Synergy of Dynamic Frequency Scaling and Demotion on DRAM Power Management: Models and Optimizations

Yanchao Lu, Bingsheng He, Xueyan Tang and Minyi Guo

Abstract—Main memory (or DRAM) is one of the most significant components to the computer system's performance and energy consumption. Dynamic frequency scaling (*DFS*) and DRAM low-power states (*Demotion*) are two main-stream techniques for DRAM power management. DFS reduces the operation frequency of memory channels and DRAM devices when the memory bandwidth is under-utilized, whereas demotion transits individual memory *ranks* to low-power states during long idle periods. Despite that there have been fruitful research work for DFS and demotion separately, little attention has been paid to the synergy between these two techniques. To bridge this gap, this paper conducts a comprehensive study on the synergy between DFS and demotion. In particular, we leverage queuing theory to develop analytical models for the energy consumption and performance of DRAM systems with DFS and demotion. These models provide valuable insights into the synergy between DFS and demotion. We further attempt to minimize the energy consumption by considering both DFS and demotion while keeping a pre-defined performance penalty budget. To reduce the optimization complexity, we develop simple yet effective heuristics to search near-optimum DFS-demotion configurations. We experimentally compare our design with other state-of-the-art DRAM energy saving policies using detailed simulations of a large set of workloads. Experimental results show the accuracy of our analytical models and the effectiveness of our optimizations.

Index Terms—Demotion, Dynamic frequency scaling, Energy consumption, Main memory systems, In-memory processing

1 Introduction

Energy consumption has become a major factor in the design and implementation of modern computer systems. In many systems, main memory (or DRAM) is a critical component for the performance and energy consumption. As processors have moved to a multi-/many-core era, more applications run simultaneously with their working sets in the main memory. Emerging applications such as in-memory data analytics [1] and RAMCloud [2] boost the memory capacity of modern computing systems. Such hunger for main memory of larger capacity makes the amount of energy consumed by main memory approaching or even surpassing that consumed by processors in many servers [3], [4]. For example, it has been reported that main memory contributes to as much as 40–46% of total energy consumption in server applications [5], [6]. For these reasons, this paper investigates whether and how we can leverage DRAM power management techniques to reduce the energy consumption of main memory.

There have been various energy saving techniques on exploiting the power management capability of main memory. Two important and common techniques are DRAM low-power states (*Demotion*) [7], [8], [9], [10] and dynamic frequency scaling (*DFS*) [11], [12]. The common theme of demotions is to transit individual memory ranks to low-power states during (long) idle periods, whereas DFS dynamically scales the operation frequency of memory channels and DRAM devices to make the active memory power proportional to memory loads. DFS can reduce the memory power when the memory bandwidth is under-utilized.

Despite that there have been fruitful research studies for DFS and demotion separately, little attention has been paid to the synergy between those two techniques. Actually, there is a complex interplay between DFS and demotion in DRAM power management. On the one hand, a lower memory frequency in DFS leads to lower background power consumptions for memory devices. However, this also results in a longer application execution time and increases the energy consumption of a memory access. Moreover, it usually reduces the length of memory idle periods, and thus degrades the effectiveness of demotions. On the other hand, operating memory devices at a higher frequency reduces the memory access energy and creates more opportunities for demotions, at the cost of increased background power consumptions and state transition overheads (i.e., the resynchronization energy and delay). We find that there is a tradeoff between DFS and demotion in optimizing memory energy consumption. Figure 1 shows the results of two state-of-the-art demotion (RAMZzz [10]) and DFS (MemScale [12]) approaches with different optimization goals (The detailed experimental setup can be found in Section 5). We study two common optimization goals, i.e., energy consumption and energy-delay^2 (ED^2). All values are normalized to those of RAMZzz. Demotion is more efficient for some cases on workloads, optimization goals and performance penalty budgets, whereas DFS wins in other cases. This interplay motivates us to study the synergy

- *Bingsheng He and Xueyan Tang are with Nanyang Technological University, Singapore. Corresponding author: Bingsheng He, bshe@ntu.edu.sg.*
- *Yanchao Lu and Minyi Guo are with Shanghai Jiao Tong University, China. The work of Yanchao Lu is done when he was a visiting student in Nanyang Technological University, Singapore.*

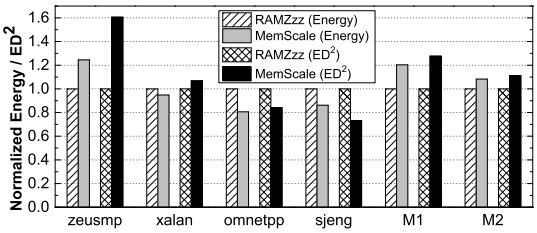


Fig. 1. The tradeoff between RAMZzz [10] and MemScale [12] across different optimization goals. The penalty budget of energy optimization is 10%, and that of ED^2 optimization is 5%.

between DFS and demotion, and to find the suitable DFS-demotion configuration for maximizing energy saving.

In this paper, we develop performance models and optimization techniques for exploiting the synergy between DFS and demotion. This is in contrast with the previous studies on DRAM power management, which focused on either DFS [11], [12] or demotion only [10], [13], [14], [15]. We start by developing queuing theory-based analytical models for the energy consumption and performance of DRAM systems with DFS and demotion. The models provide valuable insights into the synergy between DFS and demotion. We show that: (1) DFS and demotion have their own energy-efficient scopes on different workloads and DRAM architectures; (2) the optimum energy consumption depends nontrivially on the DFS-demotion configurations, which vary with different workloads and DRAM architectures.

Through modeling, we find that a naive approach of combining DFS and demotion can be far from optimal in practice. For example, Deng et al. [12] combined a single low-power state (fast-exit powerdown) with their DFS, which produces only marginal energy saving in their experiments. A dynamic and adaptive approach is more suitable than a static combination. Therefore, we propose to adjust the DFS-demotion configuration periodically. In each period, our cost model can predict the energy consumption and performance of a DFS-demotion configuration. However, a brute-force search over the entire DFS-demotion space can be too costly. Thus, we develop simple yet effective heuristic mechanisms to reduce the runtime overhead. Our models and optimizations are able to work for different goals such as energy consumption and ED^2. In this paper, we focus on the optimization goal of minimizing the energy consumption while keeping the performance penalty within a given budget.

We evaluate our design using detailed simulations of a large set of workloads. We use the SPEC 2006 benchmark to evaluate our approach in comparison with the state-of-the-art approach using demotion or DFS techniques only [10], [12] as well as the static combination of DFS and demotion [12]. The experimental results show that our approach achieves an average energy consumption reduction of 11–70% over the demotion-only approach, 20–67% over the DFS-only approach, 22–54% over the static combination of DFS and demotion. Moreover, our heuristic mechanisms produce near-optimal results with only 8% higher energy consumption on average than the optimum.

The contributions of this work are summarized as follows. First, we develop analytical models to study the synergy between DFS and demotion, which reveal the interplay between DFS and demotion. Second, leveraging the analytical models, we develop optimization techniques for dynamic DFS-demotion configurations. We further develop simple and effective heuristics to reduce the computational complexity of optimizations. To the best of our knowledge, this is the first work to dynamically adapt DFS-demotion configurations for DRAM power management. Finally, we conduct extensive experiments to show the effectiveness of our design over different workloads and DRAM architectures.

Organization. The rest of the paper is organized as follows. We introduce the background and review related work in Section 2. Section 3 presents analytical models for the synergy between DFS and demotion. Section 4 describes our proposed optimizations. The experimental results are presented in Section 5. We conclude this paper in Section 6.

2 Background and Related Work

2.1 DRAM Power Management

We study the DDR-series (e.g., DDR3 and DDR4) based main memory system in this paper. In power management, a memory *rank* is the smallest physical unit that we can control. Specifically, individual ranks serve memory requests independently and can also operate at different power states. A busy rank may work in the *active* state (ACT), while an idle rank may be set to a *low-power* (or power-down) state in order to save energy consumption. Note, *all the memory ranks should work at the same memory frequency in the current DRAM architecture.* Regardless of the memory architecture, the total power consumption of the DRAM system can be divided into two parts: operation power and background power. The operation power is the power required to activate the DRAM device to perform memory reads and writes. The background power accounts for all the power consumption when there is no memory access. Background power is a major component in the total DRAM power consumption [16], [17].

2.2 DRAM Demotion

Modern DRAM architectures support a number of low-power states, which hardware components to be disabled [18], [19]. Each state is characterized with its power consumption and the time that it takes to transition back to the active state (i.e., resynchronization time). Typically, the lower power consumption a state has, the higher the resynchronization time is. Table 1 summarizes the major power state transitions of DDR3. For each state, we show its normalized power consumption (normalized to that of ACT) and the resynchronization time back to ACT. The power consumption data are calculated from DRAM System Power Calculator [20]. The resynchronization times are obtained from DRAM manufacturers' data sheets [18].

Entering a low-power state when a rank is idle can significantly reduce the background power consumption. For example, the pre-charge power-down with fast exit state (PRE_PDN_FAST) consumes only 52% of the power of ACT.

TABLE 1
Power states for DDR3 at 1333 MHz.

Power State	Normalized Power	Resynchronization Time (ns)
ACT	1.0	0
ACT_PDN	0.612	6
PRE_PDN_FAST	0.520	18
PRE_PDN_SLOW	0.299	24
SR_FAST	0.170	768
SR_SLOW	0.104	6768

However, to exit from a low-power state, the disabled hardware components need to be reactivated and the rank needs to be restored to the active state. Transitions to different power states cause very different latencies and energy penalties. For example, the self-refresh with slow exit state (SR_SLOW) has much higher resynchronization latency and energy cost than PRE_PDN_FAST. If a memory rank makes a wrong decision to transit itself into SR_SLOW during a short idle period, the energy penalty can outweigh the saved energy. Furthermore, the high resynchronization latency can degrade the memory performance considerably if such wrong decisions happen frequently. Only sufficiently long idle periods should make use of deeper low-power states.

Existing research on demotions can be roughly divided into two categories: 1) how to make correct decisions on state transitions [7], [10], [13], [14], and 2) how to extend the idle periods effectively [10], [15], [16].

For the first category, a number of models (e.g., based on history [10], [13] and exponential distributions [7]) have been developed to guide decision making on demotions. Hur et al. [13] developed adaptive history-based scheduling in the memory controller. Diniz et al. [14] limited the energy consumption by adjusting the power states of DRAM. Fan et al. [7] developed an analytic model to estimate the idle time of DRAM chips using an exponential distribution. Wu et al. [10] developed a history-based prediction model to estimate the power-down timeout (the amount of time spent from the beginning of an idle period before a transition to a low-power state is made) for accurate control of power state transitions. Compared with all these demotion-only studies, this paper combines the demotion and DFS techniques in queuing theory based analytical models.

Page migration has been considered to be an effective approach to extend the idle periods. Huang et al. [16] stored frequently-accessed pages into hot ranks and left infrequently-used and unmapped pages in cold ranks. Kshitij et al. [15] used a similar page migration mechanism between cold and hot ranks, and always set cold ranks with a pre-selected low-power state. Wu et al. [10] developed dynamic page migrations to adapt to data access patterns. While page migrations have been demonstrated to be effective in simulations, these techniques bring some tricky implementation issues that prohibit their practical usage in current DRAM architectures. First, page migrations bring performance and energy penalty (migrating pages cause more memory reads and writes, and memory service interruptions), and are usually too complex to be integrated into current memory systems. Second, page migrations usually require modifications to not only the DRAM controller but also operating systems. For these reasons, we do not consider page migration in this study.

TABLE 2
Impacts of DFS on DDR3 DRx4 R-DIMM architectures.

Power State	Power (W) at 1333 MHz	Power (W) at 800 MHz
ACT	1.34	1.09
ACT_PDN	0.82	0.67
PRE_PDN_FAST	0.70	0.58
PRE_PDN_SLOW	0.40	0.35
SR_FAST	0.23	0.19
SR_SLOW	0.14	0.14
Power State	**Resync Time (ns) at 1333 MHz**	**Resync Time (ns) at 800 MHz**
ACT	0	0
ACT_PDN	6	8
PRE_PDN_FAST	18	20
PRE_PDN_SLOW	24	26
SR_FAST	768	1280
SR_SLOW	6768	7280
Operation	**Energy (nJ) at 1333 MHz**	**Energy (nJ) at 800 MHz**
Average energy/read	56	64.7
Average energy/write	61	72
Operation	**Latency (ns) at 1333 MHz**	**Latency (ns) at 800 MHz**
Average latency/access	51	55

2.3 DRAM Dynamic Frequency Scaling (DFS)

Memory dynamic frequency scaling (DFS) is a more recent approach to reduce the DRAM energy consumption [11], [12]. When the memory bandwidth is under-utilized, lowering the memory frequency can bring potential energy savings. Adjusting frequency on current DRAM architectures has little runtime overhead. The time of transitions between different frequencies is around $1\mu s$ [11].

Lowering the memory frequency affects both the power and performance of the memory system. We study the impacts of DFS on power consumptions, resynchronization times, and memory accesses for DDR3 architectures. Table 2 summarizes the results on the background and operation powers, low-power states' resynchronization times, and average memory access latency for DDR3 architectures at 1333MHz and 800MHz. The background powers are calculated for a 1GB DDR3 DRx4 R-DIMM, obtained from the calculation by David et al. [11]. The operation energy stands for an average energy consumption per read or write [11]. The low-power states' resynchronization times and average memory access latency are obtained from DRAM System Power Calculator [20] and DRAM manufacturers' data sheets [18]. A memory access includes activation and pre-charge operations, reading/writing data arrays, data output, and I/O termination.

From this table, we make the following observations. First, changing memory frequency has a significant impact on the power states in the DRAM architecture. Typically, the lower the memory frequency, the lower the power consumption of each power state. However, the resynchronization time of each low-power state increases with decreasing memory frequency (The resynchronization time is associated with t_{CK}, which scales with the memory frequency). DFS has a direct impact on the demotion decision. On one hand, it enhances the energy efficiency of each low-power state since the power

consumption of each low-power state is further reduced at lower memory frequencies. On the other hand, it leads to larger performance degradation as well as increased energy penalty when transiting into low-power states.

Second, the operation (read and write) energy is higher at lower memory frequencies. This is because the memory access takes longer time at lower frequencies. However, as demonstrated in [11], [12], the reduction in the background power is normally more significant than the increase in the operation energy consumption. Therefore, DFS can reduce the memory system energy consumption.

Third, lowering memory frequency increases the memory access latency. Overall, the increment of memory access latency is sub-linear to the decrement of frequency. Since the memory bus runs slower at lower frequencies, DFS can reduce the length of idle periods between memory requests.

There have been several studies on leveraging DFS to reduce DRAM energy consumption. David et al. [11] evaluated the effects of lower memory frequency on a real hardware platform, and switched the memory frequency and voltage based on memory bandwidth utilization at the runtime. Deng et al. [12] developed similar DFS schemes, based on performance and energy models. In their experiment, they developed a static approach that combines a single low-power state (fast-exit powerdown) with their DFS scheme. Our model analysis and experiment will demonstrate that the static approach is suboptimal for different workloads and DRAM architectures. Therefore, we propose adaptive selection of the DFS-demotion configuration at the runtime. A follow-up work of Deng et al. [21] combines CPU DVFS with memory DFS to save system energy. However, it does not consider main memory demotion as we do in this work.

Besides optimizations targeting at general DRAM systems, some researchers have also proposed energy saving techniques for specific applications such as databases and virtual machines. Cache-centric optimizations (either cache-conscious [22] or cache-oblivious [23]) reduce memory access and create more opportunities for energy savings. Dynamic memory allocation mechanisms such as ballooning and hot-plug [24] provide an opportunity for rank-aware DRAM power saving in virtualization environments. Finally, there has been some work on combining demotion and DFS on the CPU (e.g., [25], [26], [27]). However, their models and optimizations are specifically designed for the CPU, which are not applicable to the main memory. The synergy between demotion and DFS for the main memory has not been well studied. To the best of our knowledge, this paper is the first of its kind in studying demotion and DFS on DRAM in a systematic approach.

3 Modeling Memory DFS and Demotion

In order to understand the potential benefits of combining memory DFS and demotion techniques, we develop analytical models to estimate the energy consumption and performance of different combinations of DFS and demotion settings. The model captures the DRAM architecture features and workload parameters. With the model, we are able to gain insight into the power-performance optimization space shaped by different DFS-demotion configurations, and understand the potential improvement offered by the optimal DFS-demotion configuration. Table 3 lists the parameters used in our model.

TABLE 3
Parameters and variables used in the analytical model.

Parameters	Description
Workload Characteristics	
λ	The arrival rate of memory requests at the highest memory frequency (i.e., f_0).
ϕ	The proportion of memory reads in all memory requests.
DRAM Architecture Features	
N	The number of available power states, i.e., S_0 (active state), ..., S_{N-1}.
M	The number of available memory frequencies, i.e., f_0 (default), ..., f_{M-1}.
$P_{i,j}$	The power consumption of state S_i at frequency f_j.
$R_{i,j}$	The resynchronization time of state S_i at frequency f_j.
g_j	The memory access latency at frequency f_j.
γ_j	The average energy cost per memory read at frequency f_j.
ω_j	The average energy cost per memory write at frequency f_j.
System Configurations	
D	The maximum allowed performance degradation, i.e., the delay budget.
DFS-Demotion Configurations	
Δ_i	The power-down timeout of state S_i.
f	The memory frequency.

3.1 Memory Energy and Performance Models

We first present the assumptions for our model. First, we assume that the times between memory requests (inter-arrival times) follow a Poisson distribution. Second, the DRAM system enables both demotion and DFS techniques, with N power states (S_i, $i = 0, \ldots, N-1$) and M memory frequencies (f_i, $i = 0, \ldots, M-1$), respectively. For simplicity, we denote the *active* state by S_0, and the rest $N-1$ low-power states by S_1, S_2, ..., S_{N-1} in the descending order of their power consumptions. We also sort all the supported frequencies in the descending order: $f_0 > f_1 > f_2 > \cdots > f_{N-1}$. DFS is applied to all memory ranks, i.e., all memory ranks should have the same frequency at any time. In contrast, each rank can make its own demotion decisions. A memory rank transits to the power state S_i if the idle period exceeds a power-down timeout Δ_i ($i = 1, \ldots, N-1$). The state transition incurs a time penalty $R_{i,j}$ (the resynchronization time of state S_i at frequency f_j). We view multiple power state transitions as a chain of state transitions from higher-power states to lower-power states. We define the demotion configuration of a memory rank to be a vector of power-down timeouts $\vec{\Delta} = (\Delta_0, \Delta_1, \ldots, \Delta_{N-1})$, where Δ_i represents the power-down timeout of state S_i, $i = 1, \ldots, N-1$ (For simplicity, we set $\Delta_0 = 0$ as the power-down timeout of state S_0, i.e., the active state). When the idle period length becomes longer than Δ_i, we perform the state transition from S_{i-1} to S_i.

We model the energy consumption and performance of each memory rank individually, since the ranks operate independently. In particular, we derive the estimation for a rank

at memory frequency f_j and power-down timeouts $\vec{\Delta}$. With queuing theory, we model a memory rank as an M/D/1 queuing system with arrival rate λ (λ is the arrival rate of memory requests for a given rank at frequency f_0), and a determined service time g_j (i.e., the memory access latency at memory frequency f_j). Inspired by previous studies [5] and [28], we extend the conventional M/D/1 model with an exceptional first service time to model the effects of demotion and frequency scaling. If a memory request arrives and finds the rank busy (i.e, the rank is serving other memory requests), it has a normal memory access latency g_j at memory frequency f_j. Otherwise, the rank is idle and the memory access would be delayed by an initial setup time $\mathcal{I}$. The initial setup time $\mathcal{I}$ is the resynchronization time for powering up the rank from a low-power state. Let x be a random variable representing the idle period length for the memory rank. Then, the initial setup time $\mathcal{I}$ at memory frequency f_j and power-down timeouts $\vec{\Delta}$ is given by

$$\mathcal{I} = \begin{cases} R_{0,j}\ (=0) & \text{if } 0 \le x < \Delta_1 \\ R_{i,j} & \text{if } \Delta_i \le x < \Delta_{i+1}, i = 1, \ldots, N-2 \\ R_{N-1,j} & \text{if } x \ge \Delta_{N-1} \end{cases} \quad (1)$$

The inter-arrival time between two memory requests conforms to the exponential distribution with parameter λ. Because of the memoryless property of exponential distribution, the idle period length x conforms to the same distribution (the exponential distribution with parameter λ). Thus, the first and the second moments $E[\mathcal{I}]$ and $E[\mathcal{I}^2]$ of the initial setup time are given by

$$E[\mathcal{I}] = \int_0^\infty \mathcal{I}\lambda e^{-\lambda x}\, dx = \sum_{i=0}^{N-1} R_{i,j}(e^{-\lambda\Delta_i} - e^{-\lambda\Delta_{i+1}}) \quad (2)$$

$$E[\mathcal{I}^2] = \int_0^\infty \mathcal{I}^2\lambda e^{-\lambda x}\, dx = \sum_{i=0}^{N-1} R_{i,j}^2(e^{-\lambda\Delta_i} - e^{-\lambda\Delta_{i+1}}) \quad (3)$$

Note that we set $R_{0,j} = 0$, $\Delta_0 = 0$, and $\Delta_N = \infty$. According to Welch's previous work [28], the expected response time for an M/D/1 server with an exceptional first service time is given by

$$E[\mathcal{R}](f_j, \vec{\Delta}) = \frac{\lambda g_j^2}{2(1-\lambda g_j)} + \frac{2E[\mathcal{I}] + \lambda E[\mathcal{I}^2]}{2(1+\lambda E[\mathcal{I}])} + g_j \quad (4)$$

$E[\mathcal{R}](f_j, \vec{\Delta})$ consists of three parts: 1) the expected queueing delay for a standard M/D/1 queue, 2) the expected resynchronization delay caused by demotion (calculated by the initial setup time $\mathcal{I}$ and the probability for a memory request to be delayed by the resynchronization), and 3) the expected memory access latency g_j at memory frequency f_j. Plugging Eq. (2) and (3) into Eq. (4), we obtain the expected response time for one memory request $E[\mathcal{R}](f_j, \vec{\Delta})$ at memory frequency f_j and power-down timeouts $\vec{\Delta}$.

We now derive the expected energy consumption of one memory request. We decompose the expected energy cost of a memory request (denoted as $E[\mathcal{E}](f_j, \vec{\Delta})$) into two parts in Eq. (5): 1) $E[\mathcal{E}_{op}](f_j)$, the average energy cost per memory read/write at frequency f_j; and 2) $E[\mathcal{E}_{bk}](f_j, \vec{\Delta})$, the expected preceding background energy consumption (i.e., that between the last memory access and the current memory access).

$$E[\mathcal{E}](f_j, \vec{\Delta}) = E[\mathcal{E}_{op}](f_j) + E[\mathcal{E}_{bk}](f_j, \vec{\Delta}) \quad (5)$$

$E[\mathcal{E}_{op}](f_j)$ is given by

$$E[\mathcal{E}_{op}](f_j) = \phi \cdot \gamma_j + (1-\phi) \cdot \omega_j \quad (6)$$

where γ_j and ω_j are the average energy costs per memory read and write, respectively, and ϕ is the proportion of memory reads in all memory accesses.

To estimate the preceding background energy consumption ($E[\mathcal{E}_{bk}](f_j, \vec{\Delta})$), we first calculate the probability θ that an arrived memory request finds the memory rank idle. Since the expected busy period length for the memory rank is $E[\mathcal{B}] = \frac{g_j + E[\mathcal{I}]}{1-\lambda g_j}$ according to the results of a previous work [28], the expected number of memory requests served during a busy period is $E[\mathcal{B}]/g_j$. The first memory request of a busy period finds the system idle, while the others find it busy. Thus, θ is given by

$$\theta = \frac{g_j}{E[\mathcal{B}]} = \frac{g_j(1-\lambda g_j)}{g_j + E[\mathcal{I}]} \quad (7)$$

When an arrived memory request finds the memory rank idle, the preceding background energy consumption depends on the preceding idle period length x. We consider the situation that $\Delta_i \le x < \Delta_{i+1}$ ($i = 0, \ldots, N-1$). The DRAM first consumes an accumulated energy $AC(f_j, i) = \sum_{k=0}^{i-1}(P_{k,j}(\Delta_{k+1} - \Delta_k))$ (i.e., spending $\Delta_{k+1} - \Delta_k$ time at each state S_k with $P_{k,j}$ power, $k = 0, \ldots, i-1$), then holds $P_{i,j}$ power for $x - \Delta_i$ time at state S_i, and finally dissipates $P_{0,j}$ power for $R_{i,j}$ time when activated (i.e., the resynchronization energy penalty). Thus, the background energy cost when $\Delta_i \le x < \Delta_{i+1}$ is $P_{i,j}(x - \Delta_i) + AC(f_j, i) + P_{0,j}R_{i,j}$. Since the preceding idle period length x conforms to the exponential distribution with parameter λ, the probability for x to satisfy $\Delta_i \le x < \Delta_{i+1}$ is $e^{-\lambda\Delta_i} - e^{-\lambda\Delta_{i+1}}$. Thus, the expected background energy cost is given by

$$\begin{aligned} &E[\mathcal{E}_{bk\_idle}](f_j, \vec{\Delta}) = \textstyle\sum_{i=0}^{N-1}\{(e^{-\lambda\Delta_i} - e^{-\lambda\Delta_{i+1}}) \times \\ &(P_{i,j}(\textstyle\int_{\Delta_i}^{\Delta_{i+1}} x\lambda e^{-\lambda x}\, dx - \Delta_i) + AC(f_j, i) + P_{0,j}R_{i,j})\} \\ &= \textstyle\sum_{i=0}^{N-1}\{(e^{-\lambda\Delta_i} - e^{-\lambda\Delta_{i+1}}) \times (P_{i,j}(\frac{e^{-\lambda\Delta_i}}{\lambda}(\lambda\Delta_i + 1) - \\ &\frac{e^{-\lambda\Delta_{i+1}}}{\lambda}(\lambda\Delta_{i+1} + 1) - \Delta_i) + AC(f_j, i) + P_{0,j}R_{i,j})\} \end{aligned} \quad (8)$$

If a memory request arrives when the memory rank is busy, the background energy consumption is zero. Therefore, the expected preceding background energy consumption $E[\mathcal{E}_{bk}](f_j, \vec{\Delta})$ is given by

$$E[\mathcal{E}_{bk}](f_j, \vec{\Delta}) = \theta \cdot E[\mathcal{E}_{bk\_idle}](f_j, \vec{\Delta}) \quad (9)$$

3.2 Numerical Studies

In the following, we conduct numerical studies based on the analytical models. Specifically, we study the impact of workload features, DRAM architectures and system configurations including the budget for performance degradation. Following the current DDR3 DRAM architectures, we consider $N = 6$ power states (an active state and 5 power-down states in the DDR3 architecture) for demotion, and $M = 6$ memory frequencies (1333, 1066, 800, 667, 533, 267 MHz) for DFS.

To understand how the DFS-demotion configuration affects the energy consumption, we change the number of power states used (denoted by n) and the number of memory frequencies used (denoted by m) for DRAM. Specifically, we set n and m with 1, 2, 3, and 6 as shown in Table 4. For example, $n = 1$ means that the memory ranks stay only in ACT. $m = 1$ means that DRAM operates only at the highest memory frequency 1333 MHz.

We aim to minimize the energy consumption of the memory system while keeping the program performance penalty

TABLE 4
The number of power states used (n) and the number of frequencies used (m).

n	Power States
1	ACT
2	ACT, PRE_PDN_FAST
3	ACT, PRE_PDN_FAST, SR_FAST
6	ACT, ACT_PDN, PRE_PDN_FAST, PRE_PDN_SLOW, SR_FAST, SR_SLOW

m	Frequencies
1	1333 MHz
2	1333, 800 MHz
3	1333, 800, 533 MHz
6	1333, 1066, 800, 667, 533, 267 MHz

TABLE 5
The impact of λ (accesses per 10^3 cycles).

λ	Optimal DFS-demotion configuration		
	Freq. (MHz)	Power-down timeouts (cycles)	Normalized energy consumption
0.1	267	0, 0, 0, ∞, ∞	0.098
1.0	667	12, 16, 16, ∞, ∞	0.317
2.5	667	0, 1214, 1591, ∞, ∞	0.614
4.0	800	37, ∞, ∞, ∞, ∞	0.830
5.5	1066	16, ∞, ∞, ∞, ∞	0.987
7.0	1333	∞, ∞, ∞, ∞, ∞	1.000

within a pre-defined budget. The penalty budget is defined by a performance slowdown D (e.g., 10% performance loss) relative to the program performance without any DRAM power management. We consider the expected program execution time of an idle period of the memory system (i.e., the CPU computation phase) and one memory request (i.e., the memory access phase). For simplicity, we study a single-rank memory system. We calculate the expected program performance under the memory frequency f_j and power-down timeouts $\vec{\Delta}$ by $T(f_j, \vec{\Delta}) = T_{cpu} + T_{mem}$. We assume that the CPU computation happens in and only in the idle period of the memory system. Thus, $T_{cpu} = 1/\lambda$, which is insensitive to changes in the memory system. $T_{mem} = E[\mathcal{R}](f_j, \vec{\Delta})$, which varies with the memory frequency and power-down timeouts. Hence, the optimal DFS-demotion configuration under a given performance penalty budget D is defined as, *the optimal memory frequency f and power-down timeouts $\vec{\Delta}$ that minimize $E[\mathcal{E}](f_j, \vec{\Delta})$, while satisfying the program performance* $T(f_j, \vec{\Delta}) \leq (1+D) \times T(f_0, \vec{\Delta}_{baseline})$. $\vec{\Delta}_{baseline}$ = $(0, \infty, \ldots, \infty)$, which means the memory system is always in ACT. $T(f_0, \vec{\Delta}_{baseline})$ is maximum program performance when the memory system runs at the highest memory frequency (f_0) without any state transitions.

By default, we set the proportion of memory reads ϕ = 1.0 (i.e., all memory requests are read operations), and the penalty budget D = 10%. The power consumption and resynchronization time of each power state, and the average energy cost per memory read/write at f_0 (i.e., 1333 MHz) are obtained from manufactures' datasheet [18]. The values of these parameters at frequencies other than f_0 are scaled according to Micron's System Power Calculator [20] and Technical Note on DDR3 Power [29]. For the memory access latency g_j, we calculate it according to the DRAM specification, i.e., the sum of t_{RCD}, t_{CL}, t_{RP} and t_{BURST}. These DRAM timing parameters stand for the times of an activation command, a pre-charge command, a column access and a data burst transfer, respectively. t_{RCD}, t_{CL} and t_{RP} are around $15ns$ in DDR3, even at low memory frequencies. t_{BURST} is four bus cycles (i.e., $4 \cdot t_{CK}$), which increases linearly as the memory frequency decreases. The arrival rate of memory requests λ is presented as in the average number of memory requests per 10^3 CPU cycles. The energy consumption is normalized to that of the baseline DFS-demotion configuration (i.e., at the memory frequency f_0 and power-down timeouts $\vec{\Delta}_{baseline}$). We present the results for a single memory rank.

Workload characteristics. We first study the impact of the arrival rate of memory requests (λ). In this study, we set ϕ = 1.0, and D = 10%, and use n = 6 power states and m = 6 frequencies for DRAM. For each λ, we search the optimal DFS-demotion configuration exhaustively. Table 5 shows the best DFS-demotion configuration for different values of λ. ∞ means that DRAM devices will not be demoted to such power-down state. We have the two key observations.

First, the optimal DFS-demotion configuration varies with the workloads. As λ increases, the normalized energy consumption also increases. Since the background energy consumption becomes less significant in the total energy consumption, the frequency increases and the opportunities for demotions decrease in the optimal configuration.

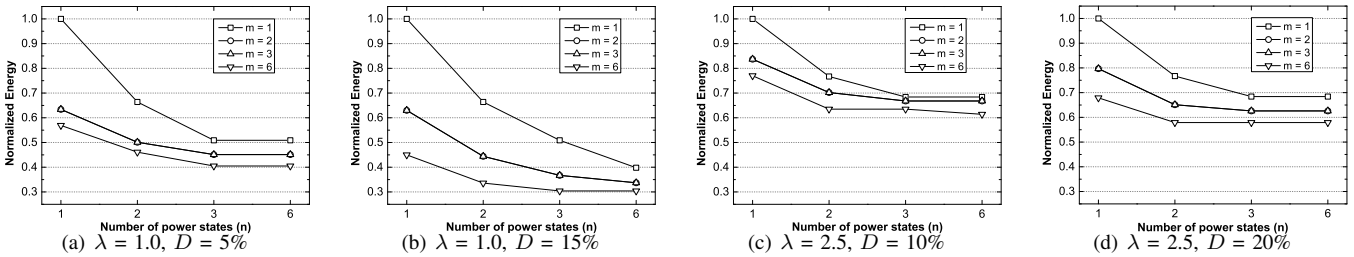
Second, both DFS and demotion contribute to the total energy saving. For some cases (e.g., λ = 1.0 and 2.5), they have the same frequency in the optimal DFS-demotion configuration, while their demotion schemes are different. For some cases (e.g., λ = 4.0 and 5.5), they have similar demotion schemes in the optimal DFS-demotion configuration, while their memory frequencies are different.

DRAM Architectures. Next, we study the impact of different DRAM architecture features as shown in Figure 2. We search the optimal DFS-demotion configuration in the space of n power states and m frequencies for each DRAM architecture exhaustively. We make the following observations from Figure 2.

First, making use of all the available power states and memory frequencies (i.e., n = 6 and m = 6) always has the lowest energy consumption across different workloads and performance penalty budgets. The energy consumption decreases with increasing numbers of power states and memory frequencies used. This implies that, both DFS and demotion contribute to the energy saving, which shows the synergy between DFS and demotion.

Second, it is quite costly to find the lowest energy consumption by exhaustive search. Since the lowest energy consumption usually exists when making use of all the available power states and memory frequencies (i.e., n=6 and m=6), the search space of DFS-demotion configurations is large. Thus, the cost of an exhaustive search for the optimal configuration is high.

Third, when the number of frequencies is large (i.e., m=6), adding more low-power states does not have much improvement for the two workloads (i.e., λ = 1.0 and 2.5), and vice versa. This is because both DFS and demotion can contribute the total energy saving. When the number of frequencies or low-power states is large, the optimization space of DFS-demotion may be large enough to pick a suitable DFS-demotion configuration for these specific workloads. However,



(a) λ = 1.0, D = 5% (b) λ = 1.0, D = 15% (c) λ = 2.5, D = 10% (d) λ = 2.5, D = 20%

Fig. 2. The impact of DRAM architecture features on energy consumption of optimal DFS-demotion configurations.

it is not appropriate to conclude that a small number of low-power states (i.e., n=3) are enough for all scenarios. In modern server workloads, the average memory access rate can be quite different for different workloads and change dynamically within one application.

System Configurations. We study the impact of the performance penalty budget. Figure 3 shows the normalized energy consumption of the optimal DFS-demotion configuration for three schemes when D varies from 5% to 20%. We search the optimal DFS-demotion configuration at each penalty budget exhaustively for the following three schemes: 1) demotion-only (with frequency at 1333MHz only), 2) DFS-only (with the active state only), and 3) the combined DFS-demotion scheme. The combined DFS-demotion scheme has the highest energy saving at all the performance penalty budgets. The energy saving becomes larger as the performance penalty budget increases. The memory frequencies of the optimal DFS-demotion configurations for the combined DFS-demotion scheme are 1066, 667, 533, 800 MHz for D = $5\%, 10\%, 15\%, 20\%$, respectively. Interestingly, the best memory frequency is higher at D = 20% compared with that at D = 15%. This again shows the synergy between DFS and demotion techniques. Though DFS alone brings less energy saving at D = 20%, combining it with a more aggressive demotion scheme achieves more significant energy saving.

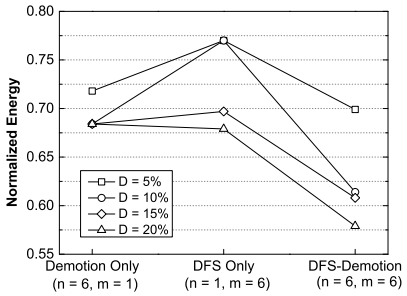


Fig. 3. The impact of penalty budget on energy consumption of optimal DFS-demotion configurations (λ = 2.5).

Summary. From the numerical studies, we observe the significant synergy between DFS and demotion. A combination of DFS and demotion can lead to much large energy savings than individual techniques. However, the optimal DFS-demotion configuration varies with the workloads, system configurations and DRAM architectures. Thus, an adaptive approach of combining DFS and demotion techniques is desirable. Furthermore, finding the optimal DFS-demotion configuration is a challenging task at the runtime, due to the large solution space. This motivates us to develop efficient heuristics for finding suitable DFS-demotion configurations.

4 Adaptive DFS-Demotion Optimizations

In this section, we develop adaptive DRAM power management that dynamically selects the suitable DFS-demotion configuration at the runtime. With the analytical models on performance and energy consumption developed in Section 3, we are able to optimize DRAM power management with respect to an arbitrary performance-energy optimization goal. In this section, we focus on minimizing the total memory energy consumption within a pre-defined budget of program performance degradation, while our method is also applicable to other optimization goals.

4.1 Overview

The workflow of the DRAM power management is given in Algorithm 1. We periodically adjust the memory frequency and each rank's power-down timeouts based on a control algorithm. The adjustment period is called an *epoch*. An epoch is defined to consist of a pre-defined number of memory requests to the DRAM. At the beginning of each epoch, we determine a suitable DFS-demotion configuration for the epoch and use it for the entire new epoch.

During an epoch, we maintain the performance information via a set of hardware performance counters (more details can be found in Section 4.2). These counters can be read by the operating system as inputs for estimating memory energy and program performance of the next epoch under different DFS-demotion configurations, or trigger specific hardware interruptions. They are reset at the beginning of each epoch. For each idle period in the epoch, demotions may be triggered according to the DFS-demotion configuration. At the beginning of each epoch, based on performance counters gathered in the previous epoch, the system estimates the maximum program performance $\mathbb{T}_{min}$ for the new epoch (Section 4.2). $\mathbb{T}_{min}$ is the program execution time when the memory system runs at the highest memory frequency and without any demotions. Then, the performance target $\mathbb{T}_{target}$ is set by the control algorithm. Next, the control algorithm tries to search all possible DFS-demotion configurations and picks up an optimal configuration that has a minimum estimated memory energy consumption, while maintaining the estimated program performance within the performance target $\mathbb{T}_{target}$ (details are presented in Section 4.3). Due to the huge search space, we explore heuristics to efficiently figure out a suitable DFS-demotion configuration that is close to the global optimum (Section 4.4). After the suitable DFS-demotion configuration

Algorithm 1 Workflow of Proposed DRAM Power Management Mechanism

Condition:
Any memory reference to rank r occurs.
Algorithm:
1: **if** rank r is in the lower-power state **then**
2: Set r to be ACT;
3: Serve the memory request;
4: Maintain performance counters; /*Section 4.2*/

Condition:
The length of the current idle period of rank r is updated.
Algorithm:
1: Perform demotions (if necessary) according to the demotion configuration for rank r;

Condition:
The processed number of memory requests equals the epoch size.
Algorithm:
1: Start a new epoch;
2: Estimate the maximum performance $\mathbb{T}_{min}$ of the new epoch; /*Section 4.2*/
3: Set the performance target $\mathbb{T}_{target}$ according to the control algorithm; /*Section 4.3*/
4: **for** each possible DFS-demotion configuration **do**
5: Estimate the total memory energy consumption and program performance; /*Section 4.2*/
6: Determine the suitable DFS-demotion configuration for the new epoch; /*Section 4.3 and 4.4*/
7: Adjust the memory frequency and each memory rank's power-down timeouts for the new epoch;
8: Reset performance counters;

is determined, we adjust the memory system with the new selected frequency, and set different power-down timeouts for different memory ranks according to the DFS-demotion configuration, and use this configuration for each rank for the entire epoch.

Let us briefly discuss some system implementation issues. Our adaptive scheme can be implemented with modest hardware and software supports. On the hardware side, we require the memory sub-system to have the ability of changing memory frequency (as proposed in previous studies [11], [12]) and demoting individual memory ranks to low-power states (already supported in current DRAM architectures) dynamically. Our design also relies on hardware performance counters in processors and memory controllers like previous studies [12], [21]. These counters can be read by the operating system as inputs for estimating memory energy and program performance of the next epoch. For example, a hardware interruption is triggered when the counter L equals the epoch size, which causes the operating system to estimate the suitable DFS-demotion configuration of a new start epoch. The amount of on-chip storage for these performance counters (about 132 bytes) is small compared to the multi-MByte shared LLC. On the software side, we offload some functionalities including new APIs for hardware performance counters and the prediction of DFS-demotion configurations to the operating system (like previous studies [10], [12], [30]). The operating system runs the prediction model and updates the DFS-demotion configuration for the memory controller (via setting corresponding registers) at the beginning of each epoch. We note that the structure complexity and storage overhead of our proposed design are similar to the previous proposals [10], [12], [16], [31].

TABLE 6
Performance Counters

Performance Counters	Description
L_r	The number of memory requests to memory rank r.
CNRR_r	The number of read requests to memory rank r.
CMRT_r	The average response time for a memory request in memory rank r.
CMLP_r	The maximal length of idle periods in memory rank r.
L	The total number of memory requests to all memory ranks.

We describe the implementation details of key components in the following sections.

4.2 Extended Performance-Energy Model

The analytical model in Section 3 is mainly for a single memory rank. In this subsection, we extend this model to estimate the total energy consumption of all memory ranks and to estimate the program execution time given a DFS-demotion configuration.

In each epoch, the system maintains a set of hardware performance counters. We list these performance counters in Table 6. An epoch ends when L reaches the pre-defined epoch size. With these counters, we leverage the analytic model developed in Section 3 and historical information in the previous epoch to predict the energy consumption of the memory system and the program performance for the new epoch. In particular, with the analytical model in Section 3, we model the expected energy consumption $E[\mathcal{E}](f_j, \vec{\Delta})$ and the expected response time $E[\mathcal{R}](f_j, \vec{\Delta})$ for a memory request. First, we need to calculate the average memory access rate λ_r for each memory rank r at the memory frequency f_0 in the previous epoch, and use it as the predicted λ_r for the new epoch. Given performance counters in Table 6 and the DFS-demotion configuration in the previous epoch, we set $E[\mathcal{R}](f_j, \vec{\Delta})$ to the actual average memory response time (CMRT_r) in the previous epoch, and get λ_r for each rank r by solving Eq. (4). This calculation is performed to all memory ranks. We denote the number of memory ranks as K, and $\vec{\Delta}_r$ ($r = 0, \ldots, K-1$) as power-down timeouts for memory rank r. Then, we can calculate the expected energy consumption $E[\mathcal{E}](f_j, \vec{\Delta}_r)$ and the expected response time $E[\mathcal{R}](f_j, \vec{\Delta}_r)$ for a memory request in memory rank r at memory frequency f_j and power-down timeouts $\vec{\Delta}_r$. Following the analytical model, the total memory energy consumption of all memory ranks $\mathbb{E}_{total}(f_j, \vec{\Delta}_0, \ldots, \vec{\Delta}_{K-1})$ is given by

$$\begin{aligned} \mathbb{E}_{rank}(r, f_j, \vec{\Delta}_r) &= \mathrm{L}_r \cdot E[\mathcal{E}](f_j, \vec{\Delta}_r) \\ \mathbb{E}_{total}(f_j, \vec{\Delta}_0, \ldots, \vec{\Delta}_{K-1}) &= \textstyle\sum_{r=0}^{K-1} \mathbb{E}_{rank}(r, f_j, \vec{\Delta}_r) \end{aligned} \tag{10}$$

where L_r is the number of memory requests to memory rank r in the previous epoch (i.e., $\mathrm{L} = \sum_{r=1}^{K-1} \mathrm{L}_r$). Given a certain memory frequency f_j, $\mathbb{E}_{rank}(r, f_j, \vec{\Delta}_r)$ calculates the total energy consumption at memory frequency f_j and with power-down timeouts $\vec{\Delta}_r$ for each memory rank r. Thus, the predicted total memory energy consumption of all ranks for the new epoch is given in $\mathbb{E}_{total}(f_j, \vec{\Delta}_1, \ldots, \vec{\Delta}_{K-1})$ by summing up each memory rank's energy.

Next, we decompose the program execution time to the total time of computation phases $\mathbb{T}_{cpu}$ and the total time of memory phases. $\mathbb{T}_{cpu}$ is insensitive to changes in DFS-demotion

configurations, while the total time of memory phases varies with memory frequencies as well as power-down timeouts. The estimated program performance $\mathbb{T}_{total}(f_j, \vec{\Delta}_1, \ldots, \vec{\Delta}_{K-1})$ is given by

$$\begin{array}{rcl} \mathbb{T}_{cpu} & = & \mathbb{T}_{actual} - max\{\mathrm{L}_r \cdot \mathrm{CMRT}_r\} \\ \mathbb{T}_{total}(f_j, \vec{\Delta}_1, \ldots, \vec{\Delta}_{K-1}) & = & \mathbb{T}_{cpu} + max\{\mathrm{L}_r \cdot E[\mathcal{R}](f_j, \vec{\Delta}_r)\} \end{array} \quad (11)$$

We record the actual epoch execution time $\mathbb{T}_{actual}$ at the end of each epoch. At the beginning of each epoch, we compute $\mathbb{T}_{cpu}$ by subtracting the maximum total time of memory phases of each memory rank (i.e., $\mathrm{L}_r \cdot \mathrm{CMRT}_r$ that recorded in performance counters) from $\mathbb{T}_{actual}$ of the previous epoch, and use it as the predicted total time of computation phases $\mathbb{T}_{cpu}$ for the new epoch. Then, we estimate the total length of memory phases in the new epoch as the maximum of $\mathrm{L}_r \cdot E[\mathcal{R}](f_j, \vec{\Delta}_r)$ ($r = 0, \ldots, K-1$). Combining them together, we obtain the estimated program performance $\mathbb{T}_{total}(f_j, \vec{\Delta}_1, \ldots, \vec{\Delta}_{K-1})$ at memory frequency f_j and power-down timeouts $\vec{\Delta}_r$ ($r = 0, \ldots, K-1$).

4.3 Slack-aware Control Algorithm

In order to ensure that a pre-defined performance target can be met over program execution, we use an adaptive control approach to perform adjustment on the performance budget at the beginning of each epoch. Following Deng et al. [12], we use slack to quantify the performance degradation. The *slack* is defined as the distance between the program's actual performance and the estimated performance as in Eq. (12).

$$\begin{array}{rcl} \mathbb{T}_{target} & = & \mathbb{T}_{min} \cdot (1 + D) \\ \mathbb{T}_{slack} & = & \mathbb{T}_{target} - \mathbb{T}_{actual} \end{array} \quad (12)$$

$\mathbb{T}_{target}$ is the target program performance for the new epoch. It has a pre-defined performance slowdown D (e.g., 10% performance loss) relative to the maximum program performance without any power management $\mathbb{T}_{min}$. With the analytical model developed in Section 4.2, $\mathbb{T}_{min}$ equals $\mathbb{T}(f_0, \vec{\Delta}_1, \ldots, \vec{\Delta}_{K-1})$, where $\vec{\Delta}_r = (0, \infty, \ldots, \infty)$, $r = 1, \ldots, K-1$. That is, $\mathbb{T}_{min}$ is the program execution time at the highest memory frequency (f_0) and without any demotions on all memory ranks. The target performance $\mathbb{T}_{target}$ is calculated based on $\mathbb{T}_{min}$. $\mathbb{T}_{actual}$ is the recorded actual program execution time of the previous epoch. Then, we get the performance slack using Eq. (12). The slack helps make performance adjustments among epochs. The accumulated slack is applied to the performance penalty budget of the new epoch. If it is larger than zero, we have a larger performance penalty budget. Thus, the control algorithm's optimization goal is defined as, *finding the optimal DFS-demotion configuration that minimizes the estimated total memory energy consumption* $\mathbb{E}_{total}(f_j, \vec{\Delta}_1, \ldots, \vec{\Delta}_{K-1})$*, and keeps the predicted program performance* $\mathbb{T}(f_j, \vec{\Delta}_1, \ldots, \vec{\Delta}_{K-1})$ *within the target performance* $\mathbb{T}_{target}$ *given the accumulated slack from previous epochs in the new epoch.*

$\mathbb{E}_{total}(f_j, \vec{\Delta}_1, \ldots, \vec{\Delta}_{K-1})$ is minimized if and only if each memory rank r's energy consumption $\mathbb{E}_{rank}(r, f_j, \vec{\Delta}_r)$ ($r = 1, \ldots, K-1$) is minimized. $\mathbb{T}(f_j, \vec{\Delta}_1, \ldots, \vec{\Delta}_{K-1})$ is satisfied if and only if each memory rank's extra latency is within the performance penalty budget.

4.4 Heuristics-based Search

Searching the optimal DFS-demotion configuration can be very costly. Even though the search process can be parallelized at the rank level, it is still a challenging task, particularly at the runtime. An exhaustive search is not feasible. The complexity of an exhaustive search for a memory rank is $O(M \cdot T^{N-1})$ steps (each step estimates the memory energy consumption and program performance for a DFS-demotion configuration), with M being the number of available frequencies, N being the number of available power states, and T being the number of possible values for a power-down timeout (e.g., the time length of an epoch). Thus, we need to explore heuristics to reduce the search time. In the following, we describe heuristics for DFS and demotion.

DFS heuristic. In Section 3.2, we have observed that the memory frequency in the optimal DFS-demotion configuration generally increases with the memory access rate. This motivates us to conduct the search along the dimension of memory frequency using a binary search with the hill-climbing optimization [32]. Specifically, the search starts at a mid-point memory frequency f (i.e., $f = f_i$, $i = \lfloor M/2 \rfloor$), and looks for the suitable power-down timeouts under f. Denote by ε the optimal energy consumption under frequency f. Then, another frequency f' that half-way between the current frequency and either of the endpoints is chosen, and the process is repeated. If the optimal energy consumption ε' for that f' is better, the binary search continues on that side, and the other side is disregarded. Otherwise, the algorithm switches to the other side (disregarding further attempts on the first side). When neither side is better, or we run out of choices, the search ends.

Demotion heuristic. An orthogonal way to reduce the search effort is the demotion dimension. Since low-power state transitions are nonzero cost processes, the *break-even* time $B_{i,j}$ denotes the minimum length of idle periods, which justifies a state transition to state S_i at memory frequency f_j, during which keeping the DRAM device in active state consumes the same amount of energy as transiting to state S_i and back to active state. In other words, $B_{i,j}$ characterizes the minimum idle length for energy-efficient state transitions. $B_{i,j}$ is given by

$$B_{i,j} = \frac{R_{i,j} \cdot P_{0,j}}{P_{0,j} - P_{i,j}} \quad (13)$$

$P_{i,j}$ and $R_{i,j}$ are the power consumption and resynchronization time of state S_i at memory frequency f_j, respectively. We keep a performance counter CMLP_r for each rank r to record the maximum idle period in the rank during an epoch, and use CMLP_r to predict the largest idle period length in the next epoch. Thus, state S_i can be disregarded directly if $B_{i,j}$ is larger than CMLP_r at frequency f_j for memory rank r.

Combining these two heuristics, we develop an efficient greedy algorithm with the branch-bound optimization to find the suitable demotion configuration for a memory rank in Algorithm 2. Given a certain frequency f_j, we first remove the low-power states, whose break-even times are larger than CMLP_r for rank r, from the set of available low-power states. The remaining eligible low-power states are kept in $\vec{S}_{eligible}$.

Algorithm 2 Greedy algorithm to find the demotion configuration $\vec{\Delta}_r$ for rank r

Input:
The memory frequency f_j, all low-power states set $\vec{S} = (S_1, \ldots, S_{N-1})$, with associated power consumptions set $\vec{P} = (P_{1,j}, \ldots, P_{N-1,j})$, and break-even times set $\vec{B} = (B_{1,j}, \ldots, B_{N-1,j})$;
Initialization:
$\vec{\Delta}_k = \Phi$, $\vec{S}_{select} = \Phi$, $\vec{S}_{eligible} = \Phi$;
for all $S_i \in \vec{S}$ **do**
  if $B_{i,j} \leq CMLP_r$ **then**
    Add S_i into $\vec{S}_{eligible}$;
$W = |\vec{S}_{eligible}|$;
while $|\vec{S}_{select}| \neq W$ **do**
  for all $S_i \in \vec{S}_{eligible}$ **do**
    Add S_i into $\vec{S}_{select}$;
    for each possible Δ_i (from CMLP$_r$ to 0) value **do**
      Calculate $\mathbb{E}_{rank}(r, f_j, \vec{\Delta}_r)$ using Eq. (10) with selected low-power states subset $\vec{S}_{select}$;
      if the program performance is violated **then**
        break;
    Find the suitable Δ_i that has the best $\mathbb{E}_{rank}(r, f_j, \vec{\Delta}_r)$;
    Remove S_i from $\vec{S}_{select}$;
  Find the low-power state S_p that has a best $\mathbb{E}_{rank}(r, f_j, \vec{\Delta}_r)$;
  Add Δ_p into $\vec{\Delta}_r$;
  Remove S_p from $\vec{S}_{eligible}$;
  Add S_p into $\vec{S}_{select}$;
for all $S_i \notin \vec{S}_{select}$ **do**
  $\Delta_i = \infty$;
  Add Δ_i into $\vec{\Delta}_r$;
Output:
Demotion times set $\vec{\Delta}_r$ at f_j frequency for rank r.

Based on $\vec{S}_{eligible}$, we choose the best low-power state and its power-down timeout which leads to the smallest estimated $\mathbb{E}_{rank}(r, f_j, \vec{\Delta}_r)$, while keeping the program performance within the budget. When finding the power-down timeout, it tries all possible values from the highest (CMLP$_r$) to the lowest (0) until the program performance is not satisfied. Next, we keep the estimated power-down timeout of the selected low-power state unchanged, and select a new low-power state and its power-down timeout from the rest eligible low-power states, which results in the smallest estimated $\mathbb{E}_{rank}(r, f_j, \vec{\Delta}_r)$ when two low-power states are applied. We repeat this process to add one more new low-power state into the previous selected subset of low-power states together with its power-down timeout in each step. Finally, we get the suitable power-down timeouts for all eligible low-power states. To further improve the prediction speed, we use an exponential search approach by iterating in the form of 2^i ($0 \leq i \leq \log_2 \text{CMLP}_r$) for each power-down timeout.

The complexity of the algorithm is $O(\log_2 M \cdot W^2 \cdot \log_2 \text{CMLP}_r)$ steps, which represents a significant improvement over the exhaustive search. W is the number of eligible low-power states, which should be much smaller than N in most cases. Thus, intuitively, the heuristic-based search should converge quickly to a good DFS-demotion configuration.

5 Evaluation

In this section, we present the quantitative evaluation of our proposed DRAM power management mechanism.

TABLE 7
Architectural characteristics of the simulated machine.

Component	Features
CPU	4 in-order core running at 2.667 GHz
TLB	64 entries
L1 I/D cache (per core)	48 KB
L2/L3 cache (shared)	256 KB/4 MB
Cache line/OS page size	64 B/4 KB
DRAM	DDR3 DRx4 R-DIMM at 1333 MHz [18]
ranks	8
capacity (GB)	8
power states	see Table 1
memory frequencies	10 frequencies (1333–133 MHz)

5.1 Methodology

We use a cycle-accurate simulator–PTLSim [33] to collect memory access traces (last-level cache misses and writebacks) from a variety of workloads, and replay the traces using our detailed memory system simulator. Our simulation models all the relevant aspects of the operating system, memory controller and DRAM devices, including page placements, memory channel, bank contention, row buffer management, DRAM device power and timing. The main architectural characteristics of the simulated machine are listed in Table 7. We evaluate our techniques with the DDR3 memory architecture, and simulate a 8GB memory system with 8 memory ranks.

Our settings for demotion and DFS are consistent with the previous studies [10], [11], [12]. By default, we consider the six power states supported in the current DDR3 architecture (DDR3 DRx4 R-DIMM) as shown in Table 1, and a wide range of memory frequencies: 1333, 1200, 1066, 934, 800, 667, 533, 400, 267 and 133 MHz. The default memory frequency is 1333 MHz. The timing and power parameters of DRAM chips at the default frequency are obtained from manufacturers' datasheet [18]. The background powers at the default frequency are obtained from the calculation by David et al. [11]. Parameters at other frequencies are scaled according to the previous study [11], [12].

Workloads. We have used 19 applications from SPEC 2006. These workloads have widely different memory access rates, footprints and localities. To assess our algorithm under the context of multi-core CPUs, we study various mixed workloads of four different applications from SPEC 2006 (Table 8). The four applications start at the same time. The mixed workloads form multi-programmed executions on a four-core CPU, ordered by the average number of memory accesses ($Mean$). The standard deviation and mean values are calculated based on memory access statistics per 5×10^8 CPU cycles. For each workload, we select the simulation period of 10^{10} CPU cycles in the original PTLSim simulation (at the default memory frequency and no demotions), which represents a stable and sufficiently long execution behavior.

We use the optimization goal of minimizing the total memory energy consumption while keeping the performance penalty within a predefined budget in this section. Due to space limitations, we do not present the results for all single applications. Instead, we report their geometric mean (**GM**), and also five representative applications: omnetpp, zeusmp, cactusADM, libquantum and mcf (denoted as S1, S2, S3, S4 and S5, respectively). They cover a wide range of memory

TABLE 8
Mixed workloads: memory footprint (FP), memory accesses statistics per 5×10^8 cycles ($Mean$ and $\frac{Stdev}{Mean}$).

Name	FP (MB)	$Mean$ (10^6)	$\frac{Stdev}{Mean}$	Applications
M1	661.3	0.6	1.02	gromacs, gobmk, hmmer, bzip
M2	1477.4	1.7	1.11	bzip, soplex, sjeng, cactusADM
M3	626.6	2.9	0.59	soplex, sjeng, gcc, zeusmp
M4	537.8	3.5	0.47	zeusmp, gcc, leslie3d, omnetpp
M5	1082.9	4.4	0.71	gcc, leslie3d, calculix, gemsFDTD
M6	1250.5	8.7	0.40	libquantum, xalan, gemsFDTD, zeusmp

accesses intensiveness (0.1, 0.9, 1.0, 4.7, 8.0 millions accesses on average per 5×10^8 CPU cycles, respectively).

Comparisons. We compare our DRAM power management mechanism (denoted as *Hybrid*) with a number of baseline and state-of-the-art DRAM power management schemes. Here are the details about the schemes in comparison:

- **No Power Management (BASE)**: The memory frequency is fixed at the default 1333 MHz and memory ranks are always kept active even when they are idle.
- **RAMZzz**: RAMZzz is one of the state-of-the-art approaches using demotion [10]. For fair comparison, we disable page migrations in RAMZzz (as explained in Section 2). RAMZzz only considers two pre-selected low-power states on DDR3 (PRE_PDN_FAST and SR_FAST).
- **RAMZzz+**: RAMZzz+ is an enhanced version of RAMZzz, which explores state transitions among all available power-down states on DDR3.
- **MemScale**: MemScale is one of the state-of-the-art approaches using DFS [12].
- **MemScale+SFD**: MemScale+SFD is a policy that combines DFS with a static demotion scheme. A memory rank immediately demotes to PRE_PDN_FAST when it is idle. This approach was adopted in the previous study [12].
- **MemScale+DFD**: MemScale+DFD is similar to MemScale+SFD, except that MemScale+DFD chooses the power-down timeout according to our control algorithm. MemScale+DFD is different from Hybrid, where it only uses a pre-selected low-power state for demotions.
- **Hybrid**: Hybrid is the power management approach developed in this paper.

We allow users to specify the epoch sizes and penalty budgets. By default, the epoch size is set to 10^6 memory requests, and penalty budget is set to 10%.

We first compare the behavior of Hybrid, BASE, MemScale+SFD, and MemScale+DFD to show the adaptivity of Hybrid under different workloads (Section 5.2.1). Then, we compare Hybrid with MemScale, RAMZzz and RAMZzz+ in order to evaluate the impact of individual techniques, and the synergy between DFS and demotion (Section 5.2.2). Third, we investigate the effectiveness of the proposed search heuristics (Section 5.3). We also study the impact of Hybrid on full system energy savings, and present some additional results for the optimization goal of ED^2 (memory subsystem energy $\times$ program execution time^2). Due to the space limitation, we put these results in Appendix A and B of the supplementary file. *All the results are normalized by those of BASE.*

5.2 Results on Energy Optimizations

5.2.1 Overall Comparison

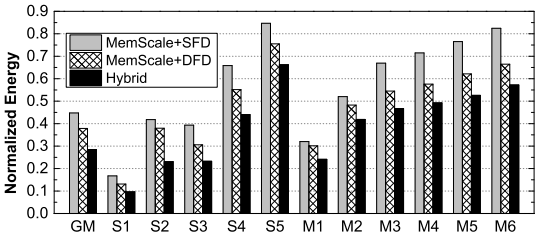


Fig. 4. Overall comparisons on energy consumptions.

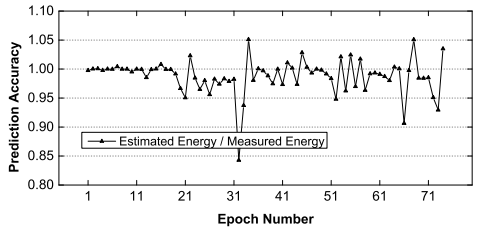
Figure 4 shows the normalized total memory energy consumption of Hybrid in comparison with MemScale+SFD, and MemScale+DFD when the penalty budget is set at 10%. The comparison with MemScale+SFD and MemScale+DFD shows the effectiveness of the adaptive feature of our proposed method. If the normalized energy consumption of an approach is smaller than 1.0, the approach is more energy efficient than BASE.

Thanks to the adaptive combination of DFS and demotion techniques, Hybrid is significantly more energy-efficient than BASE, with an average reduction of 67% in total energy consumption. The reduction is more significant for the workloads with less intensive memory accesses (such as S1, S2 and M1). This is because idle periods are generally longer for less memory-intensive workloads, and lower memory frequencies are feasible for those workloads for saving more background power.

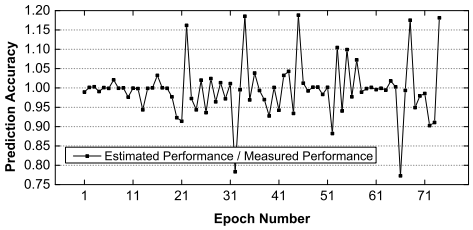
Comparing to the static scheme of DFS and demotion, Hybrid outperforms MemScale+SFD for all workloads, with an average reduction of 38% in total energy consumption. The fixed power-down timeouts in MemScale+SFD cannot adapt to different workloads when the memory frequency is changing. Furthermore, the gap between MemScale+SFD and Hybrid becomes larger for more memory-intensive workloads (such as S5 and M6). The aggressive demotion of MemScale+SFD can hurt energy efficiency because of the high latency and energy penalty of demotions on short idle periods. This penalty is even larger on memory-intensive workloads. We make a further study to compare MemScale+DFD with MemScale+SFD. MemScale+DFD is more energy-efficient than MemScale+SFD, which demonstrates that adaptively choosing the suitable DFS-demotion configuration yields significant improvement over static DFS-demotion schemes.

Hybrid also has larger energy savings compared with MemScale+DFD, with an average reduction of 26% in total energy consumption. Though MemScale+DFD chooses the power-down timeout according to our control algorithm, it only considers a single pre-selected low-power state (i.e., PRE_PDN_FAST). However, Hybrid explores five available low-power states in DDR3 architectures. Also, the demotion schemes are different for MemScale+DFD and Hybrid. Adding more low-power states brings a larger search space of DFS-demotion configurations for maximizing energy savings.

We also perform detailed studies on our analytical model. Figures 5(a) and 5(b) show the ratios of estimated energy/performance (predicted by our analytical model at the beginning



(a) Energy

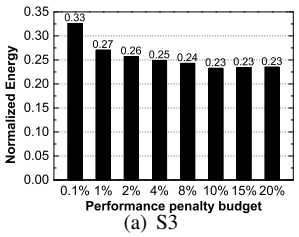


(b) Performance

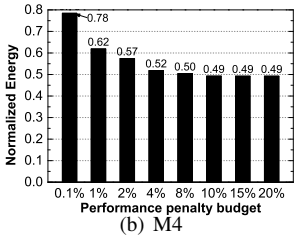
Fig. 5. Comparing estimated energy/performance with measurements of Hybrid on M4.

of an epoch) and measured energy/performance (measured at the end of the corresponding epoch) for different epochs of M4. Our estimations are very close to real measurements at different DFS-demotion settings. We observe similar results for different workloads.

We study the performance delay in more details. Figure 6 shows the breakdown of the performance delay. We divide the delay penalty into two parts, resynchronization delay (caused by state transitions) and frequency scaling delay (caused by DFS, including memory access delay and frequency switching penalty). Though Hybrid and MemScale+DFD show different breakdowns of the performance delay due to different demotion schemes, the delays of Hybrid and MemScale+DFD are well controlled under the pre-defined penalty budget (i.e., 10% in this experiment). On the other hand, MemScale+SFD cannot limit the delay within the penalty budget for workloads S5 and M6. This demonstrates the effectiveness of our control algorithm. Additionally, the resynchronization delay and frequency scaling delay vary significantly across different workloads in our Hybrid scheme. For example, the resynchronization delay is much higher than the frequency scaling delay for workload S5, which indicates that state transitions are more often for saving the DRAM power. These observations demonstrate the effectiveness of adaptive DFS-demotion configurations for different workloads.



(a) S3



(b) M4

Fig. 7. Energy consumption of Hybrid on S3 and M4.

One might think that Hybrid could have even higher energy savings, if it could keep the frequency lower and power-down timeouts smaller and approximate the performance penalty budget more closely. However, approximating the performance penalty budget more closely could also increase the DRAM energy consumption (by increasing the memory read/write energy and the resynchronization energy significantly). Thus, our policy degrades the performance only up (and sets the DFS-demotion configuration) to the point that results in the minimized total memory system energy.

Figure 7 illustrates the normalized energy consumption of Hybrid for different performance penalty budgets on S3 and M4. The normalized energy consumption is decreased when varying the penalty budget from 0.1% to 20%. A small penalty budget limits the potential for energy savings.

5.2.2 Individual Impacts

We now evaluate the individual impacts of DFS and demotion in Hybrid.

Impact of demotion. We study the impact of demotion by comparing Hybrid, MemScale (no demotion), and MemScale+SFD (a static demotion scheme), as shown in Figure 8. Hybrid has much lower energy consumption than MemScale, with a range of 20–66% reduction in total energy consumption.

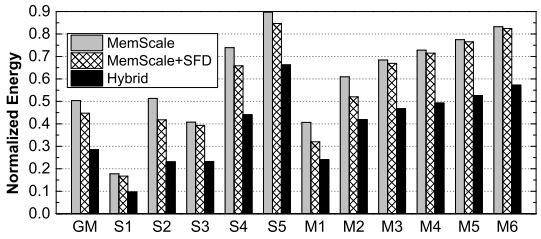
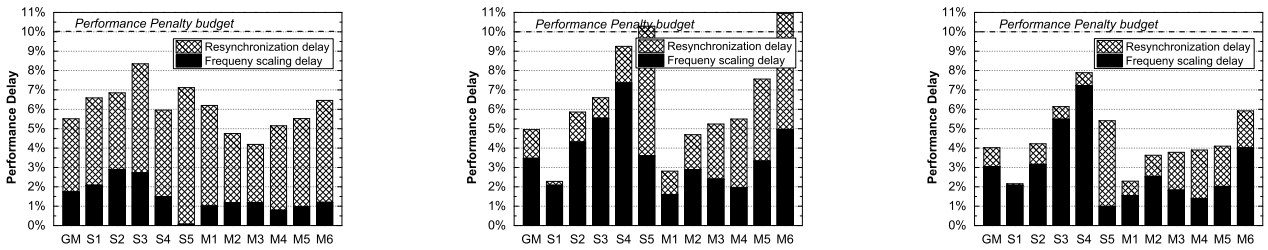


Fig. 8. Energy consumption of Hybrid, MemScale and MemScale+SFD.

We further have the following two major observations. First, compared with MemScale, Hybrid can save more background power by considering low-power states. The reduction is also encouraging for memory-intensive workloads (such as S4, S5, M5 and M6). Even though idle periods are shortened after applying DFS on those workloads, Hybrid can still use those lower-power states with short resynchronization times. Second, applying a static demotion scheme with DFS shows only marginal improvement on the energy efficiency. The normalized energy consumption of MemScale+SFD is rather close to that of MemScale. This is consistent with the previous study [12].

Impact of DFS. Figure 9 shows the energy consumption results for Hybrid, RAMZzz, and RAMZzz+. Comparing to RAMZzz, Hybrid decreases the energy consumption by 31% on average. Though lowering the memory frequency may shorten the idle periods of ranks, Hybrid can still achieve significant improvement by adaptively choosing the optimal DFS-demotion configuration.

RAMZzz+ is comparable to Hybrid for memory-intensive workloads (such as S5, M5 and M6). Lower memory frequencies are not suitable for those workloads, due to the cost of increased operational energy and memory access time. Thus, demotions play a more significant role for saving the energy consumption of DRAM. On the other hand, Hybrid can exploit much lower memory frequencies for workloads with less memory access intensiveness. Overall, Hybrid has a



(a) Hybrid (b) MemScale+SFD (c) MemScale+DFD

Fig. 6. The breakdown of performance delay.

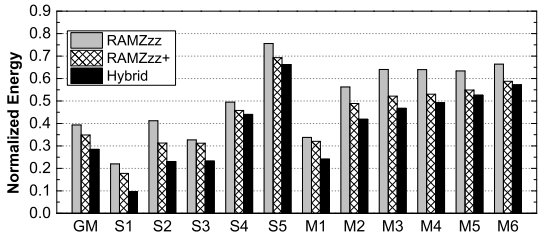


Fig. 9. Energy consumption of Hybrid, RAMZzz and RAMZzz+.

lower energy consumption than RAMZzz+, with an average energy reduction of 21%.

Compared with RAMZzz, RAMZzz+ further decreases the energy consumption, which justifies the necessity of involving more low-power states for demotions. RAMZzz only uses two pre-selected low-power states, and loses the opportunity of adapting to different workloads.

5.3 Effectiveness of Search Heuristics

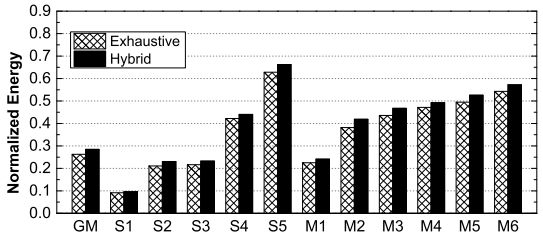


Fig. 10. Energy consumption of Hybrid and Exhaustive.

To study the effectiveness of our search heuristics, we simulate a variant of Hybrid, denoted as *Exhaustive*, which finds the optimal DFS-demotion configuration by searching the entire optimization space. Still, Exhaustive adopts branch-bound optimizations to reduce unnecessary search efforts. That is, at a certain frequency, the search of power-down timeouts starts from high to low until the target performance is violated. As shown in Figure 10, Hybrid achieves a very close energy consumption to Exhaustive for all workloads. This indicates that our proposed heuristics achieve near-optimal energy savings with reduced search space.

We further perform a detailed analysis on the effectiveness of these optimization heuristics for a selected workload M1. We compare the average computational time and normalized energy consumption for the Hybrid and Exhaustive approaches in Table 9 for different numbers of power states (denoted as n) and memory frequencies used (denoted as m). For a given n, we consider S_0 (*active* state) and other low-power states S_i $(1 \leq i < n)$. Similarly, for a given m, we consider the highest memory frequency 1333 MHz and other frequencies $f_i = 1333 - 133 \times i$ MHz $(1 \leq i < m)$. The average computational time is the average simulation time of finding the DFS-demotion configuration for an epoch. We show the speedup of the average computational time for Hybrid over Exhaustive.

As the number of power states and frequencies used increases, the normalized energy consumption becomes lower for both Hybrid and Exhaustive (from left to right, and top to bottom in Table 9). This further shows the benefits of the self-adapting feature brought by our proposed adaptive DFS-demotion configurations. Hybrid has a slightly higher energy consumption than Exhaustive in all cases, within the range of 3–8%. The average computational time of Hybrid is improved significantly over that of Exhaustive, which significantly reduces the runtime overhead.

6 Conclusions

Effectively exploiting power management techniques is critical for reducing DRAM energy consumption. In this paper, we propose a novel DRAM power management design by adaptively combining DFS and demotion. An analytical model is developed to understand the synergy between DFS and demotion. Based on the analytical model, we develop optimization techniques to efficiently search for good DFS-demotion configurations at the runtime. We further develop simple and effective heuristics to reduce the computational complexity of optimization. We evaluate our proposed models and optimizations with SPEC 2006 in comparison with baseline and state-of-the-art power saving techniques. Our simulation results demonstrate significant improvement of our mechanism in energy consumption over other power saving techniques.

Acknowledgement

The authors would like to thank anonymous reviewers for their insightful comments. This work is supported by Program for Changjiang Scholars and Innovative Research Team in University (IRT1158, PCSIRT) China. This work is also partly supported by a MoE AcRF Tier 2 grant (MOE2012-T2-1-126) in Singapore.

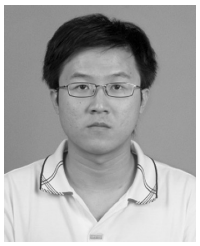
References

[1] F. Färber and et al., "Sap hana database: data management for modern business applications," *SIGMOD Rec.*, vol. 40, no. 4, 2012.

TABLE 9
Speedup (**Speedup**) and normalized energy consumption (**E**) of Hybrid (**HB**) and Exhaustive (**EX**) on M1.

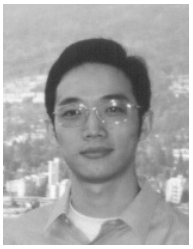
	$m=3$			$m=6$			$m=10$		
n	**E (HB)**	**E (EX)**	**Speedup**	**E (HB)**	**E (EX)**	**Speedup**	**E (HB)**	**E (EX)**	**Speedup**
3	0.51	0.49	72	0.39	0.37	92	0.30	0.29	145
4	0.36	0.35	1275	0.30	0.29	1586	0.27	0.25	2475
6	0.30	0.28	23027	0.26	0.25	35922	0.24	0.23	45824

[2] J. Ousterhout and et al., "The case for ramclouds: scalable high-performance storage entirely in dram," *SIGOPS Oper. Syst. Rev.*, vol. 43, no. 4, 2010.
[3] C. Lefurgy, K. Rajamani, F. Rawson, W. Felter, M. Kistler, and T. W. Keller, "Energy management for commercial servers," *IEEE Computer*, vol. 36, no. 12, 2003.
[4] U. Hoelzle and L. A. Barroso, *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*, 1st ed. Morgan and Claypool Publishers, 2009.
[5] D. Meisner, B. T. Gold, and T. F. Wenisch, "Powernap: eliminating server idle power," in *ASPLOS '09*, 2009.
[6] M. S. Ware and et al., "Architecting for power management: The ibm power7 approach," in *HPCA '10*, 2010.
[7] X. Fan, C. Ellis, and A. Lebeck, "Memory controller policies for dram power management," in *ISLPED '01*, 2001.
[8] V. Delaluz and et al., "Scheduler-based dram energy management," in *DAC '02*, 2002.
[9] H. Huang, P. Pillai, and K. G. Shin, "Design and implementation of power-aware virtual memory," in *USENIX ATC '03*, 2003.
[10] D. Wu, B. He, X. Tang, J. Xu, and M. Guo, "Ramzzz: rank-aware dram power management with dynamic migrations and demotions," in *SC '12*, 2012.
[11] H. David and et al., "Memory power management via dynamic voltage/frequency scaling," in *ICAC '11*, 2011.
[12] Q. Deng and et al., "Memscale: active low-power modes for main memory," in *ASPLOS '11*, 2011.
[13] I. Hur and C. Lin, "A comprehensive approach to dram power management," in *HPCA '08*, 2008.
[14] B. Diniz, D. Guedes, W. Meira, Jr., and R. Bianchini, "Limiting the power consumption of main memory," in *ISCA '07*, 2007.
[15] K. Sudan, K. Rajamani, W. Huang, and J. Carter, "Tiered memory: An iso-power memory architecture to address the memory power wall," *IEEE Trans. on Comput.*, vol. 61, no. 12, 2012.
[16] H. Huang and et al., "Improving energy efficiency by making dram less randomly accessed," in *ISLPED '05*, 2005.
[17] H. Zheng and Z. Zhu, "Power and performance trade-offs in contemporary dram system designs for multicore processors," *IEEE Trans. on Comput.*, vol. 59, no. 8, 2010.
[18] Micron Tech., Inc., *MT41J256M4JP-15E Datasheet*, 2010.
[19] Micron Tech., Inc., *MT42L128M32D1LF-25WT Datasheet*, 2011.
[20] Micron Tech., Inc., *System Power Calculator*, http://www.micron.com/products/support/power-calc, 2012.
[21] Q. Deng, D. Meisner, A. Bhattacharjee, T. Wenisch, and R. Bianchini, "Coscale: Coordinating cpu and memory system dvfs in server systems," in *MICRO '45*, 2012.
[22] B. He, Q. Luo, and B. Choi, "Cache-conscious automata for xml filtering," *IEEE Trans. on Knowl. and Data Eng.*, vol. 18, no. 12, 2006.
[23] B. He and Q. Luo, "Cache-oblivious databases: Limitations and opportunities," *ACM Trans. Database Syst.*, vol. 33, no. 2, 2008.
[24] H. Liu, H. Jin, X. Liao, W. Deng, B. He, and C.-z. Xu, "Hotplug or ballooning: A comparative study on dynamic memory management techniques for virtual machines," *IEEE Trans. on Parall. and Distrib. Syst.*, vol. PP, no. 99, 2014.
[25] F. Kong, Y. Wang, Q. Deng, and W. Yi, "Minimizing multi-resource energy for real-time systems with discrete operation modes," in *ECRTS '10*, 2010.
[26] V. Devadas and H. Aydin, "On the interplay of voltage/frequency scaling and device power management for frame-based real-time embedded applications," *IEEE Trans. on Comput.*, vol. 61, no. 1, 2012.
[27] M. E. T. Gerards and J. Kuper, "Optimal dpm and dvfs for frame-based real-time systems," *ACM Trans. Archit. Code Optim.*, vol. 9, no. 4, 2013.
[28] P. D. Welch, "On a generalized m/g/1 queuing process in which the first customer of each busy perido receives exceptional service," *Operations Research*, vol. 12, no. 1, 1964.
[29] Micron Tech., Inc., *TN-41-01: Calculating Memory System Power for DDR3*, 2007.
[30] L. E. Ramos, E. Gorbatov, and R. Bianchini, "Page placement in hybrid memory systems," in *ICS '11*, 2011.
[31] K. Sudan and et al., "Micro-pages: increasing dram efficiency with locality-aware data placement," in *ASPLOS '10*, 2010.
[32] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2003.
[33] M. T. Yourst, "Ptlsim: A cycle accurate full system x86-64 microarchitectural simulator," in *ISPASS '07*, 2007.



Yanchao Lu received the BS degree in computer science and technology from Beijing Institute of Technology, China, in 2010. He is currently a fourth year Ph.D. student at Department of Computer Science and Engineering, Shanghai Jiao Tong University, China. His research interests include low-power system design, GPGPU, parallel and distributed systems.

Bingsheng He received the bachelor degree in computer science from Shanghai Jiao Tong University (1999-2003), and the PhD degree in computer science in Hong Kong University of Science and Technology (2003-2008). He is an assistant professor in Division of Networks and Distributed Systems, School of Computer Engineering of Nanyang Technological University, Singapore. His research interests are high performance computing, distributed and parallel systems, and database systems.



Xueyan Tang received the BEng degree in computer science and engineering from Shanghai Jiao Tong University in 1998, and the PhD degree in computer science from the Hong Kong University of Science and Technology in 2003. He is currently an associate professor in the School of Computer Engineering at Nanyang Technological University, Singapore. He has served as an associate editor of IEEE Transactions on Parallel and Distributed Systems. His research interests include distributed systems, mobile and pervasive computing, and wireless sensor networks. He is a senior member of the IEEE.

Minyi Guo received the BS and ME degrees in computer science from Nanjing University, China, in 1982 and 1986, respectively, and the PhD degree in information science from the University of Tsukuba, Japan, in 1998. From 1998 to 2000, he had been a research associate of NEC Soft, Ltd. Japan. He was a visiting professor at the Department of Computer Science, Georgia Institute of Technology. He was a full professor at the University of Aizu, Japan, and is the head of the Department of Computer Science and Engineering at Shanghai Jiao Tong University, China. He has served as an associate editor of IEEE Transactions on Computers and IEEE Transactions on Parallel and Distributed Systems. His research interests include automatic parallelization and data-parallel languages, bioinformatics, compiler optimization, high-performance computing, and pervasive computing. He is a senior member of the IEEE and has published more than 150 papers in well-known conferences and journals.