RAMZzz: Rank-Aware DRAM Power Management with Dynamic Migrations and Demotions

Donghong Wu[1,2], Bingsheng He, Xueyan Tang
[1]Nanyang Technological University

Jianliang Xu
Hong Kong Baptist University

Minyi Guo
[2]Shanghai Jiao Tong University

***Abstract*—Main memory is a significant energy consumer which may contribute to over 40% of the total system power, and will become more significant for server machines with more main memory. In this paper, we propose a novel memory system design named RAMZzz with rank-aware energy saving optimizations. Specifically, we rely on a memory controller to monitor the memory access locality, and group the pages with similar access locality into the same rank. We further develop dynamic page migrations to adapt to data access patterns, and a prediction model to estimate the demotion time for accurate control on power state transitions. 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 power saving techniques.**

I. Introduction

Energy consumption has become a major factor for design and implementation of computer systems. Inside many computing systems, main memory (or DRAM) has been a critical component for the performance and energy consumption. As processors have moved to multi-/many-core era, more applications run simultaneously with their working sets in the main memory. Many online and data-centric applications require low latency accesses, and large-scale memory-resident computing systems like RAMCloud [29] are developed to host hundreds of Gigabytes or even larger data in the main memory. 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 [16], [23]. For example, it has been reported that main memory contributes to as much as 40–46% of total energy consumption in server applications [25], [34], [23]. For these reasons, this paper studies the energy saving techniques of main memory.

Current main memory architectures allow the power management on individual memory ranks. Individual ranks at different power states consume different amounts of energy (idle, refresh and precharge energies). There have been various power-saving techniques on exploiting the power management capability of main memory [17], [18], [9], [3]. The common research theme of those research studies is to exploit the transition of individual memory ranks to low-power states for energy saving. Fan et al. concluded that immediate transitions to the low-power state save the most energy consumption for most single-application workloads [12]. However, the decision can be wrong for more memory intensive workloads such as multiprogrammed executions. Huang et al. [18] has shown that only the sufficiently long idle periods can be exploited because state transitions take non-negligible amount of time and energy. Essentially, the amount of energy saving relies on the distributions of idle periods and the effectiveness on how power management techniques exploit the idle periods. Existing techniques are suboptimal in the following aspects: (1) they do not effectively extend the idle period, either with static page placement [11], [12] or with heuristics-based page migrations [17], [18]; (2) the prediction on the demotion time for a state transition is limited and static, either with heuristics [17], [18] or regression-based model [12] that is not robust for different workloads.

Those two shortcomings have hurt the effectiveness of state transition-based energy saving approaches, especially for memory intensive workloads. First, in memory-intensive workloads, many idle periods are too short to be exploited for state transitions. Advanced techniques should be invented to consolidate the short idle periods into sufficiently longer ones. Second, due to lacking of accurate prediction on the demotion time, making wrong decisions is unavoidable, leading to significant penalties in energy and delay.

To address the aforementioned issues, we propose a novel memory design named RAMZzz with rank-aware power management techniques. Instead of having static page placement, we develop dynamic page migrations to reflect the access locality changes in the workload. Pages are placed into different ranks according to their access locality so that the pages in the same rank have roughly the same hotness. As a result, ranks are categorized into hot and cold ones. The hot rank is highly utilized with reads/writes and has some 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. We further develop a prediction model to estimate the idle period distribution. The prediction model combines the historical page access frequency and historical idle period distribution, and 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). This is achieved through adjusting the demotion time for the given optimization goal.

We implement our design into a cycle-accurate simulator PTLSim [36] and compute the energy and performance of workloads. We use SPEC 2006 benchmark to evaluate

SC12, November 10-16, 2012, Salt Lake City, Utah, USA
978-1-4673-0806-9/12/$31.00

RAMZzz in comparison with representative power saving policies [22], [18], [11] and an ideal oracle approach. Our experiments show that (1) with the optimization goal of ED^2, RAMZzz achieves an average ED^2 improvement of 15.3–38.2% over other power saving policies, and achieves only 10.3% on average larger ED^2 than the ideal oracle approach; (2) with the optimization goal of energy consumption, the energy consumption of RAMZzz is on average 14.8–54.5% lower than other policies, and is only 1.2% on average higher than the ideal oracle approach.

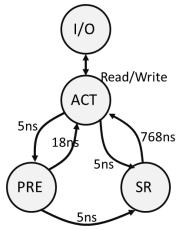
To the best of our knowledge, RAMZzz is the first cost model guided page placement algorithm for main memory power saving. The key contributions of this paper are: 1) we propose a page migration approach to dynamically consolidate the idle periods among ranks to improve the effectiveness of state transition-based power saving approaches; 2) we develop a prediction model to estimate the suitable demotion time for different optimization goals; 3) we implement our design into a cycle-accurate simulator, and conduct extensive studies to show the effectiveness of RAMZzz on multiple metrics including ED^2, energy consumption and performance.

Organization. The rest of the paper is organized as follows. We introduce the background on basic power management of DRAM in Section II. Section III describes the detailed design and implementation of RAMZzz. The experimental results are presented in Section IV. Section V summarizes the related work. We conclude this paper in Section VI.

II. Background

In this paper, we use the terminology of DDR-series memory architectures (e.g., DDR3 and DDR4 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 each 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 (usually larger than 50% among various workloads [18], [38]), and tends to be more significant in the future. On one hand, there has been a tremendous amount of research work on reducing the DRAM stalls, which translates into the reduction of active power. CPU caches are designed with large capacities. The recent Intel Core i7 CPU can have a 12MB L3 cache. Software optimizations like cache-friendly algorithms further reduce the number of memory accesses. On the other hand, memory capacity will become larger and is usually provisioned with peak usage. Therefore, it is the focus of this paper on reducing the background power consumption.

Different power states have different background power consumptions. When a rank is idle, background power is still consumed, unless the rank enters into a lower power state.



(a)

	Power State	Idle Power(W/GB)
Active Standby	ACT	1.34
Precharge Fast Powerdown	PRE	0.7
Self-refresh	SR	0.23

(b)

State transition	Energy (nJ/GB)
PRE →ACT	19.3
SR →ACT	583.7

(c)

Fig. 1. DDR3 DRx4 R-DIMM at 1333 MHz [6]: (a) delays for power state transitions, (b) power states, (c) energy consumption of state transitions.

To exit from a power down state, the disabled subcomponents need to be reactivated and the rank needs to be restored to the active state. State transitions among different power states cause latency and energy penalties.

Depending on which subcomponents are disabled, modern memory architectures support a number of power states with complicated transitions [26], [27]. Due to the complexity, previous studies [17], [18], [9], [12] usually abstract the power state transitions with simplified models of several power states. In this paper, we utilize a simplified model with three power states (namely ACT, PRE and SR, illustrated in Figure 1(a)). Note, many previous studies assume two states only (ACT and a lower-power state, e.g., PRE or SR). Our prediction model enables us to consider the state transitions of more power states. Figures 1(b) and 1(c) show their power consumption and penalty for transiting from PRE/SR back to ACT. All the statistics numbers are calculated for DDR3 at 1333 MHz with System Power Calculator [28], and one can calculate those numbers for other memory architectures similarly. PRE consumes 52% of the power of ACT, with relatively small latency and energy penalties. In contrast, SR consumes only 17% of the power of ACT, with much higher latency and energy penalties. The energy penalty by transiting from SR to ACT is over an order of magnitude higher than that of transiting from PRE to ACT.

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 active state. Thus the effective use of deeper low-power state is contingent on having long idle periods on a rank. That naturally leads to two problems: 1) how to create longer idle periods without modifying the application, and 2) how to make correct decisions on state transitions. Those two problems are essential for reducing background power consumption. In the next section, we present our approach to address those two problems.

III. Rank-Aware Power Management

In this section, we start with an overview on the design rationales of RAMZzz. Next, we give the detailed design of two key components in RAMZzz including dynamic migrations and demotions. Finally, we put all the components together and detail our memory design.

A. Overview

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

First, due to the latency and power penalty of transiting from lower-power state to active state, it requires a minimum length threshold for an idle period that is worthwhile to make the state transition. Furthermore, the threshold value varies with the amount of energy and delay penalties of different state transitions for the same DRAM architecture and of the same state transition for different DRAM architectures. Since there is a length threshold for an idle period, an energy saving technique needs to determine whether an idle period on a rank is longer than threshold or not. Ideally, if the idle period is longer than the threshold value, the rank should directly jump to the lower-power state; 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, the state transition-based power saving approaches cannot take full advantage of idle periods, especially for memory intensive workloads. 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.

We propose a novel memory design RAMZzz with dynamic migrations and 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, we develop an analytical model to periodically estimate the idle period distribution within a predefined interval (the period is called *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 estimate the suitable demotion time 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 2 illustrates the relationship between slot and epoch. RAMZzz performs prediction at the beginning of each slot and performs page migration at the beginning of each epoch.

B. 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.

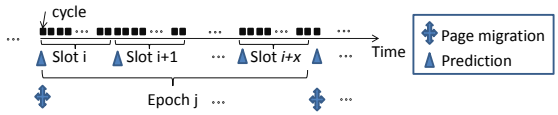


Fig. 2. Overview of RAMZzz.

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 main memory management policy named MQ [39]. 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 [39], [31]. 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 $i-1$.

An observation in MQ is that MQ has clustered the pages with similar access locality into the same queue. 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 1 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.

Algorithm 1 Obtain R page groups in the increasing hotness

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

Figure 3 illustrates an example of page placement onto the ranks. There are four ranks in DRAM, and each rank can hold two pages. At epoch i, we run Algorithm 1 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). With the page migration (the detailed algorithm

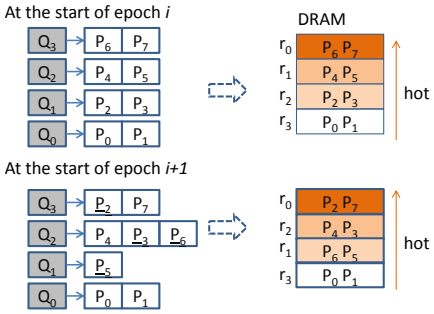


Fig. 3. An example of page placement on ranks.

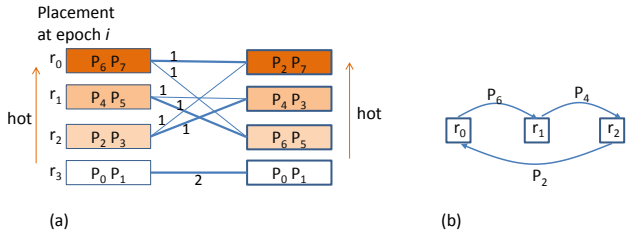


Fig. 4. An example of page migrations: (a) calculate the maximum matching on the bipartite graph; (b) calculate Eulerian cycle for page migrations.

can be found in the later description), we obtain the placement on the right. Note, page migration may change the hotness of the ranks.

Page migrations. The hotness of each rank needs to adapt to the changes in the access localities of the pages. Pages may need to be migrated from one rank to another. Hot pages are moved to hot ranks. Due to the capacity, some pages that are originally stored in the hot ranks have to be moved to the cold ranks. There are two major tasks. First, we need to determine the hotness of each rank, i.e., which rank stores which set (or group) of pages determined in Algorithm 1. 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. In other words, given the current page placement, we should find a mapping to minimize the amount of page migrations. Second, given the mapping obtained in the first task, we need to schedule the page migrations in a manner to minimize the runtime overhead.

We determine the mappings from groups to ranks with the consideration of the number of pages that do not need migration. Our goal now is to find a *rank permutation* (n_0, n_1, ..., n_{R-1}) for (0, 1, 2, ..., $R-1$) so that (1) rank n_i is hotter than rank n_j if and only if $i > j$ ($0 \leq i, j < R$), and (2) the number of page migrations is minimized (equivalently, maximizing the number of pages that do not need migration).

We formulate this problem as finding a maximum 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 rank permutation in the previous epoch (n'_0, n'_1, ..., n'_{R-1}) and page groups obtained with Algorithm 1 in the current epoch (S_{R-1}, ..., S_1, S_0) respectively. An edge between n'_i and S_j has a weight equaling to the number of pages that exist in both rank n'_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 matching of such a balanced bipartite graph with the classic HopcroftKarp algorithm. The maximum matching means the maximum number of pages that are common in both sides, and equivalently the matching minimizes the number of page migrations. Thus, given the maximum matching with the set of edges (n'_{i_j}, S_j), $0 \leq j < R$, we get the rank permutation ($n_0 = n'_{i_{R-1}}$, $n_1 = n'_{i_{R-2}}$, ..., $n_{R-1} = n'_{i_0}$) as the solution. Figure 4(a) illustrates the calculation of the maximum matching for the bipartite graph for the example in Figure 3. In this example, there are multiple possible matchings with the same maximum matching weight. All the thick edges represent one of such maximum matchings. Thus, the rank permutation is (r_3, r_1, r_2, r_0).

After the page mappings to individual ranks are determined, we know which pages should be migrated from one rank to another. Inspired by Eulerian cycle in graph theory, we develop a novel approach to perform multiple page migrations in parallel. We consider a labeled directed graph G_m where each node represents a distinct rank. An edge from node n_i to node n_j is labeled with a page descriptor, representing the page to be migrated from rank n_i to rank n_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. So, each edge is touched exactly once, meaning each page is migrated exactly once. 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 4(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 the concurrent page migrations according to Eulerian cycle, each rank is equipped with one extra row-buffer for storing the outgoing page. When migrating a page, a rank needs to first write the outgoing page to the buffer of the target rank, and then read the incoming page from its buffer.

C. Prediction Model

When a new slot starts, we run a prediction model against each rank. The model predicts the idle period distribution, followed by estimating the demotion time for the new slot.

Predicting 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. Thus, our estimation considers both the history distribution as well as the access locality of each page in the previous slot.

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 $Hist[i]$ of idle periods with

the length of i cycles. 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 the 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 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 the 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 pages in the rank. Given the actual histogram in the previous slot, $Hist'[i]$, we can estimate the histogram of the current 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'=$Hist^+[0] + \sum_{i=1}^{T}(Hist^+[i] \cdot i)$. We normalize the histogram with the value of $\frac{T}{s'}$, e.g., $Hist[i] = Hist^+[i] \times \frac{T}{s'}$. We use $Hist[i]$ as the prediction on the idle period distribution for the current slot. Next, we use the histogram estimation to make decisions on when to perform state transitions.

Determining the demotion time. 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 demotion time to be zero, thus performing instant state transitions. So we have developed a simple approach to reduce the total penalty of state transitions. The basic idea is to use one demotion time to determine the state transitions within the entire slot. That is, RAMZzz performs the state transition after some idle period threshold Δ. If the idle period is shorter than Δ, RAMZzz does not make the state transition. The Δ value is determined according to the predicted idle period distribution.

Given the estimated histogram on idle periods, we estimate the demotion time for a given optimization goal (such as energy consumption or ED^2). Here, we use the optimization goal of energy consumption, and one can similarly extend to other goals such as ED^2. Since the choice on different demotion times 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 first present our algorithm design for two power states only, and later extend it to handle multiple states as a chain of two-state transitions. Suppose the idle period length is t cycles, and the power consumption of the higher- and lower-power states h and l are P_h and P_l, respectively. If $t \leq \Delta$, there is no state transition and the energy consumption for the idle period is denoted as a function $\mathcal{B}_{short}(t)$, where $\mathcal{B}_{short}(t) = P_h \times t$. Otherwise, after time Δ, there is a state transition. At the end of time t, a memory access comes and the rank transits back to the active state. The energy consumption of the idle period is denoted as a function, $\mathcal{B}_{long}(\Delta, t)$. By definition, $\mathcal{B}_{long}(\Delta, t) = P_h \times \Delta + P_l \times (t - \Delta) + E_{l \rightarrow active}$, where $E_{l \rightarrow active}$ is the penalty of energy consumption transiting from power state l to h.

Given the histogram $Hist[i]$, i = 0, 1, ..., T, each $Hist[i]$ means there are $Hist[i]$ idle periods with length i cycles. We can calculate the total energy consumption for all the idle periods, as $E(\Delta)$ in Eq. (1). Our goal is to get the suitable demotion time Δ so that $E(\Delta)$ is minimized.

$$E(\Delta) = \sum_{i=1}^{\Delta}(\mathcal{B}_{short}(i) \cdot Hist[i]) + \sum_{i=\Delta+1}^{T}(\mathcal{B}_{long}(\Delta, i) \cdot Hist[i]) \quad (1)$$

We note that $E(\Delta)$ is neither concave nor monotonic. Therefore, we have to iterate all the possible values Δ=0, 1, ..., T, and find the suitable Δ. To make a compromise on the prediction speed and accuracy, we use an exponential increasing approach by iterating zero and $\Delta = 2^i$ $(0 \leq i \leq \log_2 T)$. RAMZzz also allows users to specify a delay budget to limit the delay penalty incurred by state transitions. We choose the Δ value that minimizes $E(\Delta)$ and has the total delay smaller than the given delay budget.

Since we model the power management of DRAM with three power states, the state transitions are viewed as a chain of state transitions. We need two threshold values Δ_1 and Δ_2 that represent the demotion time transiting from ACT to PRE and from PRE to SR, respectively. In our experiments, we observed that those two transitions are already sufficient for

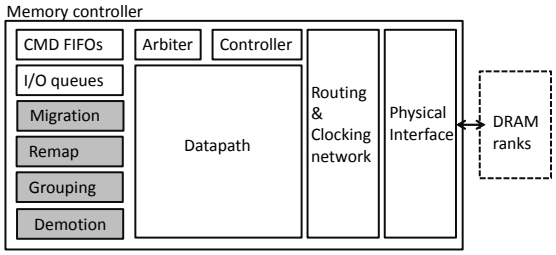


Fig. 5. Memory controller with RAMZzz's new modules highlighted.

our power management.

D. Put It All Together

The overall workflow of RAMZzz is designed as Algorithm 2. RAMZzz performs the memory references to the target rank and updates the MQ structure and histogram used in idle period estimation. At the beginning of each epoch, RAMZzz decides the page migration schedule and starts to migrate the pages to the destination ranks. At the beginning of each slot, RAMZzz updates the demotion time for each rank.

Algorithm 2 Workflow of RAMZzz.

```
1: if any memory reference to rank r then
2:     if rank r is in lower-power state then
3:         Set r to be ACT;
4:     Perform the memory reference to r;
5:     Update the MQ structure;
6:     Update the corresponding histogram counter according to the idle
       period length;
7: else
8:     Increase the current idle period, idleLen, by one cycle;
9:     if idleLen ≥ Δ2 then
10:        Transit to SR;
11:    else
12:        if idleLen ≥ Δ1 then
13:            Transit to PRE;
14: if The current cycle is the beginning of an epoch then
15:    Run page migration algorithm and schedule page migrations;
16: if The current cycle is the beginning of a slot then
17:    Estimate the suitable demotion time (Δ1, Δ2) for each rank;
```

RAMZzz adds a few new components to the memory controller. Following the previous study [31], RAMZzz extends a programmable controller [35] by adding its own new components (shaded in Figure 5). The memory controller 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. 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. In practice, some functionalities including page migrations and state transitions according to the prediction model can be offloaded to operating systems (like previous studies [31], [17]). We add a flag bit to indicate whether this request is from applications or new modules.

We briefly describe some important details of the new modules. Four new modules including Migration, Remap, Grouping and Demotion are added for implementing the functionality of page migration, page remapping for a page migration, page grouping and power state control in RAMZzz, respectively. The Grouping module polls the CMD FIFO queues and updates the MQ structure per new request. At the beginning of an epoch, we perform grouping according to the MQ structure. In the Migration module, the set of scheduled migrations is maintained in a queue. The migrations are enqueued in a manner that concurrent migrations of a Eulerian cycle are put in consecutive positions. Page migration starts from the beginning of an epoch, and is scheduled once there is idle period so that it does not add delay to the workload. The priority is given to longer segments because they involve more pages. The Remap module maintains the mapping from logical pages to physical pages. It takes the latched logical page numbers involved in a segment, and updates the page mapping in the page table. Note that the migration and remapping of a segment blocks the accesses to only the involved pages, and concurrent accesses to other pages are still possible. Finally, the Demotion module runs the prediction model and sets the demotion time.

Finally, we note that the structure complexity and storage overhead of RAMZzz are similar to the previous proposals, e.g., [18], [10], [11], [12], [32]. For example, our design has small DRAM space requirement (less than 2% of the total amount of DRAM). For page migration, only page descriptors, their mapping tables and MQ structures are stored. For the prediction model, we store the two arrays for calculating the idle period histogram per rank, and reuse the frequency counter and expiration time information in the MQ structure for prediction.

IV. Evaluation

In this section, we evaluate our design using energy, performance and ED^2 as metrics.

A. Methodology

We have integrated RAMZzz into PTLSim V3.0 simulator [36], which offers cycle-accurate simulation. 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.

The main architectural characteristics of the simulated machine are listed in Table I. We simulate DDR3 DRAM with different capacities (1GB, 2GB and 4GB etc) and different numbers of ranks (4, 8, 12 and 16). All the ranks have the same configurations and capacities. By default, we assume a 2GB DRAM with eight ranks. We calculate the memory power consumption following the System Power Calculator [28], with the power and delay illustrated in Figure 1. The power consumption characteristics of DRAM is the same as those of DDR3-1333 in the previous paper [6].

TABLE I
ARCHITECTURAL CHARACTERISTICS OF THE SIMULATED MACHINE. THE DEFAULT SETTING IS HIGHLIGHTED.

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)	256KB/4MB
Cache line/OS page size	64B/4KB
DRAM	DDR3-1333
ranks	4, **8**, 12, 16
capacity (GB)	1, **2**, 4
delay and power	see Figure 1

TABLE II
MIXED WORKLOAD: MEMORY FOOTPRINT, MEMORY ACCESSES STATISTICS PER 5×10^8 CYCLES ($Mean$ AND $\frac{Stdev}{Mean}$).

Name	Footprint (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	0.4	libquantum, milc, mcf, lbm

Workloads. We have used 19 applications from SPEC 2006. Those workloads have widely different memory footprints and localities. To assess our algorithm under the context of multi-core CPUs, we study mixed workloads of four different applications from SPEC 2006 (Table II). The mixed workloads form multi-programmed executions on a four-core CPU, ordered by the average number of memory accesses (mean) and the standard deviation ($\frac{Stdev}{Mean}$) per 5×10^8 cycles. The four workloads start at the same time. We perform measurement when all the workloads finish the initialization. For each workload, we select the simulation period of 15 billion cycles in the original PTLSim simulation, and that simulation point represents a stable and sufficiently long execution behavior.

We study the distribution of idle periods. Figure 6 shows the histogram of idle period lengths of the collected traces on Rank 0. We observed similar results on other ranks. Many idle periods are too short to be exploited for state transitions.

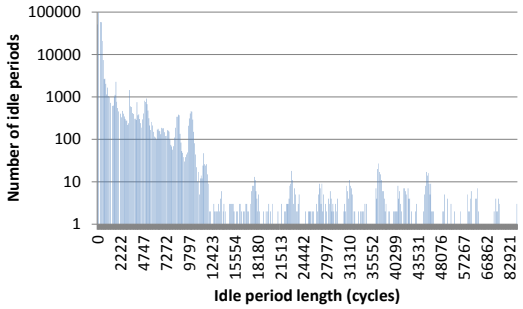


Fig. 6. The histogram of idle periods with M2 on Rank 0.

Comparisons. To evaluate the proposed techniques in RAMZzz, we have simulated the following techniques:

- **No Power Management (BASE)**: no power management technique is used, and ranks are kept active when they are idle.
- **Immediate Powerdown (IPD)**: This is the simplest form of hardware power management. It is a static technique where IPD immediately transits a rank to a lower-power state when an idle period starts. IPD was previously shown to be the most efficient [22], [12]. In this study, we chose PRE as the target lower-power state when the optimization goal is ED^2, and chose SR when the optimization goal is energy consumption.
- **Immediate Powerdown with page migration (IPM)**: this approach arguments IPD with our page migration approach. IPM is similar to the previous work [18], except the difference in the page hotness definition and the heuristics in transiting from the middle power state to the lowest power state.
- **Predicted Powerdown (PP)**: PP arguments IPD by using our histogram-based prediction on the idle period distributions and finding the suitable demotion time for state transitions.

To evaluate the effectiveness of our histogram-based prediction, we also simulate an oracle approach (**OPT**). OPT is the same as RAMZzz, except the demotion time in OPT 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, and get the real histogram of idle periods. Based on the histogram, we calculate the optimal demotion time. Finally, RAMZzz allows users to specify the slot and epoch sizes and delay budgets. By default, the slot size is 10^8 cycles and an epoch consists of five slots (5×10^8 cycles), and delay budget is set to be 4% of the slot size. We evaluate their impact in Section IV-D.

We compare the behavior of RAMZzz, BASE, IPD, IPM, PP and OPT. *All the metrics are normalized to those of BASE.* To demonstrate the flexility of our optimization metric, we assess two optimization goals: ED^2 and total energy consumption. We report the metrics for DRAM only, to remove the impact of other components in the machine. Due to space limitations, we do not present the results for all workloads of single application; instead, we report their geometric mean (**GM**), and also four applications with different memory intensiveness. They are omnetpp, cactusADM, mcf and lbm (denoted as S1, S2, S3 and S4, respectively). They cover a wide range of memory accesses intensiveness (0.1, 0.9, 8.0, 17.9 millions accesses on average per 5×10^8 cycles accordingly).

B. Results on ED^2-Oriented Optimizations

We first compare the algorithms with the optimization goal of ED^2, simply because ED^2 is a widely used metric for energy efficiency. Figure 7 presents ED^2 comparison for all the energy saving approaches.

RAMZzz has much lower ED^2 than other techniques, on average 38.2%, 15.3%, 20.7% lower than IPD, IPM and PP respectively. The reduction is more significant on the workloads of single applications (S1–4) than the mixed workloads. There are two major reasons. First, the page migration has a smaller overhead, since the single-application workload has a smaller memory footprint. The number of page migrations becomes

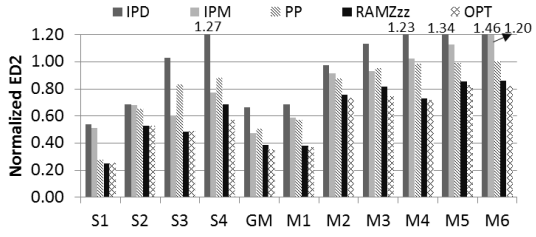


Fig. 7. Comparing ED^2 with the optimization goal of ED^2.

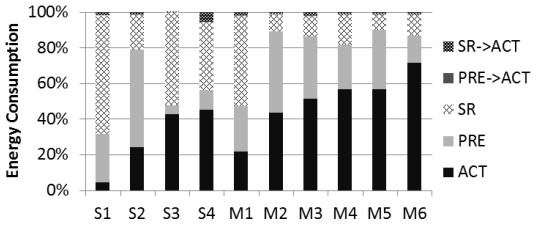


Fig. 8. The breakdown of energy consumption of RAMZzz with the optimization goal of ED^2.

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. Second, the memory access is less intensive on single-application workloads, and there are more opportunities for saving background power. Figure 8 shows the energy consumption breakdown for background power of RAMZzz. As the workload becomes more memory intensive, the portion of ACT becomes significant, indicating that many idle periods are too short and they are not worthwhile to perform state transitions (even with page migration). For the less memory intensive workloads like S1 and M1, SR and PRE have very significant portions in the total energy consumption, indicating significant energy saving compared with ACT.

Due to the significant delay incurred by the immediate power down, both IPD and IPM have very high ED^2. Our histogram-based prediction model accurately estimates the suitable demotion time for the sake of minimizing ED^2. Figure 9 compares RAMZzz's estimated demotion times to SR with OPT on ranks 0 and 2 of executing M4. Our estimation is very close to the optimal value on the two ranks. That shows the effectiveness of our estimation, with page migrations among ranks 0 and 2 and other ranks. We observe similar results for different ranks and different workloads and also for the demotion time to PRE. For the two techniques with histogram prediction (PP and RAMZzz), they achieve a much lower ED^2 than BASE. Particularly, RAMZzz has a smaller ED^2 (15.3% on average) than IPM. When page migration is disabled, PP has a smaller ED^2 (20.3% on average) than IPD. The impact of prediction model is relatively larger when page migration is disabled.

With page migration, RAMZzz achieves a significant reduction on ED^2 compared with PP (up to 42.9%, and 20.7% on average for all workloads, see Figure 7). We also note that the ED^2 improvement of page migration is even higher without the histogram prediction (with an average of 34.6% ED^2 reduction of IPM over IPD). Page migrations and accurate

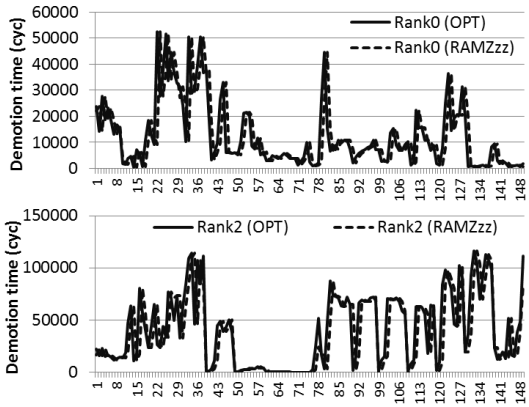


Fig. 9. Comparing the demotion time to SR of RAMZzz and OPT.

demotion time prediction are additive to the overall ED^2 improvement. Comparing the impacts of page migration and prediction model, we find that the prediction model has a larger impact. The prediction model has facilitated the energy saving while keeping the delay budget.

Due to the accuracy of the demotion time prediction, RAMZzz has only 10.3% on average larger ED^2 than OPT (see Figure 7). In other words, RAMZzz is already very close to the offline oracle approach.

We briefly discuss the experimental results for the performance and energy consumption without figures.

While RAMZzz is configured to optimize ED^2, both delay and energy consumption of RAMZzz are rather reasonable. Its delay penalty is relatively small (5% on average larger than BASE). The delay is mainly from state transitions. In contrast, without demotion time prediction, IPD and IPM may suffer from the long delay incurred from short idle periods, with delays up to 38% and 27% higher than BASE.

For RAMZzz, the energy consumption is significantly reduced, 58% on average smaller than BASE. Also, the energy consumption of RAMZzz is 18–32% smaller than other approaches. The energy consumption of page migrations is less than 5% of the total energy consumption across all workloads. We note that, IPD is actually quite competitive on the energy consumption when the workload is less memory intensive, such as single-application and M1 workloads. That is consistent with the previous result [22]. However, IPD is much worse for memory intensive applications such as M2–6.

C. Results on Energy-Oriented Optimizations

We also present the results when RAMZzz's optimization goal is set to energy consumption. We set the relatively high delay budget (20%) to unleash the potential of energy saving. Figure 10 compares the energy consumption and ED^2 for different techniques. We make the following observations. First, RAMZzz has the lowest energy consumption. The improvement over IPD, IPM and PP are 54.5%, 35.1%, 14.8%, respectively. Moreover, RAMZzz consumes only 1.2% on average more energy than OPT on all workloads. Our study on the demotion time also shows that our prediction model is very close to OPT with the optimization goal of energy

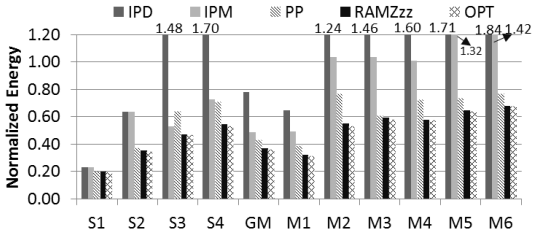


Fig. 10. Comparing energy consumption with the optimization goal of energy consumption.

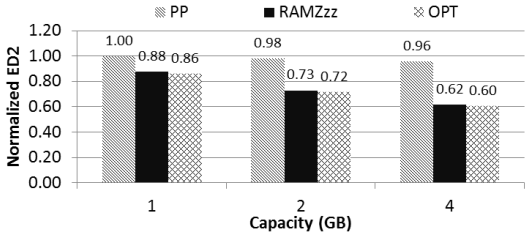


Fig. 11. Comparing ED^2 with varying DRAM capacity on M4.

consumption. Thus RAMZzz achieves the effectiveness and flexibility in different optimization goals.

We briefly present the results for ED^2 under the optimization goal of energy consumption without figures. RAMZzz still has a comparable ED^2 with OPT, and outperforms other techniques. The ED^2 of RAMZzz is 14.1% on average higher than that of OPT. In some cases, RAMZzz has a higher ED^2 than BASE, because of the compromise between the energy saving and delay penalty.

D. Sensitivity Studies

We use ED^2 as the optimization metric to study the sensitivity analysis. Since IPD and IPM have higher ED^2 than PP and RAMZzz, especially on memory intensive workloads with large footprints, we present the results for PP, RAMZzz and OPT. In those studies, we vary one parameter at a time and keep other parameters in their default settings. Due to the space limitation, we present the figures for M4 (as a modest case among all those workloads) and comment on other workloads without figures when appropriate.

DRAM parameters. We study the impact of different numbers of ranks and memory capacities of DRAM. As the number of ranks increases, we observed a rather stable ED^2 for PP, RAMZzz and OPT. For all the three approaches, when the number of ranks increases from 2 to 4, ED^2 drops less than 1%, because of a finer grained power control on ranks. When the number of ranks increases from 4, 8 to 16, ED^2 increases less than 3%. The major reason for increasing ED^2 is the increased amount of page migrations caused by increasing the rank. Figure 11 shows the results for varying the memory capacity. As memory capacities increase, all three methods achieve a lower ED^2, and the ED^2 improvement of RAMZzz over PP becomes larger. That indicates the effectiveness of our approach on larger-memory systems.

RAMZzz parameters. We study the impact of different epoch/slot sizes and delay budgets of RAMZzz.

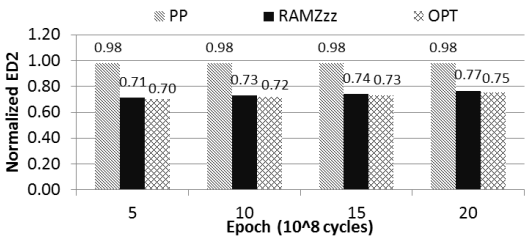


Fig. 12. Comparing ED^2 with varying epoch sizes on M4.

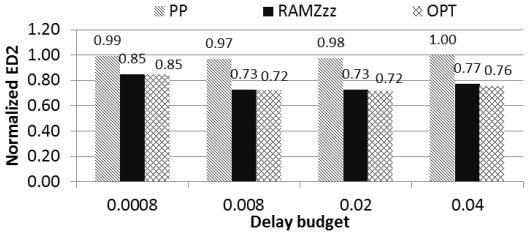


Fig. 13. Comparing ED^2 with varying delay budget per slot on M4.

Figure 12 shows the results of varying epoch sizes. PP is not sensitive to the epoch size, whereas the ED^2 of RAMZzz and OPT both increases slightly. That is because, for a longer epoch, the rank hotness does not affect the changes in page access locality in time, and the ED^2 improvement brought by page migrations is slightly reduced. We observed a similar result when varying the slot size in $(0.125 \times 10^8 \times 2^i)$ cycles (i =0, 1, ..., 3). The ED^2 of RAMZzz varies by less than 2%. In practice, we set the slot size to be 10^8 cycles as a compromise on the prediction overhead and the accuracy.

Figure 13 compares ED^2 for varying delay budget. A small delay budget limits the potential for energy saving, whereas a large delay budget leads to too aggressive energy saving and exaggerates the delay incurred by mispredictions. In practice, we set the delay budget within 1–4% for optimizing ED^2.

Finally, we study the impact of static demotion time if RAMZzz adopts one static demotion time during the entire execution. The goal is to evaluate the effectiveness of our dynamic prediction on demotion time. The result is shown in Figure 14. We observed a concave trend on increasing the static demotion time. When the static demotion time is small, the trend shows sharp drops due to the reduced penalties in energy and delay; when the static demotion time is large, the trend shows increase due to missing the chance of state transitions for better energy saving. All those static settings have a higher ED^2 than RAMZzz, justifying the necessity

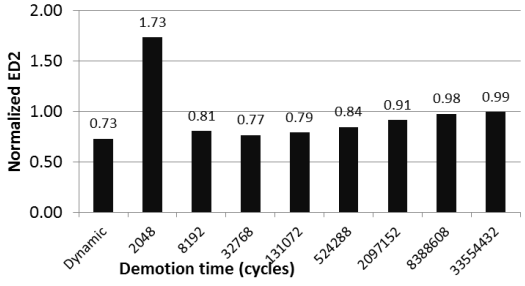


Fig. 14. Evaluating RAMZzz with static demotion time on M4.

of dynamic predictions per slot. This observation is also consistent with fluctuations on the suitable demotion time observed in Figure 9.

V. Related Work

We review the related work on energy saving with power states and with other hardware and software approaches.

A. Energy Saving with Power States

Saving energy by transiting memory power states has attracted many research efforts, covering memory controller design, compilers and operating systems.

Different power state transition approaches have been developed for DRAM systems. Hur et al. [19] developed adaptive history-based scheduling in the memory controller. Based on page migration, Huang et al. [18] further stored the 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. [22] studied different page allocation strategies. Their approach does not have the analytical model to guide the decision, or utilize both recency and frequency to capture rank hotness. Diniz et al. [11] limited the energy consumption by adjusting the power states of DRAM. Different from their approach at the granularity of ranks, our approach is finer grained with page migrations and our prediction model offers a novel way of power management on guiding page migrations and power state transitions. Fan et al. [12] developed an analytic model on estimating the idle time of DRAM chips using an exponential distribution. Their model does not consider page migrations. Instead of relying on the presumed knowledge of distribution, our prediction model combines the historical information on idle period distribution and page access locality.

DRAM power state transitions have been implemented in operating systems and compilers. Delaluz et al. [9] 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. [17] proposed power-aware virtual memory including memory allocations and page migrations. For energy efficient compilations, Delaluz et al. [7] proposed compiler optimizations for memory energy consumption of array allocations. They further combined the hardware-directed approach and compiler-directed approaches [8] for more energy saving.

Some research efforts have been devoted to reduce the power consumption of power state transitions. Bi et al. [4] 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. [5] proposed finer grained memory state transition. Those approaches are complementary to the state transition-based energy saving approaches.

B. Other Energy Saving Approaches

We review three categories of other hardware and software approaches for DRAM power management.

The first category is to adjust the refresh rate or voltage and frequency of DRAM. Flikker [24] assigns different refresh rates according to the locality of the data. The idea of Flickker is to keep the critical portion of DRAM refreshed at the regular refresh rate, while the portion containing non-critical data is refreshed at substantially lower rates. Pandey et al. [30] addressed the energy waste during DMA transfers. Memory voltage and frequency scaling is a recent approach to reduce DRAM energy consumption [6], [10].

The second category is on reducing the number of devices involved in a memory request. Mini-rank [37] uses a small bridge chip on each DRAM DIMM to break a conventional DRAM rank into multiple smaller mini-ranks. The goal is to reduce the rank power consumption in a single memory access. Micro-page [32] collocates heavily accessed small chunks from different OS pages in a row-buffer to improve the locality.

The third category is on CPU cache management. Amin et al. [2] proposed rank-aware CPU cache replacement policy for energy efficiency of memory ranks.

Recently, different architectural designs of DRAM systems [1], [20], [38], [33] are explored on multi-core processors for performance, energy, reliability and other issues. Cache-centric optimizations (either cache-conscious [15] or cache-oblivious [14], [13]) 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 [3], [21] and video processing [21].

VI. 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. One is dynamic page migration to consolidate the short idle periods into longer ones and unleash the potential of state transitions, and the other one is an accurate prediction on the demotion time to minimize the delay and energy penalty in state transitions. We evaluate RAMZzz with SPEC 2006 in comparison with other power saving techniques. Our simulation results demonstrate significant improvement on ED^2 and energy consumption over other power saving techniques. Moreover, RAMZzz performs very close to the ideal oracle approach: achieving 10.3% on average larger ED^2 with the optimization goal of ED^2, and only 1.2% on average higher energy consumption with the optimization goal of energy consumption.

Acknowledgement

The authors would like to thank anonymous reviewers for their valuable comments. The work of Donghong Wu was done when he was a visiting student in Nanyang Technological University. This work is supported by an Inter-disciplinary Strategic Competitive Fund of Nanyang Technological University 2011 for "C3: Cloud-Assisted Green Computing at NTU Campus".

REFERENCES

[1] J. H. Ahn, N. P. Jouppi, C. Kozyrakis, J. Leverich, and R. S. Schreiber. Future scaling of processor-memory interfaces. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, pages 42:1–42:12, New York, NY, USA, 2009. ACM.

[2] A. M. Amin and Z. A. Chishti. Rank-aware cache replacement and write buffering to improve dram energy efficiency. In *Proceedings of the 16th ACM/IEEE international symposium on Low power electronics and design*, ISLPED '10, pages 383–388, New York, NY, USA, 2010. ACM.

[3] C. Bae and T. Jamel. Energy-aware memory management through database buffer management. In *Third Workshop on Energy-Efficient Design*, 2011.

[4] M. Bi, R. Duan, and C. Gniady. Delay-hiding energy management mechanisms for dram. In *HPCA'10*, pages 1–10, 2010.

[5] E. Cooper-Balis and B. Jacob. Fine-grained activation for power reduction in dram. *IEEE Micro*, 30:34–47, May 2010.

[6] H. David, C. Fallin, E. Gorbatov, U. R. Hanebutte, and O. Mutlu. Memory power management via dynamic voltage/frequency scaling. In *Proceedings of the 8th ACM international conference on Autonomic computing*, ICAC '11, pages 31–40, New York, NY, USA, 2011. ACM.

[7] V. Delaluz, M. Kandemir, N. Vijaykrishnan, and M. J. Irwin. Energy-oriented compiler optimizations for partitioned memory architectures. In *Proceedings of the 2000 international conference on Compilers, architecture, and synthesis for embedded systems*, CASES '00, pages 138–147, New York, NY, USA, 2000. ACM.

[8] V. Delaluz, M. Kandemir, N. Vijaykrishnan, A. Sivasubramaniam, and M. J. Irwin. Dram energy management using software and hardware directed power mode control. In *Proceedings of the 7th International Symposium on High-Performance Computer Architecture*, HPCA '01, pages 159–, Washington, DC, USA, 2001. IEEE Computer Society.

[9] V. Delaluz, A. Sivasubramaniam, M. Kandemir, N. Vijaykrishnan, and M. J. Irwin. Scheduler-based dram energy management. In *Proceedings of the 39th annual Design Automation Conference*, DAC '02, pages 697–702, New York, NY, USA, 2002. ACM.

[10] Q. Deng, D. Meisner, L. Ramos, T. F. Wenisch, and R. Bianchini. Memscale: active low-power modes for main memory. In *Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems*, ASPLOS '11, pages 225–238, New York, NY, USA, 2011. ACM.

[11] B. Diniz, D. Guedes, W. Meira, Jr., and R. Bianchini. Limiting the power consumption of main memory. In *Proceedings of the 34th annual international symposium on Computer architecture*, ISCA '07, pages 290–301, New York, NY, USA, 2007. ACM.

[12] X. Fan, C. Ellis, and A. Lebeck. Memory controller policies for dram power management. In *Proceedings of the 2001 international symposium on Low power electronics and design*, ISLPED '01, pages 129–134, New York, NY, USA, 2001. ACM.

[13] B. He and Q. Luo. Cache-oblivious query processing. In *CIDR*, pages 44–55, 2007.

[14] B. He and Q. Luo. Cache-oblivious databases: Limitations and opportunities. *ACM Trans. Database Syst.*, 33(2):8:1–8:42, June 2008.

[15] B. He, Q. Luo, and B. Choi. Cache-conscious automata for xml filtering. *IEEE Trans. on Knowl. and Data Eng.*, 18(12):1629–1644, Dec. 2006.

[16] U. Hoelzle and L. A. Barroso. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan and Claypool Publishers, 1st edition, 2009.

[17] H. Huang, P. Pillai, and K. G. Shin. Design and implementation of power-aware virtual memory. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, pages 5–5, Berkeley, CA, USA, 2003. USENIX Association.

[18] H. Huang, K. G. Shin, C. Lefurgy, and T. Keller. Improving energy efficiency by making dram less randomly accessed. In *Proceedings of the 2005 international symposium on Low power electronics and design*, ISLPED '05, pages 393–398, New York, NY, USA, 2005. ACM.

[19] I. Hur and C. Lin. A comprehensive approach to dram power management. In *HPCA'08*, pages 305–316, 2008.

[20] Y. Kim, D. Han, O. Mutlu, and M. Harchol-Balter. Atlas: A scalable and high-performance scheduling algorithm for multiple memory controllers. In *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, pages 1 –12, 2010.

[21] K. Kumar, K. Doshi, M. Dimitrov, and Y.-H. Lu. Memory energy management for an enterprise decision support system. In *Proceedings of the 17th IEEE/ACM international symposium on Low-power electronics and design*, ISLPED '11, pages 277–282, Piscataway, NJ, USA, 2011. IEEE Press.

[22] A. R. Lebeck, X. Fan, H. Zeng, and C. Ellis. Power aware page allocation. In *Proceedings of the ninth international conference on Architectural support for programming languages and operating systems*, ASPLOS-IX, pages 105–116, New York, NY, USA, 2000. ACM.

[23] C. Lefurgy, K. Rajamani, F. Rawson, W. Felter, M. Kistler, and T. W. Keller. Energy management for commercial servers. *Computer*, 36:39–48, December 2003.

[24] S. Liu, K. Pattabiraman, T. Moscibroda, and B. G. Zorn. Flikker: saving dram refresh-power through critical data partitioning. In *Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems*, ASPLOS '11, pages 213–224, New York, NY, USA, 2011. ACM.

[25] D. Meisner, B. T. Gold, and T. F. Wenisch. Powernap: eliminating server idle power. In *Proceedings of the 14th international conference on Architectural support for programming languages and operating systems*, ASPLOS '09, pages 205–216, New York, NY, USA, 2009. ACM.

[26] Micron Technology, Inc. *MT41J128M8BY-187E*. http://download.micron.com/pdf/datasheets/dram/ddr3/1Gb DDR3 SDRAM.pdf, 2009.

[27] Micron Technology, Inc. *MT47H64M16HR-25E*. download.micron.com/pdf/datasheets/dram/ddr2/1GbDDR2.pdf, 2010.

[28] Micron Technology, Inc. *System Power Calculator*. http://www.micron.com/support/designsupport/tools/powercalc/powercalc.aspx, 2012.

[29] 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.*, 43:92–105, January 2010.

[30] V. Pandey, W. Jiang, Y. Zhou, and R. Bianchini. Dma-aware memory energy management. In *HPCA'06*, pages 133–144, 2006.

[31] L. E. Ramos, E. Gorbatov, and R. Bianchini. Page placement in hybrid memory systems. In *Proceedings of the international conference on Supercomputing*, ICS '11, pages 85–95, New York, NY, USA, 2011. ACM.

[32] K. Sudan, N. Chatterjee, D. Nellans, M. Awasthi, R. Balasubramonian, and A. Davis. Micro-pages: increasing dram efficiency with locality-aware data placement. In *Proceedings of the fifteenth edition of ASPLOS on Architectural support for programming languages and operating systems*, ASPLOS '10, pages 219–230, New York, NY, USA, 2010. ACM.

[33] 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 *Proceedings of the 37th annual international symposium on Computer architecture*, ISCA '10, pages 175–186. ACM, 2010.

[34] 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*, pages 1–11, 2010.

[35] Xilinx, Inc. *Spartan-6 FPGA Memory Controller User Guide*. http://www.xilinx.com/support/documentation/user_guides/ug388.pdf, 2010.

[36] M. T. Yourst. Ptlsim: A cycle accurate full system x86-64 microarchitectural simulator. In *ISPASS*, 2007.

[37] H. Zheng, J. Lin, Z. Zhang, E. Gorbatov, H. David, and Z. Zhu. Mini-rank: Adaptive dram architecture for improving memory power efficiency. In *Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 41, pages 210–221, Washington, DC, USA, 2008. IEEE Computer Society.

[38] H. Zheng and Z. Zhu. Power and performance trade-offs in contemporary dram system designs for multicore processors. *IEEE Trans. Comput.*, 59:1033–1046, August 2010.

[39] Y. Zhou, J. Philbin, and K. Li. The multi-queue replacement algorithm for second level buffer caches. In *Proceedings of the General Track: 2002 USENIX Annual Technical Conference*, pages 91–104, Berkeley, CA, USA, 2001. USENIX Association.