

Memory Power Management via Dynamic Voltage/Frequency Scaling

Howard David[†], Chris Fallin[§], Eugene Gorbatov[†], Ulf R. Hanebutte[†], Onur Mutlu[§]

[†]Intel Corporation
{howard.david,eugene.gorbatov,
ulf.r.hanebutte}@intel.com

[§]Carnegie Mellon University
{cfallin,onur}@cmu.edu

ABSTRACT

Energy efficiency and energy-proportional computing have become a central focus in enterprise server architecture. As thermal and electrical constraints limit system power, and datacenter operators become more conscious of energy costs, energy efficiency becomes important across the whole system. There are many proposals to scale energy at the datacenter and server level. However, one significant component of server power, the memory system, remains largely unaddressed. We propose memory dynamic voltage/frequency scaling (DVFS) to address this problem, and evaluate a simple algorithm in a real system.

As we show, in a typical server platform, memory consumes 19% of system power on average while running SPEC CPU2006 workloads. While increasing core counts demand more bandwidth and drive the memory frequency upward, many workloads require much less than peak bandwidth. These workloads suffer minimal performance impact when memory frequency is reduced. When frequency reduces, voltage can be reduced as well. We demonstrate a large opportunity for memory power reduction with a simple control algorithm that adjusts memory voltage and frequency based on memory bandwidth utilization.

We evaluate memory DVFS in a real system, emulating reduced memory frequency by altering timing registers and using an analytical model to compute power reduction. With an average of 0.17% slowdown, we show 10.4% average (20.5% max) memory power reduction, yielding 2.4% average (5.2% max) whole-system energy improvement.

Categories and Subject Descriptors

C.5.5 [Computer System Implementation]: Servers;
B.3.1 [Semiconductor Memories]: DRAM

General Terms

Measurement, Performance

1. INTRODUCTION

Power management has become a critical component of both mobile and enterprise systems in recent years. In the data center environment, thermal management and power budgeting have become significant concerns, especially as data centers become larger and pack servers more densely.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ICAC'11, June 14–18, 2011, Karlsruhe, Germany.

Copyright 2011 ACM 978-1-4503-0607-2/11/06 ...\$10.00.

The cost of operating a data center increasingly depends on its energy usage as much as its capital cost. As a result of these shifting constraints, server power consumption has become a significant focus of work, stimulating a variety of research for energy-efficient systems [4, 35].

Most proposed energy efficiency mechanisms autonomously observe system load or behavior and adjust the system's operating point periodically, moving on the performance / power curve to achieve the best efficiency. Several mechanisms operate by shutting down unused servers in clusters [10, 34], placing unused servers in a sleep state and batching work to minimize sleep transitions [10, 23], or scaling active server power proportionally to load [2]. In this paper, we focus on the last goal, known as server *energy proportionality*, which works by scaling subsystems within an individual server.

Prior work has focused mostly on *CPU* energy proportionality, adjusting frequency and voltage according to load (DVFS, or dynamic voltage/frequency scaling) [36]. While CPU DVFS, and idle powerdown states in various other system components, help to achieve scalability, we observe that the *memory system* often draws power disproportionate to its load. In modern systems, memory power can be a significant portion of system power: in our evaluations, 23% on average. Although modern systems make use of memory powerdown states during idle periods between memory requests, significant further opportunity exists. Current memory systems run at speeds that are balanced with respect to the peak computing power, optimized for memory-intensive workloads. However, for many other workloads, the performance impact of running at lower memory frequency is minimal. A slower memory frequency allows for lower voltage, furthering power reduction. Thus, we propose *memory DVFS* to dynamically adapt the memory system's operating point to current needs. We make the following contributions:

- We identify the *opportunity* for memory DVFS by presenting a detailed power model that quantifies frequency-dependent portions of memory power, showing that significant reductions are possible.
- We present a *control algorithm* based on observing memory bandwidth utilization and adjusting its frequency/voltage to minimize performance impact.
- We evaluate this on *real hardware*, obtaining performance results by emulating memory frequency with altered timing settings and modeling power reduction analytically.

The rest of this paper is organized as follows. In §2, we motivate the opportunity for memory DVFS. In §3, we discuss the components of DRAM power, and in §4 we present a model of memory power under voltage and frequency scaling. §5 discusses the impact of frequency scaling on application performance. We present our control algorithm in §6, and evaluate it in §7. We conclude with a discussion of related work and future directions for memory DVFS.

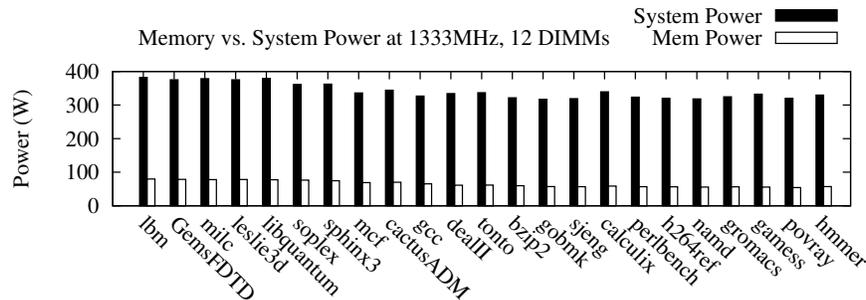


Figure 1: Memory system power in a 12-DIMM (48 GB), 2-socket system for SPEC CPU2006 benchmarks.

2. MOTIVATION

In order to motivate memory frequency/voltage scaling as a viable mechanism for energy efficiency, we must show (i) that there is significant opportunity for power reduction with this mechanism, and (ii) that common workloads tolerate the performance impact of memory frequency scaling with minimal degradation.

At a high level, two opposing forces on energy efficiency are at play when memory frequency is reduced. The efficiency depends both on power consumption and on runtime, as energy is the product of power and time. Power reduction alone will increase efficiency. However, performance also degrades at lower-power operating points, which increases runtime and thus energy. Thus, there is a tradeoff in reducing memory frequency/voltage. We will show later that statically scaling memory frequency has little performance effect on many lower-bandwidth workloads because frequency impacts only bus transfer latency, a portion of the full memory latency. In this section, we motivate that memory frequency scaling can have an effect on system power and thus energy (in the next section, we present an analytical model that incorporates voltage scaling).

2.1 Magnitude of Memory Power

We argue that (i) memory power is a significant portion of full-system power in modern systems, and (ii) the magnitude of power reduction attainable by memory frequency/voltage scaling is on par with the reduction due to CPU DVFS. First, Figure 1 shows average server memory system power in a 12-DIMM (48 GB), 2-socket server class system running 8 copies of each benchmark (see §7.1 for details) as computed by our power model (see §3). Total average system power for each run is shown for comparison. Memory power is 80W in the highest case, and 65W on average, against a 382W maximum (341W average) system power.

2.2 Potential Memory Power Reduction

Second, having seen that the *magnitude* of memory power is significant, we argue that the *potential reduction* is also significant. In order to show this, we perform a simple experiment on real Intel Nehalem hardware (§7.1). For a fixed workload (mcf from SPEC CPU2006 [32], a memory-intensive benchmark, with one copy running on each core), we measure AC power for three configurations. First, we run both the CPU and memory at full speed (2.93GHz and 1333MHz, respectively). Then, we force CPU DVFS to scale all cores down statically to 2.4GHz, with core voltage reduced to the appropriate level (controlled by hardware). Finally, we force memory speed to 800MHz, the lowest setting supported by our test system. Table 1 presents the results. Although CPU DVFS has a larger impact on system power (9.9% reduction), the impact of memory frequency scaling on system power is also significant (7.6%).

Because of limitations in existing hardware, this simple scaling experiment does not perform voltage scaling on memory, even though the CPU DVFS reduces core voltage. As we will discuss in more detail later, memory running at lower speed can operate at lower voltage as well, and this

Configuration	CPU	Mem	Avg. Power	Reduction
Baseline	2.93GHz	1333MHz	355W	
CPU scaling	2.4GHz	1333MHz	320W	9.9%
Mem scaling	2.93GHz	800MHz	338W	7.6%

Table 1: Simple (static) reduction for mcf: AC power reduction due to CPU and memory frequency scaling, in real Intel Nehalem hardware.

grants additional power reduction. In the remainder of this paper, we assume voltage as well as frequency scaling.

3. BACKGROUND: MEMORY POWER

In order to effectively improve energy efficiency by scaling memory frequency and voltage, we must first understand how this scaling affects memory system power (and thus system energy). In this section, we first briefly provide background on DRAM structure (§3.1). We then break down the components of DRAM power (§3.2) and discuss how frequency scaling impacts each component. With this understanding, we quantify power in terms of *operational power* due to memory access (§3.3.1) and *background power* that varies with memory sleep states and frequency (§3.3.2). We next discuss the ability of DRAM to tolerate voltage scaling, and its effects on power (§3.4). We address the effects that frequency/voltage scaling might have on time spent in power-down states (and the consequent trade-offs) in §3.5. Finally, we discuss some potential implementation issues for memory DVFS in §3.6. This understanding will allow us to build an analytical model in §4 in order to approximate power and energy reduction.

3.1 DRAM Structure Background

Figure 2 (a simplified version of Figure 1 in [26]) gives a general overview of the structure of a DDR3 DRAM device. A set of *devices* placed together on a DIMM comprises a *rank*. Within a rank, a number of *banks* consist of independent DRAM storage arrays in each device with associated decoders and sense amplifiers. These banks share I/O hardware (drivers and receivers) to interface with the DDR bus. Each bank is a matrix of storage cells, organized into *rows*. The *row buffer* can hold one active row (or page) at a time. An *activate* command brings a row into the buffer, after which read/write commands can access *columns* in the row. A *precharge* command returns the data to the storage array and prepares for the next activation. Detailed descriptions of memory operation can be found in datasheets, technical notes and papers [26, 24, 25, 27].

Various policies govern the way in which a memory controller uses these commands. Two common policies are *page-open* and *page-closed* policies. Page-open keeps the last-accessed row (page) active in the row buffer. This benefits performance when the next access is to the same row, because no activate is necessary. Page-closed performs a precharge as soon as the access is complete. Although this eliminates row hits, it reduces latency upon a row miss, because no precharge is necessary, only an activate. Note that we assume a page-closed policy in this paper. This is motivated by the observation that in multi-core systems,

Command	Energy (nJ) @ 1333MHz	Energy (nJ) @ 800MHz
Read (array)	18	18
Write (array)	20	20
Read I/O (1 DIMM/channel)	1	1.7
Write I/O (1 DIMM/channel)	4	7
I/O additional termination (2 DIMMs/channel)	12	20
Activate+Pre-charge (page open+close)	25	25
Average energy/read, page-closed policy, 2 DIMMs/channel	56	64.7
Average energy/write, page-closed policy, 2 DIMMs/channel	61	72

Table 2: Energy per operation for DRAM commands in a DDR3 DIMM, at 1333 and 800MHz. All energy components shown comprise the energy for a memory access, summarized in the last two rows. Note that energy is higher at lower frequencies due to the increased bus utilization (the bus is driven for more time).

Power-down State	Exit Latency to Read Command	Power @ 1333	Power @ 800	PLL, Out. Clk	IBT, ODT	DLL	Clk. Tree	Page Buf.	Decod.	Input Buf.	Self Refresh
SelfRefresh-Register Off	512 tCK + 6 μ s	0.56W	0.56W	0	0	0	0	0	0	0	1
SelfRefresh *	512 tCK	0.92W	0.77W	1	0	0	0	0	0	0	1
Precharge Slow Powerdown Register Off	tMRD + tXPDLL	1.35W	1.16W	1	0	0	0	0	0	0	0
Precharge Slow Powerdown	tXPDLL	1.60W	1.41W	1	1	1	0	0	0	0	0
Precharge Fast Powerdown *	tXP + tRCD	2.79W	2.33W	1	1	1	0	0	0	0	0
Active Powerdown	tXP	3.28W	2.71W	1	1	1	0	1	0	0	0
Precharge Standby *	tRCD	4.66W	3.87W	1	1	1	1	0	1	1	0
Active Standby	0	5.36W	4.36W	1	1	1	1	1	1	1	0

Table 3: Background Power: power states for a 4GB DDR3 DRx4 R-DIMM. Asterisks mark states our evaluation system supports. These values incorporate all device power that is not accounted for by per-operation energy above. Within the table, a 1 indicates that the given component is powered up.

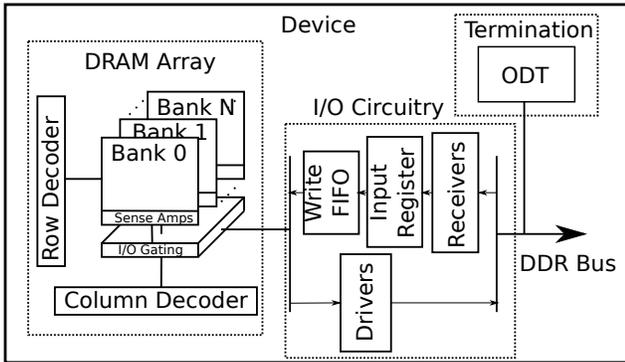


Figure 2: General overview of DRAM device structure, simplified from [26].

increasing page-conflict rates due to parallel access by many threads reduce the benefit of page-open policies [33, 29].

3.2 DRAM Power Breakdown

To provide background for this work, we will first describe the power usage of various components in a DRAM device.

DRAM Array Power: The DRAM array is the core of the memory. It is asynchronous in operation. Thus, array power is not dependent on memory frequency, only on access count (memory bandwidth). Low-power states also have no effect on DRAM array power: its power consumption is already low when idle ($\sim 10\text{mW}/\text{device}$ in x8 DDR3 DIMMs). The array draws a constant active-state power when a read, write or precharge command is active.

I/O Power: This component of device power consists of input buffers, read/write latches, DLL (delay-locked loop), data bus drivers, and control logic, and is consumed when the DRAM is idle (not powered down) or actively executing a command. I/O power is memory-frequency-dependent: it reduces with lower frequency. The portion of I/O power due to active command execution scales with bus utilization; this leads to an indirect effect when considering the energy

impact of frequency scaling, discussed below. I/O power is reduced in memory power-down states (described below).

Register Power: A registered DIMM consists of input/output registers on clock and command/address lines; register power consists of these components as well as associated logic and phase-locked loop (PLL). Like I/O power, register power is related to the bus interface and so is frequency-dependent. It also scales with low-power states. (Registers are not shown in Figure 2 for simplicity.)

Termination Power: Finally, modern DRAM devices include on-die termination (ODT) to properly terminate the bus during active operation. Termination power is dissipated in on-die resistive elements, and is adjusted to bus electrical characteristics, depending on DIMM count. With 2 DIMMs per channel, DDR3 termination power can reach 1.5-2.0W per DIMM. Termination power is not directly frequency-dependent; it depends only on bus utilization.

3.3 Operation and Background Power

In order to understand and model DRAM power quantitatively, we split it into two parts: *operation power* and *background power*. Operation power accounts for the effects of active memory operations in all four components, and is computed from the energy that each operation comprising a memory access consumes. Background power accounts for all other power in the device, and depends only on power-down state and operating frequency. Taken together, these two characterizations (given in Tables 2 and 3) describe the power usage of the memory system.

The energy and power figures given in Tables 2 and 3 are based on measurements of standby and active current (I_{DD} values) for multiple DIMMs. This process is statistically rigorous: measurements are taken of multiple vendors' DIMMs, and the values used here have certain statistical confidence based on the distributions. More details can be found in [18].

3.3.1 Operation Power

First, we must understand *operation power*, or the power required to execute a command when the device is in an

active state. We first present operation *energy*, split into several subcomponents that comprise a complete access and data transfer. The average power for a given access rate can then be determined from these components.

Table 2 shows the energy required for each operation that comprises a memory access at two operational frequencies, 1333MHz and 800MHz. A single memory access is 64 bytes (8 data transfer cycles). The first two rows correspond to actions that occur in the DRAM storage array itself. The next two table rows correspond to the energy required to drive the data onto the bus. The fifth table row accounts for additional I/O power required for bus termination when two DIMMs are present on the bus. Finally, the sixth table row shows the energy for one Activate and Pre-charge command pair required to bring a new row into the bank’s row buffer. Note that because we use a closed-page policy, operation power for a single access always includes an activate, read/write operation, and precharge. Given this, the last two rows compute the average energy per read or write operation as the sum of all components.

Note that I/O and termination energy are higher at lower frequencies due to increased bus utilization: for a given transfer size, I/O drivers and on-die termination are active for longer at lower frequencies. Higher bus utilization at lower frequency acts to increase energy more than the frequency decrease acts to reduce energy. It is important to note, however, that this affects only *operation power*. The net reduction in energy with frequency scaling comes from two sources: (i) the reduction in background power, and (ii) the voltage scaling that reduced frequency enables.

3.3.2 Background Power and Power-Down States

Background power accounts for all DRAM device power that is not operational power. In other words, it is the power that the device consumes regardless of which or how many commands it is executing, dependent only on its current state and frequency. Current DDR3 devices support a variety of power-down states to save power when inactive. In order to quantify DRAM power during a system’s execution, we must quantify the power usage in each state, and then calculate average background power weighted by the time spent in each state.

As with any power-management technique, DRAM power-down states present a tradeoff between aggressiveness and wake-up latency: turning off more of the device will save power, but wake-up will take longer. Power-down states are summarized in Table 3. Asterisks (*) mark states that our evaluation system supports.

DDR3 memory supports three power states with decreasing power consumption: standby, power-down and self-refresh. A CKE (Clock Enable) control signal is de-asserted in power-down and self-refresh states. When CKE is disabled, no commands can be issued to memory.

When DRAM is in standby, it consumes the highest amount of background power but can accept commands immediately without any latency penalty. There are two standby modes: active standby and precharge standby, with precharge mode consuming less power. Precharge mode requires all banks to have closed their row buffers (i.e., performed a precharge).

Three power-down states (Active, Precharge Fast, and Precharge Slow) consume less power than standby states at the cost of moderate exit latency. DRAM enters Active Powerdown state only if one of its banks is active when the CKE signal is de-asserted. Otherwise, DRAM enters one of the Precharge Powerdown states. In the fast mode, the DRAM DLL is on, while in the slow mode, it is stopped. The slow mode offers a trade-off between performance and power, consuming 40% less power than the fast mode, but at the cost of increased exit latency (2-4 clocks in DDR3-1333). Finally, it is possible to turn some register logic

off in Slow Powerdown state, as shown in Table 3. In this state, called Precharge Slow Powerdown – Register Off, input buffer termination (IBT) and output ODT are turned off, further reducing DIMM power.

The last DRAM low power state is self-refresh. In this state, CKE is de-asserted, the DRAM DLL is stopped, and DRAM devices are in self-refresh mode, consuming 40% less power than in Precharge Slow Powerdown state. However, this state has a significantly higher exit latency of 512 DRAM clocks. As with power-down modes, self-refresh has a special register state, called SelfRefresh – Register Off, in which register PLL is turned off, reducing power by another 40% at the expense of additional 6 μ s exit latency.

3.4 Voltage Scaling

Now that we have discussed the baseline power characteristics of DRAM, including those portions that are sensitive to frequency, we are interested in understanding how power can reduce when voltage scales. DRAM devices require a certain minimum supply voltage (V_{dd}) for stable operation at a given frequency. This voltage scales with frequency; at lower frequencies, a lower supply voltage is necessary. Note that DRAM power has components that scale by both V and V^2 . Specifically, the internal DRAM array is powered by an on-chip low-dropout linear regulator [19], and so its current is not dependent on external supply voltage. Its power thus scales linearly with V (since $P = IV$ and I remains constant). The I/O and register circuitry, however, draws current in proportion to supply voltage, and so its power scales with V^2 . This, in turn, allows for significant power reduction.

In order to understand the potential for voltage scaling in real DDR3 devices, we performed tests on 8 DIMMs in our evaluation system (detailed in §7) while manually controlling the memory voltage regulator output. The results are shown in Figure 3. At 1333, 1066, and 800MHz respectively, we observed average minimum stable voltages of 1.280V, 1.203V, and 1.173V respectively, and a maximum across the 8 DIMMs of 1.35V, 1.27V and 1.24V respectively. Thus, we conclude that the required supply voltage reduces with frequency. Later, in §4.1.2, we will model power reduction by conservatively assuming V_{dd} of 1.5V, 1.425V and 1.35V shown in this figure. Note that these voltages are well above the minimum stable voltages for the tested DIMMs.

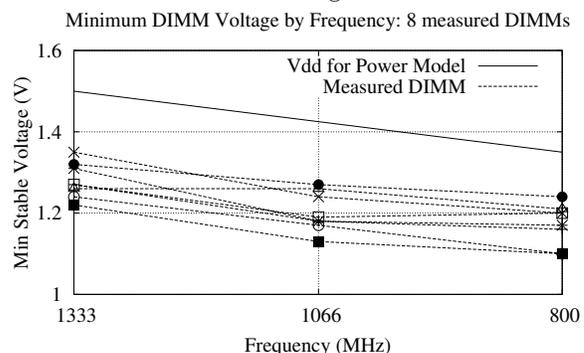


Figure 3: Minimum stable memory voltage as a function of frequency for 8 tested DDR3 DIMMs.

3.5 Indirect Effects of Frequency on Power-Down State Residency

Scaling down memory frequency can lead to another trade-off that we have not yet discussed. Because the memory bus runs more slowly, data transfers take longer. This could reduce the idle time between transfers, eliminating or reducing opportunity to place the DRAM into low-power sleep states. Frequency scaling could thus cause average memory power to increase in the worst case. In such cases, it would

be better to run the bus at a higher frequency so that the DRAM can be placed into a low-power sleep state sooner.

However, we observe that in many real workloads, there are idle periods between data transfers that are not long enough to enter a sleep state even at the highest memory frequency. In these cases, slowing the data transfers by reducing frequency will fill the time between transfers without reducing the time spent in sleep states.

To test how sleep state residency changes with frequency, we measured the change in time spent in sleep states (CKE-low residency) from 1333MHz to 800MHz in three representative workloads: `gcc`, `mcf` and `GemsFDTD` from SPEC CPU-2006. We also measured CPU core C0 (active state) residency to address concerns about reduced energy improvements due to increased runtime. The results show small deltas from 1333MHz to 800MHz: in `gcc`, time spent in a sleep state actually increases from 3.6% to 4.0%, and C0 residency (CPU utilization) increases from 93.6% to 94.0%. For `mcf`, sleep-state residency decreases from 25.2% to 23.8%, and C0 residency increases from 72.7% to 73.8%. For `GemsFDTD`, sleep-state residency decreases from 2.2% to 1.3%, and C0 increases from 95.9% to 97.8%. In all cases, these deltas are within the bounds of experimental error. Of course, the final confirmation that increased residency in higher-power states does not cancel power reduction at lower frequencies comes with our evaluation results in §7, which incorporate memory sleep-state information into the power model, and CPU utilization into the measured system power.

3.6 Potential Complexities of Scaling

Finally, we will briefly discuss the impact of DVFS on memory controller complexity, and discuss potential implementation difficulties. There are three general categories of problems that must be solved in order to make memory DVFS practical: ensuring that memory will operate correctly over the range of frequency/voltage points, actually implementing the run-time switching of frequency and voltage, and ensuring data stability across voltage changes.

Our DIMM voltage tests summarized in Figure 3 address the first point, validation: we observed the evaluation system to be stable over the range of voltages and frequencies that we propose. Initial implementations of memory DVFS could stay within voltage and frequency ranges that have been validated on existing DIMMs. In the future, DIMMs could also be validated at other operating points for potentially more power reduction.

Second, the procedure to switch frequency and voltage at run-time must be implemented in the memory controller and properly validated. This procedure can occur as follows: (1) freeze memory traffic, (2) put the memory into self-refresh, (3) stop the DIMM clock, (4) start ramping voltage toward the new setting, and re-lock the memory controller’s clock PLL at the new frequency, (5) re-start the DIMM clock when voltage stabilizes, (6) re-lock the register PLL on registered DIMMs (this takes $t_{STAB} = 6\mu s$), (7) take memory out of self-refresh, (8) re-lock the DLL in the DRAM (this takes 512 clocks, as shown in Table 3 for exit from Self-Refresh), and (9) re-start the memory traffic. We note that the only part of this procedure not possible on existing platforms is a configurable multiplier on the memory controller clock PLL. Even the memory voltage is already programmable: DRAM voltage regulators can be adjusted from BIOS settings. This procedure in total should take less than $20\mu s$ if the voltage regulator has a comparable slew rate to those used for CPU DVFS.

Finally, when switching voltage, data stability in the DRAM cells could be reduced. Because DRAM stores values as charges on capacitors, altering the DRAM core voltage will change the detection threshold when reading these capacitors, and might reduce margins. However, as noted above,

internal DRAM arrays in modern devices are powered by a linear voltage regulator [19]. Thus, V_{core} will be unaffected by changes in V_{dd} , and no changes to the DRAM core or the refresh strategy are required to accommodate voltage scaling.

4. ANALYTICAL MEMORY POWER MODEL

As described previously, we aim to evaluate memory DVFS in real hardware. However, current hardware cannot dynamically scale memory frequency and voltage (although we can emulate the performance effects of frequency scaling by changing timing parameters of the DRAM). Thus, we are limited to analytical modeling to compute power and energy reduction in this regard. In this section, we present our analytical model for DRAM power, based on the power breakdown and figures presented in §3. First, in §4.1, we will present a baseline model that assumes nominal frequency and voltage (i.e., the maximum speed, since our mechanism scales frequency and voltage down). Then, we will model the effects of frequency scaling (§4.1.1) and voltage scaling (§4.1.2).

4.1 Baseline Model

Our analytical model of memory power is based on *background power*, or power that is consumed regardless of memory operations performed and depends only on power-down state (Table 3), and *operation power*, according to the commands executed by the device (Table 2).

To model background power, we record time spent in each power-down state: active (t_{CKEH}), fast powerdown (t_{CKEL}) and self-refresh (t_{SR}), such that $t_{CKEH} + t_{CKEL} + t_{SR} = 1$. We weight power in each state by time spent in that state to arrive at average background power. Operation power is modeled by determining power per unit bandwidth, based on energy per operation, and multiplying it by bandwidth consumed. Together, background and operation power comprise memory power:

$$\begin{aligned} MemPower &= (P_{SR} * t_{SR} \\ &+ P_{CKEL} * t_{CKEL} + P_{CKEH} * t_{CKEH}) \\ &+ (P_{BW,r} * RBW + P_{BW,w} * WBW) \end{aligned}$$

where RBW and WBW indicate read and write bandwidth, respectively, to this memory channel. P_{SR} , P_{CKEL} , and P_{CKEH} come from Table 3; specifically $P_{SR} = 0.92W$, $P_{CKEL} = 2.79W$, $P_{CKEH} = 4.66W$. $P_{BW,r}$ and $P_{BW,w}$ come from per-operation energy in Table 2: we compute $P_{BW,r} = 0.939W/(GB/s)$, $P_{BW,w} = 1.023W/(GB/s)$ (for 2 DIMMs/channel). These two parameters are computed as follows: 1 GB of transfer is 16M operations, since each read or write moves 64 bytes. Then, multiplying 16M by the energy per operation gives energy per GB of transfer. Taking this value as a rate relates GB/s (bandwidth) to energy per time, or power.

4.1.1 Frequency Scaling

Next, we model the effects of frequency scaling. As discussed above, the *background* register and I/O power are frequency-dependent, and will decrease with frequency. However, the *operation* energy due to I/O and termination increases at lower frequencies, because the bus is active for a longer time for a given transfer. For a given bandwidth, this increases operation power. Said another way, bus utilization goes up at lower frequencies, increasing I/O and termination power.

We model the power reduction due to frequency scaling by taking both of these opposing effects into account. We model both effects linearly, according to the number of frequency steps N_f below nominal (maximum). For N_f frequency steps below nominal, DIMM power scales to:

$$\begin{aligned}
MemPower_f &= MemPower \\
&- (N_f * (P_{SR,save} * t_{SR} \\
&+ P_{CKEL,save} * t_{CKEL} \\
&+ P_{CKEH,save} * t_{CKEH})) \\
&+ (N_f * P_{IO,adder,r} * RBW + P_{IO,adder,w} * WBW)
\end{aligned}$$

For our modeled devices and system setup, we are interested in three operating points: 1333MHz (0 steps), 1066MHz (1 step), and 800MHz (2 steps). Our model derives from real measurements at 1333 and 800MHz; we assume linear scaling to derive the 1066MHz point, calculating a per-step delta that is half of the 800MHz-1333MHz power difference. We compute this background-power reduction per step from Table 3: $P_{SR,save} = 0.075W/step$ (from the ‘‘SelfRefresh’’ row), $P_{CKEL,save} = 0.23W/step$ (from the ‘‘Precharge Fast Powerdown’’ row), and $P_{CKEH,save} = 0.395W/step$ (from the ‘‘Precharge Standby’’ row). We then compute the ‘‘adder’’ factors from Table 2: $P_{IO,adder,r} = 0.073W/(GB/s)/step$ (from the second-to-last row), and $P_{IO,adder,w} = 0.092W/(GB/s)/step$ (from the last row), corresponding to increased operation energy at lower frequencies.

4.1.2 Voltage Scaling

As we described earlier, DRAM devices require lower voltage at lower frequencies. Thus, we assume that memory voltage will scale as the DRAM devices move between voltage steps, and model the power reduction here. Details of this mechanism are beyond the scope of this paper; however, we note that existing memory voltage regulators have software-configurable voltage settings, and slew rates should be fast enough that transition times are amortized over reasonable-length epochs (as with CPU DVFS).

In §3.4, we measured minimum stable operating voltage for 8 DIMMs at our three frequency operating points. From those figures, we add some margin and choose voltages that are commonly available on existing motherboards: 1.5V, 1.425V and 1.35V for 1333, 1066 and 800MHz respectively. As noted earlier, I/O circuitry scales power as V^2 , and the storage array power scales as V . With V^2 scaling, these voltage steps correspond to 10% reduction per step; with V scaling, 5%. From our background and operational power figures, we derive the percentage of power drawn by I/O circuitry (V^2 scaling). At 800MHz with two registered DIMMs per channel, the combination of I/O power and register power (estimated at $0.67W$ from a DDR register data sheet [17]) ranges from 25% (at 2GB/s bandwidth) to 29% of power (at both idle and maximum bandwidth); we assume that the remainder of power scales linearly with V . Total power reduction per voltage step is thus at least $25% * 10% + 75% * 5% = 6.25%$. We conservatively take a value of 6%. Thus, the power reduction due to memory voltage scaling is as follows, with $P_{vstep} = 0.06$:

$$MemPower_{f,v} = MemPower_f - MemPower_f * P_{vstep} * N_f$$

5. PERFORMANCE IMPACT OF FREQUENCY SCALING

We have quantified the power implications of memory frequency and voltage; in order to develop a dynamic control algorithm that adjusts memory frequency based on bandwidth demand, we now aim to understand how workload bandwidth demand varies across a spectrum (§5.1), how reduced memory frequency increases memory latency for different bandwidth demands (§5.2), and how this increased latency due to frequency scaling affects performance (§5.3). These insights will lead directly to a simple, intuitive control algorithm.

5.1 Memory Bandwidth Utilization

Figure 4 shows memory bandwidth for SPEC CPU2006 applications running on our evaluation system (see §7.1).

As is shown, memory bandwidth is highly variable, and depends on many factors: memory access rate, LLC residency, memory- and bank-level parallelism (MLP [13] and BLP [28]) and the ability to tolerate memory latency, for example. An application that is entirely resident in last-level cache will have zero memory bandwidth; at the other extreme, an application that exhibits perfect streaming behavior, accessing memory continually with no dependent loads, and that has enough outstanding memory accesses in parallel, should be able to maximize memory-system bandwidth.

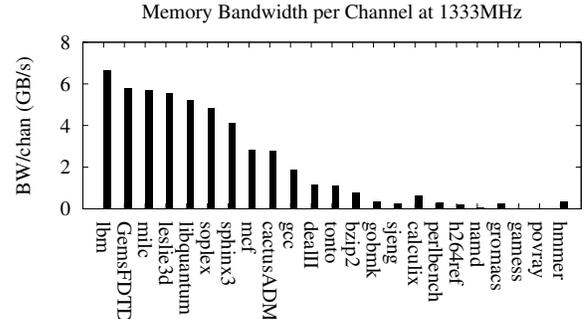


Figure 4: Memory system bandwidth utilization for SPEC CPU2006 applications.

5.2 Impact of Frequency on Memory Latency

It is important to understand the performance-energy tradeoff across the range of bandwidth. So far, we have introduced multiple memory-frequency operating points and discussed the power implications of frequency/voltage scaling. By evaluating the ability of each operating point to handle a given bandwidth, we can choose the best operating point for optimal efficiency.

We characterize memory latency as a function of memory frequency. We measured actual latency using a carefully-constructed microbenchmark that exhibits cache misses with dependent loads (i.e., MLP of 1) at an adjustable rate. The resulting bandwidth-latency curves for three memory frequencies are shown in Figure 5 (for closed-page mode).

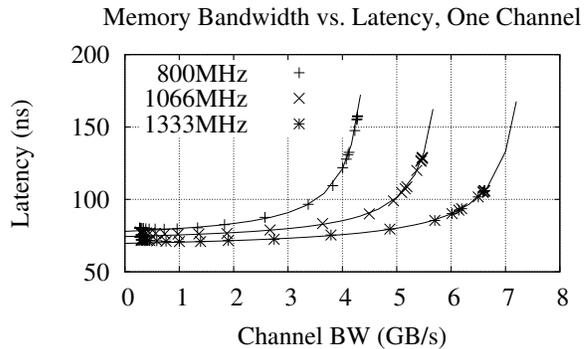


Figure 5: Memory latency as a function of bandwidth demand.

Note that latency remains nearly flat as bandwidth increases up to some point. At that point, the average queue length begins to add a non-negligible *wait time* to each request before it becomes the active request. Eventually, as the request rate (benchmark bandwidth) rises toward the peak service rate (memory system bandwidth), the average queue length grows, and memory latency increases superlinearly. As shown by the fitted curves in Figure 5, we found that the measured points closely matched the curve predicted by the queueing equation for a single-queue, single-server system [3]:

$$MemLatency = IdleLatency + slope * \frac{BW}{PeakBW - BW}$$

where the parameters *IdleLatency*, *slope* and *PeakBW* are fixed for a given system configuration and memory frequency.

5.3 Impact of Frequency on Performance

By varying memory frequency, and thus peak bandwidth, we alter the memory latency perceived by user workloads. As memory latency increases, an out-of-order core is less able to hide the latency, and stall time increases, yielding reduced performance. Ultimately, this effect depends on the application characteristics. However, as we will show below, knowing the *bandwidth* demand of an application suffices in most cases to bound *performance impact* for workloads in SPEC CPU2006, which consists of CPU- and memory-intensive applications. This is due to the fact that a bandwidth-based scaling mechanism with properly set thresholds will transition to a higher frequency, with a lower latency curve and higher saturation point, as soon as queuing delay starts to become significant.

Figure 6 shows performance degradation for SPEC CPU-2006 when memory runs at 800MHz and 1066MHz (degraded from baseline 1333MHz). Benchmarks are sorted by baseline average bandwidth utilization (Fig. 4). In general, benchmarks with higher baseline bandwidth experience more performance degradation at lower memory frequencies, because the baseline bandwidth is closer to (or beyond) the peak bandwidth at the lower frequency. As the latency curves in Figure 5 show, latency rises considerably as utilization approaches maximum bandwidth. These benchmarks show a reduction in actual bandwidth at lower memory frequencies as execution slows due to memory throughput limits.

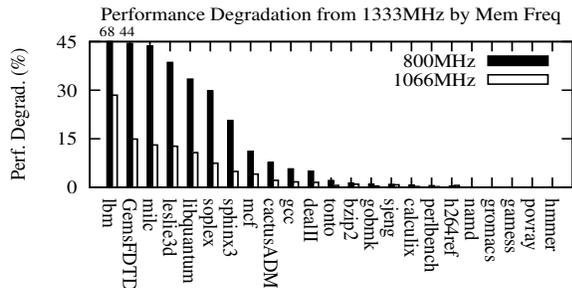


Figure 6: Performance degradation as memory is varied statically from 1333MHz baseline to 1066MHz and 800MHz. Benchmarks with high bandwidth demand show greater degradation.

6. FREQUENCY CONTROL ALGORITHM

As the previous section has shown, benchmarks with lower bandwidth demand are, in general, more tolerant to lower memory frequency. We can reduce average power by scaling down memory frequency when it has little effect; because runtime is not affected, or at worst impacted only a little, *energy efficiency* will improve.

The *key insight* our control algorithm builds on is that at low bandwidth demand, memory latency is not significantly affected by memory frequency. This is because only bus transfer latency is impacted by frequency; other components of memory access latency do not change. By choosing a lower memory frequency at low bandwidth demand, we have little impact on performance, as Figure 6 shows. As bandwidth demand increases, we scale up frequency. This effectively creates a *piecewise latency curve* as a function of bandwidth demand that approximates the shape of the highest memory frequency latency curve.

To implement this, we simply pick a fixed bandwidth threshold for each frequency transition. A controller algorithm runs periodically, at fixed epochs, and measures average bandwidth usage for the previous epoch. Based

on this measurement, it picks the corresponding memory frequency. Algorithm 1 implements this functionality. For our evaluations, we need to specify two thresholds: for the 800 to 1066MHz transition, and the 1066MHz to 1333MHz transition. We evaluate two threshold settings: $BW(0.5, 1)$ transitions to 1066 and 1333MHz at 0.5GB/s and 1.0GB/s per channel, respectively, and $BW(0.5, 2)$ transitions at 0.5GB/s and 2.0GB/s. These thresholds are conservative in that they are below the knees on the latency curves in Fig. 5. Rather, they are chosen based on the range of average per-channel bandwidth measurements in Fig. 4. As our results show in §7, these parameter choices result in minimal performance impact.

Algorithm 1 Bandwidth-based Frequency Selection Policy

```

while true do
  wait for  $t_{sample}$ 
  sample average memory bandwidth per thread as  $BW$ 
  if  $BW < T_{f_1}$  then
    set memory frequency to  $f_1$ 
  else if  $T_{f_1} \leq BW < T_{f_2}$  then
    set memory frequency to  $f_2$ 
  else if  $T_{f_2} \leq BW$  then
    set memory frequency to highest frequency  $f_0$ 
  end if
end while

```

7. EVALUATION

7.1 Methodology

We evaluate our proposed memory-frequency scaling algorithm on real hardware, described in Table 4. We emulate the performance effects of dynamically-variable memory frequency by altering internal memory controller timing registers: in particular, t_{RCD} (RAS-to-CAS delay) and t_{B2BCAS} (back-to-back CAS delay) are set so that the effective memory latency and bandwidth approximate those at the emulated memory frequency. We prefer this methodology to simulation because allows full-length runs of benchmarks and captures full-system behavior.

SPEC CPU2006 benchmarks are run to completion in our evaluation. Table 4 shows the evaluation system’s parameters. In each run, effective memory frequency is either static or is controlled by our algorithm. Memory always physically runs at 1333MHz. Dynamic control is implemented in a userspace daemon, with a custom kernel module to read bandwidth counters and set timing registers in the memory controller.

Power and energy results are computed using real-system measurements and analytically-modeled power reduction. Because memory frequency does not actually change, memory will show no power reduction as we emulate lower memory speeds. However, by taking periodic measurements from an AC power-meter attached to the evaluation system, and subtracting our analytical model’s predicted power reduction (based on our model in §4), we compute full-system power for the frequency-scaled runs. We compensate for 85% AC-DC PSU efficiency and 85% voltage-regulator efficiency by scaling our memory power reductions to correspond to AC power reductions. From average AC power and runtime, we compute energy per workload, in order to evaluate energy efficiency as energy per fixed work unit (one benchmark run).

7.2 Performance Impact

Fig. 7 shows slowdown from baseline for two bandwidth-based policies, $BW(0.5, 1)$ and $BW(0.5, 2)$, alongside static 800MHz and 1066MHz-memory slowdowns for comparison. All slowdowns are relative to a 1333MHz-memory baseline. Additionally, Fig. 8 shows this memory frequency distribution shift as bandwidth decreases. Fig. 9 shows frequency-switching behavior over time for two representative workloads, *bzip2* and *gcc*.

Our bandwidth-based policy is successful in limiting performance impact: at the high-bandwidth end (left), mem-

Processor	two Intel Xeon Nehalem (X5500-series), 4 cores each (8 cores total)
SMT	disabled (one logical thread per core) for repeatability
Power management settings	TurboBoost disabled for repeatability
Memory controllers	three on-chip per 4-core CPU, 6 total, DDR3, 1333MHz (timing altered to emulate the performance effects of frequency scaling)
Memory timing registers	1333 baseline: $t_{RCD} = 9$, $t_{B2BCAS} = 3$; 1066 emulated: $t_{RCD} = 12$, $t_{B2BCAS} = 6$; 800 emulated: $t_{RCD} = 14$, $t_{B2BCAS} = 9$
Memory	12x 4GB DIMMs, registered, dual-rank x4, PC10667 (1333MHz), two per channel
Motherboard	Intel server platform reference board
Hard drive	500GB SATA
OS	Fedora Core 9 (GNU/Linux), single-user mode
SPEC CPU2006	Compiled with Intel C++ compiler 11.0, <i>base</i> optimization, <i>rate</i> run-type, <i>ref</i> input sets (8 copies)
Power supply	Redundant server power supply, 650W, 120V
Instrumentation	Yokogawa WT210 power-meter, 1 sample/second

Table 4: System configuration parameters for evaluation.

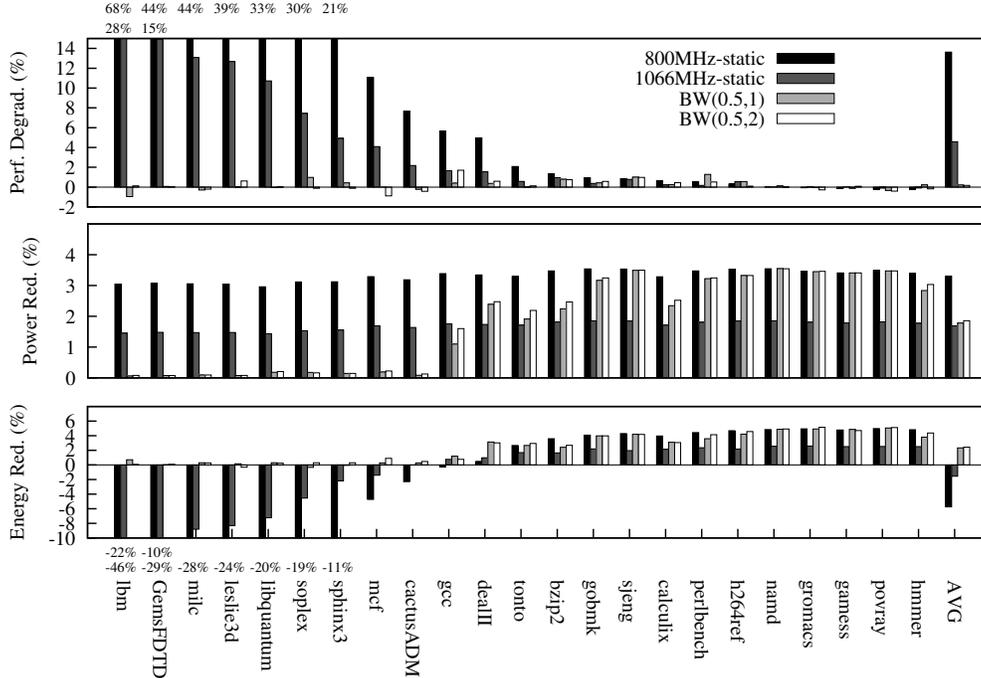


Figure 7: Performance degradation, power reduction, and energy reduction with our memory DVFS policy compared to static-frequency baselines.

ory bandwidth is easily above the threshold, and memory runs at 1333MHz most or all of the time. As the static scaling shows, the performance impact of lower memory frequencies with these workloads would be catastrophic. As memory bandwidth decreases to the right, frequency scaling starts to choose lower settings, and slowdowns become nonzero. However, because thresholds are chosen conservatively, the worst-case slowdown is also minimal, under 2%. For the middle benchmarks, slowdown is still less than that for static scaling, because memory spends only some time in lower frequency states. Finally, for the lowest-bandwidth benchmarks, slowdown again decreases to negligible levels, because these benchmarks are almost completely unaffected by lower memory frequency. As the frequency distribution shows, these workloads spend most or all of their time in the lower-frequency states.

Note that in several cases, measurement errors inherent in a real-system evaluation yield a slightly negative slowdown. We ran each application five times to minimize this error.

7.3 System Power and Energy

Our fundamental goal is to improve energy efficiency while minimally impacting performance. To this end, we determine the total energy (whole-system) taken to run each benchmark. As described above in §7.1, full-system power is computed by subtracting (analytical) memory-power reduction from measured full-system power. We compute energy by multiplying average power with runtime.

Figure 7 shows energy reduction under our control algorithm as well as static frequency reduction for comparison. The results show that the bandwidth-based policy is able to attain as much energy-efficiency improvement as static frequency reduction for low-bandwidth benchmarks, while avoiding the significant efficiency reductions (due to increased runtime and thus inflated total energy) when running high-bandwidth benchmarks. Fig. 7 shows average system power reductions: at low bandwidth, the reduction is significant, proportional to time spent at lower frequencies.

On average, across SPEC CPU2006, the $BW(0.5, 2)$ policy reduces memory power by $6.05W$ ($11.31W$ max) (DC power) for 0.17% average (1.69% max) slowdown. Taking only benchmarks including and to the right of *gcc* when sorted by bandwidth (that is, with bandwidth roughly less than 1.1GB/s per channel), we reduce memory system power by $9.62W$ on average. This is significant when average memory system power at 1333MHz is $65.1W$.

In a whole-system context, $BW(0.5, 2)$ provides 2.43% average (5.15% max) energy-efficiency improvement. It is important to note that (i) power reduction and thus energy-efficiency improvements due to DVFS in memory can give similar power reductions to CPU DVFS (§2.2), and (ii) this reduction comes with negligible performance reduction.

8. RELATED WORK

Memory Power Management: *MemScale* [8], work done concurrent to ours, proposes DVFS in memory systems

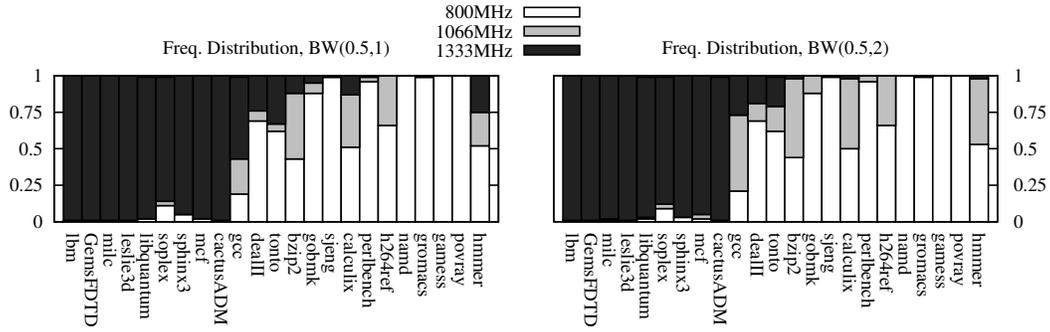


Figure 8: Memory frequency distribution for bandwidth-based policies: as memory intensity decreases to the right, the distribution generally shifts toward lower memory frequencies.

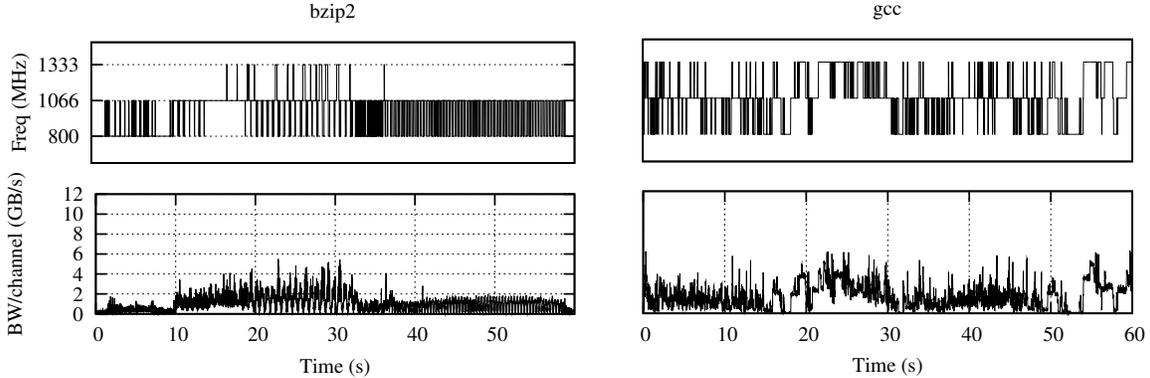


Figure 9: Fine-grained behavior for two representative benchmarks, `bzip2` and `gcc`, over 60-second slices of benchmark execution. The chosen memory frequency follows the general trend of application behavior. `bzip2` spends most time at 800 and 1066MHz, and jumps to 1333 periodically during a memory-intensive phase. `gcc` shows an upward trend between 20 and 30 seconds that causes more frequent shifts to 1333MHz.

as well. While Deng et al. in [8] evaluated memory frequency/voltage scaling in simulation, the primary distinction of our work is the real-system evaluation methodology. By emulating the effects of lower memory frequency on a real hardware platform, we captured the performance effects across the full software stack and thus accurately measured the opportunity for memory DVFS. Our proposal also differs in algorithm design: MemScale estimates performance impact with a model, while our mechanism switches frequency based on memory bandwidth utilization.

Other work also examines memory power management in various ways. First, Puttaswamy et al. [31] examined voltage/frequency scaling of an off-chip memory in an embedded system. Although this work makes use of the same opportunity that we do, it differs in two key ways: (i) most importantly, it evaluates only static voltage/frequency operating points, with no mechanism for dynamic switching; and (ii) it uses SRAM, while we use DRAM, with different power and performance characteristics. Several works have examined the tradeoffs inherent in *memory sleep states*, and how to manage the system to increase opportunity for them. Fan et al. in [11] give an analytical model for sleep-state benefit, and Lebeck et al. in [20] explored the interaction of page placement and memory sleep states. Hur and Lin in [16] present several techniques to increase opportunity for DRAM power-down states, and [1] presents a mechanism to judge when speculative DRAM accesses are power-efficient. All of these techniques are orthogonal to frequency/voltage scaling, and could be combined with our approach.

Another body of work examines *power-limiting* as the primary constraint, either for thermal reasons or for power-management in dense server environments. Lin et al. [21, 22] propose two thermal-motivated mechanisms, both of which throttle the CPU to reduce memory traffic. The first, *adaptive core gating*, uses clock-gating on processor cores when memory nears its thermal limit; the second, *co-*

ordinated DVFS, uses DVFS to slow the cores in the same situation. David et al. in [5] describe *RAPL* (running average power limit), in which memory power is explicitly modelled and limited by throttling. Their power model is similar to ours. However, it is used for a different purpose: while we require a model only for evaluation of memory DVFS, RAPL’s power model is at the center of its algorithmic control loop (to maintain an average power limit). Therefore, RAPL calibrates its weights against measurements from an in-system memory power meter. Diniz et al. [9] propose several algorithms to limit power by turning off a subset of the memory devices in a system that can control each device individually. Felter et al. [12] propose *power shifting*, in which throttling is used to manage power consumption across the system, including the memory subsystem. All these approaches differ from ours by taking a power target/limit as the first-order constraint; we optimize for energy-efficiency instead, aiming to minimally impact performance. However, throttling could be combined with frequency/voltage scaling when scaling alone does not meet a power budget. Or, frequency/voltage scaling alone could be used to meet a power budget, likely yielding better performance than throttling due to long-latency delays when DRAM access is throttled.

Additional work examines how *software-level* decisions, in particular how software allocates and accesses memory, can affect memory system power. Delaluz et al. in [6] describe how compiler analysis of workloads to direct memory mode control can lead to better energy efficiency; in [7], they propose controlling DRAM power states from the OS scheduler. Huang et al. [14] integrate awareness of DRAM power states into the OS virtual memory system, allocating pages intelligently from different banks. Huang et al. in [15] take VM integration a step further by migrating frequently-accessed pages to “hot” ranks, concentrating utilization there and allowing “cold” ranks to enter sleep states

more frequently. Pandey et al. in [30] examine memory accesses due to I/O DMA activity, and propose alignment and data-placement techniques to increase efficiency. All of these higher-level techniques are orthogonal to our work, and could be combined; we mention them simply to show that memory power management is not limited to hardware-level (software-transparent) mechanisms.

Energy Efficiency Techniques: As described previously, energy-proportional computing [2] is the broad research direction that characterizes this work: scaling energy use proportionally to workload demand. Beyond component-specific approaches in the CPU and memory, several works approach energy-proportionality on a larger scale. PowerNap [23] proposes a full-system sleep state in lieu of finer-grain power management. The work demonstrates that for server workloads with idle time between requests, it is more efficient to serve requests quickly and then enter the low-power sleep state than to scale performance down. While this approach is valid when such idle time exists, workloads that run continuously but for which the system is imbalanced (e.g., heavily CPU-bound or memory-bound) do not contain such idle periods. Our approach specifically targets such CPU-bound workloads that underutilize memory bandwidth. However, the techniques are compatible: PowerNap can take advantage of idle periods, while memory frequency scaling can help to achieve energy efficiency while the system is active.

At the datacenter scale, Elnozahy et al. [10] propose scaling the size of a server cluster according to load. By concentrating load onto fewer, more highly-utilized servers, this technique approximates energy proportionality by reducing idle-time waste. Tolia et al. [34] optimize a cluster through a combination of CPU DVFS and consolidation. As above, this general approach solves the problem for a different class of workload, but is compatible with our proposal.

9. FUTURE WORK

We have introduced the basic tradeoffs in memory frequency scaling, and performed an initial evaluation using a simple and intuitive algorithm. However, more work remains to be done. First, our mechanism is simple, and there is a large design space, both in predicting and measuring performance impact and on predicting the future impact of memory frequency changes. Further work can investigate both the measurement and prediction aspects of this problem, and characterize how various types of workloads respond to increased memory latency. Additionally, the interaction between memory scaling and CPU frequency/voltage scaling (DVFS) has not been examined in this work. Certainly, the two mechanisms could supply hints to each other. It could also be the case that better efficiency improvements are possible through coordinated control than when the two operate independently. Finally, we considered only SPEC CPU2006 in this work; further evaluations are necessary to quantify performance impact in other workloads.

10. CONCLUSIONS

In this work, we propose and evaluate *memory voltage/frequency scaling* in order to reduce memory power and increase energy efficiency. Starting from the observation that a significant portion of memory-system power is frequency-dependent, we present a control algorithm that reduces memory frequency while limiting performance impact. The key observation is that at low memory bandwidth utilization, lowering memory frequency does not significantly alter memory access latency. By observing memory bandwidth utilization, our proposed control algorithm increases memory frequency when utilization crosses a threshold, bounding the performance impact. We conclude that memory DVFS can be an effective energy efficiency technique, especially when memory bandwidth utilization is low.

Acknowledgments

We thank the anonymous reviewers for their feedback. We gratefully acknowledge members of the Power Management Architecture group at Intel Labs. Chris Fallin was an intern at Intel Labs while conducting this work. He is currently supported by an NSF Graduate Research Fellowship. We acknowledge the support of Intel and the Gigascale Systems Research Center. This research was partially supported by an NSF CAREER Award CCF-0953246.

References

- [1] N. Aggarwal, J. F. Cantin, M. H. Lipasti, and J. E. Smith. Power-efficient DRAM speculation. *HPCA*, 2008.
- [2] L. Barroso and U. Holzle. The case for energy-proportional computing. *IEEE Computer*, Dec 2007.
- [3] U. N. Bhat. *An Introduction to Queueing Theory: Modeling and Analysis in Applications*. Birkhauser, 2008.
- [4] R. Bianchini and R. Rajamony. Power and energy management for server systems. *IEEE Computer*, 37, Nov 2004.
- [5] H. David et al. RAPL: Memory power estimation and capping. *ISLPED*, 2010.
- [6] V. Delaluz et al. DRAM energy management using software and hardware directed power mode control. *HPCA*, 2001.
- [7] V. Delaluz et al. Scheduler-based DRAM energy management. *DAC*, 2002.
- [8] Q. Deng et al. MemScale: Active low-power modes for main memory. *ASPLOS*, 2011.
- [9] B. Diniz et al. Limiting the power consumption of main memory. *ISCA-34*, 2007.
- [10] M. Elnozahy, M. Kistler, and R. Rajamony. Energy-efficient server clusters. *Power-Aware Computer Systems*, 2003.
- [11] X. Fan, H. Zeng, and C. Ellis. Memory controller policies for DRAM power management. *ISLPED*, 2001.
- [12] W. Felter et al. A performance-conserving approach for reducing peak power consumption in server systems. *ICS*, 2005.
- [13] A. Glew. MLP yes! ILP no! *ASPLOS*, 1998.
- [14] H. Huang et al. Design and implementation of power-aware virtual memory. *Usenix Annual Tech. Conf.*, 2003.
- [15] H. Huang et al. Improving energy efficiency by making DRAM less randomly accessed. *ISLPED*, 2005.
- [16] I. Hur and C. Lim. A comprehensive approach to DRAM power management. *HPCA*, 2008.
- [17] IDT. SSTE2882HLB (DDR3 Register / PLL) Datasheet. <http://www.idt.com/?catID=18710992>.
- [18] Intel Corporation. Intel memory 3-sigma power analysis methodology. <http://edc.intel.com/>.
- [19] B. Keeth et al. *DRAM Circuit Design: Fundamental and High-Speed Topics*. Wiley-IEEE Press, 2007.
- [20] A. Lebeck et al. Power aware page allocation. *ASPLOS*, 2000.
- [21] J. Lin et al. Thermal modeling and management of DRAM memory systems. *ISCA-34*, 2007.
- [22] J. Lin et al. Software thermal management of DRAM memory for multicore systems. *SIGMETRICS*, 2008.
- [23] D. Meisner et al. PowerNap: eliminating server idle power. *ASPLOS*, 2009.
- [24] Micron Technology Inc. 2Gb: x4, x8, x16 DDR3 SDRAM (datasheet). http://download.micron.com/pdf/datasheets/dram/ddr3/2Gb_DDR3_SDRAM.pdf.
- [25] Micron Technology Inc. Tn-41-01: calculating memory system power for DDR3. http://download.micron.com/pdf/technotes/ddr3/TN41_01DDR3%20Power.pdf.
- [26] Micron Technology Inc. Tn-46-05: general DDR SDRAM functionality. <http://download.micron.com/pdf/technotes/TN4605.pdf>.
- [27] O. Mutlu and T. Moscibroda. Stall-time fair memory access scheduling for chip multiprocessors. *MICRO-40*, 2007.
- [28] O. Mutlu and T. Moscibroda. Parallelism-aware batch scheduling: enhancing both performance and fairness of shared DRAM systems. *ISCA-35*, 2008.
- [29] C. Natarajan, B. Christenson, and F. A. Briggs. A study of performance impact of memory controller features in multi-processor server environment. *WMPPI*, 2004.
- [30] V. Pandey et al. DMA-aware memory energy management. *HPCA*, 2006.
- [31] K. Puttaswamy et al. System level power-performance trade-offs in embedded systems using voltage and frequency scaling of off-chip buses and memory. *ISSS*, 2002.
- [32] Standard Performance Evaluation Corporation. SPEC CPU2006. <http://www.spec.org/cpu2006/>.
- [33] K. Sudan et al. Micro-Pages: increasing DRAM efficiency with locality-aware data placement. *ASPLOS*, 2010.
- [34] N. Tolia et al. Delivering energy proportionality with non energy-proportional systems – optimizing the ensemble. *HotPower*, 2008.
- [35] U.S. EPA. Report to Congress on server and data center energy efficiency. *U.S. Environmental Protection Agency, Tech. Rep.*, Aug 2007.
- [36] M. Weiser et al. Scheduling for reduced CPU energy. *OSDI-1*, 1994.