Rethinking DRAM Power Modes for Energy Proportionality

Krishna T. Malladi† Ian Shaeffer‡ Liji Gopalakrishnan‡
David Lo† Benjamin C. Lee§ Mark Horowitz†
Stanford University † Rambus Inc‡ Duke University§
{ktej, davidlo, horowitz}@stanford.edu †, {ians, lijig}@rambus.com‡, {benjamin.c.lee}@duke.edu

Abstract

We re-think DRAM power modes by modeling and characterizing inter-arrival times for memory requests to determine the properties an ideal power mode should have. This analysis indicates that even the most responsive of today's power modes are rarely used. Up to 88% of memory is spent idling in an active mode. This analysis indicates that power modes must have much shorter exit latencies than they have today. Wake-up latencies less than 100ns are ideal.

To address these challenges, we present MemBlaze, an architecture with DRAMs and links that are capable of fast powerup, which provides more opportunities to powerdown memories. By eliminating DRAM chip timing circuitry, a key contributor to powerup latency, and by shifting timing responsibility to the controller, MemBlaze permits data transfers immediately after wake-up and reduces energy per transfer by 50% with no performance impact.

Alternatively, in scenarios where DRAM timing circuitry must remain, we explore mechanisms to accommodate DRAMs that powerup with less than perfect interface timing. We present MemCorrect which detects timing errors while MemDrowsy lowers transfer rates and widens sampling margins to accommodate timing uncertainty in situations where the interface circuitry must recalibrate after exit from powerdown state. Combined, MemCorrect and MemDrowsy still reduce energy per transfer by 50% but incur modest (e.g., 10%) performance penalties.

1. Introduction

In an era of big data and datacenter computing, memory efficiency is imperative. More than 25% of datacenter energy can be attributed to memory and this fraction will only grow with demands for memory capacity [13, 24, 28].

Recent efforts to improve efficiency study memory that is active and transferring data. The resulting architectures focus on reducing energy per transfer. By tailoring DRAM page width, memory core energy is made proportional to the amount of data requested [2, 37, 41]. However, none of these architectures address a different source of inefficiency: idle memories kept in an active power mode.

One approach to address this problem is to use mobile-class DRAM [27] which have much lower active idle power. But using LPDDR2 requires a static decision to trade bandwidth for efficiency. Alternatively, we could use dynamic powerdown modes but controllers have difficulty invoking them. Transfers are separated by idle periods but they are often too short to justify powerdown.

Indeed, witness the sophistication and complexity of efforts in the compiler, operating system, and architecture to consolidate memory activity to a small number of active ranks [14, 15, 23]. By attempting to lengthen idle periods in other ranks, these approaches acknowledge the unwieldy nature of today's power modes and build systems to accommodate them.

In this paper, we present a fundamentally different approach. Instead of shaping memory activity to produce idleness suited to existing power modes, we re-think the power modes themselves. In an application-driven approach, we model and characterize inter-arrival times for memory requests to determine the properties an ideal power mode should have. This analysis indicates that power modes must have much shorter exit latencies than they have today.

To architect power modes with fast exits, we identify the key contributor to powerup latency: DRAM timing circuitry. The most efficient modes turn off delay-locked loops (DLLs) and clocks. But turning them on again requires expensive recalibration (e.g., 700+ns). Few applications have idle periods long enough to justify this latency. Thus, existing modes offer an unattractive energy-delay trade-off.

We improve this trade-off with a new I/O architecture that shifts timing circuitry from DRAMs to the controller while preserving high bandwidth. In this architecture, the first transfer after wake-up completes in a few nanoseconds. Such responsiveness is orders of magnitude faster than the exit latency of today's most efficient power mode, which must recalibrate timing after wake-up. We make the following contributions:

- **Understanding Power Mode Inefficiency.** Even the most responsive of today's power modes are rarely used. Up to 88% of memory time is spent idling in an active mode. Addressing limitations in existing DRAMs could improve energy efficiency by 40-50%.
- **Understanding Memory Activity.** We study memory activity and its implications for power mode design. A probabilistic analysis establishes a clear path from fast wake-up to attractive energy-delay trade-offs. A workload characterization indicates wake-up in $\leq$100ns is ideal.
- **Rethinking Power Modes.** We present *MemBlaze*, a DRAM I/O architecture that is capable of fast wake-up while ensuring high bandwidth. Alternatively, we propose two new mechanisms: *MemCorrect*, which detects timing errors, and *MemDrowsy*, which lowers transfer rates to widen timing margins. These architectures allow memory transfers immediately after wake-up.
- **Saving Energy.** MemBlaze reduces energy per transfer by up to 50% with negligible performance penalty since data transfers begin immediately after wake-up. If timing is less than perfect, a combination of MemCorrect and MemDrowsy provide similar energy savings with a 10% performance penalty incurred to correct timing errors.

2. Background and Motivation

Today's DRAM interfaces provide performance but dissipate high idle power. Moreover, these interfaces include power modes which are disconnected from architectural requirements. To address these challenges, we architect new DRAM interfaces for fast transitions between power modes.

2.1. DRAM Systems

Each DRAM device contains two-dimensional arrays of memory cells. Multiple devices comprise a rank and multiple ranks share a data bus. Figure 1 illustrates a memory system with four ranks that

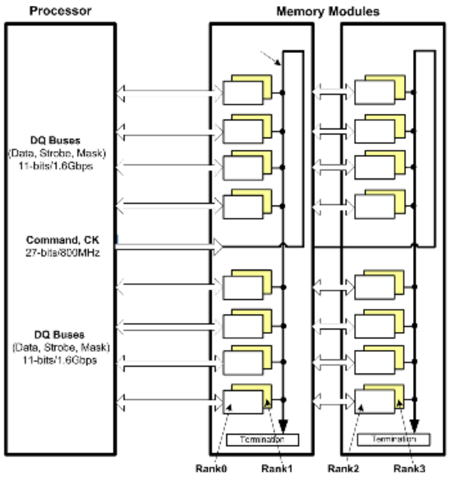

Figure 1: DDR3 DRAM Memory System.

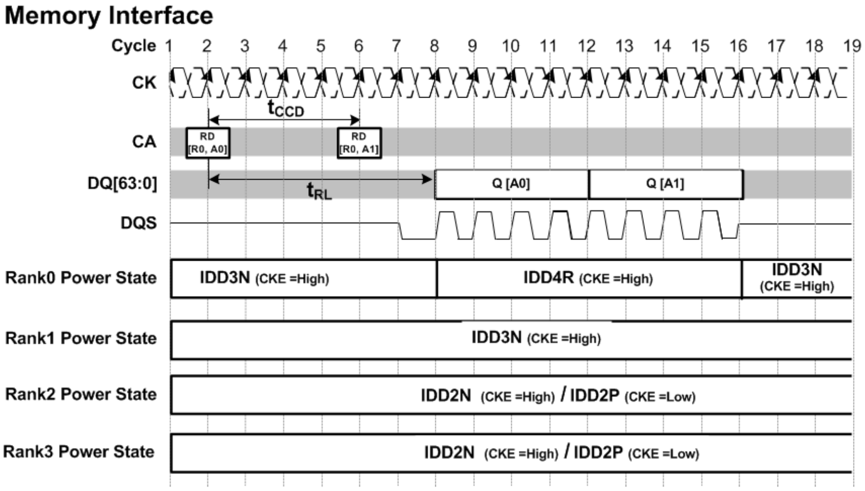

Figure 2: DDR3 DRAM Timing.

Power Mode	DIMM Idle Power (W)	Exit Latency (ns)	Mechanism
Active idle	5.36	0	none
Precharge-idle	4.66	14	pages closed
Active powerdown	3.28	6	clock, I/O buffers, decode logic off
Fast exit powerdown	2.79	19.75	active powerdown + pages closed
Slow exit powerdown	1.60	24	fast exit powerdown + DLL frozen
Self Refresh	0.92	768	fast exit powerdown + DLL, CK off
Self Refresh + registers off	0.56	6700	self refresh + register PLLs off
Disabled	0	disk latency	DIMMs off

Table 1: Power Modes for a 4GB DDR3-x4-1333 RDIMM [5, 30]

share a x64 channel. The number of channels and the interface's data rate determine system bandwidth.

Each channel is attached to a memory controller, which is integrated on the processor die. To activate a row, the controller issues a row access strobe (RAS) to enable word lines and buffers a row's data. To read and write, a column access strobe (CAS) transfers buffered data to IO interfaces. Prefetching 8 bits across a 64b wide channel produces 64B to fill a processor cache line.

2.2. DRAM Interfaces

The controller and DRAMs are connected by CA and DQ buses for control and data signals. To synchronize signals, the controller generates and forwards a clock (CK) to the DRAMs. Controller circuitry aligns this clock with command and enable signals. Because these signals have lower bandwidth and experience the same loading conditions and discontinuities en route to DRAMs, skew is not an issue. Thus, commands and writes are synchronized.

However, synchronizing reads is more difficult. During a read, data signals are generated by DRAMs (DQ) while clock signals are generated by the controller (CK). Originating on different dies, these signals are subject to different loading conditions and variations in process, voltage, and temperature. Under these conditions, the controller has difficulty using CK edges to sample DQ for arriving read data, especially at high frequencies and narrow data windows.

To facilitate read synchronization, DRAMs explicitly communicate data timing to the controller with a data strobe signal (DQS) that is aligned with the clock (CK) and bus data (DQ). The controller samples DQ on DQS edges as illustrated in Figure 2. Data is available some latency after receiving a read command (RD on CA produces Q on DQ after tRL).

DQS edges and data windows must align with the controller-generated clock. DRAMs ensure alignment in two ways. First, during initialization, DQS and CK are calibrated to eliminate any skew due to wire length while the controller specifies worst-case tolerance for timing differences (tDQSCK). Second, during operation, delay-locked loops (DLLs) dynamically adjust the DRAM clock delays to compensate for voltage and temperature variations and keep the position of the DQS at the controller constant to reduce timing uncertainty when sampling data at high frequencies.

2.3. DRAM Power Mode Limitations

Consider two scenarios in which DLLs affect efficiency. In the first, the DRAM is idling in an active power mode. In such an 'active-idle' state, the DLL and clocking power are a large fraction of the total power. For example, DDR3 active-idle current is 2× that of LPDDR2 and much of this difference is attributed to the interface [27].

In a second scenario, which we call 'idle-idle', the DRAM is in a powerdown mode. More efficient modes have higher powerup latencies (e.g., self-refresh in Table 1). While this state seems energy-efficient, the next reference pays the cost as the DRAM spends tDLLK=512 active memory cycles (768ns) powering up the interface. This is a lot of energy. In addition, applications slow down, as indicated in Figure 3(a). Thus, existing DRAM interfaces impose unattractive performance and power tradeoffs.

Static mechanisms to reduce interface power fare no better. We can configure the memory mode registers (MR) in the BIOS [30], eliminating DLLs but this imposes performance penalties. First, the peak data rate is halved as channel frequency must be lowered to ensure signal integrity. Furthermore, without DLLs, timing is less certain and controllers must assume worst-case margins (i.e., tDQSCK=10ns [30]). Conservative timing increases critical word latency, affecting application performance as shown in (Figure 3(b)).

Due to these punishing trade-offs, memory controllers invoke power modes conservatively. Modern controllers recommend a powerdown threshold no lower than 15 idle memory cycles [17]. Figure 4(a) shows the percent of time the DRAMs stay in each power state for this aggressive threshold (A), a moderate (M) threshold 10× larger, and a conservative (C) threshold 100 × larger. With such thresholds, up to 88% of memory time is in active-idle.

Potential for Efficiency. Suppose we were to address limitations in today's interfaces and power modes so that the most efficient

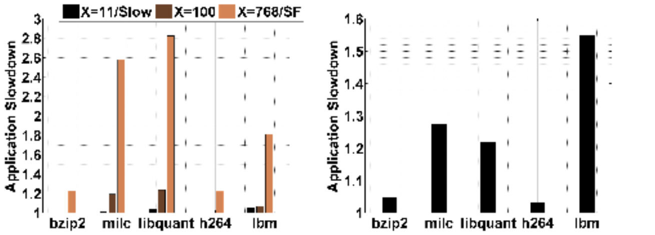

Figure 3: Performance sensitivity to (a) dynamic power down modes at different exit latencies and (b) static BIOS programming to disable DLLs.

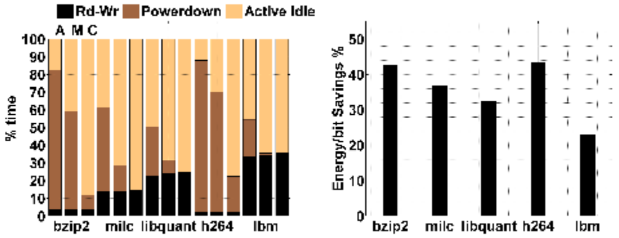

Figure 4: (a) Memory time breakdown with aggressive (A), moderate (M), and conservative (C) thresholds; (b) Potential efficiency from new power modes.

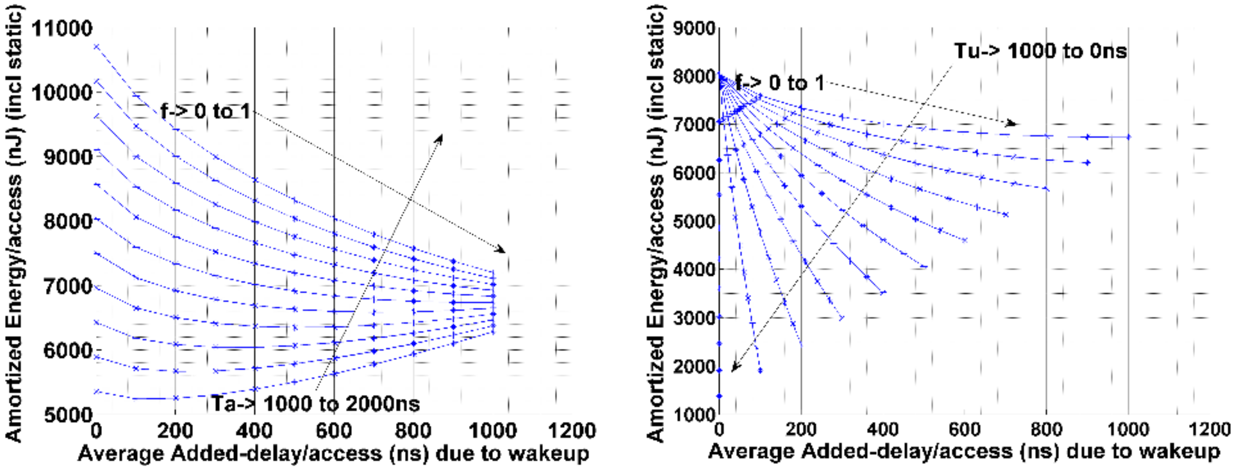

Figure 5: Probabilistic energy-delay trade-offs when powerup latency is exposed. E-D plots with different lines for different (a) memory request inter-arrival times and (b) varying powerup times. Each line's points sweep powerdown fraction f.

modes could be exploited. We would enter these modes aggressively (A) and leave them instantaneously upon the next access. Efficiency could improve by 40-50% (Figure 4(b)). Moreover, performance penalties would be negligible.

In this paper, we re-think energy-delay trade-offs with new DRAM architectures. We consider a high-performance system that requires sustained bandwidth; we cannot simply eliminate DLLs and operate at lower data rates. We present architectures that reduce power mode exit latencies and interface power by replacing DLLs with another synchronization mechanism or using existing DLLs differently.

3. Understanding Memory Activity

The precise benefits of fast exit power modes depend on the interaction between memory activity and the exit latency, which we study in two ways. First, we probabilistically model memory requests in order to understand fundamental energy-delay trends. Then, we precisely capture memory request inter-arrival times from emerging big data applications.

3.1. Probabilistic Energy-Delay Analysis

We model a stream of memory requests as a Poisson process. This analysis assumes that the time between requests follow an exponential distribution and these inter-arrival times are statistically independent, which roughly match our data. Histograms for memory inter-arrival times resemble exponential probability densities and the autocorrelation between inter-arrival times is nearly zero.

Let T_i be an exponentially distributed random variable for the idle time between two memory requests. The exponential distribution is parameterized by $1/T_a$ where T_a is the average inter-arrival time. Let P_d and P_u denote power dissipated in powerdown and powerup modes. The memory powers-down if idleness exceeds a threshold T_t. And it incurs a latency T_u when powering-up again.

Power-down is invoked with probability $f = P(T_i > T_t) = e^{-T_t/T_a}$. In this scenario, DRAM dissipates P_d for $T_i - T_t$ time while powered-down and dissipates P_u for $(T_t + T_u)$ time while powered-up. T_i is the only random variable; $\mathbb{E}[T_i] = T_a$.

$$\begin{aligned}\mathbb{E}[E] &= f \times \mathbb{E}\left[P_d(T_i - T_t) + P_u T_t + P_u T_u\right] + (1-f) \times \mathbb{E}\left[P_u T_i\right] \\ &= f \times \left[P_d(T_a - T_t) + P_u T_t + P_u T_u\right] + (1-f) \times \left[P_u T_a\right] \\ &= P_d\left[f(T_a - T_t)\right] + P_u\left[f(T_t + T_u) + (1-f)T_a\right]\end{aligned}$$

With this probabilistic formulation, the expectation for memory energy is given by $\mathbb{E}[E]$. And the expected impact on delay is $\mathbb{E}[\Delta D] = fT_u$, which conservatively assumes that powerup latency is exposed on the critical path.

Clearly, we would prefer to frequently use an efficient powerdown mode (i.e., large f and $P_d << P_u$). Energy falls as inter-arrival time increases beyond the threshold. Conversely, energy increases if powerup latency is large.

Energy-Delay Trade-offs. The relationship between $\mathbb{E}[E]$ and $\mathbb{E}[\Delta D]$ depends on average inter-arrival times (T_a), powerdown threshold (T_t), and powerup latency (T_u).

First, consider various inter-arrival times and powerdown thresholds. Each curve in Figure 5(a) plots trade-offs for a particular inter-arrival time T_a at $T_u = 1000$ns and points along a curve represent varying thresholds T_t. Short inter-arrival times ($T_a = 1000$ns) mean that the added energy costs to power back up are expensive than the savings by invoking powerdown. Thus both the energy and

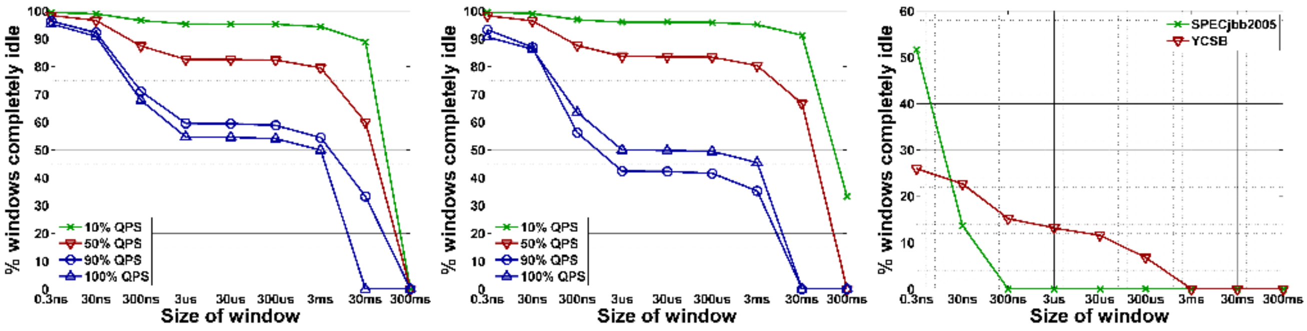

Figure 6: Activity graphs for (a-b) memcached at value sizes of 100B and 10KB. (c) SPECjbb2005 and YCSB.

delay increase with the powerdown fraction f. As inter-arrival times increase (T_a→2000ns), the energy saving in powerdown offsets the overhead of wakeups.

But by implementing different thresholds for powerdown, memory controllers can explore steep and interesting trade-offs between energy and delay. Each curve in Figure 5(b) plots trade-offs for a particular powerup latency and clearly shows the cost of slow wakeups. At one end of the spectrum, zero latency powerup reduces energy with no delay penalty ($T_u = 0$ns). Waiting to go to powerdown only costs more energy, as the energies are higher for low values of f. In contrast, today's approach to disabling DLLs and clocks is expensive, producing a horizontal trend line ($T_u = 1000$ns).

Ideally, power modes reduce energy with little delay impact. This scenario would manifest as instantaneous powerup and produce vertical lines in Figure 5(b). In practice, today's efficient power modes require nearly 1000ns to powerup, producing nearly horizontal energy-delay trends in these figures. To close the gap between the ideal and practice, we re-think memory architecture to reduce powerup latency and accommodate practical inter-arrival statistics in real applications.

3.2. Characterizing Emerging Applications

The nature of computing has changed significantly in the last decade [11] and many emerging datacenter applications have memory behavior that is not well understood. While a prior study quantifies memory idleness in websearch [29], it does so for coarse, 100ms, time periods. At this granularity, which is many orders of magnitude larger than device access times (e.g., 100ns), understanding application requirements for power modes is difficult.

To study memory behavior at fine granularity, we use a custom simulation infrastructure with x86 instrumentation and a built-in scheduler [33] to benchmark a spectrum of real applications. From the spectrum of emerging data driven workloads, we characterize three representative workloads: memcached for distributed memory caching, Yahoo! Cloud Serving Benchmark (YCSB) for OLTP and data serving, and SPECjbb2005 for conventional enterprise computing.

Applications. Distributed memory caching is used by many popular websites (e.g., Facebook and Twitter) to improve query times while OLTP applications are popular in cloud computing. On the other hand, Java-based middleware servers are still popular in many enterprises.

Memcached is a popular open source distributed key-value store that caches data in DRAM for fast retrieval [32]. As the cache fills, evictions occur according to an LRU policy. Memcached hashes keys to distribute load and data. Memcached activity is a function of data popularity and query rate. We model popularity with a zipf distribution and use a large α parameter to create a long tail. We model query inter-arrival times with an exponential distribution. Such models are consistent with observed memcached queries in datacenters [34].

Yahoo! Cloud Serving Benchmark (YCSB) is a benchmark for online transaction processing that interfaces to cloud databases, such as Cassandra, BigTable, and HBase [4]. To characterize YCSB, we first populate a 6.2GB database. Next, we use the YCSB client model to generate a zipf distribution with operations that have a 95:5 read to write ratio, which is representative of modern, read-heavy applications.

SPEC Java Server Benchmark emulates a three-tier client/server system with an emphasis on the middle tier. SPECjbb performs work that is representative of business logic and object manipulation to simulate the middle tier. It exercises the Java Virtual Machine, Just-In-Time compiler, garbage collection, threads and some aspects of the OS [35].

Memory Activity. To help understand how applications will interact with low power modes, we use rank-level activity graphs to visualize memory behavior [29]. These graphs characterize bus activity using windows that define a period of time. We sweep a window over the timeline of application execution and count the number of completely idle windows for varying different window sizes. If applicable, this measurement is also taken across various application loads, which is measured in queries per second (QPS) relative to the system's peak load (denoted as %QPS).

At small value sizes (100B), memcached is CPU-bound as the CPU must cope with many small, incoming packets. At large value sizes (10KB), memcached saturates the network connection. With limited network bandwidth (e.g., 10Gb/s), memcached rarely stresses memory bandwidth (e.g., 80Gb/s).

Although memcached does not saturate memory bandwidth, we must determine whether the memory channel has uniformly low utilization or has bursty traffic with many periods of idleness. The former would make power mode design difficult but the latter would benefit from many existing DRAM power modes, and even full-system power modes.

Figure 6(a-b) shows rank-level activity graphs for memcached configured at 100B and 10KB. A large percentage of short windows (e.g., 100ns) are idle. At typical loads between 10-50%, 95% of the windows encounter completely idle memory. Moreover, these idle periods are long. Even as we increase window size towards microsecond granularities, 80-90% of these windows encounter idle memory. However, idleness is difficult to find as application load increases to 90-100% or when windows widen beyond μs granularities.

While memcached exhibits such idleness, conventional datacenter workloads in data serving and online-transaction processing have even fewer opportunities to exploit existing power-modes when run at 100% QPS. Figure 6(c) illustrates few idle windows for SPECjbb and YCSB even at small windows. At lower utilizations that are typical in datacenters [29], the idle fractions could be higher but the opportunities are scarce beyond 1μs.

Clearly, with idle windows at the order of 1μs or less for emerging workloads, power modes that can transition in the order of 100ns are necessary. However, today's modes are insufficient to take advantage of the memory idle times in many workloads like memcached. They either have wakeup times of a few ns with consequently small energy savings, or they require nearly a μs to wakeup. Such power modes are not applicable to these applications and new modes are needed.

4. Architecting Effective Power Modes

Probabilistic analysis and workload characterization highlight the importance of fast wake-up for memory efficiency. We review high-speed interfaces to explain today's long wake-up times. Then, we propose several architectures with much shorter idle to active transitions but differ in how conservatively they enforce timing after wake-up.

4.1. High-Speed Interfaces

A reliable, high-speed interface performs two critical tasks. First, the interface converts a sequence of bits into a voltage waveform. Then, it drives that waveform on a wire with enough margin so that the receiver can distinguish between the voltages that represent ones and zeros. For high data rates, we engineer the wires as transmission lines and use termination to avoid reflections. Even so, loss in the wires and process variations cause high and low voltage levels to become distorted and mixed when they arrive at the receiver. Equalization cleans up the waveforms.

But getting the signal to the receiver is only half the battle. The other half is knowing when to sample the signal to get the correct value of the bit. At a data rate of 1.6Gb/s, the time window for each bit is only 625ps, and this time includes transitions from the previous bit and to the next bit. To build a reliable link, the interface needs to sample the bit in the middle of the stable region.

Analog circuits drive and receive the bits, and align the clock so that bits are sampled at the right time. These circuits use voltage and current references for their operation, and often use feedback to learn the right corrections (e.g. sample time or equalization) that optimize link operation.

Because they are turned off during powerdown, these circuits must re-learn their connection settings before the link can be operated again. Worse, this re-learning cannot begin until voltage and current references stabilize after powerup. Because analog circuits have lower bandwidth than digital ones, yet demand precision, μs settling latencies are typical.

One of the critical circuits in high-speed links is a delay locked loop (DLL), which uses feedback to align the phase (timing) of two clocks. In links, DLLs align the sample clock to data, or align data and strobes to the system clock. DLLs compensate for changes in timing that would otherwise occur from variations in process, voltage, and temperature (PVT). Since voltage and temperature are dynamic, DLLs continue to run after initial calibration to track and remove their effect [3, 21].

Interfaces that rapidly transition from idle to active mode apply several strategies, such as digitally storing "analog" feedback state, using simpler analog circuits that power off quickly, and designing bias networks that power off and on quickly. Applying these strategies allows DRAMs to wake-up quickly.

4.2. Fast Wake-up DRAMs

Existing link interfaces are generally symmetric: circuitry on both sides of the link need to be the same. But, symmetry is not optimal in a memory system that has a large number of DRAMs but usually a small number of controllers. Furthermore, because DRAM process technology is optimized for density, the speed of its transistors is much worse than that of transistors in a comparable logic process. Thus, we would rather shift link circuitry from DRAMs to the controller.

MemBlaze DRAMs. We present a post-DDR4 DRAM architecture with an asymmetric link interface that removes clock delay circuitry from DRAMs and places them on the memory controller. Because such circuitry determines wake-up latency in today's DRAMs, the system is capable of much faster power mode transitions. The controller and memory interfaces, which we have implemented in silicon, are shown in Figure 7.

Synchronization. In this architecture, DRAMs no longer have DLLs for timing adjustment. For arriving commands and writes, DRAMs simply sample inputs at the rising edge of link clocks, CK and DCK received from the controller. But synchronizing reads (i.e., data from DRAM to controller) requires special treatment. DRAMs no longer send data strobes along with data, which raises two new issues. The first is how the controller can learn the correct timing, and the second is that this timing may be different for each rank.

To address these challenges, the controller uses a clock and data recovery (CDR) circuit to update its clock, and thus update its sample points for data reads, based on a timing reference signal (TRS) received from DRAMs. The DRAMs time-multiplex the TRS on a pin used for error detection and correction (EDC). For every read and write, DRAMs calculate and transmit an 8-bit EDC to the controller. The remainder of the 32-bit EDC burst transmits DRAM clock information.

Thus, during normal rank operation, the EDC pin transmits correction codes interleaved with a toggling pattern that guarantees some minimum edge density. The controller tracks timing variations for each DRAM in a rank as long as that rank sees activity and communicates edges across the EDC pin. Activity on one rank provides no timing for other ranks.

Accommodating Idle Ranks. With regular accesses to a rank, the controller tracks rank timing. But gaps in activity produce gaps in phase updates. Because our interfaces rely on these updates, DRAMs specify the maximum amount between rank accesses. Ranks with longer idle periods incur a recalibration latency before further data transfers.

Alternatively, data-less pings can maintain timing when data is not needed from the memory core but toggling patterns are needed on the EDC pin for phase updates. The ping furnishes a toggling pattern without page activation or column access strobe. In this scenario, the system uses EDC signals for timing and ignores DQ signals.

Recalibration or data-less pings are small overheads that are rarely incurred. But when pings do occur, they coincide with periods of low channel utilization and thus do not interfere with normal traffic.

Fast Wake-up Protocol. Because MemBlaze DRAMs do not have DLLs, the critical latency during wake-up shifts from clock delay circuitry to the datapath. MemBlaze defines an extra control pin (DCKE) to enable the data clock domain, quickly powering the datapath, data clock buffering, and data I/O circuitry (shaded blocks in Figure 7). DCKE observes timing constraints to avoid a latency penalty.

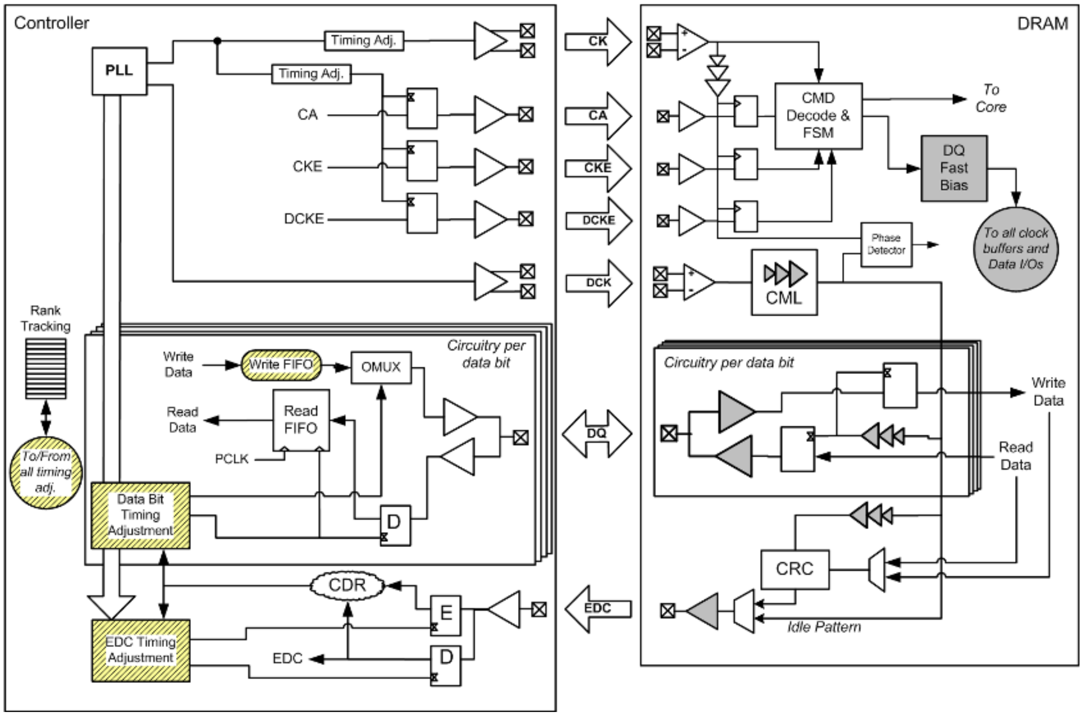

Figure 7: Proposed architecture introduces clock data recovery (CDR) circuitry to the controller, which uses the timing reference signal (TRS) transmitted across error detection and correction (EDC) pins.

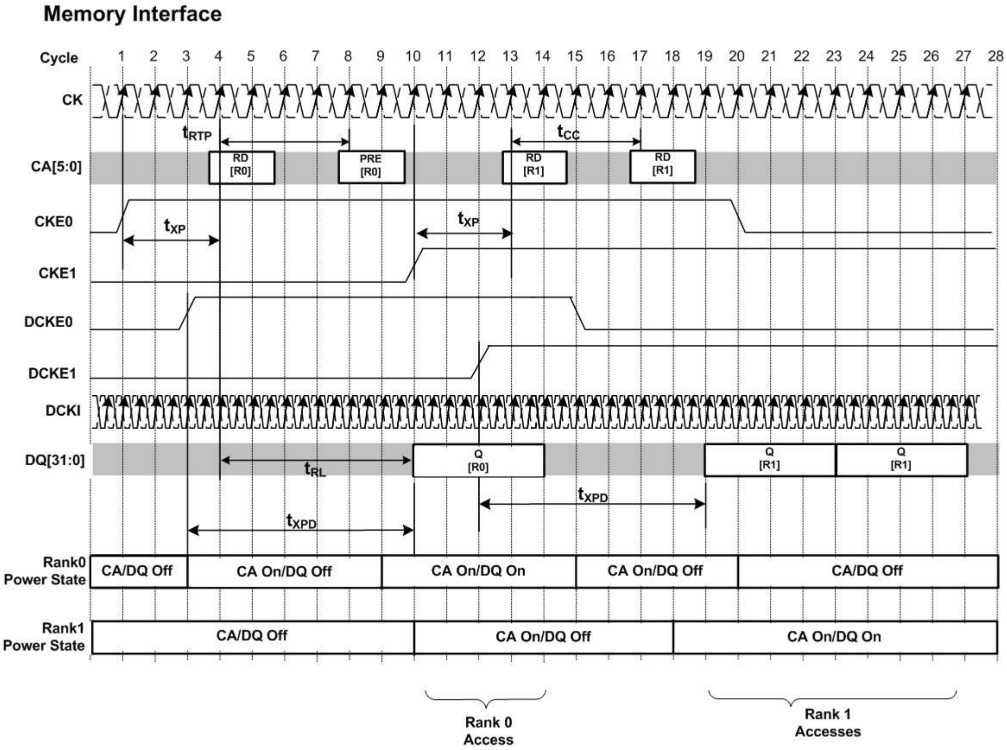

Figure 8: Timing diagram illustrates separate power management for command (CA) and data (DQ) paths.

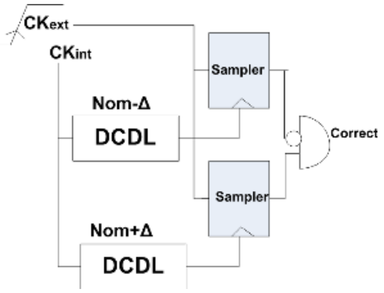

Figure 9: MemCorrect Error Detection with Digitally Controlled Delay Lines (DCDL) to sample CK_{ext} using delayed versions of nominal CK_{int}.

Figure 8 illustrates operation in a two-rank MemBlaze system. Initially, data and clock enables (DCKE, CKE) for both ranks are de-asserted, which powers-down command and data blocks. At cycle 1, CKE0 for rank 0 is asserted and the command block powers-up; the data block remains powered-down. Command receivers (CA) are awake in time to receive a read for rank 0. At cycle 3, DCKE0 is asserted and the data block powers-up to transmit read data. Exit latency for command and data blocks are tXP and tXPD.

Similarly, the second rank exits command standby at cycle 10 and exits data standby at cycle 12. Reads arrive at cycles 13 and 17, satisfying constraints on consecutive reads, which is denoted by tCC. Power-up does not affect read latency as long as DCKE is asserted early enough (i.e., tXPD before first read data). By separating command and data block enable signals, the DQ interface circuitry can remain powered-down even as precharge and refresh commands arrive.

Hiding Datapath Wake-up Latency. By default, datapath wake-up requires approximately 10ns. MemBlaze defines the DCKE pin to enable the data block early enough to avoid affecting latency and completely hide it under column access (CAS). Thus, we can quickly power the datapath only when needed and separate the powerups for command and data blocks. For fast DRAM interface wake-up, MemBlaze exploits a number of circuit innovations including common-mode logic (CML) clock trees and fast-bias circuitry to powerup links quickly. Further, if we leverage more insights from a recently implemented serial link interface that transitions from idle to active well under 10ns [40], we could simply use read or write commands to trigger datapath powerup eliminating the need for DCKE.

Timing, Datarate and Power. Both the MemBlaze Memory Controller and DRAM PHYs were taped out in a 28nm process and the chip was rigorously tested for functionality, correctness and the proposed fast wakeup speed in an industry-strength, serial-links laboratory[19]. In lieu of a DRAM core, the test chip pairs the new PHY blocks with test pattern generation and checking logic for emulating memory read and write transactions. The laboratory operation of the timing is demonstrated in Appendix §A.

The transmit eye diagram at the DQ pins had clear eyes with sufficient timing and voltage margins at 6.4Gbps. The architecture also reduces power in both active-standby and precharge-standby power modes. Compared to today's power modes, this provides the performance of fast-exit powerdown with the DLL-off efficiency.

Specifically, this matches deep power down mode's power at a reduced exit latency of 10ns making it useful for many different applications including emerging ones that have short idle periods. The power difference between the active idle mode and the most efficient powerdown mode is attributed to DLLs, clock tree, and command pins. By powering down these components, MemBlaze lowers power in all other states except during active reads/writes.

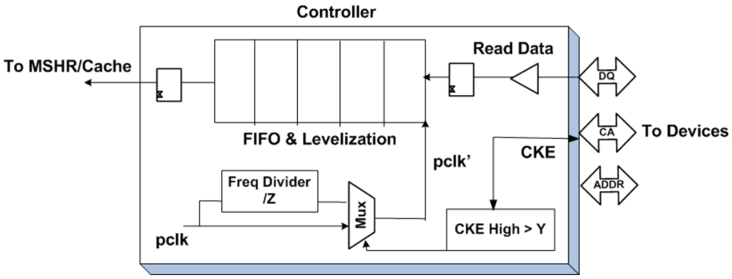

Figure 10: MemDrowsy Architecture.

Although standby efficiency improves, dynamic burst power is largely unchanged. During bursts, DLL power savings are offset by new power costs in current-mode clock circuitry and injection locked-oscillator power costs.

4.3. Imperfect Wake-up Timing

MemBlaze provides an ideal solution and perfect timing upon power-mode wake-up. Alternatively, we propose two new mechanisms in which DRAMs retain their DLLs and turn them on/off very aggressively. Systems that would prefer not to modify the device interface (as MemBlaze does) and can tolerate modest performance degradation will benefit from such techniques. But such DRAMs face synchronization challenges, and we propose mechanisms to mitigate or avoid timing errors. Without changing the controller-device link interface and operating speed these systems retain the DDRx memory links. However, they greatly reduce DRAM idle power by simply changing the sampling rate/decision inside the controller using a few flops.

Reactive Memory Interfaces (MemCorrect). For interfaces that track timing less precisely, we introduce speculative DRAM data transfers immediately after waking up from deep powerdown modes. To guarantee that each memory transaction completes correctly, we architect error detectors. Our fast wake-up memory interface implements this error check on each transaction (Figure 9), which ensures that the clock transition is within a window (±Δ) of its nominal location. This check handles cases when variations and drift during powerdown affect link operation. The error is communicated to the controller through a dedicated pin, *Correct*. If an error occurs, it is because the controller has issued a command too soon after powerup. The controller could simply wait a longer period of time before retrying the command or could send a timing calibrate command to expedite wake-up.

Errors are unlikely in systems with modest voltage and temperature variations. Most memory systems fit this description since boards are designed for tolerance against voltage fluctuations and temperatures vary slowly. Moreover, these variations have a modest effect on timing margins if ranks powerdown for short periods as drift is less likely to have accumulated to affect timing margins.

Drowsy Memory Interfaces (MemDrowsy). Rather than wait for calibration, a controller might begin transfers immediately but mitigate timing errors by halving the data rate for a certain period of time (Y) after wake-up. This slower rate more than doubles the timing margin of the link, greatly improving tolerance to small timing errors induced by VT variations.

Thus, MemDrowsy reduces the effective data rate and relaxes timing precision after wake-up, for reads that need a locked DLL. The

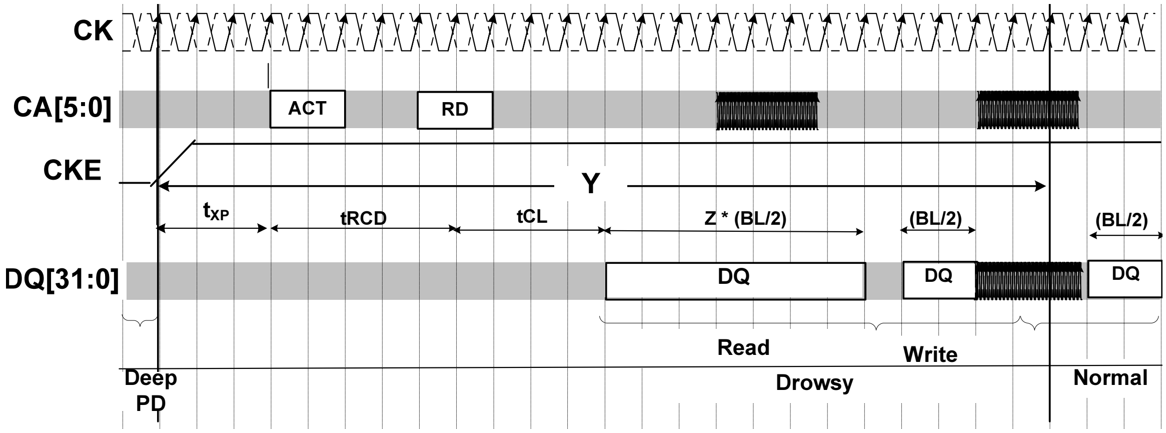

Figure 11: MemDrowsy Timing Diagram.

clock speed is still maintained at the full rate but the link effectively transmits each bit twice. This enables transmitting data while recalibrating and results in lengthening the valid data window. Of course, the controller must also shift the point at which data is sampled. After recalibration, the link operates at nominal data rates.

Figure 10 illustrates extensions to the memory controller. A rank is powered-down simply by disabling the clock (CKE low). Upon powerup, the clock is enabled (CKE high) and a timer starts. The clock operates at the nominal frequency f, providing a sufficient density of clock edges needed to facilitate timing feedback and recovery.

However, given timing uncertainty after wake-up, we use a frequency divider to reduce the rate at which we sample the incoming data; the drowsy rate is f/Z. A multiplexer chooses between sampling at the nominal clock rate or at a divided rate. During drowsy mode, the valid read window is lengthened by a factor of Z as illustrated in Figure 11.

Since the drowsy sampling period is an integral multiple of nominal sampling period, the controller clock is unchanged; it is simply sampled every Z-th cycle.[1] Sampling returns to the nominal frequency only after timing recalibration.

5. Evaluation

We evaluate system implications from two types of memory architectures. The first type, MemBlaze, provides perfect timing and synchronization after wake-up by eliminating expensive interface circuitry from the DRAMs. The second type, exemplified by MemCorrect and MemDrowsy, provides imperfect timing and requires corrective mechanisms.

MemBlaze provides the efficiency of powerdown without performance trade-offs. But MemCorrect and MemDrowsy's mechanisms to ensure correct timing can negatively affect performance. We quantify these effects.

5.1. Experimental Methodology

Simulators. We use an x86_64 execution-driven processor simulator based on a Pin front-end [26, 33]. We use eight out-of-order (OOO) cores at 3GHz matched with Intel's Nehalem microarchitecture and cache latencies as shown in Table 2.

The memory simulator is extended to model three architectures: MemBlaze, MemCorrect, and MemDrowsy. MemBlaze is implemented in silicon and chip measurements are used to configure the simulator. For the other memory architectures, the simulator draws timing estimates from JEDEC specifications and energy estimates from Intel's analyses [5, 16]. In general, DDR3 systems dissipate about 1-1.5W/GB on average [30] and about 2.5W/GB at peak [12]. We validate that our experiments produce numbers in this range. Other memory simulator parameters are described in Table 2.

Processor	Eight 3GHz x86 Out-of-Order cores
L1 cache	private, 8-way 32KB, cycle time = 1, 64B cache-lines
L2 cache	private, 8-way 256KB, cycle time = 7, 64B cache-lines
L3 cache	shared, 16-way 16MB, cycle time = 27, 64B cache-lines
Memory controller	Fast powerdown with threshold timer = 15 mem-cycles [17] Closed-page, FCFS scheduling
Main memory	32GB capacity, 2Gb x4 1333MT/s parts, single ranked 4GB-RDIMMs, four channels, 2DIMMs/channel [5, 16]

Table 2: Baseline System Simulation Parameters.

Classification	**Multi-Programmed (MP) Benchmarks**
High B/W (MP-HB)	433.milc, 436.cactusADM, 450.soplex, 459.GemsFDTD, 462.libquantum, 470.lbm, 471.omnetpp, 482.sphinx3
Med. B/W (MP-MB)	401.bzip2, 403.gcc, 434.zeusmp, 454.calculix, 464.h264ref 473.astar
Low B/W (MP-LB)	435.gromacs, 444.namd, 445.gobmk, 447.dealll, 456.hmmer, 458.sjeng, 465.tonto
Classification	**Multi-Threaded (MT) Benchmarks**
High B/W (MT-HB)	applu, art, canneal, streamcluster, swim, mgrid
Med. B/W (MT-MB)	apsi, blackscholes, equake
Low B/W (MT-LB)	ammp, fluidanimate, wupwise

Table 3: Benchmark Classification.

Workloads. We evaluate memory activity and the proposed architectures on datacenter workloads like memcached. [2] During evaluation, we fast-forward initialization phases and perform accurate simulations during the measurement phase by running for fixed number of instructions across power modes.

In addition, we evaluate a variety of multi-programmed (MP) SPEC CPU2006 as well as multi-threaded (MT) SPEC OMP2001 and PARSEC benchmarks, following prior memory studies [2, 37, 22, 7, 18]. Each core runs a copy of the program/thread depending on the benchmark and the number of application threads or processes are matched to the cores. We fast-forward 10 to 20 billion instructions to skip warm-up and initialization stages and focus on memory behavior in steady state for weighted IPC calculations. We classify MP and MT applications into 3 groups (HB, MB, LB) as shown in Table 3.

Metrics. For each memory architecture, we plot efficiency and performance. Efficiency is measured in energy per bit (mW/Gbps). In this metric, static and background power are amortized over useful data transfers. Performance penalties measure the impact on cycles

[1] MemDrowsy clock rates are unchanged, differentiating it from work in channel frequency scaling [5].

[2] 100b value denoted by *a* and 10KB by *b*

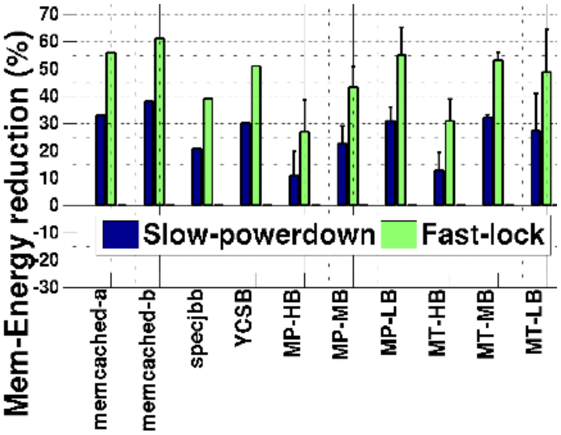

Figure 12: MemBlaze (fast-lock) energy savings relative to DDR3 DRAM baseline (fast-powerdown) and compared against DDR3 DRAM baseline (slow-powerdown).

per instruction (CPI). In each workload group, worst case performance penalty and best case energy savings are plotted on the top of each bar.

Energy savings are measured relative to baseline DDR3 DRAM that aggressively exploits fast-powerdown whenever encountering 15 idle memory cycles [17]. This low threshold gives an optimistic baseline. Realistic, high-performance systems would set the threshold an order of magnitude higher, which would only magnify both active-idle energy costs and our architectures' advantages.

5.2. MemBlaze

MemBlaze efficiency arises from two key features. First, it eliminates DRAMs' DLLs and clocks, thus eliminating long-latency DLL recalibration, which is on the critical path for today's mode exits. Capable of fast exits, MemBlaze can spend more time in powerdown and less time in active-idle.

Second, for any remaining time spent in active idle (i.e., neither bursting data nor in powerdown), MemBlaze consumes very little energy. With MemBlaze links that are capable of fast wake-up, DRAMs' data blocks are powered-on by DCKE precisely when they are needed and no earlier. Only the command blocks remain active, consuming a small fraction of the original active-idle power.

Given these advantages, MemBlaze energy savings are substantial. Even though silicon results indicated feasibility at much larger datarates, we conservatively use DDR3-1333 transfer rate in our simulations to make the comparison fair. Figure 12 compares savings from MemBlaze (fast-lock) and the most efficient power-mode in today's DRAMs (slow-powerdown). When DLLs in today's DRAMs are kept in a quiescent state, slow-power down improves efficiency by 22%. But this efficiency requires a performance trade-off. Exit latency is 24ns, which affects the critical word latency.

MemBlaze fast-lock improves efficiency by 43%. Even memory-intensive applications, like *433.milc* and *471.omnetpp*, dissipate 25-36% less energy. Applications that demand little bandwidth (LB) like *444.namd* consume 63% less energy. With compared against a baseline that uses power-modes more conservatively, these savings would increase by 2×.

Moreover, efficiency comes with better performance than the baseline since MemBlaze power mode exit latency is comparable to that of fast-powerdown in today's DRAMs; neither incur DLL-related wake-up latencies. Fast links powerup the datapath in 10ns. Because this latency is hidden by the command access, we reduce energy with no performance impact.

With attractive energy savings and no delay trade-off, MemBlaze is an order of magnitude better than approaches that aggressively power-off DLLs at run-time or modify the BIOS to disable DLLs at boot-time. These mechanisms all require large performance trade-offs since today's high-performance DRAMs rely on DLLs for timing.

5.3. MemCorrect

While MemBlaze provides perfect timing, other interfaces (including today's DRAMs with DLLs) may be susceptible to timing errors when aggressively exploiting power modes. MemCorrect provides circuitry to detect timing errors, allowing the system to speculate that the timing was correct. We assess performance and energy relative to the DDR3 DRAM baseline in Figure 13.

We evaluate MemCorrect based on the probability p of correct timing. In the best-case, p=100% and timing is never affected when using power-modes. And p=0% is the worst-case in which every wake-up requires a long-latency recalibration. Smaller values for p degrade performance. When p=50%, performance degrades by as much as 100%.

In practice, systems are more likely to encounter correct timing. Boards can be designed with decoupling capacitors to tolerate voltage fluctuations and thermal effects have long time constants. If timing is correct for 99% or 90% of transfers immediately following a wake-up, we incur modest performance penalties of 1% and 10%, respectively.

In exchange for the occasional delay, MemCorrect can exploit power-modes more aggressively. DRAMs with DLLs might bypass recalibration, start transfers immediately after wake-up, and detect errors as they occur. In such a system, MemCorrect energy savings are 38% and 30% when timing is correct for p =99% and p =90% of the transfers. However, if errors are too common, workloads encounter large penalties and low, or even negative, energy savings.

To increase the likelihood of correct timing, we might characterize phase sensitivity to temperature (T) and voltage (V) while the part is operating. During powerdown, we could store the current phase, voltage, and temperature. And changes in T and V during powerdown could be used to calculate a small correction to the last phase. Upon powerup, this correction is added to the phase. We draw lessons from processors in which canaries predict critical path delay across PVT corners and find frequencies that meet timing constraints [9].

5.4. MemDrowsy

If correct timing cannot be ensured at nominal data sampling rates, the system could operate in drowsy mode and reduce its sampling rate by a factor of Z for Y=768ns (tDLLk). In practice Z depends on timing margins at the DRAM interface. $Z = 2$ is realistic because existing LPDDR2 systems eliminate DLLs and transfer at half the data rate to ensure timing. We also assess sensitivity to more conservative margins ($Z = 4$, $Z = 8$).

Reducing the data sample rate more aggressively produces larger penalties in Figure 14(a); latency-sensitive *streamcluster* sees a 32% penalty when $Z = 8$. Less drowsy transfers have far more modest penalties, ranging from 1-4%.

MemDrowsy is also parameterized by how long the DRAM must operate in drowsy mode. In practice, this parameter is defined by the nominal wake-up latency. In other words, transfers are drowsy until the interface can ensure correct timing (e.g., current DRAM DLLs require 768ns). Performance is insensitive to the duration of drowsy operation as only the first few transfers after wake-up are slowed.

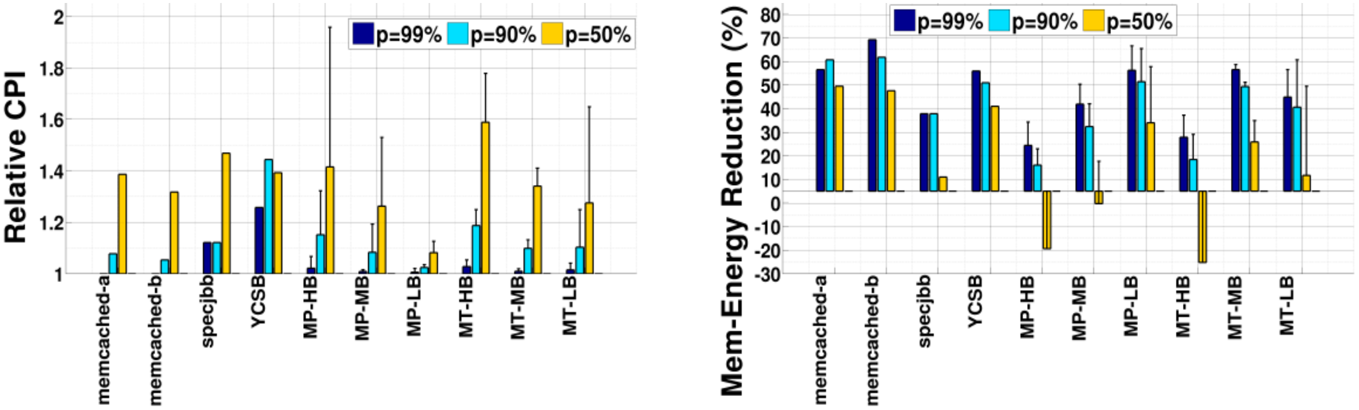

Figure 13: MemCorrect (a) performance as measured in Cycles per Instruction (CPI) and (b) energy relative to DDR3 DRAM baseline. The probability p of correct timing for transfer immediately after wakeup is varied. Plotted for MP, MT, datacenter benchmarks. Error bars represent ranges over mean value in the group.

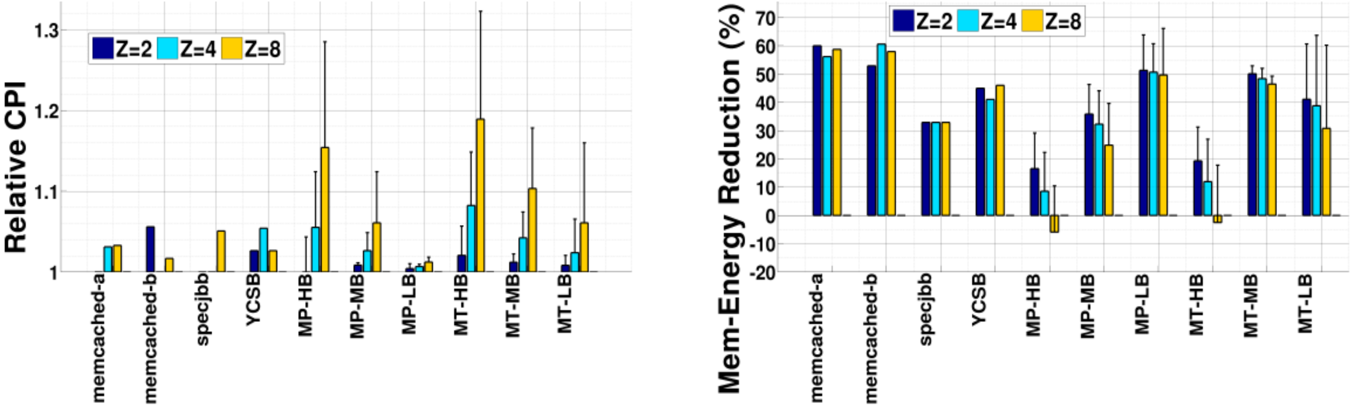

Figure 14: MemDrowsy (a) performance as measured in Cycles per Instruction (CPI) and (b) energy relative to DDR3 DRAM baseline. The drowsy rate reduction factor Z for transfer is varied by using Y=768ns. Plotted for MP, MT, datacenter benchmarks. Error bars represent ranges over mean value in the group.

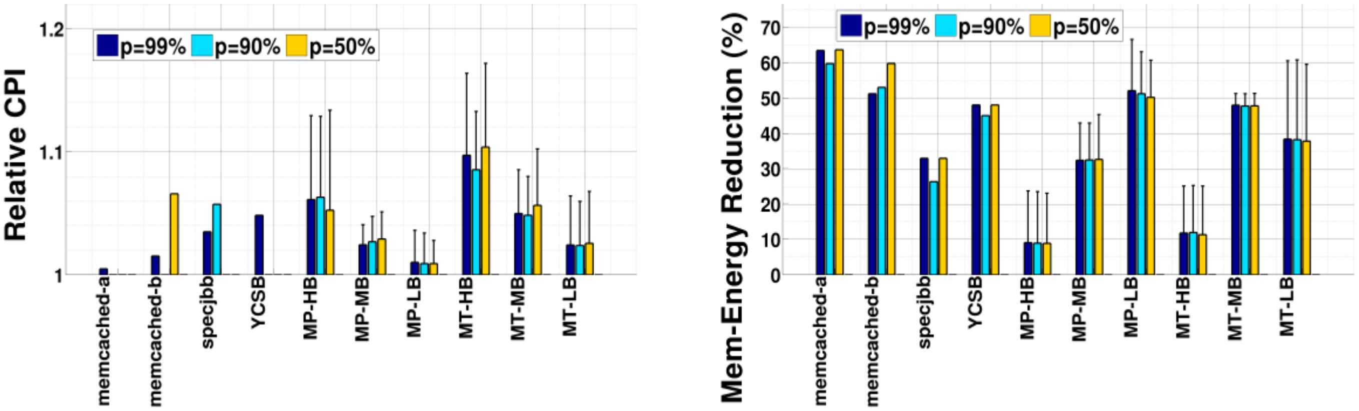

Figure 15: MemCorrect+MemDrowsy (a) performance as measured in Cycles per Instruction (CPI) and (b) energy relative to DDR3 DRAM baseline. The probability p of correct timing for transfer immediately after wakeup with $Z = 4$ is varied. Plotted for MP, MT, datacenter benchmarks. Error bars represent ranges over mean value in the group.

For these modest penalties, MemDrowsy achieves significant energy savings in Figure 14(b). Drowsy transfers allow applications to enter power modes more often with fewer penalties. Clearly, applications that demand memory bandwidth (HB) are more sensitive to drowsy operation. Indeed, average energy per transfer might increase due to larger termination energy from higher bus utilization and also the idle power during the extra cycles.

5.5. MemCorrect and MemDrowsy

Suppose MemCorrect detects a timing error for a transfer immediately following a wake-up. Instead of delaying the transfer for the nominal wake-up latency, the system invokes MemDrowsy and begins the transfer immediately at a slower rate. Clearly, performance and efficiency in MemCorrect+MemDrowsy will be better than either approach applied individually. Immediately after wake-up, transfers begin immediately either at the nominal or reduced data rate.

With MemCorrect+MemDrowsy, exploiting power modes and transferring data immediately after wake-up has performance penalties between 10-20%, as shown in Figure 15(a). In exchange, power modes are more often exploited and energy savings are more consistent.

When implemented alone, MemCorrect energy savings are very sensitive to the probability of correct timing after wake-up. In combination, however, MemCorrect+MemDrowsy is insensitive to timing risk, as shown in Figure 15(b). As MemDrowsy improves memory channel utilization, background power is amortized over more transfers.

MemBlaze promises large energy savings with an architecture that provides perfect timing information. Without such timing guarantees, however, MemCorrect+MemDrowsy provide the next best thing: comparable efficiency and modest (<10%) performance degradation for many applications.

6. Related Work

Much prior work reduces power in conventional server memory. Memory systems can statically set voltage and frequency at boot time, typically in the BIOS [7]. Frequency scaling reduces power but since the static power is amortized over few accesses at low utilizations, the energy per memory access is still expensive. The energy per access could also increase due to higher bus utilization from scaling [5].

Malladi et al. studied LPDDR2 in servers to trade bandwidth for reduced active-idle power [27]. In the current paper, we convert DDR3 active-idle time to time in efficient powerdown without affecting bandwidth. Lim et al. consider various grades of DDR memory [25, 24] while Kgil et al. consider memory-processor stacking [20]. In contrast, we propose changes to power-hungry DRAM interfaces.

Prior work also manages DRAM data placement, increasing access locality and creating opportunities to transition between power states [10, 23, 36, 8, 31, 1]. Prior work studies compiler strategies for cluster accesses by inserting NOPs or reordering to coalesce requests [6]. Also, pages might be redirected to particular DRAM ranks to create hot and cold memory spaces [14]. Memory controllers can throttle requests to manage power [15]. In contrast, our work improves powermode efficiency as multiple studies highlight increasing difficulty of finding usable idle times [7, 29].

We build upon detailed studies of Meisner et al. about subsystem characterization [29], and Ferdman et al. insights on scale out workloads [11]. We study memory bus activity at finer granularities, enabling the analysis and design of DRAM power modes.

Given wide accesses internal to DRAM, chips have been proposed to reduce the number of parts activated. One approach reduces access granularity through separately controlled parts (e.g.,chips, ranks, banks, etc.) to create smaller, independent memory spaces [2, 37, 39, 41]. However, reducing the size of the DRAM activated increases the number of peripheral circuits and degrades density [38].

7. Conclusion

In server memory, idle power can comprise 60-70% of the total. Memory ranks spend 45-60% of their time idle. The spectrum of memory architectures presented in this paper demonstrate new interfaces and architectures that address this problem. By eliminating or mitigating long-latency DLL wake-ups, these systems aggressively uses efficient powerdown states during short idle periods with negligible performance penalty.

Benefits are particularly pronounced for high-capacity, multi-rank systems with frequent idleness. We demonstrate energy savings of up to 68% in a four-rank memory system. While MemBlaze reduces idle power with no performance impact on the system, MemDrowsy and MemCorrect accomplish similar power savings with low penalties. A MemBlaze test chip has also been fabricated and demonstrated to function at a high datarate of 6.4Gbps while the exit latencies and idle power are verified with hardware measurements. Overall, we demonstrate possible techniques to build scalable, energy-proportional memory systems for the future.

8. Acknowledgments

We sincerely thank Yi Lu for helping us with lab measurements, James Tringali, Hongzhong Zheng, Jared Zerbe for their useful discussion. This project is supported in part by a Google Focused Research Award. Krishna Malladi is supported by Benchmark-Capital Stanford Graduate Fellowship. Benjamin Lee is supported in part by NSF grant CCF-1149252.

A. Laboratory Measurements

The fabricated test chip has been tested extensively and Figure 16 demonstrates the timing operation of the powermodes.

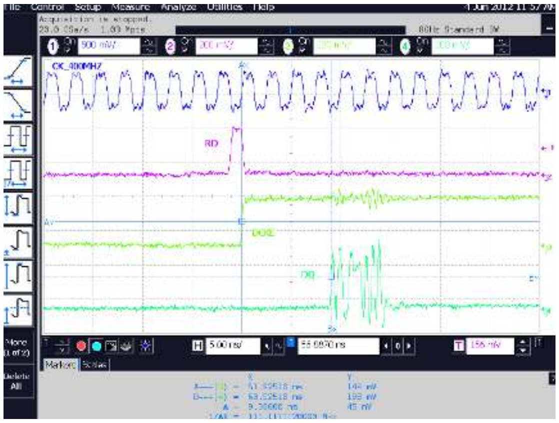

Figure 16: Timing operation for the MemBlaze test chip's fast lock link demonstrating DQ operation within 10ns from powermode wakeup.

References

[1] N. Aggarwal et al. Power-energycient DRAM speculation. In *High Performance Computer Architecture*, 2008.

[2] J. H. Ahn, N. P. Jouppi, C. Kozyrakis, J. Leverich, and R. S. Schreiber. Future scaling of processor-memory interfaces. In *SC*, 2009.

[3] Chih-Kong and K. Yang. Delay-locked loops - an overview. In *Phase Locking in High-Performance Systems. IEEE Press*, 2003.

[4] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking cloud serving systems with YCSB. In *ACM Symposium on Cloud Computing*, 2010.

[5] H. David, O. Mutlu, et al. Memory power management via dynamic voltage/frequency scaling. In *ICAC*, 2011.

[6] V. Delaluz, M. Kandemir, N. Vijaykrishnan, A. Sivasubramaniam, and M. Irwin. DRAM energy management using software & hardware directed power mode control. In *High Performance Computer Architecture*, 2001.

[7] Q. Deng, D. Meisner, L. Ramos, T. F. Wenisch, and R. Bianchini. Memscale: Active low-power modes for main memory. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2011.

[8] B. Diniz, D. Guedes, and R. Bianchini. Limiting the power consumption of main memory. In *International Symposium on Computer Architecture*, 2007.

[9] D. Ernst, N. S. Kim, S. Das, S. Lee, D. Blaauw, T. Austin, T. Mudge, and K. Flautner. Razor: A low-power pipeline based on circuit-level timing speculation. In *International Symposium on Microarchitecture*, 2003.

[10] X. Fan, C. Ellis, and A. Lebeck. Memory controller policies for DRAM power management. In *International Symposium on Low Power Electronics and Design*, 2001.

[11] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi. Clearing the clouds: a study of emerging scale-out workloads on modern hardware. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2012.

[12] Hewlett-Packard. DDR3 memory technology. Technology brief TC100202TB, Hewlett-Packard, 2010.

[13] U. Hoelzle and L. Barroso. *The Datacenter as a Computer*. Morgan and Claypool, 2009.

[14] H. Huang, K. G. Shin, C. Lefurgy, and T. Keller. Improving energy efficiency by making DRAM less randomly accessed. In *International Symposium on Low Power Electronics and Design*, 2005.

[15] I. Hur and C. Lin. A comprehensive approach to DRAM power management. In *High Performance Computer Architecture*, 2008.

[16] Intel. Intel memory 3-sigma power analysis methodology. Data sheet, Intel.

[17] Intel. Intel xeon processor e3-1200 family datasheet. Data sheet, Intel, 2011.

[18] A. Jaleel, K. B. Theobald, S. C. S. Jr, and J. Emer. High performance cache replacement using re-reference interval prediction (RRRIP). In *International Symposium on Computer Architecture*, 2010.

[19] K. Kaviani et al. A 6.4-Gb/s near-ground single-ended transceiver for dual-rank dimm memory interface systems. In *International Solid-State Circuits Conference*, February 2013.

[20] T. Kgil et al. PicoServer: Using 3D stacking technology to enable a compact energy efficient chip multiprocessor. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2006.

[21] J. Kim, M. Horowitz, and G.-Y. Wei. Design of cmos adaptive-bandwidth plls/dlls: A general approach. In *IEEE Transactions on Circuits and Systems-II: Analog and Digital Signal Processing*, 2003.

[22] Y. Kim, M. Papamichael, O. Mutlu, and M. Harchol-Balter. Thread cluster memory scheduling: Exploiting differences in memory access behavior. In *International Symposium on Microarchitecture*, 2010.

[23] A. Lebeck, X. Fan, H. Zeng, , and C. Ellis. Power aware page allocation. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2000.

[24] K. Lim, J. Chang, T. Mudge, P. Ranganathan, S. K. Reinhardt, and T. F. Wenisch. Disaggregated memory for expansion and sharing in blade servers. In *International Symposium on Computer Architecture*, 2009.

[25] K. Lim, P. Ranganathan, J. Chang, C. Patel, T. Mudge, and S. Reinhardt. Understanding and designing new server architectures for emerging warehouse-computing environments. In *International Symposium on Computer Architecture*, 2008.

[26] C. Luk et al. Pin: Building customized program analysis tools with dynamic instrumentation. In *PLDI*, 2005.

[27] K. Malladi, F. Nothaft, K. Periyathambi, B. Lee, C. Kozyrakis, and M. Horowitz. Towards energy-proportional datacenter memory with mobile DRAM. In *International Symposium on Computer Architecture*, 2012.

[28] D. Meisner, B. Gold, and T. Wensich. PowerNap: Eliminating server idle power. In *International Symposium on Computer Architecture*, 2009.

[29] D. Meisner, C. M. Sadler, L. A. Barroso, W.-D. Weber, and T. F. Wenisch. Power management of online data-intensive services. In *International Symposium on Computer Architecture*, 2011.

[30] Micron. Micron 2Gb: x4, x8, x16 DDR3 SDRAM. Data Sheet MT41J128M16HA-125, Micron, 2010.

[31] V. Pandey, W. Jiang, Y. Zhou, and R. Bianchini. DMA-aware memory energy management. In *High Performance Computer Architecture*, 2006.

[32] P. Saab. Scaling memcached at Facebook. *Facebook Engineering Note*, 2008.

[33] D. Sanchez et al. The ZCache: Decoupling ways and associativity. In *International Symposium on Microarchitecture*, 2011.

[34] N. Sharma, S. Barker, D. Irwin, and P. Shenoy. Blink: Managing server clusters on intermittent power. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2011.

[35] K. Shiv et al. SPECjvm2008 performance characterization. In *SPEC Benchmark Workshop on Computer Performance Evaluation and Benchmarking*, 2009.

[36] M. Tolentino et al. Memory MISER: Improving main memory energy efficiency in servers. *IEEE Trans*, 2009.

[37] A. N. Udipi, N. Muralimanohar, N. Chatterjee, R. Balasubramonian, A. Davis, and N. P. Jouppi. Rethinking DRAM design and organization for energy-constrained multi-cores. In *International Symposium on Computer Architecture*, 2010.

[38] T. Vogelsang. Understanding the energy consumption of dynamic random access memories. In *International Symposium on Microarchitecture*, 2010.

[39] F. Ware and C. Hampel. Improving power and data efficiency with threaded memory modules. In *International Conference on Computer Design*, 2006.

[40] J. Zerbe, B. Daly, L. Luo, B. Stonecypher, W. D. Dettloff, J. C. Eble, T. Stone, J. Ren, B. S. Leibowitz, M. Bucher, P. Satarzadeh, Q. Lin, Y. Lu, and R. Kollipara. A 5gb/s link with matched source synchronous and common-mode clocking techniques. In *IEEE Journal of Solid-State Circuits*, 2011.

[41] H. Zheng, J. Lin, Z. Zhang, E. Gorbatov, H. David, and Z. Zhu. Mini-rank: Adaptive DRAM architecture for improving memory power efficiency. In *International Symposium on Microarchitecture*, 2008.