Improving Energy Efficiency by Making DRAM Less Randomly Accessed

Hai Huang, Kang G. Shin
University of Michigan
{haih,kgshin}@eecs.umich.edu

Charles Lefurgy, Tom Keller
IBM Austin Research Lab
{lefurgy,tkeller}@us.ibm.com

ABSTRACT

Existing techniques manage power for the main memory by passively monitoring the memory traffic, and based on which, predict when to power down and into which low-power state to transition. However, passively monitoring the memory traffic can be far from being effective as idle periods between consecutive memory accesses are often too short for existing power-management techniques to take full advantage of the deeper power-saving state implemented in modern DRAM architectures. In this paper, we propose a new technique that will actively reshape the memory traffic to coalesce short idle periods — which were previously unusable for power management — into longer ones, thus enabling existing techniques to effectively exploit idleness in the memory.

Categories and Subject Descriptors

D.4.2 [**Main Memory**]: Storage Management; H.3.4 [**Performance Evaluation**]: Systems and Software

General Terms

Experimentation, Measurement, Performance

Keywords

DDR, low power, memory system

1. INTRODUCTION

This paper focuses on improving energy efficiency of main memory built with DRAM. This is motivated by a continual increase in the power budget allocated to the memory subsystem. For example, it has been reported that as much as 40% of the total system energy is consumed by the main memory subsystem in a mid-range IBM eServer machine [8]. As applications are becoming increasingly data-centric, we expect main memory to remain as a significant energy consumer because achieving good overall system performance will be more likely to depend on having higher-performance and larger-capacity DRAM.

Recently, various power-saving techniques have been proposed by exploiting power-management capabilities built into modern DRAM devices. Lebeck *et al.* [7, 4] studied the effects of static and dynamic memory controller policies on power and performance using extensive simulation in a single-process environment. Delaluz *et al.* [2] proposed various threshold predictors to determine after how long of an idle period should the memory controller transition a DRAM device to a low-power state. Regardless of which memory controller policy or threshold predictor we use, the presence of idle period in DRAM is essential for reducing power. However, not all idle periods can be exploited because state transitions take non-negligible amount of time and energy. Rather, it is only beneficial to transition a memory device to a low-power state if it stays idle for longer than this state's *break-even* time [4]. However, the break-even time can vary significantly among the different low-power states, and the deeper power-saving states usually have longer break-even time as more components are disabled and would take more time and energy to transition out of these states to service new memory requests. Therefore, for us to utilize these deep power-saving states, long idle periods in the memory traffic are essential. Unfortunately these long idle periods are not commonly found in realistic workloads as physical memory is usually randomly accessed and driven completely by the current process' execution. Since existing power-management techniques only passively monitor memory traffic, due to lack of these long idle periods, deep power-saving states are rarely fully exploited. In this paper, we propose a new technique to minimize short and unusable idle periods and create longer ones. From such reshaped memory traffic, existing techniques are able to make better use of the idleness in the memory, thus saving more energy. Lebeck *et al.* [7] briefly mentioned a frequency-based technique that is similar to our work, but they failed to recognize how such technique can be used to complement existing power-management techniques. Furthermore, unlike their work, we propose a practical technique that could be implemented in real systems using conventional operating systems and hardware. A more thorough evaluation is also presented here to fully assess benefits and problems of this technique.

In addition to these hardware-controlled techniques, some software-controlled techniques have also been proposed. Delaluz *et al.* [3] demonstrated a simple scheduler-based power-management policy. Huang *et al.* [5] later implemented Power-Aware Virtual Memory to improve upon this work. Even though software techniques usually have less performance impact, hardware techniques can save more energy by leveraging finer-grained run-time information, which is mostly unavailable to system software. Software-hardware cooperative technique [6] has also been proposed.

In the next section, we will first give some background information on the current state-of-art of DRAM technology. Section 3 describes our memory traffic reshaping mechanism. Our simulation setup and evaluation are presented in Section 4, and finally, we conclude the paper in Section 6.

2. BACKGROUND

In this paper, we use the terminology of the Double-Data Rate (DDR) memory architecture to describe our approach, simply because DDR is becoming the most common type of memory used in today's PC and server systems. However, by no means our approach is limited to only DDR; one can easily apply this technique to other memory types, e.g., SDR and RDRAM. We will now give some background information on DDR memory architecture and discuss its performance–energy tradeoff.

ISLPED'05, August 8–10, 2005, San Diego, California, USA.

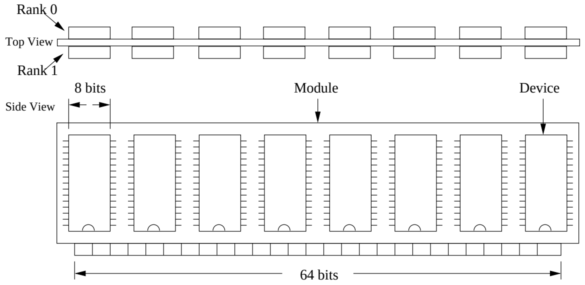
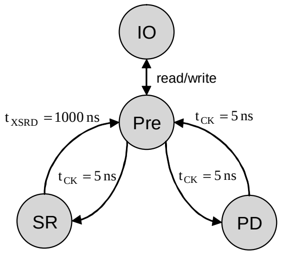


Figure 1: A memory module, or a DIMM, that is composed of 2 ranks (front and back), and each with 8 devices.

2.1 Double-Data Rate Memory Model

DDR is usually packaged as modules, or DIMMs, each of which usually contains either 1, 2, or 4 *ranks*, which are commonly composed of 4, 8, or 16 physical devices (shown in Figure 1). When managing power for the memory, a rank is the smallest physical unit we can control. Power can be reduced on a rank when some of its subcomponents (i.e., row and column decoders, sense amplifiers, bus drivers, etc.) are disabled by switching this rank to one of the several pre-defined low-power states. However, if a rank is accessed while at a low-power state, performance penalty, called *re-synchronization cost*, is incurred to transition this rank from the current low-power state to an active state so it can be accessed again.



	State	Power
IO	Read/Write	1200 mW
PRE	Precharge	137.5 mW
PD	Powerdown	12.5 mW
SR	Self Refresh	10 mW

(a)

PLL	Power
Low-power	0.05 mW
High-power	750 mW

Register	Power
Low-power	0.025 mW
High-power	117.5 mW

(b)

Figure 2: Part (a) shows the power dissipation of each state and the delays to transition between them for a single 512-Mbit DDR-400 device. For Read/Write state, we show its maximum power dissipation when all banks on a device are actively reading. Part (b) shows the power dissipation of a TI PLL device (one per DIMM) and a TI register.

DDR has many power states defined and even more possible transitions between them [10]. These states and transitions are simulated in our memory simulator, which we used to evaluate our work in Section 4.3. However, for simplicity of presentation, we only show four of these power states here — Read/Write, Precharge, Powerdown, and Self Refresh — listed in a decreasing order of power dissipation. In Figure 2(a), we show the power dissipation of these states and the state transition delays. Note that the power numbers shown here are for a single device. Therefore, to calculate the total power dissipated by a rank, we need to multiply this power by the number of devices in the rank. For a 512MB registered DIMM consisting of 8 devices, the expected power draw values are 10.47 W, 1.97 W, 0.97 W, and 0.08 W (including energy consumed by PLL and registers), respectively for the four power states considered here. Details of these power states are as follows.

- **Read/Write:** Dissipates the most power, but it is only briefly entered when a read/write operation is in progress.

- **Precharge:** When a rank is neither reading nor writing, Precharge is the highest power state, or the most-ready state, in which read and write operations can start immediately at the next clock edge.

- **Powerdown:** When this state is entered, the input clock signal is gated except for the auto refresh signal. I/O buffers, sense amplifiers and row/column decoders are all deactivated in this state.

- **Self Refresh:** In addition to all components that are deactivated in Powerdown, the phase-lock loop (PLL) device and registers can also be put to low-power state. This gives maximum power reduction as the PLL and the registers (Figure 2(b)) can consume a significant portion of the total energy on each DIMM. However, when exiting from Self Refresh, a 11 *μsec* delay is incurred — 10 *μsec* is due to re-synchronizing both the PLL and the registers and the other 1 *μsec* is due to re-synchronizing DRAM's internal Delay Lock Loop (DLL) device with the PLL.[1]

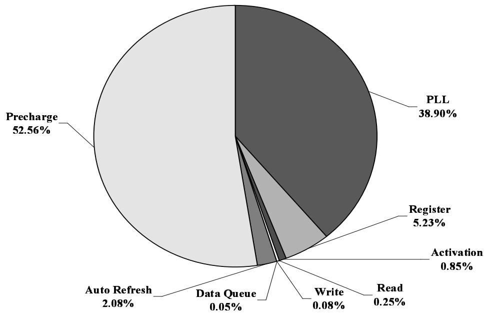


Figure 3: Breakdown of the energy consumed by DRAM.

2.2 Power Management

Even though read and write operations dissipate the most amount of power, they do not consume a significant amount of energy due to their short duration. Instead, most of the energy is consumed when memory is idling. We show in Figure 3 that for a SPECjbb workload, energy is mostly consumed in Precharge state and by the peripheral components, i.e., PLL and registers (details on the workload and on our memory simulator are given in Section 4). This suggests that power can be significantly reduced by transitioning memory devices and the peripheral components to their low-power state during idle periods.

Powerdown is one of the two low-power states implemented in DDR memory devices, and it uses 49% of the Precharge power. Having only a 5 nsec re-synchronization latency, using Powerdown, power can be reduced even with short idle periods; however, it is not nearly as power-efficient as Self Refresh where we can also put the PLL and the registers to their low-power state. Its benefit is clearly shown in Figure 3. However, due to having a much longer re-synchronization latency when exiting from Self Refresh, idle periods of at least 19 *μsec* are needed just to break even. This is more than 3 orders of magnitude larger than the break-even time for entering Powerdown. We calculate the break-even time using the Energy×Delay metric as shown in [4]. Its calculation is omitted here due to space limitation.

Unfortunately, in realistic workloads, due to the randomness in memory accesses, long idle periods are rarely observed. As a result, it inhibits the use of Self Refresh, which severely limits the amount of power saved from using existing power-management

[1] Registered memory is almost always used in server systems to better meet timing needs and provide higher data integrity, and the PLL and registers are critical components to take into account when evaluating registered memory in terms of performance and energy.

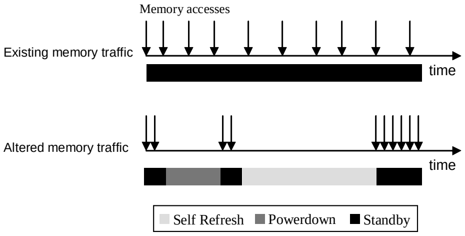


Figure 4: In the first case (above figure), gaps between consecutive memory accesses are too short for entering low-power states to obtain any savings. In the second case, by delaying and batching memory accesses, we can create longer idle periods, thus allowing power management to take advantage of various low-power states.

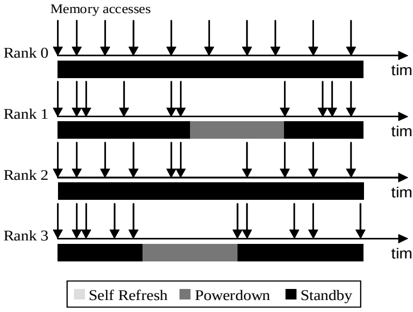


Figure 5: An example showing that if memory traffic is left unshaped, power management cannot take full advantage of deeper power-saving states since most idle periods are too short.

techniques. This is illustrated by an example shown in Figure 4, where we show that simply making power-management decisions based on the monitored memory traffic is often not enough — the observed memory traffic might not present the necessary energy-saving opportunities. However, if we can alter the traffic pattern in a certain way, it is possible to create longer idle periods, from which power can be more effectively reduced. Unfortunately, the particular technique we show in Figure 4 is not very useful in practice as we cannot control memory accesses at such a fine granularity. Additionally, by delaying and batching memory accesses, we pay a severe performance penalty. In the following section, we illustrate a more practical and low-overhead method of reshaping memory traffic to improve energy efficiency.

3. MEMORY TRAFFIC RESHAPING

To reshape the memory traffic for our benefit, we must make memory accesses less random and more controllable. Conventional memory traffic often seems random because (1) the operating system arbitrarily maps virtual pages to physical pages, and (2) different pages are often accessed very differently at run-time. As a result of such randomness, the interarrival characteristic of memory requests observed on each rank might not be favorable for existing techniques to manage power.

To give an example, we use a 4-rank system shown in Figure 5. Due to the arbitrary OS's page mapping, memory requests are likely to be randomly distributed among the 4 ranks. This creates a large number of small and medium-sized idle periods. The smaller idle periods are often completely useless and cannot be used for saving energy. As for the medium-sized ones, we can transition memory devices to Powerdown and obtain a moderate amount of power savings. However, to significantly reduce power, we need to take advantage of Self Refresh's ultra-low power property. Unfortunately, as it can be seen from this example, due to the lack of long idle periods, Self Refresh is very infrequently utilized.

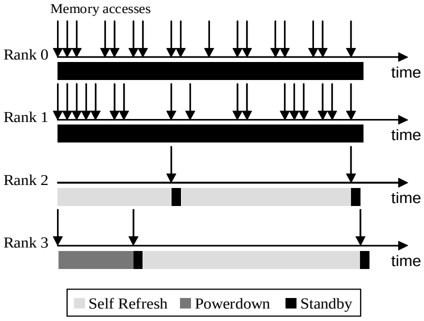


Figure 6: An example showing that if memory traffic can be loosely controlled (e.g., by migrating pages), some ranks will, as a result, have much longer idle periods, thus allowing the use of the deeper power-saving states.

3.1 Hot Ranks and Cold Ranks

To elongate idle periods, we introduce the concepts of *hot* and *cold* ranks. Hot ranks are used to hold frequently-accessed pages, which leaves infrequently-used and unmapped pages on cold ranks. Hot ranks are created by migrating frequently-accessed pages from cold ranks to hot ranks. The mechanism to migrate pages from one rank to another was previously described in full detail in [5]. The result of making this differentiation among ranks is shown in Figure 6. Here we assume that Rank 0 and 1 are used as hot ranks and Rank 2 and 3 are used as cold ranks. Essentially, we are increasing the utilization of hot ranks and decreasing the utilization of cold ranks. As a result, the additional memory requests imposed upon these hot ranks will “fill-in” between the idle gaps. As most of these gaps were small and could not be used for saving power, by servicing additional requests during such times, we are making more efficient use of the power dissipated by the hot ranks. Although this might cause hot ranks to lose some energy-saving opportunities to use Powerdown (e.g., Rank 1 shown in Figure 5 and Figure 6), but as a result of that, more valuable opportunities are created on cold ranks where Self Refresh can be more utilized. In our experiments, we found that the average interarrival time was elongated by almost 2 orders of magnitude on cold ranks.

3.2 Reducing Migration Overhead

Migrating pages causes additional memory traffic, which results in more queuing delays and contentions. Therefore, only a small number of pages can be moved without causing noticeable overhead. Fortunately, empirical observations from our experiments gave us some hints that allow us to do just that and still be able to reshape the memory traffic according to our needs. Memory traces collected from several workloads indicate that only a small percentage of pages are responsible for a majority of the memory traffic. This is shown in Figure 7, and we summarize the results in Table 3.2. From this table, it is clear that we can reshape the memory traffic to meet our needs by migrating only a very small percentage of pages. For example, if we want to control 90% of all memory traffic, we only need to control 1.5–14.3% of all pages. Only half of these pages would need to be migrated because pages are randomly allocated, and on average, 50% of the frequently-accessed pages should have already been allocated on the hot ranks and do not need to move. Furthermore, since migration overhead is only a one-time cost, the longer a migrated page stays hot, the more we can amortize its migration cost over time. We can also think of other heuristics that we can use to further reduce the number of migrations and the migration overheads. For example, we can use process profiling to better predict the appropriate initial location where we should allocate pages for each process so that the number of page migrations can be reduced. Additionally, we can reduce migration overhead by avoiding moving heavily-shared pages, page-cache pages, and buffer-cache pages as there are more overheads in moving these types of pages. In particular, to move a shared page, we would need to change the page table entries of each of the

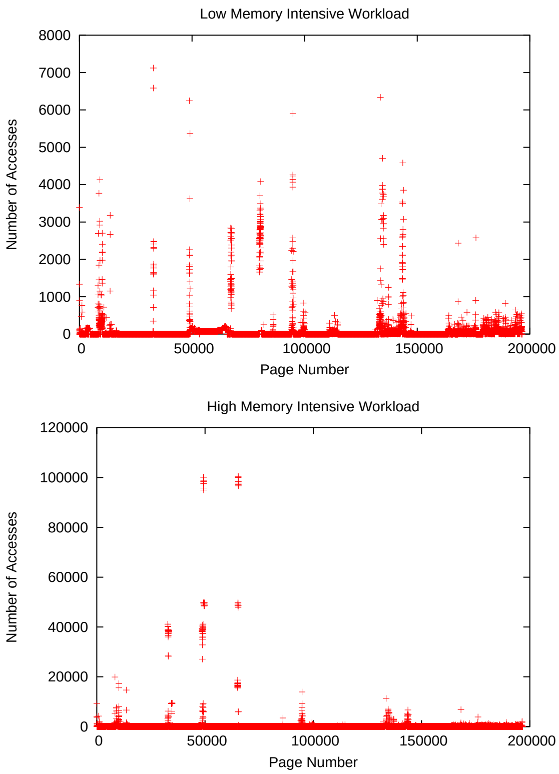


Figure 7: Number of times each physical page in the memory is accessed for low memory-intensive workload and high memory-intensive workload. These workloads are described in Section 4.

	75 Percentile	90 Percentile	95 Percentile	99 Percentile
LowMem Workload	5.68%	14.27%	18.60%	25.81%
HighMem Workload	0.53%	1.49%	4.98%	16.38%

Table 1: Shows what percentage of all pages is responsible for 75%, 90%, 95% or 99% of all memory accesses.

processes sharing this page; the more processes sharing this page, the more page tables we would need to modify. Therefore, the best candidate pages to migrate are frequently-accessed private pages.

3.3 Implementation

To determine which page to migrate, we keep a count on the number of times each page was accessed. This is used by a kernel thread to find frequently access pages on cold ranks so these pages can be migrated. Currently, the page access-count table is maintained by the memory controller in our simulator. Alternatively, the same can be achieved by using software page faults to avoid hardware modification — sampling may be used to reduce page fault overheads.

4. EVALUATION

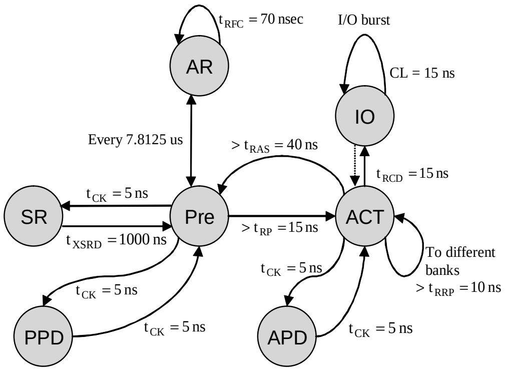
In this section, we evaluate the energy benefits in reshaping the memory traffic. We first describe our simulation setup and the workloads we used, in Section 4.1 and 4.2, respectively. In Section 4.3, we show results from our simulations.

4.1 Simulation Setup

Mambo [1] is a full-system simulator that can simulate various PowerPC® machine architectures, and it is currently in active use by multiple research and development groups at IBM. We used it for running workloads and collecting memory traces. In our simulation study, the Mambo-simulated machine is parameterized as that shown in Table 2. We also implemented a trace-driven main memory simulator using the CSIM [9] library. It can accurately model performance and power dissipation of the memory by simulating a detailed DDR state machine (shown in Figure 8). Furthermore, it can also simulate various effects of queuing and contention occurring at the memory controller, synchronous memory interfaces (SMIs), and on various buses. Power dissipation of memory devices is calculated by keeping track of the state information for each bank on a per-cycle basis, as was described by [11].

Component	Parameter
Processor	64-bit 1.6 GHz PowerPC
DCache	64KB 2-way Set-Associative
ICache	32KB 4-way Set-Associative
L2-Cache	1.5MB 4-way Set-Associative
DTLB	512 entries 2-way Set-Associative
ITLB	512 entries 2-way Set-Associative
DERAT	128 entries 4-deep
IERAT	128 entries 4-deep
SLB	16 entries
Memory	DDR-400 768MB (64Mbx8)
Linux Kernel	2.6.5-rc3 w/ PAVM patch

Table 2: System parameters used in Mambo. All cache lines are 128 Bytes long.



	State	Power
PRE	Precharge	137.5 mW
PPD	Precharge-Powerdown	12.5 mW
ACT	Active	150 mW
APD	Active-Powerdown	112.5 mW
SR	Self-refresh	10 mW
AR	Auto-refresh	27.5 mW
IO	Read/Write	1200 mW

Figure 8: Detailed DDR state machine that we simulate in our memory simulator. Some minor states and transitions are omitted from this graph for better viewing.

4.2 Workloads

In our evaluation, we use two workloads, classified as either "*low memory-intensive*" or "*high memory-intensive*", based on L2 miss rates [12], to study the effect of memory access intensity on our proposed technique. For the low memory-intensive workload, we run SPECjbb having 8 warehouses in parallel with *bzip2* and *crafty* from the SPEC CPU2K benchmarks, and for the high memory-intensive workload, we run SPECjbb in parallel with *mcf* and *art* from the SPEC CPU2K benchmarks. *Reference* input sets are used for all the SPEC CPU2K benchmarks.

4.3 Results

Our memory simulator can simulate various power-management techniques. To compare our technique with the previously-

```
$t_{REFI}$          = Auto refresh interval (7.8 usec)
$_iT_{i+1}$         = Time interval between $i^{th}$ and $(i+1)^{th}$ refresh interval
$_iN_{i+1}[x]$      = Number of memory accesses within $_iT_{i+1}$ on rank x
$Th[x]$             = Threshold to enter Self Refresh on rank x
$Th_{Max}$          = Maximum threshold value

    Threshold Prediction Algorithm:

    At the end of the $i^{th}$ refresh interval
      For each rank x
        if ($_iN_{i+1}[x] == 0$) then
          if ($Th[x] < t_{REFI}$) then
            $Th[x] = Th[x]/2$
          else $Th[x] = Th[x] - t_{REFI}$
        else
          $Th[x] = Th[x] + {_iN_{i+1}}[x] * t_{REFI}$
          if ($Th[x] > Th_{Max}$)
            $Th[x] = Th_{Max}$
        if (rank x is not currently in Self Refresh)
          enter(Powerdown)
          if (idle time has exceeded Th[x]) then
            enter(Self Refresh)
```

Figure 9: Threshold prediction algorithm used in the hardware-controlled power-management technique.

Power Management	Power	Normalized Runtime	Average Response Time
NOPM	49.97 W	1.000	72.01 ns
IPD	25.54 W	1.018	82.17 ns
ISR	6.23 W	1.992	697.60 ns
HW	10.24 W	1.035	90.14 ns
HW_5	6.26 W	1.056	95.91 ns

Table 3: Summary of the energy and performance results for the low memory-intensive workload.

Power Management	Power	Normalized Runtime	Average Response Time
NOPM	52.15 W	1.000	84.13 ns
IPD	28.30 W	1.050	92.18 ns
ISR	14.47 W	3.795	537.45 ns
HW	14.79 W	1.056	93.21 ns
HW_5	9.52 W	1.190	106.25 ns

Table 4: Summary of the energy and performance results for the high memory-intensive workload.

proposed ones, we evaluate five techniques in power and performance, which are listed as follows.

- **No Power Management (NOPM):** Here, no power-management technique is used, and ranks are transitioned to Precharge when they are idle.
- **Immediate Powerdown (IPD):** This is the simplest form of hardware power management. It is a static technique where the memory controller immediately transitions a rank to Powerdown when all memory requests on this rank have completed.
- **Immediate Self Refresh (ISR):** Same as IPD, but transitions to Self Refresh instead of to Powerdown.
- **Dynamic Hardware Technique (HW):** This is a dynamic hardware power-management technique and is similar to the History-Based Predictor described in [2]. It monitors past memory accesses, and based on which, predictions after how long of an idle period should it transition a rank to Self Refresh (the threshold prediction algorithm is shown in Figure 9). Transitions to Powerdown have a zero threshold, which was previously shown to be the most efficient [7].
- **HW with a Reshaped Memory Traffic (HW_x):** Using the same HW technique as above but with a reshaped memory traffic. The x in HW_x represents the percentage of all pages that we migrate when reshaping the memory traffic, i.e., HW_0 is the same as HW.

Results for the low memory-intensive and the high memory-intensive workloads are shown in Tables 3 and 4, respectively. In these tables, we show the average power dissipation, the normalized runtime (with respect to no power management), and the average response time of memory accesses for each of the power-management techniques. We can see that even with the simplest static power management, IPD, a significant amount of power (45.73–48.89%) can be reduced without causing much impact on the performance (1.8–5.0%). Using ISR, additional power can be reduced. However, due to having a static policy to transition into Self Refresh and a much higher resynchronization latency when exiting from Self Refresh, ISR's overwhelming performance penalty (99.2–279.5%) makes it almost impractical to use in realistic workloads. On the other hand, using the dynamic hardware technique (HW), we show that if Self Refresh is utilized more carefully, a significant amount of power can be reduced (71.64–78.57%) but without significantly affecting the performance (3.5–5.6%).

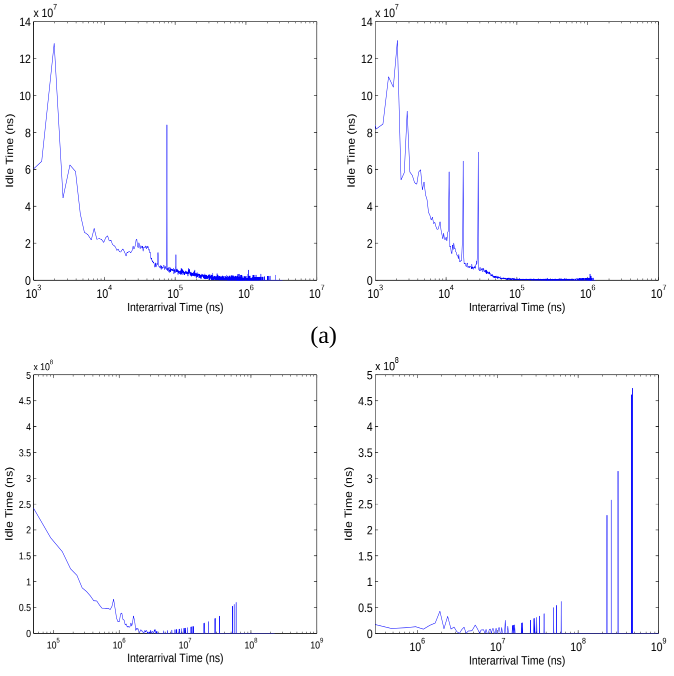
4.3.1 Effect of Reshaping on Memory Traffic

Among these four techniques, HW is by far the most effective because it can dynamically adapt its power-management decisions as the memory traffic's characteristic changes. To understand the implications of memory traffic reshaping, we re-evaluated HW with a reshaped memory traffic. However, before we delve into that, we will first look at the effects of migrating frequently-accessed pages from cold ranks to hot ranks on the existing memory traffic. In Figure 10, we show how the idle time characteristic (i.e., the distribution of the total idle time among different-sized idle periods) on a hot rank and a cold rank has changed after we reshaped the memory traffic. Due to the need to serve more memory requests, the average idle period on hot ranks decreased from 3,630 nsec to 472 nsec. This causes some opportunities to be lost for entering low-power state, but since we can benefit from entering Powerdown even with short idle periods, not much is really lost here. Moreover, by redirecting a significant number of memory accesses from cold ranks to these hot ranks, much longer idle periods are created on cold ranks. We found that after the migration, the average idle period increased from 1,520 nsec to 122,210 nsec (Figure 10(b)). This created more valuable opportunities where Self Refresh can be exploited.

Results of using HW to manage power on a reshaped memory traffic are shown in Tables 3 and 4, labeled as HW_5, for the low memory-intensive and the high memory-intensive workloads, respectively. Here, we migrated 5% of pages to reshape the memory traffic. By doing so, we achieved 35.63–38.87% additional power savings than the original HW technique. However, due to the additional contention created at the hot ranks and the extra memory accesses needed to migrate pages, the performance is degraded. In the low memory-intensive workload, the performance degradation is only 2.0% compared to HW. However, in the high memory-intensive workload, performance is degraded by 12.7%. The reason for this is that pages are much more frequently accessed in the high memory-intensive workload,[2] and therefore, as a result of migrating frequently-accessed pages onto the hot ranks, the contention created on hot ranks is much more severe in the high memory-intensive workload.

To study the effect of memory traffic reshaping in more detail, we compare the results of migrating 1%, 5%, and 10% of pages. These are shown in Figure 11, where we normalized average power and average runtime to that of using HW with a unshaped memory traffic. Here we can see, migrating only 1% of pages gives only lim-

[2] Both workloads run for the same amount of time, but the high memory-intensive workload has 6 times more memory accesses than the low memory-intensive workload.



(a)

(b)

Figure 10: Part (a) shows idle time characteristic of a hot rank before (left) and after (right) migrating frequently-accessed pages. Part (b) shows idle time characteristic of a cold rank before (left) and after (right) migrating frequently-accessed pages. These are derived from the low memory-intensive workload. High memory-intensive workload gives similar result, thus is omitted here.

ited benefits in power reduction. On the other hand, migrating 10% of pages does not give any additional energy benefit beyond that of migrating 5%. In addition, it also suffers from more performance penalty due to having to migrate more pages. Therefore, migrating 5% of pages gives the best result for the workloads we ran.

4.3.2 Discussion

When memory is extensively accessed, as in the high memory-intensive workload, performance degradation due to contention on hot ranks can be more of a concern. However, this can be alleviated by implementing a detection mechanism that stops migration or even triggers a "reverse migration" when excessive contention is observed on hot ranks. However, this only minimally alleviates the problem. As we can see from Figure 11, migrating 1% as opposed to 5% of pages does not give much benefit in reducing performance penalty.

To solve the problem at its root, it calls for an alternative main memory design, where we should use high-performance, highly parallel memory on hot ranks and low-performance/low-power memory on cold ranks. This allows faster access time for more frequently-accessed pages and fewer contentions on hot ranks. It also allows for more energy savings with the use of low-power memory on cold ranks. Additionally, by using low-performance memory on cold ranks, this heterogenous main memory design can potentially lower the monetary cost of building the main memory subsystem.

5. ACKNOWLEDGMENT

We would like to thank Karthick Rajamani for his help on Memsim and Jim Peterson on Mambo at IBM Austin. This work is partially funded by AFOSR, Grants F49620-01-1-0120 and FA9550-05-0287.

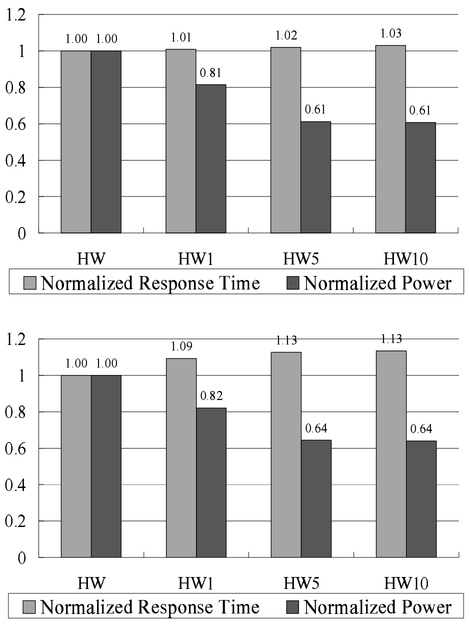


Figure 11: Effects of actively reshaping memory traffic by migrating 1%, 5%, and 10% of pages for the low memory-intensive workload (above) and high memory-intensive workload (below).

6. CONCLUSION

In this paper, we propose how to actively reshape memory traffic to produce longer idle periods so we can more effectively exploit idleness in the memory. Our extensive simulation in a multitasking system shows that a 35.63–38.87% additional energy can be saved by complementing existing power-management techniques with this proposed technique. Our result also indicates that an alternative main memory design could be more efficient than today's homogenous design in power efficiency, performance and cost.

7. REFERENCES

[1] P. Bohrer *et al.*, "Mambo —A Full System Simulator for the PowerPC Architecture", *ACM SIGMETRICS Performance Evaluation Review*, 31(4), 8–12, 2004.

[2] V. Delaluz *et al.*, "Hardware and Software Techniques for Controlling DRAM Power Modes", *IEEE Transactions on Computers*, 50(11),1154–1173, 2001.

[3] V. Delaluz *et al.*, "Scheduler-based DRAM Energy Power Management", *Design Automation Conference 39*, 697–702, 2002.

[4] X. Fan and C. S. Ellis and A. R. Lebeck, "Memory Controller Policies for DRAM Power Management", *International Symposium on Low Power Electronics and Design (ISLPED)*, 129–134, 2001.

[5] H. Huang and P. Pillai and K. G. Shin, "Design and Implementation of Power-Aware Virtual Memory", *USENIX Annual Technical Conference*, 57–70, 2003.

[6] H. Huang *et al.*. "Cooperative Software–Hardware Power Management for Main Memory", *Workshop on Power-Aware Computer Systems*, 2004.

[7] A. R. Lebeck *et al.*, "Power Aware Page Allocation", *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 105–116, 2000.

[8] C. Lefurgy and K. Rajamani and F. Rawson and W. Felter and M. Kistler and Tom Keller, "Energy Management for Commercial Servers", *IEEE Computer*, 39–48, 2003.

[9] Mesquite Software, "http://www.mesquite.com".

[10] Micron, "http://www.micron.com".

[11] Micron, "http://download.micron.com/pdf/technotes/TN4603.pdf".

[12] Karthikeyan Sankaralingam *et al.*, "Exploiting ILP, TLP, and DLP with the Polymorphous TRIPS Architecture", *ISCA*, 422–433, 2003.