memory until the end of the application.

Figure 11 shows the node energy saving of footprint-based DIMM hotplug. We focus only on the package and

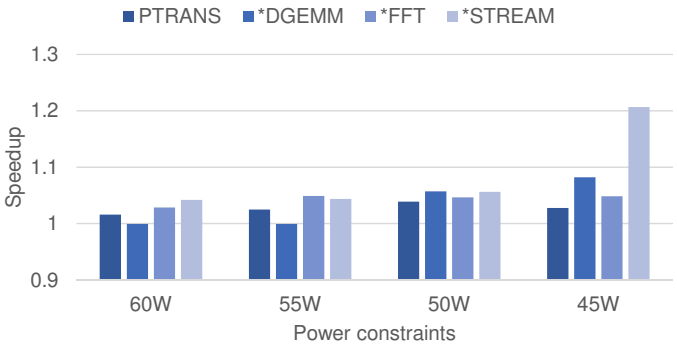


Fig. 12. Speedup in power-capped scenarios.

DRAM power reported by RAPL [45] because the power of the other components within a node highly depends on the node architecture. Since RAPL does not model the power of DRAM standby modes [45], we computed the node energy of the proposed technique by subtracting the DIMM energy saving shown in Figure 9 from the amount of energy measured with RAPL. Note that our experiment includes the impact of our software modification on both power of CPU and dynamic power of DRAM, which are modeled by RAPL.

The figure shows that the longer an application with a small memory footprint runs, the more our proposed technique saves energy. For example, the proposed technique saves the energy of 1.50 Wh for QBOX with 256 atoms (at point A). This amount of energy saving is larger than that of QBOX with 384 atoms (at point B), as QBOX with 384 atoms has little chance to power the second DIMMs off due to its large memory footprint.

From Figures 8 to 11, we can conclude that the proposed technique has a good ability to control the standby power of DIMMs matching the memory footprints of the applications.

Additionally, our footprint-based DIMM hotplug has the ability to improve performance of power-constrained HPC systems [5]. Figure 12 shows the relative performance of the HPCC applications with Ns=30,000 running on our modified kernel compared to running on the original one. We selected the best combination of package and DRAM power budgets by using a brute-force search. The figure shows that our technique can improve the performance of memory-intensive applications in stringent power-constraint scenarios. E.g., ∗STREAM shows a speedup of 1.21x at a power budget of 45 W.

6.2 Sensitivity Studies

Figure 13 shows the performance sensitivity of footprint-based DIMM hotplug against T_{on}. D is 200 ms. The top figure shows that a larger T_{on} is more helpful to applications with large memory footprints (16.4 GB). We emphasize that this feature clearly appears in case of applications with a short execution time. As shown in the bottom figure, the difference in T_{on} has no impact on the performance of QBOX, which runs for a long time.

Figure 14 shows the performance sensitivity of the proposed technique against D. T_{on} is 4 GB. There is no remarkable difference in application performance between different values for D, as the value of D (up to 200 ms) is two or more

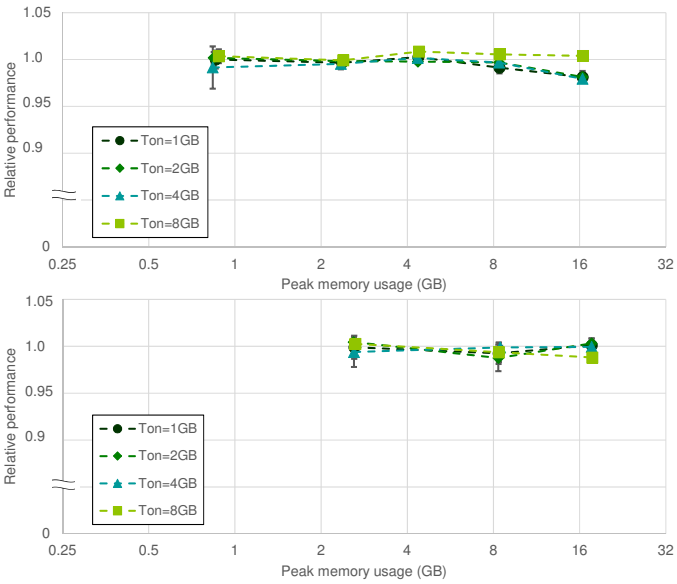


Fig. 13. Senstivity of footprint-based DIMM hotplug against power-on thresholds (top: ∗FFT, bottom: QBOX).

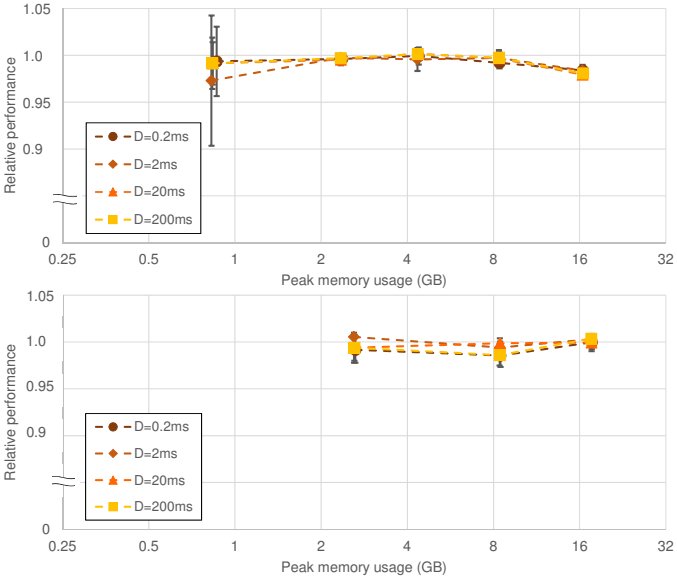


Fig. 14. Sensitivity of footprint-based DIMM hotplug against firmware and hardware overheads (top: ∗FFT, bottom: QBOX).

orders of magnitude shorter than the execution time of the applications (a few seconds or longer). Further, these two figures show that the firmware and hardware overheads are not dominant in the performance of footprint-based DIMM hotplug. Reducing the overhead of operating system kernels (e.g., the cost to stop page management systems for the page relocation) is more important to increase the performance.

6.3 Multi-Node Experiments

We tested our proposed technique with all four nodes; Figures 15 and 16 show the results. T_{on} was 4 GB and D was 200 ms. We used Ns =20,000–60,000, np =40–60, 256–768 atoms, 0.8–3.2M rows and 2^{23}–2^{25} for the HPCC applications, `HACC`, `QBOX`, `BDAS` and `GRAPH`, respectively,

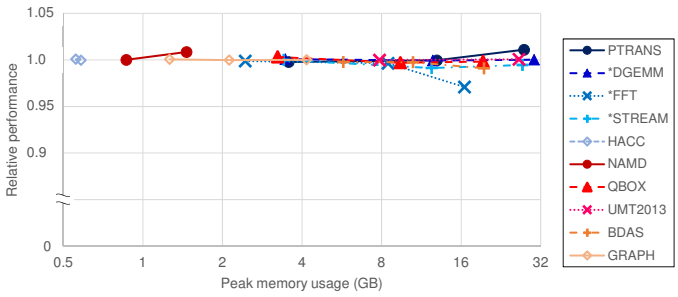


Fig. 15. Performance of footprint-based DIMM hotplug on 4 nodes.

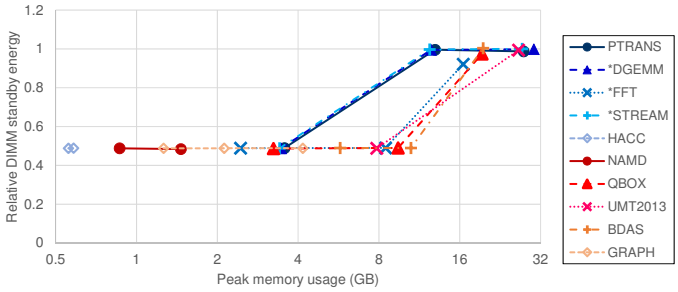


Fig. 16. DIMM standby energy of footprint-based DIMM hotplug on 4 nodes.

and apoa1 and kv1.2 inputs for `NAMD`. For `UMT2013`, the processor block was 4x2x2 and we tested with 12x12x24 and 12x24x24 zones. Similar to Figure 8, Figure 15 shows that our technique does not affects the performance of many applications, as our proposed technique has no additional overhead for controlling DIMMs in multi-node systems.

Figure 16 shows the DIMM standby energy saving of the proposed technique. As shown in the figure, the proposed technique can save 51.1–51.6% of the DIMM standby energy if the peak memory usage per node is less than 10.0 GB. On the other hand, the proposed technique has little ability to save the DIMM standby energy if the peak memory usage per node exceeds 12.4 GB. This result is quite similar to Figure 9, as the memory usage of these applications is well-balanced between nodes and all nodes therefore show an almost identical power mode, despite that they independently monitor and control their own memory systems.

6.4 Impact on Real Workloads

Finally, we estimate impact of our footprint-based DIMM hotplug on performance of production supercomputing systems. Since our approach needs to modify Linux kernels and memory configurations, real experiment of our approach on a production system may cause some operational issues. Instead, we use three real workloads shown in Figure 1 and a performance model for this experiment.

Performance of a job under footprint-based DIMM hotplug ($P_{hotplug}$) can be defined as follows.

$$P_{hotplug} = P_{org} + P_{overhead} \quad (1)$$

where P_{org} and $P_{overhead}$ represent the original execution time of the job and the performance overhead of powering DIMMs on/off, respectively. $P_{overhead}$ is zero if the peak

TABLE 4
Performance of footrpint-based DIMM hotplug for real workloads.

CX400				
Class	# of jobs [%]	$\sum P_{org}$ (s)	$\sum P_{hotplug}$ (s)	SD
S	61,048 [98.7]	1.40×10^{9}	1.40×10^{9}	1.00
LS	27 [4.37×10^{-4}]	530	540.8	1.02
LL	760 [0.0123]	5.95×10^{7}	5.95×10^{7}	1.00
Total	61,835	1.46×10^{9}	1.46×10^{9}	1.00
PIK IPLEX				
Class	# of jobs [%]	P_{org} (s)	$P_{hotplug}$ (s)	SD
S	688,405 [94.4]	9.65×10^{10}	9.65×10^{10}	1.00
LS	2,662 [3.65×10^{-3}]	29,356	30,421	1.04
LL	37,804 [0.0519]	3.28×10^{9}	3.28×10^{9}	1.00
Total	728,871	9.98×10^{10}	9.98×10^{10}	1.00
RICC				
Class	# of jobs [%]	P_{org} (s)	$P_{hotplug}$ (s)	SD
S	440,918 [98.5]	9.36×10^{10}	9.36×10^{10}	1.00
LS	397 [8.87×10^{-4}]	3,332	3,481	1.05
LL	6,479 [0.0145]	7.98×10^{8}	7.98×10^{8}	1.00
Total	447,794	9.44×10^{10}	9.44×10^{10}	1.00

memory usage of the job is lower than the half memory capacity minus T_{on}. Otherwise, we assume a uniform overhead of 400 ms (each 200 ms for turing DIMMs on and off, respectively) for the simplicity.

To make an effect of our approach clear, we classify the jobs included in the workloads into the following three types: **S**, **LS** and **LL**. **S** represents jobs that have small memory footprints, and our approach does not need to switch power mode for the jobs. Both **LS** and **LL** mean jobs that have large memory footprints, and our approach therefore powers DIMMs on/off during the execution of the jobs. The difference between **LS** and **LL** are length of execution time (i.e., short or long). We assume that memory footprints per node less than 28 GB, 12 GB and 2 GB are small for CX400, PIK IPLEX and RICC, respectively, and the execution time less than 40 s (i.e., 100x of the overhead) is short.

Table 4 shows performance of footprint-based DIMM hotplug for real workloads. As shown in the table, most jobs executed on the production systems are classified into Class **S** (i.e, 98.7%, 94.4% and 98.5% for CX400, PIK IPLEX and RICC, respectively), in which jobs do not suffer from the control overhead. Many of the remaining jobs are categorized to Class **LL**, in which the control overhead is hidden by the large execution time. Very few jobs (i.e., Class **LS**) suffer from the control overhead, so that the overall impact of our footprint-based DIMM hotplug on performance of production systems would be negligible.

7 Related Work

The efficient usage of low power DRAM modes has been identified as a promising technique in other projects as well. Delaluz et al. propose a power-aware OS scheduler that changes DRAM power modes during context switches depending on process memory usage [46]. Zhou et al. propose optimizing page allocation based on page miss ratio curves in order to increase energy efficiency [47]. In addition, several power-aware memory controllers have been proposed [48], [49], [50]. Our approach complements these techniques and adds the ability to completely shutdown DIMMs.

Further, there have been many efforts to optimize refresh operations for DRAM power savings. Ghosh and Lee propose a technique to stop refresh operations for those DRAM rows that have been recently accessed [51]. Isen and John propose an architecture-level technique to halt refresh operations for free memory region with OS support [52]. Liu et al. propose a framework that stops refresh operations for non-critical data a programmer designates [53]. Wilkerson et al. propose a technique to reduce refresh rates with the help of multi-bit ECC [54]. In contrast to these techniques, our technique completely powers off all DRAMs in a DIMM at the OS level.

Some techniques using memory DVFS [22], [55], [56] have been proposed. Similar to CPU DVFS, memory DVFS reduces the memory power consumption by adjusting a set of voltage and frequency levels at runtime. However, to the best of our knowledge, there is no hardware to support memory DVFS, yet. Moreover, memory DVFS has limited capability of saving DRAM standby power due to the following two reasons. First, typical memory DVFS controls power of DIMMs at a spatially coarse granularity, requiring all DIMMs to change their memory clock frequencies concurrently. Second, memory DVFS has no ability to reduce the standby power to zero.

Zhang et al. proposed DIMMer, which turns DRAM ranks on/off based on an analysis of a datacenter trace [19]. However, no real implementation or system experiment is shown. Further, our work provides easier transparent integration of OS-level DIMM-on/off.

8 Conclusions

This paper presented a new power management technique called footprint-based DIMM hotplug. It enables a compute node to dynamically adjust the number of DIMMs that are powered matching the memory footprint requirements of a running application. We implemented this approach as part of an optimized page management system with small control overhead, using the existing DIMM hotplug APIs in Linux. We demonstrated our technique on a real server, and show that it has the capability to save substantial DIMM power.

Our future work will focus on verifying the effectiveness of our technique on real HPC systems. Additionally, powering down also the first DIMMs, while mitigating the impact on bandwidth, offers further opportunities for power savings.

Acknowledgments

This work was partially supported by the Japan Science and Technology Agency under Core Research for Evolutional Science and Technology. We sincerely thank the Research Institute for Information Technology in Kyushu University for providing the job log of the CX-400 supercomputer.

References

[1] GREEN500, "The green 500," *Internet:www.top500.org/green500/*, [Apr. 3, 2017].
[2] P. Messina and S. Lee, "The U.S. exascale computing project," in *The 2016 ACM/IEEE Conference on Supercomputing (Birds of a Feather)*, 2016.

[3] *Final Minutes, Advanced Scientific Computing Advisory Committee, August 14-15, 2012*. U. S. Department of Energy, Aug 2012.

[4] TOP500, "Top500 supercomputer sites," *Internet:www.top500.org/*, [Aug. 10, 2017].

[5] T. Patki, D. K. Lowenthal, B. Rountree, M. Schulz, and B. R. de Supinski, "Exploring hardware overprovisioning in power-constrained, high performance computing," in *Proceedings of the 27th International Conference on Supercomputing*, 2013, pp. 173–182.

[6] K. P. Saravanan, P. M. Carpenter, and A. Ramirez, "A performance perspective on energy efficient HPC links," in *Proceedings of the 28th International Conference on Supercomputing*, 2014, pp. 313–322.

[7] X. Yang, Z. Zhou, S. Wallace, Z. Lan, W. Tang, S. Coghlan, and M. E. Papka, "Integrating dynamic pricing of electricity into energy aware scheduling for HPC systems," in *Proceedings of the 2013 ACM/IEEE Conference on Supercomputing*, 2013, pp. 60:1–60:11.

[8] I. Alan, E. Arslan, and T. Kosar, "Energy-aware data transfer algorithms," in *Proceedings of the 2015 ACM/IEEE Conference on Supercomputing*, 2015, pp. 44:1–44:12.

[9] Y. Inadomi, T. Patki, K. Inoue, M. Aoyagi, B. Rountree, M. Schulz, D. Lowenthal, Y. Wada, K. Fukazawa, M. Ueda, M. Kondo, and I. Miyoshi, "Analyzing and mitigating the impact of manufacturing variability in power-constrained supercomputing," in *Proceedings of the 2015 ACM/IEEE Conference on Supercomputing*, 2015, pp. 78:1–78:12.

[10] S. Miwa and H. Nakamura, "Profile-based power shifting in interconnection networks with on/off links," in *Proceedings of the 2015 ACM/IEEE Conference on Supercomputing*, 2015, pp. 37:1–37:11.

[11] A. Venkatesh, A. Vishnu, K. Hamidouche, N. Tallent, D. D. Panda, D. Kerbyson, and A. Hoisie, "A case for application-oblivious energy-efficient MPI runtime," in *Proceedings of the 2015 ACM/IEEE Conference on Supercomputing*, 2015, pp. 29:1–29:12.

[12] S. Wallace, X. Yang, V. Vishwanath, W. E. Allcock, S. Coghlan, M. E. Papka, and Z. Lan, "A data driven scheduling approach for power management on HPC systems," in *Proceedings of the 2016 ACM/IEEE Conference on Supercomputing*, 2016, pp. 56:1–56:11.

[13] K. T. Malladi, I. Shaeffer, L. Gopalakrishnan, D. Lo, B. C. Lee, and M. Horowitz, "Rethinking DRAM power modes for energy proportionality," in *Proceedings of the 45th Annual IEEE/ACM International Symposium on Microarchitecture*, 2012, pp. 131–142.

[14] Al Gara, "Technology Challenges and Trends a Look through a 2030 Crystal Ball," Presentation at Next-Gen HPC: The Path to Exascale Artifical Intelligence and Personalized Medicine, Munich, Germany, June 2018.

[15] Parallel Workload Archives, "The Potsdam Institute for Climate Impact Research (PIK) IBM iDataPlex cluster log," *Internet:www.cs.huji.ac.il/labs/parallel/workload/l_pik_iplex/index.html*, [July 6, 2017].

[16] ——, "The RICC log," *Internet:www.cs.huji.ac.il/labs/parallel/workload /l_ricc/index.html*, [July 6, 2017].

[17] C. S. Bae and T. Jamel, "Energy-aware memory management through database buffer control," in *Proceedings of the third Workshop on Energy-Efficient Design*, 2011.

[18] T. Zhang, K. Chen, C. Xu, G. Sun, T. Wang, and Y. Xie, "Half-DRAM: A high-bandwidth and low-power DRAM architecture from the rethinking of fine-grained activation," in *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, 2014, pp. 349–360.

[19] D. Zhang, M. Ehsan, M. Ferdman, and R. Sion, "DIMMer: a case for turning off DIMMs in clouds," in *Proceedings of the ACM Symposium on Cloud Computing*, 2014, pp. 11:1–11:8.

[20] J. Beckett, "Memory performance guidelines for Dell PowerEdge 12th generation servers," Whitepaper, 2016.

[21] H. Park, S. Baek, J. Choi, D. Lee, and S. H. Noh, "Regularities considered harmful: Forcing randomness to memory accesses to reduce row buffer conflicts for multi-core, multi-bank systems," in *Proceedings of the 18th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2013, pp. 181–192.

[22] H. David, C. Fallin, E. Gorbatov, U. R. Hanebutte, and O. Mutlu, "Memory power management via dynamic voltage/frequency scaling," in *Proceedings of the 8th ACM International Conference on Autonomic Computing*, 2011, pp. 31–40.

[23] S. Desrochers, C. Paradis, and V. M. Weaver, "A validation of DRAM RAPL power measurements," in *Proceedings of the Second International Symposium on Memory Systems*, 2016, pp. 455–470.

[24] Micron Technology Inc., "4Gb: x4, x8, x16 DDR3L SDRAM description," Whitepaper, 2011.

[25] ——, "512Mb: x4, x8, x16 DDR SDRAM features," Whitepaper, 2000.

[26] ——, "2Gb: x4, x8, x16 DDR2 SDRAM features," Whitepaper, 2006.

[27] ——, "4Gb: x4, x8, x16 DDR3 SDRAM features," Whitepaper, 2009.

[28] ——, "4Gb: x4, x8, x16 DDR4 SDRAM features," Whitepaper, 2014.

[29] T. Chen, "Introduction to ACPI based memory hot-plug," LinuxCon Japan, 2013.

[30] Y. Ishimatsu, "Memory hotplug," LinuxCon Japan, 2013.

[31] S. Miwa and H. Honda, "Memory hotplug for energy savings of HPC systems," in *Proceedings of the 2015 ACM/IEEE Conference on Supercomputing (poster)*, 2015, pp. –.

[32] T. Cloyd, "Installing and upgrading DDR3 memory: Quick reference guide for new dell PowerEdge servers," Whitepaper, 2009.

[33] Hewlett Packard Enterprise, "Overview of DDR4 memory in HPE ProLiant Gen9 servers with Intel Xeon E5-2600 v3: Best practice guidelines," Whitepaper, 2015.

[34] AMD Inc., "High bandwidth memory, reinventing memory technology," *Internet:www.amd.com/en-us/innovations/software-technologies/hbm*, [Apr. 3, 2017].

[35] Hybrid Memory Cube Consortium, "About hybrid memory cube," *Internet:www.hybridmemorycube.org/technology.html*, [Apr. 3, 2017].

[36] Hewlett Packard Enterprise, "Exascale: A race to the future of HPC – the next generation of computing," Whitepaper, 2016.

[37] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting phase change memory as a scalable dram alternative," in *Proceedings of the 36th Annual International Symposium on Computer Architecture*, 2009, pp. 2–13.

[38] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable high performance main memory system using phase-change memory technology," in *Proceedings of the 36th Annual International Symposium on Computer Architecture*, 2009, pp. 24–33.

[39] A. Asato, "Extracting the performance of multi-core processor in supercomputer," in *The 12th International Forum on Embedded MPSoC and Multicore (keynote)*, 2012.

[40] Micron Technology Inc., "Power calcs," *Internet:www.micron.com/~/media/documents/products/power-calculator/ddr3_ddr3l_power_calc.xlsm*, [May 20, 2018].

[41] HPC Callenge, "HPC challenge benchmark," *Internet://icl.cs.utk.edu/hpcc/index.html*, Jun. 18, 2012 [Apr. 3, 2017].

[42] CORAL, "CORAL (collaboration of oak ridge, argonne and livermore) benchmark codes," *Internet:asc.llnl.gov/CORAL-benchmarks/*, Jun. 19, 2014 [Apr. 3, 2017].

[43] ——, "CORAL-2 benchmarks," *Internet:asc.llnl.gov/coral-2-benchmarks/*, [May 21, 2018].

[44] GRAPH500, "Graph500," *Internet:graph500.org/*, [May 21, 2018].

[45] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le, "RAPL: memory power estimation and capping," in *Proceedings of the 16th ACM/IEEE International Symposium on Low Power Electronics and Design*, 2010, pp. 189–194.

[46] V. Delaluz, A. Sivasubramaniam, M. Kandemir, N. Vijaykrishnan, and M. J. Irwin, "Scheduler-based DRAM energy management," in *Proceedings of the 39th Annual Design Automation Conference*, 2002, pp. 697–702.

[47] P. Zhou, V. Pandey, J. Sundaresan, A. Raghuraman, Y. Zhou, and S. Kumar, "Dynamic tracking of page miss ratio curve for memory management," in *Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2004, pp. 177–188.

[48] I. Hur and C. Lin, "A comprehensive approach to DRAM power management," in *Proceedings of the 14th International Symposium on High Performance Computer Architecture*, 2008, pp. 305–316.

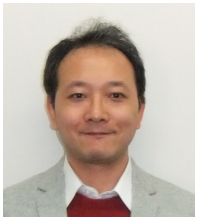
[49] Y. Lee and S. Kim, "RAMS: DRAM rank-aware memory scheduling for energy saving," *IEEE Transaction on Computers*, vol. 65, no. 10, pp. 3210–3216, 2016.

[50] D. Wu, B. He, X. Tang, J. Xu, and M. Guo, "RAMZzz: Rank-aware dram power management with dynamic migrations and demotions," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, 2012, pp. 32:1–32:11.

[51] M. Ghosh and H.-H. S. Lee, "Smart refresh: An enhanced memory controller design for reducing energy in conventional and 3D die-stacked DRAMs," in *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, 2007, pp. 134–145.

[52] C. Isen and L. John, "ESKIMO: Energy savings using semantic knowledge of inconsequential memory occupancy for DRAM sub-

system," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2009, pp. 337–346.
[53] S. Liu, K. Pattabiraman, T. Moscibroda, and B. G. Zorn, "Flikker: Saving DRAM refresh-power through critical data partitioning," in *Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2011, pp. 213–224.
[54] C. Wilkerson, A. R. Alameldeen, Z. Chishti, W. Wu, D. Somasekhar, and S.-l. Lu, "Reducing cache power with low-cost, multi-bit error-correcting codes," in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, 2010, pp. 83–93.
[55] Q. Deng, D. Meisner, A. Bhattacharjee, T. F. Wenisch, and R. Bianchini, "CoScale: Coordinating CPU and memory system DVFS in server systems," in *Proceedings of the 45th Annual IEEE/ACM International Symposium on Microarchitecture*, 2012, pp. 143–154.
[56] Q. Deng, D. Meisner, L. Ramos, T. F. Wenisch, and R. Bianchini, "MemScale: active low-power modes for main memory," in *Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2011, pp. 225–238.



Shinobu Miwa received the Ph.D. degree in Informatics from Kyoto University in 2007. He joined Lawrence Livermore National Laboratory as a Visiting Scientists and Professionals, and is now an Associate Professor at the University of Electro-Communications. His research interests are computer architecture and high performance computing. He is a member of ACM and IEEE.

Masaya Ishihara was born in 1990 and got the M.E. from the University of Electro-Communications in 2017. He currently works at the Mitsubishi Electric Corporation. His research interest is high performance computing.

Hayato Yamaki received the Ph.D. in Engineering from Keio University, Japan, in 2016. He is now an assistant professor at the University of Electro-Communications. His research interests are Internet architecture and computer architecture. He is a member of ACM, IEEE, and IPSJ.

Hiroki Honda received the BS, MS, and PhD degrees in electrical engineering from Waseda University, Japan, in 1984, 1986 and 1991, respectively. He is currently a professor of The University of Electro-Communications, Japan. He was the chair of IEEE Computer Society Japan Chapter from 2009 to 2011. His research interests include parallel processing, parallelizing compiler, parallel computer architecture. He is a member of IEEE, IEEE-CS and ACM.

Martin Schulz is a Full Professor and Chair for Computer Architecture and Parallel Systems at the Technische Universität München (TUM), which he joined in 2017. Prior to that, he held positions at the Center for Applied Scientific Computing (CASC) at Lawrence Livermore National Laboratory (LLNL) and Cornell University. He earned his Doctorate in Computer Science in 2001 from TUM and a Master of Science in Computer Science from UIUC.