A Comprehensive Approach to DRAM Power Management

Ibrahim Hur[†]

[†]IBM Corporation
Systems and Technology Group
Austin, TX
ibrahur@us.ibm.com

Calvin Lin[‡]

[‡]The University of Texas at Austin
Department of Computer Sciences
Austin, TX
lin@cs.utexas.edu

Abstract

This paper describes a comprehensive approach for using the memory controller to improve DRAM energy efficiency and manage DRAM power. We make three contributions: (1) we describe a simple power-down policy for exploiting low power modes of modern DRAMs; (2) we show how the idea of adaptive history-based memory schedulers can be naturally extended to manage power and energy; and (3) for situations in which additional DRAM power reduction is needed, we present a throttling approach that arbitrarily reduces DRAM activity by delaying the issuance of memory commands. Using detailed microarchitectural simulators of the IBM Power5+ and a DDR2-533 SDRAM, we show that our first two techniques combine to increase DRAM energy efficiency by an average of 18.2%, 21.7%, 46.1%, and 37.1% for the Stream, NAS, SPEC2006fp, and commercial benchmarks, respectively. We also show that our throttling approach provides performance that is within 4.4% of an idealized oracular approach.

1 Introduction

Because DRAMs can account for a significant amount of a system's total power consumption [24], chip designers have begun to seek active ways to manage DRAM power. Two possible power management goals have emerged. The first goal is to improve energy efficiency, which translates into lower energy bills. The second goal is to provide a mechanism for throttling the flow of memory commands to ensure that power consumption falls within some power budget. Such throttling may decrease performance, but it has at least two benefits.

- Throttling can reduce system costs. Because of the large disparity between worst case and expected case power consumption, it is expensive—both in terms of the power supply and the cooling system—to provision a system for the worst case. If a system can instead dynamically throttle its power to reduce worst case power consumption, the system can be provisioned to be much less expensive.

- Throttling supports *Power Shifting* [12], a technique that dynamically assigns a power budget to different system components, such as the CPU and DRAM, to maximize performance for a given workload and a given power budget. Power Shifting assumes that each subsystem can throttle its power consumption to stay within its given budget.

For DRAM, one mechanism for addressing both of these power goals is to put idle memory devices into a low power mode. Although DRAMs with low power modes are commercially available, no specific policy for their use has been evaluated for commercially available server-class systems. Because there are latencies associated with entrance into and exit from the various low power modes, it is difficult to know when to transition into and out of low power mode. A policy that toggles modes too frequently can increase the latency of memory commands, thereby reducing performance. A policy that transitions to low power mode too slowly will miss opportunities to save power, while a policy that transitions out of low power mode too slowly will unnecesarily degrade performance.

Low power mode also plays an important role in throttling. By forcing memory commands to wait in the memory controller, DRAM structures can remain in low power mode for arbitrarily long periods of time, thereby modulating the DRAM's average power consumption over some small time interval. The key difficulty is to determine the minimum *throttling delay*—the period of time for which memory commands will be blocked in the memory controller—that is needed to stay below a given power threshold. The determination of this delay is complicated by the complex

parallel structure of modern DRAMs and the workload-dependent distribution of memory commands. Consider a particular point in the execution where a throttling delay of t cycles is ideal for a given power threshold. If the power management system is only able to estimate that a delay of $t \pm \delta$ cycles will suffice, then the power management system will be forced to conservatively choose a longer target delay of $t + \delta$ cycles, resulting in unnecessary performance loss. Thus, for a given power threshold, a more accurate estimate of throttling delay translates to increased performance.

This paper addresses both of the above power management goals by describing policies for putting memory devices into low power mode. We propose small changes to the memory controller that significantly improve DRAM energy efficiency and support accurate power throttling. We evaluate our solutions by using extremely accurate simulators for the IBM Power5+ processor and a DDR2-533 SDRAM. In particular, this paper makes three contributions:

1. We describe and evaluate a simple and practical policy for using the DRAM power-down mechanism. We show that when compared against a baseline system that does not use the power-down mechanism, our policy increases DRAM energy efficiency by an average of 11.6%, 18.1%, 43.4%, and 34.2% for the Stream, NAS, SPEC2006fp, and commercial benchmarks, respectively.

2. We present a small change to the Adaptive History-Based Scheduler (AHB) [16] that adds power consumption as a scheduling criterion. This modified AHB scheduler increases the average idle duration of each rank, thereby increasing the utility of the power-down unit. When combined with our power-down policy, our scheduler increases DRAM energy efficiency by an average of 18.2%, 21.7%, 46.1%, and 37.1% for the Stream, NAS, SPEC2006fp, and commercial benchmarks, respectively, and it decreases performance by 2.7%, 1.2%, 0.8%, and 0.6%, respectively.

3. We present a throttling approach that uses an accurate delay estimation model. This delay model is the main conceptual contribution of this paper, and the key idea is to build an offline regression model that uses only a small number of input parameters, which allows the dynamic overhead of the estimator to be low. Our delay estimation model provides performance that is within 4.4%, 0.9%, 1.3%, and 2.7% of a perfect oracular model, for the Stream, NAS, SPEC2006fp, commercial benchmarks, respectively. By contrast, our baseline model, which was proposed by others [12], degrades performance by 29.6%, 20.7%, 18.9%, and 16.4% for these same benchmark suites.

The remainder of this paper is organized as follows. The next section places our work in the context of prior work. Section 3 describes our solution. We then describe our experimental methodology in Section 4, present our empirical evaluation in Section 5, and conclude in Section 6.

2 Related Work

Much of the early work in memory system power management has focused on embedded systems and laptops, where performance loss has been less of an issue [2].

Delaluz et al. [8] control the use of low power mode by having the memory controller predict the idle duration of various memory devices. They demonstrate good results for cacheless systems using Rambus DRAM. Fan et al. [11] extend this work for systems with multi-level caches, and Irani et al. [19] give a theoretical analysis of dynamic power management in memory controllers. These approaches are difficult to tune because they use thresholds, which are system and application dependent.

Previous hardware-based approaches for DRAM power savings assume FIFO scheduling in the memory controller. However, it has been shown that better memory scheduling approaches can substantially improve performance [30, 6, 28, 33, 16]. Such approaches improve performance by reducing gaps between commands. Since threshold-based predictive algorithms passively monitor memory traffic to schedule power-down commands, we expect that shorter gaps will make those algorithms less effective. By contrast, our work takes an active approach and reorders commands to save power while preserving performance.

Compiler-directed [21, 35, 27, 7] and operating system-based methods [26, 36, 23, 9] have also been proposed to save DRAM power. For modern systems with multi-level caches, multiple threads, or shared memory controllers, the role of compiler for DRAM power savings is limited. Our scheduling methods appear to be complementary to OS-based approaches, which operate at a much coarser granularity. For example, a recent OS-based method by Huang et al. [13] is similar to our command reordering approach, but it reshapes memory traffic at the page granularity.

Various throttling approaches have been proposed, including dynamic voltage scaling, dynamic frequency scaling, and decode throttling. Brooks and Martonosi [5] discuss these throttling methods in the context of CPU power management. Our throttling approach is similar to decode throttling in the sense that it reduces the flow of commands, but unlike the previous studies, we focus on DRAM power management.

Felter et al. [12] were the first to present a throttling approach for DRAM power management (and the first to propose the idea of Power Shifting), and we take their solution as the baseline for comparison. Our study differs from their

work in two ways: (1) we develop a much more accurate method of estimating DRAM power, significantly reducing performance degradation; and (2) we describe how to implement our method in the memory controller, whereas they leave implementation details as future work.

More recently, Diniz et al. [10] present a set of throttling techniques that provide extremely low performance degradation. Their key is to compute, for each memory command that is issued, a complete, fine-grained power estimate for every DRAM structure. Our work shows that a simpler, lower-cost solution can also be quite effective.

Recent work by Li et al. [25] present a power-down mechanism that solves the dual of our problem, guaranteeing that performance degradation falls within some specified limit.

Recent studies have shown the importance of addressing DRAM power consumption in large server systems [24, 3]. In contrast to most prior work, we introduce techniques for server-class memory controllers with mechanisms suitable for server-class memory topologies.

3 Our Solution

This section describes our approach to memory controller design, which makes the memory controller both power-aware and performance-aware. To provide context, we first briefly describe the Power5+'s memory controller. We then present our additions to current memory controllers in two subsections. First, we describe a power-down mechanism to schedule power-down/up signals; and we present an augmented form of adaptive history-based schedulers [16, 17, 14, 18, 15] that includes power criteria. Second, we introduce an adaptive throttling mechanism that can arbitrarily reduce DRAM power consumption.

The Power5+ Memory Controller. As shown in Figure 1, the Power5+ memory controller sits between the L2/L3 caches and DRAM. As memory commands enter the memory controller, they are placed in the reorder queues. On each cycle, the scheduler selects from the reorder queues a command, which is then sent to the CAQ, which in turn transmits commands to DRAM in FIFO order. The Power5+ memory controller uses a command bus to transmit memory commands to DRAM. Every command on this bus has a command type and an address. The DRAM is organized as 4 *ranks*, where each rank is an organizational unit consisting of 4 *banks*.

3.1 Effective Use of Low-Power Modes

With multiple ranks in current DRAMs, it is possible that at any given instant, some fraction of devices is idle. While DRAM power consumption is significantly lower

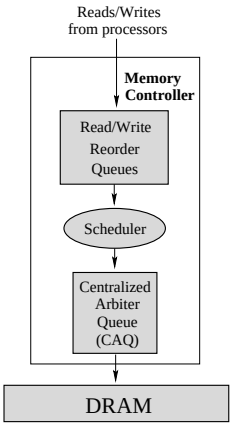

Figure 1. The IBM Power5+ Memory System.

when idle, the low power modes reduce power consumption by another order of magnitude [31]. To effectively use low power modes without adversely affecting performance, we present two additions to current memory controllers: a power-down strategy for generating rank power-down/up commands and an augmented form of adaptive history-based schedulers that includes a power reduction criterion.

The non-optimal use of power-down/up commands can limit performance in three ways. First, power-down/up commands consume command bus bandwidth between the memory controller and DRAM. Second, there may be unnecessary switches between low and high power modes, which waste two DRAM cycles for each switch. Third, in most modern DRAM chips, when a rank enters a low power mode, it has to remain in that mode for a certain number of cycles. Thus, powering down a rank prematurely can increase the latency for memory commands that are waiting for the powered-down rank.

3.1.1 Queue-Aware Power-Down Mechanism

We now describe a new technique for powering down/up ranks of DRAM; the basic idea is to be aware of the commands that are resident in the memory controller. Our queue-aware power-down mechanism generates commands to put idle DRAM ranks into low power mode. We introduce a new type of memory command, in which the ranks to be powered down/up are encoded in the address bits of the command. In the power-down mechanism, we maintain two hardware components for each rank: a status bit and a counter. The status bit is set to 1 when the rank is in the low power mode. The counter maintains the number of cycles remaining until the rank becomes idle. Each time a Read or a Write is sent to any bank of a rank, the rank's counter is initialized to the maximum of the current value

and the latency of the new command; otherwise the counter is decremented by one on every cycle.

We now present a protocol to decide when to send a power-down command to DRAM. On every cycle, the power-down mechanism checks the rank counters, rank status bits, and commands waiting in the CAQ. A power-down command is generated for the ranks that meet all of the following conditions. (1) The rank counter is zero, which indicates that the rank is idle. (2) The rank status bit is zero, because otherwise the rank is already in low power mode. (3) The command at the front of the CAQ cannot be issued in this cycle, which implies that regular commands have priority over power-down commands. (4) There is no command in the CAQ with the same rank number; this condition avoids powering down a rank if a Read or Write to that rank is imminent. The fourth condition can be extended to include the reorder queues as well, but we don't evaluate that option in this paper.

To generate power-up commands, the mechanism keeps track of the commands entering the CAQ. Whenever a new Read or a Write command enters the CAQ from the reorder queues, a power-up command is generated for the appropriate rank, the rank status bit is set to zero, and the rank counter is initialized.

3.1.2 Power-Aware Memory Scheduler

In this section, we describe how to modify an Adaptive History-Based (AHB) scheduler [16] to make it power-aware (PA-AHB).

An intelligent memory scheduler would seem to be a natural partner with the low power modes, but the scheduling goals of low power and good performance are at odds. For good performance, the scheduler typically selects commands that avoid hardware conflicts, essentially spreading the commands across many physical memory devices. On the ohter hand, to reduce power consumption, the scheduler attempts to cluster commands to a subset of the physical devices, allowing one or more of them to be put into low power mode.

An adaptive history-based scheduler uses the history of recently scheduled memory commands when selecting the next memory command. In particular, scheduling goals are encoded in finite state machines (FSM). Previously, two scheduling goals were used to improve performance: (1) minimize the latency of the scheduled commands, and (2) match some desired balance of Reads and Writes. We augment the AHB scheduler by adding power savings as a third goal.

To satisfy the power savings goal of the scheduler, we create a new FSM that groups same-rank commands, in the memory queue as close as possible, so that the total number of rank power-down operations is reduced. In the new state machine, we define the priorities for each possible command in the reorder queues as follows: The set of commands destined for the same rank as the last command sent to the memory queue has the highest priority, the set of commands to the same rank as the second from the last command has second priority, and so on. Since there may be more than one command in each of these sets, our approach breaks ties using performance as the second criterion.

Because both performance and power goals are important, we probabilistically combine the new FSM with the finite state machines of the original AHB, giving each of the three FSM's equal weighting. (We find that the behavior of our solution is not very sensitive to these weightings.) The result is a history-based scheduler that is optimized for both performance and power, but for one particular mix of Read/Writes. To accommodate a wide range of Read/Write mixes, we use adaptivity in the same sense as the original AHB scheduler, namely, our adaptive scheduler observes the recent command pattern and periodically chooses the most appropriate of the multiple history-based schedulers.

3.2 Adaptive Memory Throttling

Our throttling approach blocks commands inside the memory controller for all DRAM ranks for some fixed period of processor cycles, which we refer as the throttling delay. Commands that are blocked cannot proceed to the CAQ, so they accumulate in the reorder queues, reducing bandwidth between the memory controller and DRAM, and allowing ranks to remain in low power mode for longer periods of time.

To reduce DRAM power consumption to a target level, accurate estimation of the throttling delay is crucial, so we augment the memory controller with a *delay estimator*, which takes as input a power threshold and some information about the state of the DRAM, and produces as output an estimated throttling delay. The next section provides details about this delay estimator and its input parameters, but at a high level, the estimator uses a linear model of delay that is embedded in the memory controller. To calculate the throttling delay for a given power threshold, the estimator multiplies the relevant input parameters with corresponding model coefficients and sums the results. Because both the model parameters and the target power level can change over time, the estimator periodically calculates a new throttling delay; we refer to this period as an *epoch*. Our approach thus makes two assumptions: (1) the measured command flow in the current epoch is similar that of the next epoch; and (2) the epoch length is sufficiently long (we use one million processor cycles) that the overhead of delay calculation is negligible.

The coefficients of the model are computed by a software

tool, the *model builder*, which performs measurements on a set of workloads by applying linear regression on the measured data. Unlike the estimator, the model builder is active only at system installation or configuration time.

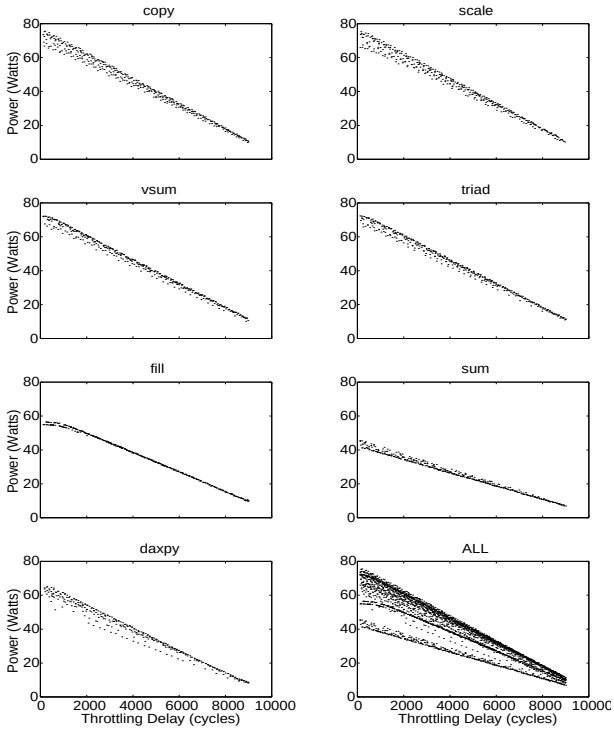

Figure 2. Relation between DRAM power consumption and the throttling delay for the Stream benchmarks (interval length is 10,000 cycles).

3.2.1 The Delay Estimator Model

This section explains our model for estimating the throttling delay. Our goal is to produce an accurate closed form equation that has a minimal number of input parameters. Aside from P, the target power threshold, the inputs to the model describe the state of the memory system, because the delay for a given command is affected by the states of the various physical substructures of the DRAM. Many possible input parameters could be considered, including the number of Reads issued in an epoch, the number of Writes issued in an epoch, the number of bank conflicts in an epoch, and many other measurements of the state of DRAM.

We conducted a number of experiments to determine the importance of various parameters. We cannot show all of our experiments here, but we show one example in Figure 2. Each graph in this figure is obtained by generating a random sampling of data alignments for the inputs to the Stream benchmarks, and each point in a graph represents the delay for such an experiment and the resulting power consumption. The lower right graph summarizes the results over the entire benchmark suite, and we see that the power varies widely with the relative alignment of data; in particular, for a target power consumption of 40 Watts, the corresponding delay varies between about 500 and 5,000 cycles, depending on the specific benchmark and the relative alignment of data. From Figure 2 we infer that the number of bank conflicts is an important input parameter to our model, because as the relative alignment of data changes, so does the number of bank conflicts. Because the number of bank conflicts is expensive to compute, our model instead uses a simpler, related value, namely, the number of cycles for which no command could be moved from the reorder queues to the CAQ due to a bank conflict.

From our experiments, we identify four important parameters: P, the power threshold; R, the number of Reads in the epoch; W, the number of Writes in the epoch, and B, the number of times in an epoch that bank conflicts prevent the movement of a command from the reorder queues.

The next subsection explains how we compute coefficients for these parameters, i.e., we explain the role of the model builder.

3.2.2 Determining Model Coefficients

The model builder determines the coefficients of the regression model for the estimator in two steps. First, for various values of throttling delay, it performs experiments for a set of workloads and collects data. We believe that the Stream benchmarks with multiple array offsets are good candidates for this purpose. Second, the model builder sets up a system of equations, where the known values are the measurement data and the unknowns are the model coefficients. We solve this system to determine the values of the model coefficients.

Linear regression models for throttling delay can be defined as:

$$y_i = \beta_0 + \beta_1 \Phi_{i1} + \beta_2 \Phi_{i2} + ... + \beta_p \Phi_{ip}, \qquad i = 1, 2, ..., n. \tag{1}$$

where n is the number of measurements, p is one less than the number of coefficients in the model, and the y_i's are the throttling delays in the experiments. The same equation can also be stated in matrix form as:

$$\mathbf{y} = \mathbf{\Phi}\beta \tag{2}$$

The elements of the $\mathbf{\Phi}$ matrix are known, and each column of the matrix represents one feature of the model. A sample matrix might be: the first column represents the measured DRAM power, the second column the number of

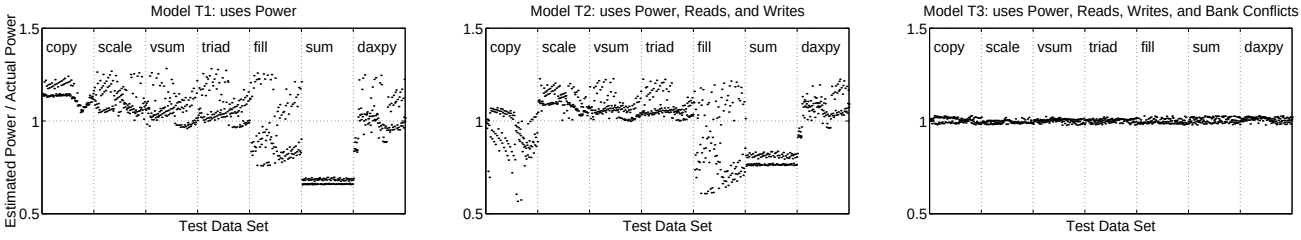

Figure 3. Estimated power vs. actual power for the three regression models. A y-axis value of 1 represents the ideal estimator. Each dot represents an element of the test data set.

Reads, the third column the number Writes, and the fourth column the number of processor cycles when no command could be moved from the reorder queues to the CAQ due to a bank conflict. To find the value of the β vector, i.e. the coefficients of the model, the model builder uses a least squares method, which is defined as:

$$\beta = \mathbf{\Phi}^{+}\mathbf{y} \tag{3}$$

where $\mathbf{\Phi}^{+}$ is the pseudo-inverse of $\mathbf{\Phi}$ [4].

To calculate the unknown coefficients of the regression models, we perform experiments using the Stream benchmarks. For each benchmark, we collect 200 data points by blocking commands in the memory controller with different throttling delays. Each data point represents one of 16 different offsets between data arrays and includes the number of Reads, number of Writes, DRAM power consumption, and the number of cycles for which no command could be sent to the CAQ from the reorder queues due to a bank conflict. We designate half of the measurements from each benchmark as the training data set and the other half as the test data set.

3.2.3 Evaluating Model Accuracy

Using the model builder, we created three models, which we now evaluate. Model $T1 = f(P)$ is the simplest possible model, which does not consider any information about the state of the DRAM; model $T2 = f(P, R, W)$ considers the number of Reads and Writes and is similar to the model proposed by Felter et al. [12]; and model $T3 = f(P, R, W, B)$ is our proposed model.

To fit the regression models to our measured data, we apply linear regression to the training data set, and we calculate the R^2 statistic [22] using the test data set. For the entire test data set of the Stream benchmarks, we obtain R^2 values of 0.1911, 0.1218, and 0.0032 for models $T1$, $T2$, and $T3$, respectively. As indicated by its low R^2 value, the $T3$ model achieves the best accuracy, and it is also the only model that satisfies the $<$0.01 criterion for the R^2 statistic.

Figure 3 shows how the estimated power produced by each of the three models compares with actual power. In these graphs, a perfect model would produce a horizontal line at 1. From the rightmost graph, we see that the $T3$ model comes close to this perfect model. By contrast, the graphs for the other two models show great variation from the actual power.

The models we have discussed are called first-order regression models, because the exponent of each Φ_j is one. Alternatively, we can define higher order models. Although higher order models may sometimes provide better fit for measured data, they might not generalize well. Thus, we have only considered first-order models.

4 Methodology

In this section, we describe our simulation methodology, our simulated system, and the benchmarks that we use in evaluating our techniques.

4.1 Simulation Methodology

To evaluate performance, we use a cycle-accurate simulator for the IBM Power5+, which has been verified to within 1% of the performance of the actual hardware. This simulator, one of several used by the Power5+ design team, is on the order of 1 million lines of high-level language code, and it uses execution traces to simulate both the processor and the memory system. To simulate our benchmarks, which have billions of dynamic instructions, we use uniform sampling, taking 50 uniformly chosen samples that each consist of 2 million instructions. This Power5+ simulator is integrated with Memsim [32], a DRAM simulator that jointly models power and performance of the main memory subsystem. In this simulation environment, Memsim models all the memory system activity, including refreshes, while synchronizing with the Power5+ simulator on every processor cycle.

Parameter	Value	Parameter	Value
Number of ports	2	tRRD: Row active to row active delay	7.5 ns
SMIs per port	4	tWTR: Write to Read command delay	10 ns
DIMMs per SMI	4	tRFC: Auto-refresh command period	105 ns
DIMM width	64	tREFI: Average periodic refresh interval	7.8 μs
Devices per DIMM	16	tXP: Exit precharge power-down to any non-Read command	7.5 ns
Device width	8	tCKE: Minimum high/low time	11.25 ns
DRAM burst length	4	IDD0: One bank active-precharge current	80 mA
Number of ranks	4	IDD2P: Precharge power-down current	7 mA
Number of banks	4	IDD2N: Precharge standby current	45 mA
Rows per bank	128	IDD3P: Active power-down current	30 mA
Columns per bank	1024	IDD3N: Active standby current	55 mA
tRP: Row precharge time	15 ns	IDD4R: Burst Read current	145 mA
tRCD: RAS to CAS delay	15 ns	IDD4W: Burst Write current	140 mA
tRAS: Row active time	45 ns	IDD5: Burst refresh current	170 mA
CL: CAS latency	15 ns	IDD6: Self refresh current	7 mA
tRC: Row cycle time	60 ns	IOL: Output minimum sink DC current	13.4 mA
tWR: Write recovery time	15 ns	maxVdd: Maximum supply voltage	1.9 V
AL: CAS additive latency	0	nomVdd: Nominal supply voltage	1.8 V

Table 1. Memory system details.

4.2 Simulated System

We evaluate our techniques in the context of the IBM Power5+ [20]. The Power5+ has one memory controller—with an AHB memory scheduler—and two processors per chip, where each processor supports two SMT threads. The Power5+ has also a hardware data prefetching unit.

We simulate a Power5+ running at 2.132GHz. Our simulator models all three levels of the cache. The L1D cache is 32KB with 4-way set associativity, and the L1I cache is 64KB with 2-way set associativity. The L2 cache is a 3×640KB shared cache, with 10-way set associativity and a line size of 128B. The off-chip L3 cache is 36MB.

The Power5+ memory controller has two ports to memory. Each port is connected to memory via Synchronous Memory Interface (SMI) chips [34]. We evaluate our techniques on a configuration with 4 SMIs and DDR2 SDRAM running at 533MHz, a common configuration for high-end Power5+ systems. In Table 1 we present some of the significant memory system parameters that we use in our study. We use the precharge power-down mode, which has the lowest possible power consumption for the DRAM type that we evaluate. More details about the DRAM chips that we model can be found in the datasheet from Micron [31].

4.3 Benchmarks

Our evaluation uses the Stream [29], NAS [1], and SPEC2006fp benchmarks suites, along with a set of internal IBM commercial benchmarks. We combine the original Stream and Stream2 benchmarks to create an extended set of the Stream benchmarks that consists of seven vector kernels. The NAS benchmarks are a group of eight programs derived from computational fluid dynamics applications; we use serialized versions of the class B benchmarks. The commercial benchmarks consist of four server applications, namely, *tpcc*, *cpw2*, *trade2*, and *sap*. Tpcc is an online transaction processing workload; cpw2 simulates the database server of an online transaction processing environment; trade2 is an end-to-end web application that models an online brokerage; and sap is a database workload. We exclude SPEC2006int, because they have low memory pressure, so even a trivial power-down approach will suffice for those benchmarks.

5 Evaluation

In this section, we empirically evaluate our three techniques. Our baseline for evaluation is a Power5+ system that uses an AHB memory scheduler. We evaluate our techniques in terms of performance, power, and energy efficiency.

5.1 Effects of Queue-Aware Power-Down Mechanism

We first evaluate the queue-aware power-down mechanism by comparing it against a simpler power-down policy, *greedy power-down*, which is similar to queue-aware power-down except it omits the fourth condition given in Section 3.1.1, so it greedily powers down a rank as soon as it can. We find that greedy power-down improves energy efficiency, on average, by 4.3%, 12.6%, 41.2%, and 32.7% for the Stream, NAS, SPEC2006fp, and commercial benchmarks, respectively. The queue-aware power-down policy improves energy efficiency over the baseline by 11.6%, 18.1%, 43.4%, and 34.2% for the four benchmark sets, respectively.

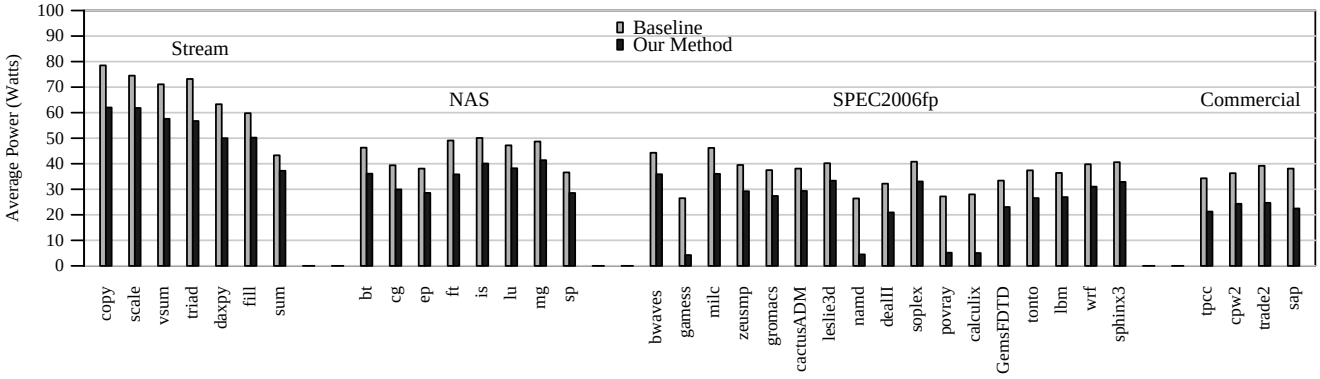

Figure 4. Power effects of the Queue-Aware Power-Down mechanism and the Power-Aware AHB scheduler over the baseline system.

Benchmark	Power Consumption (baseline) (Watts)	Power Consumption (our method) (Watts)	Power Reduction (%)	Performance Degradation (%)	Energy Efficiency Improvement (%)
Stream	65.2	53.0	18.7	2.7	18.2
NAS	44.1	34.5	21.9	1.2	21.7
SPEC2006fp	35.6	19.1	46.4	0.8	46.1
Commercial	36.9	23.1	37.3	0.6	37.1

Table 2. Power, performance, and energy efficiency comparison of our approach to the baseline system.

5.2 Effects of Power-Aware Scheduling

Figure 4 shows that there is a benefit to combining the queue-aware power-down mechanism and the power-aware AHB scheduler. Both techniques attempt to reduce the frequency of transitions into and out of low power mode, so we conclude that neither approach subsumes the other.

Table 2 summarizes the results, and also shows the performance and energy efficiency effects of our techniques. Not shown in this table is the benefit of the Power-Aware scheduler with the greedy power-down mechanism, which yields energy efficiency improvements over the baseline of 13.7%, 18.6%, 43.1%, and 33.8% for the Stream, NAS, SPEC2006fp, and commercial benchmarks, respectively.

5.3 Effects of the Memory Scheduler

Our baseline system uses the AHB scheduler, but many other systems use simpler schedulers. In this section we investigate the interaction of the power-down mechanism with other type of schedulers [16, 17], namely the in-order, memoryless, and AHB scheduler. The *in-order* scheduler implements the simple FIFO policy used by most general purpose memory controllers today. The *memoryless* scheduler implements the ideas proposed by Rixner et al. [33]. Basically, the memoryless scheduler avoids long bank conflict delays by selecting commands from the reorder queues that do not conflict with commands in DRAM.

To explore the effects of the memory schedulers, we first present detailed results for the *daxpy* kernel; we then provide results for the entire Stream benchmark set. Throughout Section 5.3 the power-down policy is the queue-aware power-down policy.

Daxpy Results. Figure 5 shows how the three previously studied memory schedulers compare in terms of power (left graph) and performance (right graph). We find that the more sophisticated schedulers provide better performance at the expense of higher average power consumption.

Figure 6 compares the power and performance of the three schedulers when combined with the queue-aware power-down policy. These results are all normalized with respect to the in-order scheduler with no power-down policy. We find that queue-aware power-down policy lowers the power consumption of the in-order, memoryless, and AHB schedulers by 19.7%, 7.3%, and 10.2%, respectively. Comparing the right graphs of Figures 5 and 6, we see that the power-down policy degrades performance by a small amount. Execution time increases by 1.2% for the in-order scheduler, by 1.4% for the memoryless scheduler, and by

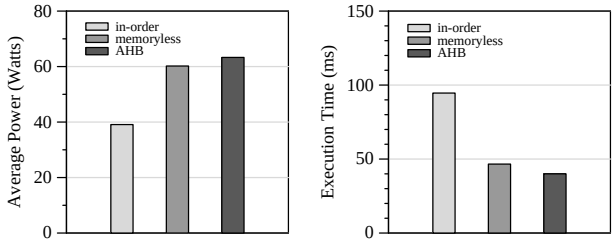

Figure 5. Left: Power consumption of In-order, Memoryless, and Adaptive History-Based schedulers (without the power-down mechanism). Right: Performance of the three schedulers.

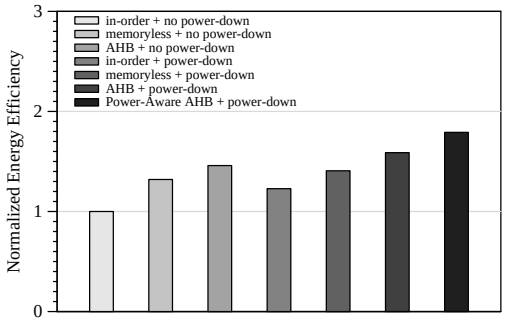

Figure 7. Energy efficiency comparison of the schedulers for the daxpy benchmark.

1.9% for the AHB scheduler. Figure 6 also shows the results for the Power-Aware AHB scheduler, which when compared with the AHB scheduler that uses queue-aware power-down, reduces power by 9.3% and degrades performance by 0.8%.

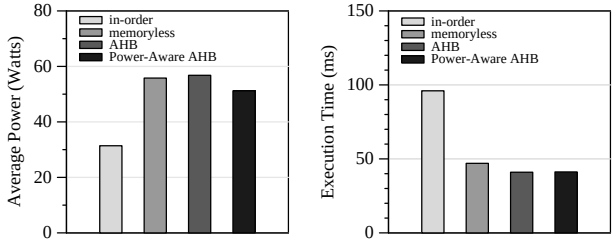

Figure 6. Left: Power consumption of the In-order, Memoryless, and Adaptive History-Based schedulers with the power-down mechanism. Right: Performance of the schedulers with the power-down mechanism.

Finally, Figure 7 shows that our two techniques—the PA-AHB scheduler and the queue-aware power-down policy—combine to provide the best energy efficiency. In particular, we find that such a system is 8.6% more energy efficient than a system that uses the AHB scheduler in combination with queue-aware power-down.

Stream Results. Figure 8 compares the four schedulers with and without queue-aware power-down for the Stream benchmarks. We find that the Power-Aware AHB scheduler provides the best energy efficiency for all benchmarks. In particular, when the memory controller uses queue-aware power-down, a system with the PA-AHB scheduler is, on average, 14.7%, 19.1%, and 11.6% more efficient than the systems with the in-order, the memoryless, and the AHB scheduler, respectively.

5.4 Evaluation of Adaptive Memory Throttling

In this section, we evaluate the adaptive memory throttling approach presented in Section 3.2. We compare the baseline model and our proposed model described in Section 3.2.3 against a perfect estimator that can calculate the exact throttling delay for any target power level.

Table 3 summarizes our experimental results for our four benchmark suites. For the Stream results, we use the test data described in Section 3.2. For the NAS, SPEC2006fp, and commercial benchmarks, we perform experiments for three different power target levels: 75%, 50%, and 25% of the original. As expected, the more accurate $T3$ model leads to significantly better energy efficiency.

Figure 9 illustrates the performance effects of the two regression models on the Stream benchmarks. We see that due to inaccuracies in estimating the throttling delay, model $T2$ degrades performance by up to 115.9% beyond the ideal throttling, while model $T3$ degrades performance by up to 5.8% beyond the ideal.

5.5 Multithreading Results

We have repeated the above experiments on a system that uses two SMT threads on the same processor, but for space reasons we omit the graphs. For these experiments, we use the same trace file for both threads, but we start the second thread 1 million instructions after the first thread.

We find that the queue-aware power-down and power-aware AHB techniques improve energy efficiency of the SMT runs, on average, by 17.4%, 19.6%, 34.7%, and 28.1% for the Stream, NAS, SPEC2006fp, and commercial benchmarks, respectively. For the benchmarks that already have high utilization of memory bandwidth, e.g., the Stream, the

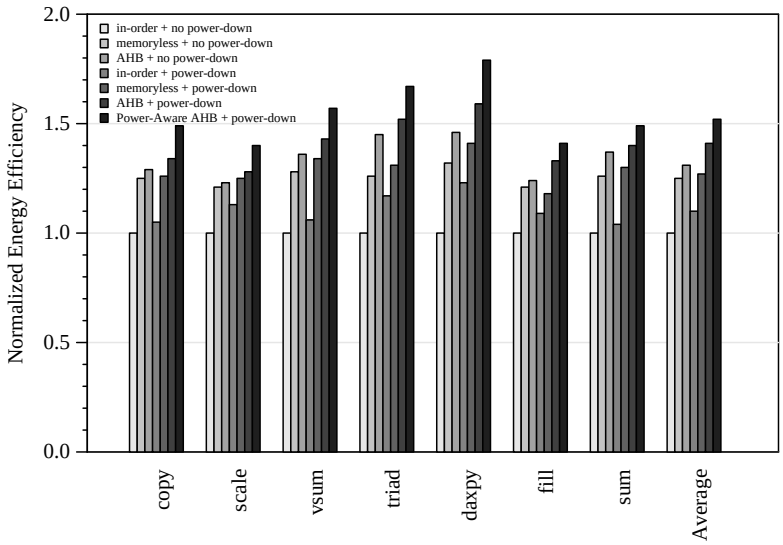

Figure 8. Energy efficiency comparison of the schedulers for the Stream benchmarks.

Benchmark	Model	Target Power {75%, 50%, 25%} (Watts)	Estimated Power {75%, 50%, 25%} (Watts)	Power Estimation Error (%)	Performance Degradation (%)	Energy Efficiency Degradation (%)
Stream	T2 (baseline)	{39.8, 26.5, 13.3}	{29.3, 21.7, 9.7}	23.7	29.6	46.3
	T3 (our model)	{39.8, 26.5, 13.3}	{39.3, 26.1, 13.1}	1.2	4.4	5.6
NAS	T2 (baseline)	{25.9, 17.3, 8.6}	{20.6, 12.8, 7.1}	21.4	20.7	37.7
	T3 (our model)	{25.9, 17.3, 8.6}	{25.0, 16.2, 8.4}	4.2	0.9	5.1
SPEC2006fp	T2 (baseline)	{14.3, 9.6, 4.8}	{11.4, 7.7, 4.1}	17.8	18.9	33.4
	T3 (our model)	{14.3, 9.6, 4.8}	{13.9, 9.1, 4.7}	3.1	1.3	4.3
Commercial	T2 (baseline)	{17.3, 11.6, 5.8}	{13.7, 9.6, 4.6}	19.3	16.4	32.5
	T3 (our model)	{17.3, 11.6, 5.8}	{17.1, 11.3, 5.7}	1.9	2.7	4.5

Table 3. Comparison of the throttling delay estimators (averages): Our model achieves significantly more accurate results than the baseline.

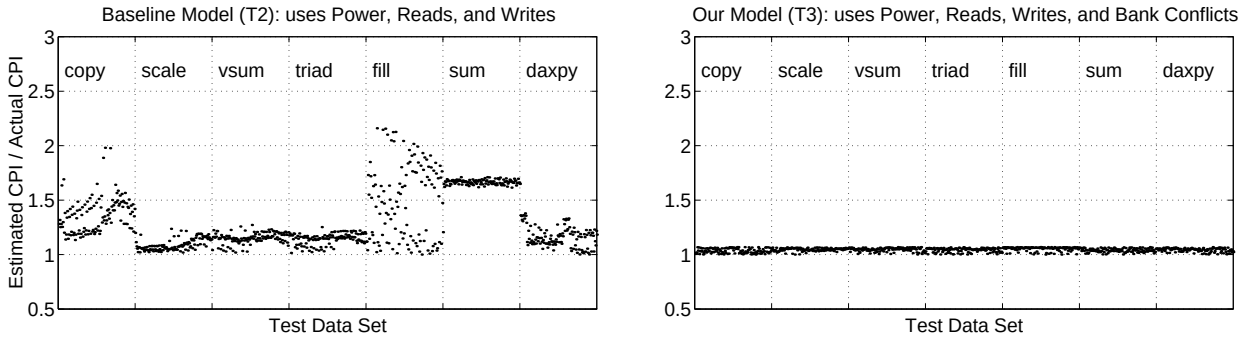

Figure 9. Performance effects of the regression models. A Y-axis value of 1 represents the ideal estimator, so model T3 is much more accruate than T2.

improvements from our techniques are about the same as the single-threaded results. As the memory bandwidth utilization of the single thread decreases, the power saving opportunities for the SMT runs decrease as well. However, as previous studies [18] confirm, in modern memory systems, constraints in the memory controller make it almost impossible to keep all the ranks busy all the time. Thus, we expect our techniques to be effective even beyond two threads.

The improvements relative to the baseline from our adaptive throttling technique are about the same as for the single-threaded results. We find that our approach provides performance that is within 4.9%, 1.4%, 1.9%, and 3.8% of a perfect oracular model, for our four benchmark suites, respectively. By contrast, the baseline model loses 32.7%, 23.1%, 19.2%, and 18.3% of performance for the same benchmark suites.

5.6 Hardware and Power Costs

The current Power5+ memory controller occupies about 1.6% of the entire chip area. The power-down mechanism that we propose requires an 8-bit counter and a status bit per rank. We conservatively assume that the Power-Aware AHB scheduler will double the size of the already existing AHB scheduler of the Power5+. For the throttling scheme, the circuitry to detect bank conflicts and to count Reads/Writes in an epoch already exists, but we need an additional 16-bit counter to keep track of the bank conflict information, and we need space to store four 64-bit model coefficients. Using an implementation of the Power5+ to provide detailed estimates of transistor counts, we estimate that our power management extensions increase the area of the memory controller by about 2.5%, which increases the overall chip's transistor count by about 0.04%.

Of course, the implementation of the power management mechanisms itself also consumes power. We do not have benchmark-specific analyses of this power usage, but we know that the current memory controller on the Power5+ consumes about 1% of the chip's power. With an area-based estimation, we find that our power management mechanisms increase the chip's total power by about 0.025%. As a reference, the Power5+ chip typically consumes roughly 2-4 times the power as the DRAM chips for our workloads.

6 Conclusions

We have presented a three-pronged approach to managing DRAM power and energy. Our first two prongs are conceptually simple but combine to reduce power with only minimal reduction in performance. Our third prong supports the notion of throttling, where the key problem is to accurately estimate the necessary throttling delay to stay below a given power threshold.

Our main conceptual contribution lies in our method of estimating throttling delay. Previous work has shown that good throttling decisions can be made by maintaining detailed information about the state of all of DRAM's physical sub-structures [10]. We instead show how to produce good estimates at much lower cost. The key is to do as much of the work statically as possible. Thus, we first conducted experiments to identify three key parameters for describing the state of DRAM. We then used an offline model builder to determine specific coefficients for each of these parameters. This model builder is trained once at system configuration time. The resulting hardware estimator maintains only a small amount of dynamic state and a small amount of logic to compute the throttling delay. We believe that similar models can be used by other microarchitectural structures that need to estimate the effects of complex behavior, including instruction throttling for the CPU.

Acknowledgments. We thank Alper Buyuktosunoglu for his helpful expertise on power consumption. We thank the entire IBM Power5 team, in particular, Cheryl Chunco, Steve Dodson, Gary Morrison, Stephen J. Powell, and Karthick Rajamani. This work was supported by NSF grant ACI-0313263 and by an IBM Faculty Partnership Award.

References

[1] D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagum, R. Fatoohi, S. Fineberg, P. Frederickson, T. Lasinski, R. Schreiber, H. Simon, V. Venkatakrishnan, and S. Weeratunga. The NAS parallel benchmarks (94). Technical Report RNR-94-007, NASA Ames Research Center, March 1994.

[2] L. Benini, A. Macii, and M. Poncino. Energy-aware design of embedded memories: A survey of technologies, architectures, and optimization techniques. *ACM Transactions on Embedded Computing Systems*, 2(1):5–32, 2003.

[3] R. Bianchini and R. Rajamony. Power and energy management for server systems. Technical Report DCS-TR-528, Rutgers University, June 2003.

[4] C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[5] D. Brooks and M. Martonosi. Dynamic thermal management for high-performance microprocessors. In *Proceedings of the Seventh International Symposium on High-Performance Computer Architecture*, pages 171–184, 2001.

[6] J. Corbal, R. Espasa, and M. Valero. Command vector memory systems: High performance at low cost. In *Proceedings of the 1998 International Conference on Parallel Architectures and Compilation Techniques*, pages 68–79, 1998.

[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*, pages 138–147, 2000.

[8] V. Delaluz, M. Kandemir, N. Vijaykrishnan, A. Sivasubramaniam, and M. Irwin. DRAM energy management using software and hardware directed power mode control. In *Proceedings of the Seventh International Symposium on High-Performance Computer Architecture*, pages 159–170, 2001.

[9] V. Delaluz, A. Sivasubramaniam, M. Kandemir, N. Vijaykrishnan, and M. Irwin. Scheduler-based DRAM energy management. In *Proceedings of the 39th Conference on Design Automation*, pages 697–702, 2002.

[10] B. Diniz, D. Guedes, J. Wagner Meira, and R. Bianchini. Limiting the power consumption of main memory. In *Proceedings of the 34th Annual International Symposium on Computer Architecture*, pages 290–301, 2007.

[11] 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*, pages 129–134, 2001.

[12] W. Felter, K. Rajamani, T. Keller, and C. Rusu. A performance-conserving approach for reducing peak power consumption in server systems. In *Proceedings of the 19th Annual International Conference on Supercomputing*, pages 293–302, 2005.

[13] 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*, pages 393–398, 2005.

[14] I. Hur. *Enhancing Memory Controllers to Improve DRAM Power and Performance*. PhD thesis, The University of Texas at Austin, 2006.

[15] I. Hur. Method and system for creating and dynamically selecting an arbiter design in a data processing system. *U.S. Patent 7,287,111, assigned to International Business Machines Corporation*, 2007.

[16] I. Hur and C. Lin. Adaptive history-based memory schedulers. In *Proceedings of the 37th Annual ACM/IEEE International Symposium on Microarchitecture*, pages 343–354, 2004.

[17] I. Hur and C. Lin. Adaptive history-based memory schedulers for modern processors. *IEEE Micro*, 26(1):22–29, 2006.

[18] I. Hur and C. Lin. Memory scheduling for modern microprocessors. *ACM Transactions on Computer Systems*, 25(4), December 2007.

[19] S. Irani, S. Shukla, and R. Gupta. Online strategies for dynamic power management in systems with multiple power-saving states. *ACM Transactions on Embedded Computing Systems*, 2(3):325–346, 2003.

[20] R. Kalla, B. Sinharoy, and J. Tendler. IBM Power5 chip: A dual-core multithreaded processor. *IEEE Micro*, 24(2):40–47, 2004.

[21] M. Kandemir. Impact of data transformations on memory bank locality. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 506–511, 2004.

[22] T. O. Kvalseth. Cautionary note about R2. *The American Statistician*, 39(4):279–285, November 1985.

[23] 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*, pages 105–116, 2000.

[24] C. Lefurgy, K. Rajamani, F. L. Rawson III, W. Felter, M. Kistler, and T. W. Keller. Energy management for commercial servers. *IEEE Computer*, 36(12):39–48, 2003.

[25] X. Li, Z. Li, F. David, P. Zhou, Y. Zhou, S. Adve, and S. Kumar. Performance directed energy management for main memory and disks. In *Proceedings of the Eleventh International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 271–283, 2004.

[26] Y.-H. Lu, L. Benini, and G. D. Micheli. Operating-system directed power reduction. In *Proceedings of the 2000 International Symposium on Low-Power Electronics and Design*, pages 37–42, 2000.

[27] C.-G. Lyuh and T. Kim. Memory access scheduling and binding considering energy minimization in multi-bank memory systems. In *Proceedings of the 41st Annual Conference on Design Automation*, pages 81–86, 2004.

[28] B. Mathew, S. McKee, J. Carter, and A. Davis. Design of a parallel vector access unit for sdram memory systems. In *Proceedings of the Sixth International Symposium on High-Performance Computer Architecture*, pages 39–48, 2000.

[29] J. D. McCalpin. Memory bandwidth and machine balance in current high performance computers. *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter*, December 1995.

[30] S. A. McKee. Hardware support for dynamic access ordering: Performance of some design options. Technical Report CS-93-08, University of Virginia, 1993.

[31] Micron. http://download.micron.com/pdf/datasheets/dram/ddr2/512MbDDR2.pdf, 2004.

[32] K. Rajamani. Memsim user's guide, IBM research report RC23431. 2004.

[33] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens. Memory access scheduling. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, pages 128–138, 2000.

[34] J. M. Tendler, J. S. Dodson, J. S. Fields Jr., H. Lee, and B. Sinharoy. Power4 system microarchitecture. *IBM Journal of Research and Development*, 46(1):5–26, 2002.

[35] Z. Wang and X. S. Hu. Power aware variable partitioning and instruction scheduling for multiple memory banks. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 312–317, 2004.

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