Rank-Aware Dynamic Migrations and Adaptive Demotions for DRAM Power Management

Yanchao Lu, Donghong Wu, Bingsheng He, Xueyan Tang, Jianliang Xu and Minyi Guo

Abstract—Modern DRAM architectures allow a number of low-power states on individual memory *ranks* for advanced power management. Many previous studies have taken advantage of demotions on low-power states for energy saving. However, most of the demotion schemes are statically performed on a limited number of pre-selected low-power states, and are suboptimal for different workloads and memory architectures. Even worse, the idle periods are often too short for effective power state transitions, especially for memory intensive applications. Wrong decisions on power state transition incur significant energy and delay penalties. In this paper, we propose a novel memory system design named RAMZzz with rank-aware energy saving optimizations including dynamic page migrations and adaptive demotions. Specifically, we group the pages with similar access locality into the same rank with dynamic page migrations. Ranks have their hotness: hot ranks are kept busy for high utilization and cold ranks can have more lengthy idle periods for power state transitions. We further develop adaptive state demotions by considering all low-power states for each rank and a prediction model to estimate the power-down timeout among states. We experimentally compare our algorithm with other energy saving policies with cycle-accurate simulation. Experiments with benchmark workloads show that RAMZzz achieves significant improvement on energy-delay^2 and energy consumption over other energy saving techniques.

Index Terms—Demotion, Energy consumption, Main memory systems, In-memory processing, Page migrations

1 Introduction

Energy consumption has become a major factor for the design and implementation of computer systems. Inside many computing systems, main memory (or DRAM) is a critical component for the performance and energy consumption. The 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 [1], [2]. For example, it has been reported that main memory contributes to as much as 40–46% of total energy consumption in server applications [2], [3], [4]. For these reasons, this paper studies the energy saving techniques of main memory.

Current main memory architectures allow power management on individual memory ranks. Individual ranks at different power states consume different amounts of energy. There have been various energy-saving techniques on exploiting the power management capability of main memory [5], [6], [7], [8]. The common theme of those research studies is to exploit the transition of individual memory ranks to low-power states (i.e., *demotion*) for energy saving. Fan et al. concluded that immediate transitions on Direct Rambus DRAM (RDRAM) to the low-power state saved the most energy consumption for most single-application workloads [9]. However, the decision can be wrong for more memory intensive workloads such as multi-programmed executions, and for different memory architectures. Existing techniques are suboptimal in the following aspects: (1) they do not effectively extend the idle period, either with static page placement [9], [10] or with heuristics-based page migrations [5], [6]; (2) the prediction on the power-down timeout (*the amount of time spent since the beginning of an idle period before transferring to a low-power state*) for a state transition is limited and static, either with heuristics [5], [6] or regression-based model [9]; (3) most of the demotion schemes are statically performed on a limited number of pre-selected low-power states (e.g., Huang et al. [6] selects two low-power states only, out of five in DDR3). The static demotion scheme is suboptimal for different workloads and different memory architectures.

To address the aforementioned issues, we propose a novel memory design named RAMZzz with rank-aware power management techniques including dynamic page migrations and adaptive demotions. Instead of having static page placement, we develop dynamic page migration mechanisms to exploit the access locality changes in the workload. As a result, ranks are categorized into hot and cold ones. The hot rank is highly utilized and has very short idle periods. In contrast, the cold rank has a relatively small number of long idle periods, which is good for power state transitions for energy saving.

Instead of adopting static demotion schemes, we develop adaptive demotions to exploit the power management capabilities of all low-power states for individual ranks. The decisions are guided by a prediction model to estimate the idle period distribution. The prediction model is specifically designed with the consideration of page migrations among ranks. Based on the prediction model, RAMZzz is able to optimize for different goals such as energy saving and energy-delay^2 (ED^2). In this paper, we focus on the optimization goal of minimizing ED^2 (or energy consumption) of the memory system while

- *Y. Lu, D. Wu and M. Guo are with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China.*
- *B. He and X. Tang are with the School of Computer Engineering, Nanyang Technological University, Nanyang Ave, Singapore 639798.*
- *J. Xu is with the Department of Computer Science, Hong Kong Baptist University, Kowloon Tong, Hong Kong.*

keeping the program performance penalty within a pre-defined performance slowdown relative to the maximum performance without any power management (e.g., 10% performance loss).

We evaluate our design using detailed simulations of different workloads including SPEC 2006 and PARSEC [11]. We evaluate RAMZzz in comparison with representative power saving policies [6], [10], [12] and an ideal oracle approach. Our experiments with the optimization goal of ED^2 (for a maximum acceptable performance degradation of 4%) on three different DRAM architectures show that (1) both page migrations and adaptive demotions well adapt to the workload; (2) with both page migrations and adaptive demotions, RAMZzz achieves an average ED^2 improvement of 45.2–64.2% over the basic approach without power management, and achieves only 3.7–5.7% on average larger ED^2 than the ideal oracle approach. The experiments with the optimization goal of energy consumption have demonstrated similar results.

Organization. The rest of the paper is organized as follows. We introduce the background on basic power management of DRAM and review related work in Section 2. Section 3 gives an overview of RAMZzz design, followed by detailed implementations in Section 4. 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

In this paper, we use the terminology of DDR-series memory architectures (e.g., DDR2 and DDR3 etc) to describe our approach. DDR is usually packaged as modules, or DIMMs. Each DIMM contains multiple ranks. In power management, a rank is the smallest physical unit that we can control. Individual ranks can service memory requests independently and also operate at different *power states*. The power consumption of a memory rank can be divided into two main categories: active power and background power. Active power consists of the power that is required to activate the banks and service memory reads and writes. Background power is the power consumption without any DRAM accesses. Background power is a major component in the total DRAM power consumption, and tends to be more significant in the future [6], [13]. For example, Huang et al. [6] found that the background power contributes to 52% of the total DRAM power in their evaluation. Therefore, we focus on reducing the background power consumption.

Depending on which hardware components are disabled, modern memory architectures support a number of power states with complicated transitions [14], [15]. Each state is characterized with its power consumption and the time that it takes to transition back to the active state (resynchronization time). Typically, the lower power consumption the low-power state has, the higher the resynchronization time is. Table 1 summarizes the major power state transitions of three typical DRAM architectures: DDR3, DDR2 and LPDDR2. For each state, we show its dynamic power consumption (normalized to that of ACT) and the resynchronization times back to ACT. The power consumption values are calculated with DRAM System Power Calculator [17]. The resynchronization times are obtained from DRAM manufacturers' data sheets [14], [15], [16].

TABLE 1
Power states for three typical DRAM architectures.

Power State	Normalized Power	Resynchronization Time (ns)
DDR3 DRx4 at 1333 MHz [14]		
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
DDR2 DRx8 at 800 MHz [15]		
ACT	1.0	0
ACT_PDN_FAST	0.619	5
ACT_PDN_SLOW	0.325	18
PRE_PDN	0.237	25
SR	0.178	500
LPDDR2 DRx16 at 800 MHz [16]		
ACT	1.0	0
ACT_PDN	0.523	8
PRE_PDN	0.303	26
SR	0.194	100

From Table 1, we have the following observations on state demotions on different memory architectures.

First, on a specific memory architecture, power states have quite different latency and energy penalties as well as different power consumptions. Second, different memory architectures have their own specifications on power states as well as power state energy consumption and resynchronization time. First, different memory architectures may have different sets of power states. For example, DDR3 has a special low-power state, i.e., self-refresh with slow exit state (SR_SLOW), whereas DDR2 and LPDDR2 do not have any equivalent state. SR_SLOW has a very high resynchronization time and consumes only 10% of the power of ACT. Second, the energy consumption or the resynchronization time of the same power state can vary for different memory architectures. Take self-refresh states (SR) as an example. While SR consumes a similar normalized power consumption for the three architectures (about 17–19%), the resynchronization time varies significantly. The resynchronization times on DDR3, DDR2, LPDDR2 are 768ns (SR_FAST), 500ns (SR) and 100ns (SR), respectively.

The above-mentioned observations have significant implications to DRAM power management design.

First, the above-mentioned observations clearly show the deficiency of the static demotion schemes [5], [6], [7], [9]. The static demotion schemes are performed on the pre-selected low-power states (even for all ranks in the same architecture, and for different memory architectures). On a specific memory architecture, the static decision loses the opportunities for demoting to the most energy-effective low-power state for different idle period lengths. Moreover, with different memory architectures, the static decision loses the opportunities for adapting to different memory architectures.

Second, because the latency and energy penalty for switching from deeper low-power states is substantially higher than the penalty of switching from shallower states, entering deep power-down states for short idle times could in fact hurt energy efficiency because the power savings might not be able to offset the high latency penalty of switching back to the active state.

The design of RAMZzz are guided by the aforementioned two implications. It embraces dynamic migrations and adaptive demotions, adapting to different workloads and different memory architectures.

2.2 Related Work

Different power state transition approaches have been developed for DRAM systems. Hur et al. [18] developed adaptive history-based scheduling in the memory controller. Based on page migration, Huang et al. [6] stored frequently-accessed pages into hot ranks and left infrequently-used and unmapped pages on cold ranks. Their decisions on page migrations are based on heuristics. Lebeck et al. [12] studied different page allocation strategies. Their approach does not have any analytical model to guide the decision, or utilize both recency and frequency to capture rank hotness. Our prediction model offers a novel way of power management on guiding page migrations and power state transitions. Fan et al. [9] developed an analytic model on estimating the idle time of Direct Rambus DRAM (RDRAM) chips using an exponential distribution. Their model did not consider page migrations. For demotions, they adopted a simple approach that, when there is an idle period, immediate transitions on RDRAM to the low-power state are made. However, the decision can be wrong for memory intensive workloads such as multi-programmed executions, and for different memory architectures. Kshitij et al. [19] used a similar page migration mechanism between cold and hot ranks, but always set cold ranks with a pre-selected low-power state. Instead of relying on the presumed knowledge of distribution, our prediction model combines the historical information on idle period distribution and page access locality. More importantly, compared with all previous studies that pre-define a number of fixed states for all ranks [6], [9], [10], [12], [18], [19], this paper develops adaptive demotions to exploit the energy-saving capabilities of all power states, and the adaptation is on the granularity of individual ranks for different memory architectures.

DRAM power state transitions have been implemented in operating systems and compilers. Delaluz et al. [7] present an operating system based solution letting the scheduler decide the power state transitions. This approach requires the interfaces of exposing and controlling the power states. Huang et al. [5] proposed power-aware virtual memory systems. For energy efficient compilations, Delaluz et al. [20] proposed compiler optimizations for memory energy consumption of array allocations. They further combined the hardware-directed approach and compiler-directed approaches [21] for more energy saving.

There are other approaches for reducing the DRAM power consumption. We review three representative categories. The first category is to reduce the active power consumption. Fang et al. [22] suggested the subdivision of a conventional DRAM rank into mini-ranks comprising of a subset of DRAM devices to improve DRAM energy efficiency. Anh et al. [23] proposed Virtual Memory Devices (VMDs) comprising of a small number of DRAM chips. Decoupled DIMMs [24] proposed the DRAM devices at a lower frequency than the memory channel to reduce DRAM power. The second category is to reduce the power consumption of power state transitions. Bi et al. [25] took advantage of the I/O handling routines in the OS kernel to hide the delay incurred by memory power state transitions. Balis et al. [26] proposed finer grained memory state transition. The third category is to adjust the voltage and frequency of DRAM. Memory voltage and frequency scaling (DVFS) is a recent approach to reduce DRAM energy consumption [27], [28]. Lu et al. [29] conducted a comprehensive study on the synergy between DVFS and demotion on DRAM architectures. Those approaches are complementary to the state transition-based energy saving approaches.

Recently, different architectural designs of DRAM systems [13], [23], [30], [31] are explored on multi-core processors for performance, energy, reliability and other issues. Cache-centric optimizations (either cache-conscious [32] or cache-oblivious [33], [34]) reduce memory access and create more opportunities for energy saving. Besides optimizations targeting at general DRAM systems, some researchers have also proposed energy saving techniques for specific applications such as databases [8], [35] and video processing [35].

A preliminary version of RAMZzz has been presented in a previous paper [36]. This paper goes beyond the preliminary version with two major improvements. First, we have enhanced the cost model and the design of RAMZzz with adaptive demotions. Adaptive demotions can exploit energy-saving capabilities of all power states for different memory architectures and different workloads, and the adaptation is on the granularity of individual memory ranks. Second, we have extended the evaluation to study the effectiveness of RAMZzz on three DRAM architectures (i.e., DDR2, DDR3 and LPDDR2), and demonstrated the self-tuning feature of RAMZzz for different workloads (SPEC 2006 and PARSEC) and different memory architectures. Note, our preliminary version [36] evaluates only SPEC 2006 on DDR3.

3 Design Overview

In this section, we give an overview of the design rationales and workflow of RAMZzz.

3.1 Motivations

Our goal is to reduce the background power of DRAM. Due to the inherent power management mechanisms of DRAM, there are three obstacles in the effectiveness of reducing the background power.

First, due to the latency and power penalty of transiting from low-power state to active state, it requires a minimum length threshold for an idle period that is worthwhile to make the state transition. Ideally, if the idle period is longer than the threshold value, the rank should jump to the low-power state at the beginning of the idle period; otherwise, we should keep the rank in the active state. However, it is not easy to predict the length of each idle period, due to dynamic memory references.

Second, in memory intensive workloads, the number of idle periods is large, and many of the idle periods are too short to be exploited for power saving. It is desirable to reshape the page references to different ranks so that the idle periods become longer and the number of idle periods is minimized.

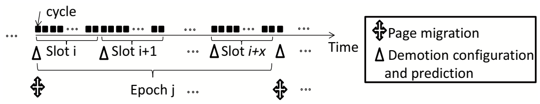


Fig. 1. Overview of RAMZzz.

Third, static demotion schemes cannot adapt to different workloads and different memory architectures. With page migrations, we further need adaptation for power management on individual ranks (differentiating the rank hotness).

3.2 Workflow of RAMZzz

We propose a novel memory design RAMZzz with dynamic migrations and adaptive demotions to address the aforementioned obstacles. We develop a dynamic page placement policy that is likely to create longer idle periods. The policy takes advantage of recency and frequency of pages stored in the ranks, and ranks are categorized into hot and cold ones. The hot ranks tend to have very short idle periods, and the cold ranks with relatively long idle periods. Page migrations are periodically performed to maintain the rank hotness (the period is defined as *epoch*). With dynamic page migrations, short idle periods are consolidated into longer ones and the number of idle periods is reduced on the cold ranks. On the other hand, the configuration for adaptive demotions is determined periodically (the period is called *slot*). For each slot, a *demotion configuration* (i.e., power-down timeouts for all power states) is used to guide demotions within the slot.

We further develop an analytical model to periodically estimate the idle period distribution of one slot. Our analytical model is based on the locality of memory pages and the idle period distribution of the previous slot. Given an optimization goal (such as minimizing energy consumption or minimizing ED^2), we use the prediction model to determine the demotion configuration for the new slot. Since the prediction has much lower overhead than the page migration, a slot is designed to be smaller than an epoch. In our design, an epoch consists of multiple slots. Figure 1 illustrates the relationship between slot and epoch. RAMZzz performs demotion configuration and prediction at the beginning of each slot and performs page migration at the beginning of each epoch.

The overall workflow of RAMZzz is designed as shown in Algorithm 1. RAMZzz maintains the performance model by updating the data structures used in the prediction model (Section 4.2). As the idle period length increases, actions of the adaptive demotion scheme may be triggered. At the beginning of each epoch, RAMZzz decides the page migration schedule and starts to migrate the pages to the destination ranks (Section 4.1). At the beginning of each slot, RAMZzz performs prediction and determines the demotion configuration for the new slot (Section 4.3). The next section will describe the design and implementation details of each component.

4 Design and Implementation Details

After giving an overview on RAMZzz, we describe the details for the following components in rank-aware power management: dynamic page migration, prediction model and adaptive demotions. Finally, we discuss some other implementation issues in integrating RAMZzz into memory systems.

Algorithm 1 Workflow of RAMZzz

if any memory reference to rank r **then**
  if rank r is in the low-power state **then**
    Set r to be ACT;
  Maintains the prediction model; /*Section 4.2*/
else
  Update the current idle period of rank r;
  Perform demotions (if necessary) according to the demotion configuration of rank r; /*Section 4.3*/
if the current cycle is the beginning of an epoch **then**
  Run page migration algorithm and schedule page migrations; /*Section 4.1*/
if the current cycle is the beginning of a slot **then**
  Determine the demotion configuration for the new slot; /*Section 4.3*/

4.1 Dynamic Page Migration

When an epoch starts, we first group the pages according to their locality and each group maps to a rank in the DRAM. Next, pages are migrated according to the mapping from groups to ranks.

Rank-aware page grouping. We place the pages with similar hotness into the same rank. We adopt the MQ structure [37] to capture the memory access locality. Ideally, RAMZzz can work with other replacement algorithms. We use MQ because it can well capture the recency and frequency of data accesses, as shown in the previous studies [38]. We briefly describe the idea of MQ, and refer the readers to the original paper for more details. MQ has M LRU queues numbered from 0 to M-1. We assume $M = 16$ following previous studies [37], [38]. Each queue stores the page descriptor including the page ID, a frequency counter and a logical expiration time. The queue with a larger ID stores the page descriptors of those most frequently used pages. On the first access, the page descriptor is placed to the head of queue zero, with initialization on its expiration time. A page descriptor in Queue i is promoted to Queue $i+1$ when its frequency counter reaches 2^{i+1}. On the other hand, if a page in Queue i is not accessed recently based on the expiration time, its page descriptor will be demoted to Queue $i-1$. We use a modified MQ structure to group physical memory pages [38]. The updates to the MQ structure are performed by the memory controller, which is designed to be off the critical path of memory accesses (more details can be found in Section 4.4).

An observation in MQ is that MQ has clustered the pages with similar access locality into the same queue [36], [38]. Moreover, unlike LRU, MQ considers both frequency and recency in page accesses. As a result, we have a simple yet effective approach to place the pages in the ranks. Suppose each rank has a distinct hotness value. We assign the rank that a page is placed in a manner such that: given any two pages p and p' with the descriptors in Queues q and q', p and p' are stored in ranks r and r' (r is hotter than r') if and only if $q > q'$ or if $q = q'$ and p is ahead of p' in the queue. That means, the pages whose descriptors are stored in a higher queue in MQ are stored in hotter ranks. Within the same queue in MQ, the more recently accessed pages are stored in hotter ranks. Algorithm 2 shows the process of grouping the pages into R sets, and each set of pages is stored in a memory rank. Each rank has a capacity of C pages.

Figure 2 illustrates an example of page placement onto the ranks. There are four ranks in DRAM, and each rank can

Algorithm 2 Obtain R page groups in the increasing hotness

```
initiate R empty sets, S_0, S_1, ..., S_{R-1};
curSet = 0;
for Queue i = M - 1, M - 2, ..., 0 in MQ do
    for Page p from head to tail in Queue i do
        Add p to S_{curSet};
        if |S_{curSet}| = C then
            curSet++;
```

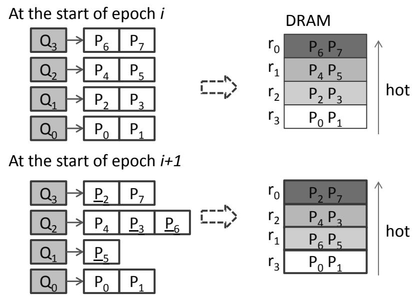


Fig. 2. An example of page placement on ranks.

hold two pages. At epoch i, we run Algorithm 2 on the MQ structures, and obtain the page placement on the right. For example, P_6 and P_7 belong to Q_3, which are the hottest pages, and they are placed into the hottest rank (here r_0). At epoch $i+1$, there are some changes in the MQ (the underlined page descriptors) and the update page placement is shown on the right.

Page migrations. To update page placement at each epoch, we first need to determine the mappings from groups to ranks, i.e., which rank stores which set (or group) of pages determined in Algorithm 2. According to the current page placement among ranks, different mappings from groups to ranks can result in different amounts of page migrations, leading to different amounts of penalty in energy and latency. We should find a mapping to minimize page migrations.

We formulate this problem as finding a maximum weighted matching on a balanced bipartite graph. The bipartite graph is defined as G whose partition has the parts U and V. Here, U and V are defined as the page placement among ranks in the previous epoch and the page groups obtained with Algorithm 2 in the current epoch respectively. An edge between r_i and S_j has a weight equaling to the number of pages that exist in both rank r_i and S_j. Since $|U| = |V|$, that is, the two subsets have equal cardinality, G is a balanced bipartite graph. We find the maximum weighted matching of such a balanced bipartite graph with the classic Hopcroft-Karp algorithm. The maximum weighted matching means the maximum number of pages that are common in both sides, and equivalently the matching minimizes the number of page migrations. Figure 3(a) illustrates the calculation of the maximum matching for the bipartite graph for the example in Figure 2. In this example, there are multiple possible matchings with the same maximum matching weight. The thick edges represent one of such maximum matchings.

After the page mappings to individual ranks are determined, we know which pages should be migrated from one rank to another. Then, we need to schedule the page migrations in a manner to minimize the runtime overhead. Inspired by the Eulerian cycle in graph theory, we develop a novel approach to perform multiple page migrations in parallel. We consider

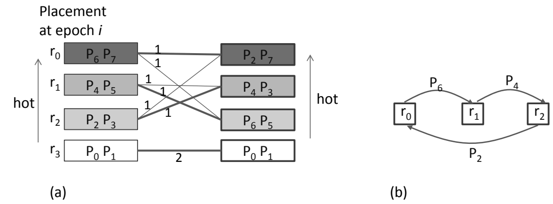


Fig. 3. An example of page migrations: (a) calculate the maximum matching on the bipartite graph; (b) calculate Eulerian cycle for page migrations.

a labeled directed graph G_m where each node represents a distinct rank. An edge from node r_i to node r_j is labeled with a page descriptor, representing the pages to be migrated from rank r_i to rank r_j.

Each strongly connected component of G_m has Eulerian cycles. According to graph theory, a directed graph has a Eulerian cycle if and only if every vertex has equal in degree and out degree, and all of its vertices with nonzero degree belong to a single strongly connected component. By definition, each strongly connected component of G_m satisfies both properties, and thus we can find Eulerian cycles in G_m. The page migration follows the Eulerian cycle. We divide the Eulerian cycle into multiple segments so that each segment is a simple path or cycle. Then, the page migrations in each segment can be performed concurrently. Figure 3(b) illustrates one example of Eulerian cycle according to the maximum matching on the left. The three migrations form a Eulerian cycle, and they are performed in one segment.

To facilitate concurrent page migrations according to the Eulerian cycle, each rank is equipped with one extra row-buffer for storing the incoming page. When migrating a page, a rank first writes the outgoing page to the buffer of the target rank, and then reads the incoming page from its buffer (more details can be found in Section 4.4).

4.2 Prediction Model

When a new slot starts, we run a prediction model against each rank. The model predicts the idle period distribution. Our estimation should be adapted to the potential changes in the page locality as well as the set of pages in each rank.

We use the histogram to represent the idle period distribution. Suppose the slot size is T cycles, and the histogram has T buckets. We denote the histogram to be $Hist[i]$, $i = 0$, 1, ..., T. The histogram means there are $Hist[i]$ number of idle periods with the length of i cycles each. One issue is the storage overhead of the histogram. A basic approach is to store the histogram into an array, and each bucket is represented as a 32-bit integer. However, the storage overhead of this basic approach is too high. Consider a slot size of 10^8 cycles in our experiments. The basic approach consumes around 400MB per rank. In practice, the histogram is usually very sparse, and there are at most $\sqrt{T}$ idle periods longer than $\sqrt{T}$ cycles. Thus, we develop a simple approach to store the short and the long idle periods separately. In particular, we maintain two small arrays: the histogram counters for the short idle periods no longer than $\sqrt{T}$ cycles, and another array of $\sqrt{T}$ integers to store the actual lengths of the long idle periods that are longer than $\sqrt{T}$ cycles. This simple approach reduces the storage overhead to $2\sqrt{T}$ integers. It takes only 80KB per rank to support a slot size of 10^8 cycles. We calculate the histogram for idle periods longer than $\sqrt{T}$ cycles with just one scan on the array.

Our estimation specifically consider page migrations. If the new slot is *not* the beginning of an epoch, there is no page migration and we use the actual histogram in the previous slot, $Hist'[i]$, to be the prediction of the current slot, i.e., $Hist[i] = Hist'[i]$ $(0 \leq i \leq T)$. Otherwise, we need to combine the access locality of the migrated pages with the historical histogram.

Our estimation after page migration works as follows. We model the references to the same page conforming to a Poisson distribution. Suppose a page i is accessed with f times in a slot. Under the Poisson distribution, the probability of having one access to page i within a cycle is $p_i = \frac{g \cdot f}{T}$, where g is the memory access latency. In our implementation, we take advantage of the frequency counter and the expiration time in the MQ structure (as described in the previous section) to approximate p_i. This already offers a sufficiently accurate approximation in practice. Given a rank consisting of N pages (pages 0, 1, ..., $N-1$), the probability of an idle cycle in the rank is $Q = (1-p_0) \cdot (1-p_1) ... \cdot (1-p_{N-1})$. Based on Q, we can estimate the probability of forming an idle period with length of k cycles (followed by a busy cycle in $(k+1)^{th}$ cycle). That is, the probability of having an idle period of k cycles is $W_k = Q^k \cdot (1-Q)$.

We denote the old values of those probability values in the previous epoch to be W'_k (k=0, 1, 2, ..., T). After page migrations, we calculate W_k (k=0, 1, 2, ..., T) according to the updated pages in the rank. Given the actual histogram in the previous slot, $Hist'[i]$, we can estimate the histogram of the new slot with the ratio W_i/W'_i, that is, $Hist^+[i] = W_i/W'_i \cdot Hist'[i]$. Finally, we normalize the histogram so that the histogram represents the total time length of a slot. Denote $s'=\sum_{i=0}^{T}(Hist^+[i] \cdot (i+g))$. We normalize the histogram with the value of $\frac{T}{s'}$, i.e., $Hist[i] = Hist^+[i] \times \frac{T}{s'}$. We use $Hist[i]$ as the prediction on the idle period distribution for the new slot.

Based on the prediction model, we will estimate the power-down timeout for the new slot in the next sub-section.

4.3 Adaptive Demotions

With the predicted idle period distribution, there are opportunities to avoid the state transitions upon those short idle periods, and to have instant state transitions for long idle periods. For example, if we know all the idle periods are expected to be very long, we can set the power-down timeout to be zero, thus performing instant state transitions. Thus, we have developed a simple approach to reduce the total penalty of state transitions. The basic idea is, for each low-power state, we use one power-down timeout to determine the state transition within the entire slot. Suppose a DDR-series memory architecture has M low-power states, denoted as $S_1, \ldots, S_M$ in the descending order of their power consumptions. For each low-power state S_i, RAMZzz performs the state transition to S_i after an idle period threshold Δ_i. If the idle period is shorter than Δ_i, RAMZzz does not make the state transition to S_i.

Since we need to exploit all power states in order to adapt to different workloads and different memory architectures, a naive approach is to consider all the possible state transitions. However, the demotion configuration of the naive approach is too complex to derive. Instead of considering all state transitions, we view multiple state transitions as a chain of state transitions from higher-power states to lower-power states. We will show that our adaptive demotion scheme can identify the unnecessary power states in a chain of states, and thus further simplify the demotion scheme. We define the demotion configuration to be a vector of power-down timeouts $\vec{\Delta} = (\Delta_1, \ldots, \Delta_M)$ where Δ_i represents the power-down timeout of low-power state S_i, $i = 1, \cdots, M$. In the chain, when the idle period length is longer than Δ_i, we perform states transition from S_{i-1} to S_i.

Given the estimated histogram on idle periods, we estimate the demotion configuration of each rank for a given optimization goal. We use energy consumption as the optimization goal to illustrate our algorithm design on estimating the demotion configuration. One can similarly extend it to other goals such as ED^2. Since the choice on different power-down timeouts does not affect the energy consumption of memory reads and writes, our metric can be simplified as the total energy consumption of background power and the state transition penalty.

We analyze the energy consumption on different demotions over an idle period. Suppose the idle period length is t cycles, and the power consumption of active state ACT and a low-power state S_i are P_{ACT} and P_{S_i} ($i = 1, \cdots, M$), respectively. Given a demotion configuration $\vec{\Delta}$, if $t \leq \Delta_1$, there is no state transition to low-power states. Otherwise, denote $I(t)$ to be the maximum i such that $\Delta_i < t$ ($i = 1, \cdots, M$). In the chain, there are at most $I(t)$ state transitions, from S_1 to $S_{I(t)}$. At the end of the idle period, a memory access comes and the rank transits from low-power state $S_{I(t)}$ back to ACT. Thus, the energy consumption of the idle period can be calculated as $\mathcal{B}(\vec{\Delta}, t)$ in Eq. (1).

$$\begin{aligned}\mathcal{B}(\vec{\Delta}, t) \quad = \quad & P_{ACT} \cdot \Delta_1 + \textstyle\sum_{j=1}^{I(t)-1}(P_{S_j} \cdot (\Delta_{j+1} - \Delta_j)) \\ & + P_{S_{I(t)}} \cdot (t - \Delta_{I(t)}) + E_{S_{I(t)}}\end{aligned} \tag{1}$$

where $E_{S_{I(t)}}$ is resynchronization energy penalty from low-power state $S_{I(t)}$ back to ACT.

Given the histogram $Hist[t]$ ($t = 0, 1, \cdots, T$), each $Hist[t]$ means there are $Hist[t]$ idle periods with length t cycles. We can calculate the total energy consumption for all the idle periods, as $E(\vec{\Delta})$ in Eq. (2).

$$E(\vec{\Delta}) = \sum_{t=0}^{\Delta_1}(P_{ACT} \cdot t \cdot Hist[t]) + \sum_{t=\Delta_1+1}^{T}(\mathcal{B}(\vec{\Delta}, t) \cdot Hist[t]) \tag{2}$$

RAMZzz also allows users to specify a delay budget to limit the delay penalty incurred by state resynchronization. We calculate the total resynchronization delay as $D(\vec{\Delta})$ in Eq. (3).

$$D(\vec{\Delta}) = \sum_{t=\Delta_1+1}^{T}(R_{S_{I(t)}} \cdot Hist[t]) \tag{3}$$

where $R_{S_{I(t)}}$ is resynchronization delay from low-power state $S_{I(t)}$ back to ACT. Our goal is to determine the suitable demotion configuration $\vec{\Delta}$ so that $E(\vec{\Delta})$ is minimized. If a

Algorithm 3 The greedy algorithm to find the suitable demotion configuration $\vec{\Delta}$

Input:
All low-power states set $\vec{S} = (S_1, \ldots, S_M)$, and associated power consumptions set $\vec{P} = (P_{S_1}, \ldots, P_{S_M})$;
Initialization:
$\vec{\Delta} = \phi$, $\vec{S}_{select} = \phi$;
while $|\vec{S}_{select}| \neq M$ **do**
  for all $S_i \in \vec{S}$ **do**
    Add S_i into $\vec{S}_{select}$;
    for each possible Δ_i value **do**
      Calculate $E(\vec{\Delta})$ using Eq. (2) with selected low-power states subset $\vec{S}_{select}$;
    Find the suitable Δ_i that has the best $E(\vec{\Delta})$;
    Remove S_i from $\vec{S}_{select}$;
  Find the low-power state S_k that has a best $E(\vec{\Delta})$;
  Add Δ_k into $\vec{\Delta}$;
  Remove S_k from $\vec{S}$;
Output:
power-down timeout set $\vec{\Delta}$

delay budget is given, we choose the $\vec{\Delta}$ value that minimizes $E(\vec{\Delta})$ with the constraint that the total delay $D(\vec{\Delta})$ is no larger than the given delay budget.

We note that $E(\vec{\Delta})$ is neither concave nor monotonic. Therefore, we have to iterate all the possible values for Δ_i=0, 1, ..., T ($i = 1, \cdots, M$), and find the best combination of Δ_i ($i = 1, \cdots, M$). The complexity of this naive approach of increases exponentially with the number of low-power states in the DRAM architecture. In the following, we develop an efficient greedy algorithm to find a reasonably good demotion configuration (illustrated in Algorithm 3).

We start by assuming that only one low-power state is used in the entire slot, and select the best suitable low-power state and its power-down timeout which leads to a smallest estimated $E(\vec{\Delta})$ among all M low-power states. Then, 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 $M - 1$ low-power states, which results in a smallest estimated $E(\vec{\Delta})$. 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. Algorithm 3 has much lower computational complexity than the naive approach.

Algorithm 3 has a low runtime overhead in most cases. First, it does not need to iterate through all values from 0 to T (T is the slot size). Instead, it only searches those values with non-zero frequencies in the predicted histogram. This number is far smaller than T in practice. Second, as more low-power states are selected during the process (one state per step), the search space for rest low-power states is further reduced since the power-down timeout of S_i is bounded by that of S_{i-1} and S_{i+1}, i.e., $\Delta_{i-1} \leq \Delta_i \leq \Delta_{i+1}$. Moreover, we further optimize Algorithm 3 in two ways. First, we adopt the branch-bound optimization in order to further reduce the search space (That is, we try possible values from the highest to the lowest until the program performance penalty violates the given budget). Second, we use an exponential search approach by iterating in the form of 2^i ($0 \leq i \leq log_2 T$) for each power-down timeout. On the current architectures, the greedy algorithm has a low runtime overhead and provides near-optimal demotion configurations, as shall be shown in our evaluation (Section 5).

The adaptive demotion scheme is applied on each rank at the beginning of a slot. The demotion configurations can be different among different ranks and at different slots. This is a distinct feature of adaptive demotion, in comparison with the previous work on static demotion schemes [5], [6], [7], [9].

4.4 Other Implementation Issues

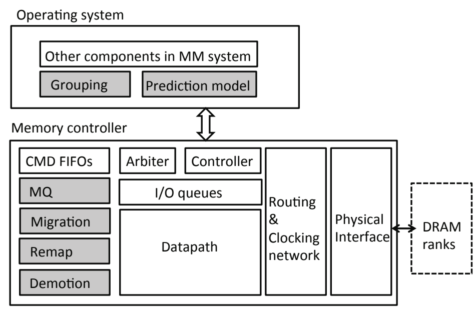


Fig. 4. Memory controller and operating system with RAMZzz's new modules highlighted.

RAMZzz can be implemented with a combination of modest hardware and software supports. Following the previous study [38], RAMZzz extends a programmable controller [39] by adding its own new components (shaded in Figure 4). Other functionalities including page grouping and the prediction model are offloaded to operating systems (like previous studies [5], [38]).

Memory Controller Structure. The memory controller (MC) receives read/write requests from the cache controller via the CMD FIFOs. The Arbiter dequeues requests from those FIFO queues, and the controller converts those requests into the necessary instructions and sequences required to communicate with the memory. The Datapath module handles the flow of reads and writes between the memory devices. The physical interface converts the controller instructions into the actual timing relationships and signals required for accessing the memory device. We assume the MC exploits cache-block-level bank interleaving and page-level channel interleaving following previous studies [28], [38]. This address mapping scheme is a common cache-line interleaving technique used in real systems. Our proposed mechanism can also be applied to other address mapping schemes (or interleaving schemes).

Four new modules including MQ, Migration, Remap and Demotion are added into the memory controller for implementing the functionality of page grouping, page migration, page remapping and power state control in RAMZzz, respectively. All the logics of the new modules are performed by the memory controller, and are designed off the critical path of memory accesses, giving the priority to the memory accesses from applications. We add a flag bit to indicate whether this request is from applications or new modules. The total on-chip storage of new MC modules in our design is 112KB (as described in the following).

MQ Module. To avoid performance degradation, MQ module contains the small on-chip cache (64KB with 4K entries) to store the MQ structure and a separate queue (10KB) for the updates to the MQ structure. To find the MQ entry of a physical page, MC uses hashing with the corresponding page number. Misses in the entry cache produce requests to DRAM. MQ module's logic snoops the CMD FIFO queue, creating one update per new request. The updates to the MQ structure are performed by the MC off the critical path of memory accesses (via the aforementioned flag bit). The update queue is implemented as a small circular buffer, where a new update precludes any currently queued update to the same entry. In our design, each physical page descriptor in the MQ queues takes 124 bits. Each descriptor contains the corresponding page number (22 bits), the reference counter (14 bits), the queue number in MQ (4 bits), the last-access time (27 bits), the pointers to other descriptors (54 bits), and the reserved bit for flags (3 bits). The space overhead of our design is low. For the 2GB DRAM, the total space taken by the descriptors is about 8MB (only 0.4% of the total DRAM space).

Migration Module. The Migration module contains the queue of scheduled migrations. The migrations are enqueued in a manner such that concurrent migrations of a Eulerian cycle are put in consecutive positions. At the beginning of each epoch, the OS accesses the current MQ structure to perform grouping and calculate the Eulerian cycle. Then, the OS updates the queue of scheduled migrations (10KB) which is stored in the Migration module. Page migrations start from the beginning of an epoch, and is scheduled once there are idle periods. Priority is given to longer segments because they involve more pages. Memory requests are buffered until the migration is concluded. To facilitate concurrent page migrations according to the Eulerian cycle, each rank is equipped with one extra row-buffer for storing the incoming page. When migrating a page, a rank first writes the outgoing page to the buffer of the target rank, and then reads the incoming page from its buffer.

Remap Module. Similar to the previous design [38], we introduce a new layer of translation between physical addresses assigned by the OS (and stored in the OS page table) and those used by the MC to access DRAM devices. Specifically, the MC maintains the *Remapping Table*, a hash table for translating physical page addresses coming from the LLC to actual remapped physical page addresses. The OS can access the *Remapping Table* as well. After the migration is completed at the beginning of an epoch, the *Remapping Table* is updated accordingly. Periodically or when the table fills up (at which point the MC interrupts the CPU), the OS commits the new translations to its page table and invalidates the corresponding TLB entries. For example, if the OS uses a hashed inverted page table, e.g., UltraSparc and PowerPC architectures, it considerably simplifies the commit operation. Then, the OS sets a flag in a memory-mapped register in the MC to make sure that the MC prevents from migrating pages during the commit process, and clears the *Remapping Table*.

When a memory request (with physical address assigned by the OS) arrives at the MC, it searches the address in the *Remapping Table*. On a hit, the new physical page address is used by the MC to issue the appropriate commands to retrieve the data from its new location. Otherwise, the original address is used. In terms of access latency, the remapping operation happens when a request is added to the MC queues and does not extend the critical path in the common case because queuing delays at the MC are substantial. For memory-intensive workloads, memory requests usually wait in the MC queues for a long time before being serviced. The above translation can begin when the request is queued and the delay for translation can be easily hidden behind the long waiting time. The notion of introducing *Remapping Table* for the MC has been widely used in the past [19], [38], [40].

The Remap module maintains the *Remapping Table* (28KB with 4K entries) and the logic to remap target addresses. At the end of migration, the Migration module submits the migration information to the Remap module, which creates new mappings in the *Remapping Table*. The Remap module snoops the CMD queue to check if it is necessary to remap its entries. We assume each *Remapping Table* lookup and each remapping take 1 memory cycle. However, these operations only delay a memory request if it finds the CMD queue empty (which is not the common case). Note that the migration and remapping of a segment blocks the accesses to only the pages involved, and concurrent accesses to other pages are still possible.

Demotion Module. The Demotion module performs the demotion to control the power state of each rank according to its demotion configuration. The demotion configuration of each rank is updated by the OS at the beginning of a slot.

OS Modules. Two major new components Grouping and Prediction Model are added to the memory management subsystem in operating system. The Grouping module performs grouping and calculates the Eulerian cycle according to the MQ structure at the beginning of an epoch. At the beginning of each epoch, the OS accesses the current MQ structure to perform grouping and calculate the Eulerian cycle. Then, the OS updates the queue of scheduled migrations which is stored in the Migration module. The Prediction Model module runs the prediction model and obtains the demotion configuration for memory controller at the beginning of each slot.

5 EVALUATION

5.1 Methodology

Our evaluation is based on trace-driven simulations. In the first step, we use cycle-accurate simulators to collect memory access traces (last-level cache misses and writebacks) from running benchmark workloads. In the second step, we replay the traces using our detailed memory system simulator. Our simulation models all the relevant aspects of the OS, memory controller, and memory devices, including page replacements, memory channel and bank contention, memory device power and timing, and row buffer management. We evaluate workloads from SPEC 2006 and PARSEC [11].

SPEC 2006 Workloads. We use PTLSim [41] to collect memory access traces of SPEC 2006 workloads. The main architectural characteristics of the simulated machine are listed in Table 2. We model and conduct the evaluation with an in-order processor following previous studies [28], [38]. More complex and recent processors are studied with Sniper-based simulations. We evaluate our techniques with three different memory architectures, as shown in Table 1 (Section 2.1).

TABLE 2
Architectural configurations of PTLSim. The default setting is highlighted.

Component	Features
CPU	In-order cores running at 2.66GHz
Cores	4
TLB	64 entries
L1 I/D cache (per core)	48KB
L2/L3 cache (shared)	256KB/4MB
Cache line/OS page size	64B/4KB
DRAM	DDR3-1333, DDR2-800, LPDDR2-800
Channels	4
Ranks	4, **8**, 12, 16
Capacity (GB)	1, **2**, 4
Delay and Power	see Table 1

TABLE 3
Mixed workload: 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	988.2	7.8	0.40	libquantum, milc, mcf, lbm

Those memory architectures are used in different computing systems. We simulate different capacities (1GB, 2GB and 4GB) and different numbers of ranks (4, 8, 12 and 16) for the memory system. All the ranks have the same configurations (DRAM parameters) and capacities. By default, we assume a 2GB DRAM with 8 ranks. We pick these small memory sizes to match the footprint of the workloads' simulation points. We calculate the memory power consumption following Micron's System Power Calculator [17], with the power and delay illustrated in Table 1. The energy and performance overheads caused by new MC and OS modules (e.g., remapping, migration and demotion) are derived from our analysis in Section 4, which are consistent with those of others [38], [40], [42].

We have used 19 applications from SPEC 2006 with the *ref* inputs. These workloads have widely different memory memory access rates, footprints and localities. Due to space limitations, we do not present the results for single applications; instead, we report their geometric mean (**GM**), and also four particular applications with different memory intensiveness. They are omnetpp, cactusADM, mcf and lbm (denoted as S1, S2, S3 and S4, respectively). To assess our algorithm under the context of multi-core CPUs, we study mixed workloads of four different applications from SPEC 2006 (Table 3). The four workloads 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 15×10^9 cycles in the original PTLSim simulation, which represents a stable and sufficiently long execution behavior.

PARSEC Workloads. Since current PTLSim cannot support PARSEC benchmarks, we use another simulator–Sniper [43] to collect memory access traces of PARSEC. By default, we use the simulated CPU architecture as shown in Table 4 (Intel's Gainestown CPUs), which simulates a four-core processor running at 2.66 GHz based on the Intel's Nehalem micro architecture. By default, we simulate a four-core CPU, and 2GB DRAM with 8 ranks. The memory architecture has

TABLE 4
Architectural configurations of Sniper. The default setting is highlighted.

Component	Features
CPU	Out-order cores running at 2.66GHz
Cores	**4**, 8, 16
DTLB/ITLB	64/128 entries
L1 I/D cache (per core)	32KB/32KB
L2 cache (per core)	256KB
L3 cache (shared)	8MB
Cache line/OS page size	64B/4KB
DRAM	DDR3-1333
Channels	4
Ranks	**8**, 16
Capacity (GB)	**2**, 32, 64
Delay and Power	see Table 1

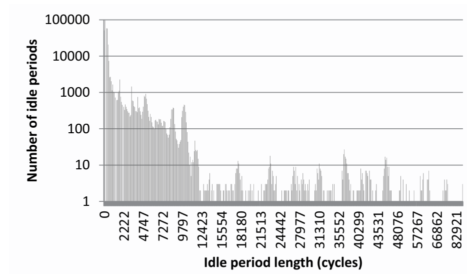


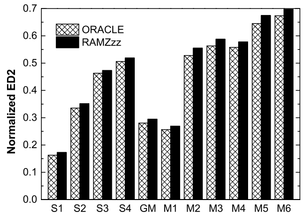
Fig. 5. The histogram of idle periods with M2 on Rank 0.

the same power consumption and performance configurations as the PTLSim-based simulations. We also observe similar results on simulated machines with larger number of cores and memory capacities. Each PARSEC workload runs with four threads, and each thread is assigned to one core. We use the *sim-medium* inputs for PARSEC workloads, and perform the measurement on the specified Region-of-Interest (ROI) of PARSEC workloads [11].

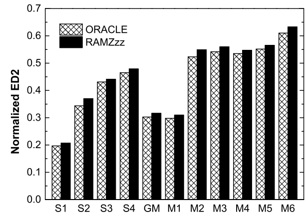
Comparisons. In our previous study [36], we have already demonstrated that the preliminary version of RAMZzz outperforms other state-of-the-art power management techniques [6], [9], [12] in terms of both ED^2 and energy consumption. For completeness, we show the comparison between RAMZzz and other state-of-the-art power management policies in Section 5.5. Overall, RAMZzz has significantly outperformed the state-of-the-art approaches [6], [9], [12], [36]. In this paper, we consider two RAMZzz variants namely **RZ–SP** and **RZ–SD** to evaluate the impact of individual techniques. They are the same as RAMZzz except that RZ–SP uses the static page management scheme without page migrations, whereas RZ–SD uses the static demotion scheme. The static demotion scheme simply transits a rank to a pre-selected low-power state according to the prediction model.

In addition to RAMZzz variants, we also simulate the following techniques for comparison. All our experimental results of RAMZzz and its variants have included the energy and performance penalty caused by page migrations and adaptive demotions. *All the metrics reported in this paper are normalized to those of BASE.*

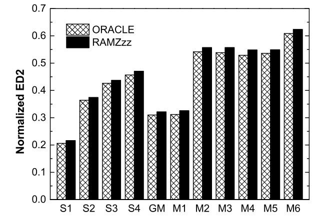
- **No Power Management (BASE)**: no power management technique is used, and ranks are kept active even when they are idle.
- **Ideal Oracle Approach (ORACLE)**: ORACLE is the same as RAMZzz, except the power-down timeout in ORACLE is determined with the future information, instead of history. Specifically, at the beginning of each slot, we perform an offline profiling on the current slot,



(a) Results on ED^2 for DDR3

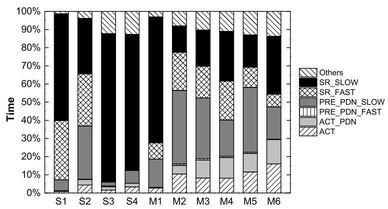


(b) Results on ED^2 for DDR2

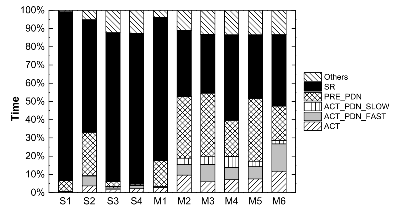


(c) Results on ED^2 for LPDDR2

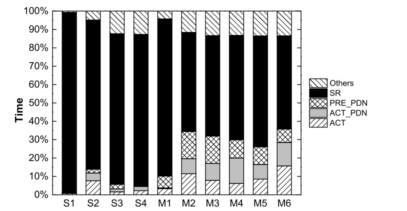
Fig. 6. Comparing ED^2 of RAMZzz and ORACLE with the optimization goal of ED^2 on three memory architectures.



(a) The breakdown of time for DDR3



(b) The breakdown of time for DDR2



(c) The breakdown of time for LPDDR2

Fig. 7. The breakdown of time stayed in different power states for RAMZzz with the optimization goal of ED^2.

and get the real histogram of idle periods. Based on the histogram, we calculate the optimal power-down timeout.

RAMZzz allows users to specify the slot and epoch sizes and delay budgets. In our simulation, we pre-set default values for RAMZzz as a compromise between the prediction overhead and the accuracy. By default, the slot size is 10^8 cycles and an epoch consists of ten slots (10^9 cycles), and delay budget is set to be 4% of the slot size.

Idle Period Distribution. We study the distribution of idle periods. Figure 5 shows the histogram of idle period lengths of the collected traces on Rank 0 on DDR3 under BASE approach. Many idle periods are too short to be exploited for state transitions, e.g., shorter than the threshold idle period length for demoting to SR_FAST (2500 cycles on DDR3). We observe similar results on other ranks.

In the following sections, we first study the behavior of RAMZzz, BASE, ORACLE, RZ–SP and RZ–SD to show the effectiveness of RAMZzz on different memory architectures, and the impact of individual techniques (Sections 5.2–5.4). Next, we compare RAMZzz with other state-of-the-art memory power management policies in Section 5.5. We focus on the DRAM component. Our models and optimizations are able to work for different goals such as energy consumption and ED^2. In this section, we focus on the optimization goal of minimizing the ED^2 while keeping the performance penalty within a given budget. For the optimization goal of minimizing energy consumption, we find that RAMZzz also has a high potential of energy saving, which is consistent with our observations in the ED^2 experiment. We have evaluated the individual impact of RAMZzz (page migrations and adaptive demotions) by comparing RAMZzz, RZ–SP and RZ–SD. The results show that page migrations achieve an average ED^2 improvement of 17.1–23.3% over schemes without page migrations, and adaptive demotions achieve an average ED^2 improvement of 22.4–36.4% over static demotions. Due to the space constraint, we put those results in Appendix A of the supplementary file.

5.2 Results on SPEC 2006 Workloads

We first compare the algorithms with the optimization goal of ED^2 on SPEC 2006 workloads, because ED^2 is a widely used metric for energy efficiency.

We study the overall impact of RAMZzz in comparison with BASE and ORACLE. The comparison with BASE shows the overall effectiveness of energy saving techniques of RAMZzz, and the comparison with ORACLE shows the effectiveness of our prediction model. Figure 6 presents normalized ED^2 results for RAMZzz and ORACLE approaches on three different DRAM architectures. If the normalized ED^2 of an approach is smaller than 1.0, the approach is more energy efficient than BASE.

Thanks to the rank-aware power management, RAMZzz is significantly more energy-efficient than BASE. Compared with BASE, the reduction on ED^2 is 64.2%, 63.3% and 63.0% on average on DDR3, DDR2 and LPDDR2, respectively. The reduction is more significant on the workloads of single applications (e.g., S1–S4) than the mixed workloads. There are two main reasons. First, since the single-application workload has a smaller memory footprint, the page migration has a smaller overhead and the number of cold ranks is larger. The number of page migrations becomes very small after the first few epochs. In contrast, the execution process of the workloads with a large memory footprint (such as M5 and M6) consistently has a fair amount of page migrations at all epochs. Secondly, on single-application workloads, there are more opportunities for saving background power using lower-power states (such as SR_FAST and SR_SLOW in DDR3, SR in DDR2 and LPDDR2). Figure 7 shows the breakdown of time stayed in different power states for RAMZzz on DDR3, DDR2 and LPDDR2. In Figure 7, each power state represents the percentage of time when ranks are in this state during the total simulation period. And *Others* represents the percentage of time that includes DRAM operations, page remapping delay, page migration delay and resynchronization delay. As the workload becomes more memory-intensive, the portion of time that a rank is in lower-power states becomes

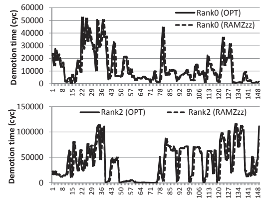


Fig. 8. Power-down timeouts comparison.

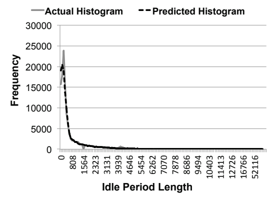


Fig. 9. The predicted idle histogram: Case 1.

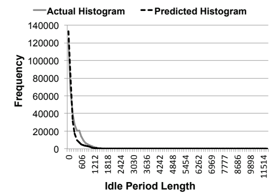


Fig. 10. The predicted idle histogram: Case 2.

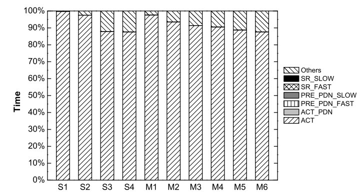


Fig. 11. The breakdown of time for BASE.

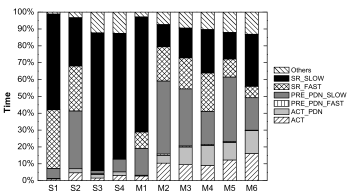


Fig. 12. The breakdown of time for ORACLE.

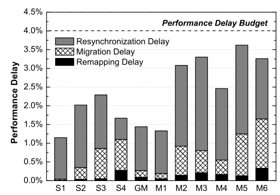


Fig. 13. The breakdown of delay for RAMZzz.

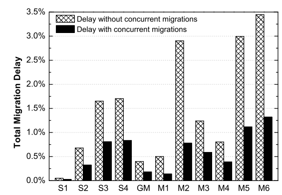


Fig. 14. The optimization of page migration delay.

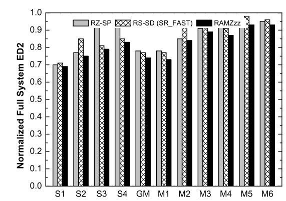


Fig. 15. Comparing full-system ED^2 on DDR3.

less significant, indicating that many idle periods are too short and they are not worthwhile to perform state transitions into lower-power states (even with page migration).

We briefly present the results of the extra energy overhead of RAMZzz on DDR3. The energy penalties of page migrations and adaptive demotions (i.e., resynchronization energy consumption) contribute to 0.4% and 0.8% of the total energy consumption on average, respectively (less than 1.4% and 1.6% on all workloads). The energy overhead is much smaller than the energy saving gained by RAMZzz (67% on average).

It can also be seen from Figure 6 that RAMZzz achieves a very close ED^2 to ORACLE on all workloads and memory architectures. RAMZzz achieves 5.7%, 4.4% and 3.7% on average larger ED^2 than ORACLE on DDR3, DDR2 and LPDDR2, respectively. This good result is because our histogram-based prediction model is able to accurately estimate the suitable power-down timeout for the sake of minimizing ED^2. Figure 8 compares RAMZzz's estimated power-down timeouts to SR_FAST with ORACLE on ranks 0 and 2 of executing M4 on DDR3. Our estimation is very close to the optimal value on the two ranks. We observe similar results for different ranks and different workloads and also for the power-down timeouts of other low-power states and other DRAM architectures. We also find that our model has high accuracy in predicting rank idle period distribution. We compare the predicted idle histogram to the actual idle histogram of RAMZzz on rank 0 of executing M4 on DDR3. The predicted histogram is close to the actual histogram in our evaluation in both cases: 1) the slot is not the beginning of an epoch as shown in Figure 9; 2) the slot is the beginning of an epoch as shown in Figure 10.

We have further made the following observations on the result of breakdown in Figure 7. First, on a specific memory architecture, the portion of time for different low-power states varies significantly across different workloads. Different workloads have different choices on the most energy-effective low-power state. For most single-application workloads, RAMZzz makes the decision to demote into SR_SLOW on DDR3 in most idle periods, whereas the decision of demotion is to SR_FAST or PRE_PDN_SLOW for the mixed workloads. Second, on different DRAM architectures, the portion of time for different low-power states varies significantly, even for the same workload. SR on LPDDR2 has a much higher significance in all workloads than on DDR3 and DDR2. That is because, as we have seen in Table 1, SR on LPDDR2 consumes a similar normalized power consumption but a relative smaller resynchronization time when compared with the other two DRAM architectures. These two observations have actually demonstrated the effectiveness of adaptive demotions of RAMZzz for different workloads and different memory architectures.

Figures 11 and 12 show the breakdown of time stayed in different power states for BASE and ORACLE on DDR3, respectively. Compared with Figure 7(a), RAMZzz has a very similar power state distribution to ORACLE on all workloads, which again demonstrates the effectiveness of our estimation. Compared to BASE, both RAMZzz and ORACLE significantly reduce the percentage of time when ranks are in the *ACT* state by the adaptive use of all available low-power states. We observe similar results for other workloads and DRAM architectures.

Next, we study the performance delay in detail. Figure 13 shows the breakdown of performance delay for RAMZzz on DDR3. We divide the delay into three parts: resynchronization delay (caused by state transitions), migration delay (caused by page migrations) and remapping delay (caused by *Remapping Table* lookup and address remapping). The performance delay of RAMZzz is well controlled under the pre-defined penalty budget (i.e., 4% in this experiment). The results demonstrate that our model is able to limit the performance delay within the pre-defined threshold. The resynchronization delay contributes the largest portion of performance delay on most workloads (2% on average).

As seen from Figure 13, the migration delay is higher on the workloads with a large memory footprint (such as S3, M5 and M6). To further study the migration delay, Figure 14 presents the total migration delay of RAMZzz with/without

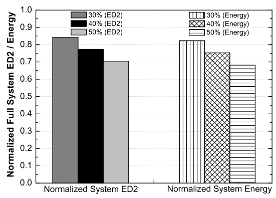


Fig. 16. The impact of memory power ratio.

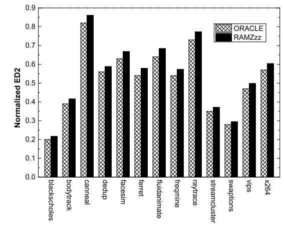


Fig. 17. The overall results of PARSEC workloads.

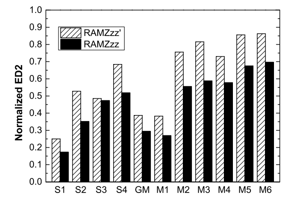


Fig. 18. Comparison of RAMZzz' and RAMZzz.

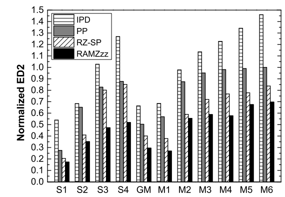


Fig. 19. Comparison of IPD, PP, RZ–SP, RAMZzz.

our graph-based optimizations on DDR3. Thanks to our graph-based optimizations (as described in Section 4.1), the total migration delay is significantly decreased, with the reduction of 50.0% to 74.4%. Concurrent migrations prevent significant performance degradation in all workloads.

Finally, we discuss the overhead of calculating the demotion configuration and the migration information (Eulerian cycle). We find that the number of those values with non-zero frequencies in the predicted histogram is far smaller than the slot size (10^8) in practice. Thus, the search space of Algorithm 3 is acceptable at runtime. The average time for calculating of the demotion configuration is around several milliseconds on current architectures. Such calculation is performed only once per slot. Moreover, the average time for calculating the migration information is around tens of milliseconds on current architectures, which is much smaller than our selected epoch size. Thus, their overheads are low on current architectures.

5.3 Studies on Full System ED^2

In this section, we evaluate the impact of RAMZzz on full-system energy consumption and performance with SPEC 2006 workloads. We start by performing back-of-envelop calculations, following previous studies [28], [44]. We assume that the average power consumption of memory system accounts for 40% of the total system power in the baseline policy (i.e., BASE), and calculate a fixed average power estimate (i.e., the remaining 60%) for all other components. Thus, the energy consumption of all other components (i.e., non-memory system energy) is proportional to the program execution time, which is usually consistent with the real-world case [28], [44]. This ratio (40%) has been chosen as the current contribution of memory system to entire system power consumption [1], [45], [46]. We also study the impact of varying this ratio in this evaluation. Architectural characteristics and experimental parameters are the same as those used in Section 5.2.

Figure 15 presents full system ED^2 of RAMZzz, RZ–SP and RZ–SD (SR_FAST is used as the pre-selected low-power state) when the optimization metric is set to ED^2 on DDR3. All three approaches still outperform BASE on all workloads in terms of full system ED^2. Compared with BASE, the reduction in full system ED^2 is 23.0%, 18.0% and 17.8% on average for RAMZzz, RZ–SP and RZ–SD, respectively. RAMZzz outperforms both RZ–SP and RZ–SD in full-system ED^2, but leads to slightly higher performance degradations. We observe that RAMZzz has an average reduction of 4.8% (from 1.6% to 17.9%) and 5.6% (from 1.7% to 8.6%) over RZ–SP and RZ–SD in full system ED^2, respectively. We observe similar results on other DRAM architectures.

We further study the ratio of power consumption of the memory subsystem to the overall power consumption of the full system. Particularly, we vary the ratio from 30% to 50%. Figure 16 shows that the fraction of memory power has a significant effect on both full system ED^2 and energy consumption. Increasing the ratio from 30% to 50% (i.e., the power contribution of other components are reduced from 70% to 50%), the normalized full-system ED^2 and energy consumption of RAMZzz decrease from 0.84 to 0.70 and 0.83 to 0.68, respectively.

5.4 Results on PARSEC Workloads

Figure 17 shows the normalized ED^2 results of RAMZzz and ORACLE approaches on DDR3 architecture using PARSEC workloads. We use the default experimental setting (e.g., the delay budget is 4%). RAMZzz is also significantly more energy-efficient than BASE on PARSEC workloads, with an average reduction of 45.2%. We observe similar results to those on the SPEC 2006 workloads. For example, the reduction is more significant for the workloads with less intensive memory accesses (such as blackscholes). RAMZzz achieves a very close ED^2 to ORACLE on PARSEC workloads (as shown in Figure 17). Furthermore, the comparisons between RAMZzz and RZ-SP/RZ-SD show that on average, page migrations bring 16.2% ED^2 saving, and adaptive demotions bring 25.3% ED^2 saving. RAMZzz is consistently and significantly more energy-efficient than other approaches on both PARSEC and SPEC 2006 benchmarks.

5.5 Comparisons with Other Approaches

For completeness, we show the comparison between RAMZzz and two typical state-of-the-art memory power management policies: the preliminary version of RAMZzz [36] (namely RAMZzz'), and the approach developed in [9] (namely IPD).

Comparison with RAMZzz'. Figure 18 presents normalized ED^2 results for RAMZzz and RAMZzz' approaches on SPEC 2006 workloads for DDR3 memory architecture. Architectural characteristics and experimental parameters are the same as those used in Section 5.2. Note, RAMZzz' only uses two pre-selected low-power states (PRE_PDN_FAST and SR_FAST) for demotions.

RAMZzz is vastly superior to the preliminary one proposed in [36] (i.e., RAMZzz'), with an average reduction of 24% in ED^2 on DDR3. Furthermore, RAMZzz' cannot work with other DRAM architectures without modifications, such as

DDR2 and LPDDR2. Since RAMZzz' only uses two pre-selected low-power states (PRE_PDN_FAST and SR_FAST on DDR3) for demotions, it loses the opportunities in exploiting the most energy-effective low-power state for different workloads and different memory architectures. This observation demonstrates the effectiveness of adaptive demotions of RAMZzz.

Comparison with IPD. We conduct a detailed study on the comparison between RAMZzz and IPD [9] with SPEC 2006 workloads on DDR3 memory architecture. To evaluate the effectiveness of RAMZzz over IPD, we simulate the following techniques:

- **Immediate Power-down (IPD)**: IPD is the approach developed in [9]. We choose PRE_PDN_FAST as the target low-power state when the optimization goal is ED^2.
- **Predicted Power-down (PP)**: PP arguments IPD by using our histogram-based perdition on the idle period distributions and finding the suitable power-down timeout for state transitions.
- **RZ–SP**: RZ–SP arguments PP with adaptive demotions.

Figure 19 presents normalized ED^2 comparison for these energy saving approaches. Architectural characteristics and experimental parameters are the same as those used in Section 5.2. RAMZzz has much lower ED^2 than other techniques, on average 54%, 40% and 23% lower than IPD, PP and RZ–SP, respectively.

RAMZzz outperforms IPD with three main reasons: 1) a histogram-based prediction model that estimates the idle period distributions and the suitable power-down timeout (which brings 23% ED^2 saving on average); 2) an adaptive demotion scheme that exploits energy saving capabilities of all power states for different memory architectures and different workloads (which brings 22% ED^2 saving on average); 3) a page migration approach that consolidates the idle periods among memory ranks (which brings 23% ED^2 saving on average). The former two aspects form the adaptive demotions technique developed in RAMZzz. All these techniques are additive to the overall ED^2 improvement of RAMZzz. Particularly, the difference between IPD and PP represents the saving from the histogram-based power-down timeout prediction, the difference between PP and RZ–SP represents the saving from the adaptive demotion scheme, and the difference between RZ–SP and RAMZzz represents the saving from page migrations.

6 Conclusion

In this paper, we have proposed a novel memory design RAMZzz to reduce the DRAM energy consumption. It embraces two rank-aware power saving techniques to address the major obstacles in state transition-based power saving approaches: dynamic page migrations and adaptive demotions. A cost model is developed to guide the optimizations for different workloads and different memory architectures. We evaluate RAMZzz with SPEC 2006 and PAESEC benchmarks in comparison with other power saving techniques on three main memory architectures including DDR3, DDR2 and LPDDR2. Our simulation results demonstrate significant improvement in ED^2 and energy consumption over other power saving techniques.

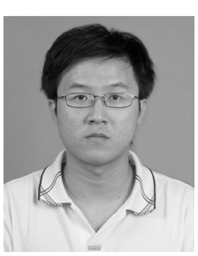
Acknowledgement

The authors would like to thank anonymous reviewers for their insightful comments. This work is partially supported by a MoE AcRF Tier 2 grant (MOE2012-T2-1-126) in Singapore. This work is also partly supported by the National Basic Research Program of China (973 Project Grant No. 2015CB352400), Program for Changjiang Scholars and Innovative Research Team in University (IRT1158, PCSIRT) China, NSFC (Grant No. 61261160502, No. 61272099) and Scientific Innovation Act of STCSM (No. 13511504200). The work of Yanchao Lu and Donghong Wu was done when they were visiting students in Nanyang Technological University, Singapore. Bingsheng He is the corresponding author.

References

[1] 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.
[2] 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.
[3] D. Meisner, B. T. Gold, and T. F. Wenisch, "Powernap: eliminating server idle power," in *ASPLOS '09*, 2009.
[4] M. S. Ware, K. Rajamani, M. S. Floyd, B. Brock, J. C. Rubio, F. L. R. III, and J. B. Carter, "Architecting for power management: The ibm power7 approach," in *HPCA '10*, 2010.
[5] H. Huang, P. Pillai, and K. G. Shin, "Design and implementation of power-aware virtual memory," in *USENIX ATC '03*, 2003.
[6] H. Huang, K. G. Shin, C. Lefurgy, and T. Keller, "Improving energy efficiency by making dram less randomly accessed," in *ISLPED '05*, 2005.
[7] V. Delaluz, A. Sivasubramaniam, M. Kandemir, N. Vijaykrishnan, and M. J. Irwin, "Scheduler-based dram energy management," in *DAC '02*, 2002.
[8] C. Bae and T. Jamel, "Energy-aware memory management through database buffer management," in *WEED '11*, 2011.
[9] X. Fan, C. Ellis, and A. Lebeck, "Memory controller policies for dram power management," in *ISLPED '01*, 2001.
[10] B. Diniz, D. Guedes, W. Meira, Jr., and R. Bianchini, "Limiting the power consumption of main memory," in *ISCA '07*, 2007.
[11] C. Bienia, "Benchmarking modern multiprocessors," Ph.D. dissertation, Princeton University, January 2011.
[12] A. R. Lebeck, X. Fan, H. Zeng, and C. Ellis, "Power aware page allocation," in *ASPLOS '00*, 2000.
[13] 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.
[14] Micron Tech., Inc., *MT41J256M4JP-15E Datasheet*, 2010.
[15] Micron Tech., Inc., *MT47H128M8CF-25E Datasheet*, 2007.
[16] Micron Tech., Inc., *MT42L128M32D1LF-25WT Datasheet*, 2011.
[17] Micron Tech., Inc., *System Power Calculator*, http://www.micron.com/products/support/power-calc, 2012.
[18] I. Hur and C. Lin, "A comprehensive approach to dram power management," in *HPCA '08*, 2008.
[19] 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.
[20] V. Delaluz, M. Kandemir, N. Vijaykrishnan, and M. J. Irwin, "Energy-oriented compiler optimizations for partitioned memory architectures," in *CASES '00*, 2000.
[21] V. Delaluz, M. Kandemir, N. Vijaykrishnan, A. Sivasubramaniam, and M. J. Irwin, "Dram energy management using software and hardware directed power mode control," in *HPCA '01*, 2001.
[22] K. Fang, H. Zheng, J. Lin, Z. Zhang, and Z. Zhu, "Mini-rank: A power-efficientddrx dram memory architecture," *IEEE Trans. on Comput.*, vol. 63, no. 6, 2014.
[23] J. H. Ahn, N. P. Jouppi, C. Kozyrakis, J. Leverich, and R. S. Schreiber, "Future scaling of processor-memory interfaces," in *SC '09*, 2009.
[24] H. Zheng, J. Lin, Z. Zhang, and Z. Zhu, "Decoupled dimm: Building high-bandwidth memory system using low-speed dram devices," in *ISCA '09*, 2009.

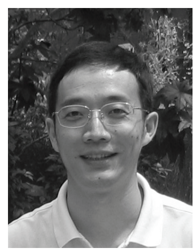
[25] M. Bi, R. Duan, and C. Gniady, “Delay-hiding energy management mechanisms for dram,” in *HPCA ’10*, 2010.
[26] E. Cooper-Balis and B. Jacob, “Fine-grained activation for power reduction in dram,” *IEEE Micro*, vol. 30, 2010.
[27] H. David, C. Fallin, E. Gorbatov, U. R. Hanebutte, and O. Mutlu, “Memory power management via dynamic voltage/frequency scaling,” in *ICAC ’11*, 2011.
[28] Q. Deng, D. Meisner, L. Ramos, T. F. Wenisch, and R. Bianchini, “Memscale: active low-power modes for main memory,” in *ASPLOS ’11*, 2011.
[29] Y. Lu, B. He, X. Tang, and M. Guo, “Synergy of dynamic frequency scaling and demotion on dram power management: Models and optimizations,” *IEEE Trans. on Comput.*, 2015.
[30] Y. Kim, D. Han, O. Mutlu, and M. Harchol-Balter, “Atlas: A scalable and high-performance scheduling algorithm for multiple memory controllers,” in *HPCA ’10*, 2010.
[31] A. N. Udipi, N. Muralimanohar, N. Chatterjee, R. Balasubramonian, A. Davis, and N. P. Jouppi, “Rethinking dram design and organization for energy-constrained multi-cores,” in *ISCA ’10*, 2010.
[32] 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.
[33] B. He and Q. Luo, “Cache-oblivious databases: Limitations and opportunities,” *ACM Trans. Database Syst.*, vol. 33, no. 2, 2008.
[34] B. He and Q. Luo, “Cache-oblivious query processing,” in *CIDR ’07*, 2007.
[35] K. Kumar, K. Doshi, M. Dimitrov, and Y.-H. Lu, “Memory energy management for an enterprise decision support system,” in *ISLPED ’11*, 2011.
[36] 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.
[37] Y. Zhou, J. Philbin, and K. Li, “The multi-queue replacement algorithm for second level buffer caches,” in *USENIX ATC ’01*, 2001.
[38] L. E. Ramos, E. Gorbatov, and R. Bianchini, “Page placement in hybrid memory systems,” in *ICS ’11*, 2011.
[39] Xilinx, Inc., *Spartan-6 FPGA Memory Controller User Guide*, 2010.
[40] K. Sudan, N. Chatterjee, D. Nellans, M. Awasthi, R. Balasubramonian, and A. Davis, “Micro-pages: increasing dram efficiency with locality-aware data placement,” in *ASPLOS ’10*, 2010.
[41] M. T. Yourst, “Ptlsim: A cycle accurate full system x86-64 microarchitectural simulator,” in *ISPASS ’07*, 2007.
[42] X. Guo, E. Ipek, and T. Soyata, “Resistive computation: Avoiding the power wall with low-leakage, stt-mram based computing,” in *ISCA ’10*, 2010.
[43] T. E. Carlson, W. Heirman, and L. Eeckhout, “Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulations,” in *SC ’11)*, 2011.
[44] N. Chatterjee, M. Shevgoor, R. Balasubramonian, A. Davis, Z. Fang, R. Illikkal, and R. Iyer, “Leveraging heterogeneity in dram main memories to accelerate critical word access,” in *MICRO ’45*, 2012.
[45] J. Ousterhout, P. Agrawal, D. Erickson, C. Kozyrakis, J. Leverich, D. Mazières, S. Mitra, A. Narayanan, G. Parulkar, M. Rosenblum, S. M. Rumble, E. Stratmann, and R. Stutsman, “The case for ramclouds: scalable high-performance storage entirely in dram,” *SIGOPS Oper. Syst. Rev.*, vol. 43, no. 4, 2010.
[46] D. Tsirogiannis, S. Harizopoulos, and M. A. Shah, “Analyzing the energy efficiency of a database server,” in *SIGMOD ’10*, 2010.



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, and cloud computing.

Donghong Wu received the bachelor degree in computer science and engineering from Shanghai Jiao Tong University, China, in 2010. He is currently a master student at Department of Computer Science and Engineering, Shanghai Jiao Tong University, China. His research interests are in-memory processing systems and high performance computing.

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 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. His research interests include distributed systems, mobile and pervasive computing, and wireless sensor networks.

Jianliang Xu is a Professor in the Department of Computer Science, Hong Kong Baptist University. He received his BEng degree in computer science and engineering from Zhejiang University, Hangzhou, China and his PhD degree in computer science from Hong Kong University of Science and Technology. He held visiting positions at Pennsylvania State University and Fudan University. His research interests include data management, mobile computing, wireless sensor networks, and distributed systems.

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. His research interests include automatic parallelization and data-parallel languages, bioinformatics, compiler optimization, high-performance computing, and pervasive computing.