Coordinated Refresh: Energy Efficient Techniques for DRAM Refresh Scheduling

Ishwar Bhati
University of Maryland
ibhati@umd.edu

Zeshan Chishti
Intel Corporation
zeshan.a.chishti@intel.com

Bruce Jacob
University of Maryland
blj@umd.edu

***Abstract*—As the size and speed of DRAM devices increase, the performance and energy overheads due to refresh become more significant. To reduce refresh penalty we propose techniques referred collectively as "Coordinated Refresh", in which scheduling of low power modes and refresh commands are coordinated so that most of the required refreshes are issued when the DRAM device is in the deepest low power Self Refresh (SR) mode. Our approach saves DRAM background power because the peripheral circuitry and clocks are turned off in the SR mode. Our proposed solutions improve DRAM energy efficiency by 10% as compared to baseline, averaged across all the SPEC CPU 2006 benchmarks.**

I. INTRODUCTION

Technology scaling trends have led to a manifold increase in the density and speed of DRAM devices over several technology generations. As DRAM devices become faster and denser, they consume more energy, even when the memory system is not servicing any requests. The increase in device speed leads to higher background power dissipation by the peripheral circuitry, and the increase in device density results in higher refresh energy. For instance, it is projected that refresh operations contribute to as much as 50% DRAM power while simultaneously degrading DRAM throughput by 50% in future 64Gb devices [1]. These trends have caused the memory subsystem to become an major contributor of energy consumption in current and future computing platforms [2].

Commodity DRAM devices employ low power operating modes to reduce the background power consumed by the peripheral circuitry. For example, in the deepest low power Self Refresh (SR) mode, the entire clocked DRAM circuitry is turned off, resulting in no additional power dissipation beyond the power required to refresh the DRAM cells. Many previous papers have proposed intelligent schemes to utilize these low power modes to save DRAM power [3–8]. The key idea behind these schemes is to switch a DRAM rank to a lower power mode whenever the rank stays idle for a time period longer than a pre-determined threshold.

While idle period tracking was originally proposed for leveraging low power modes, idle periods can also be used for intelligent scheduling of refresh operations. For instance, to mitigate the impact of DRAM refreshes on performance, Stuecheli et al. proposed *Elastic Refresh* [9], which postpones up to eight refresh commands for a busy DRAM rank and then issues those pending refresh requests when that rank becomes idle.

Even though idle period tracking can be leveraged to implement both intelligent low power mode switching and intelligent refresh scheduling, we observe that these two sets of techniques are in conflict with each other and often render each other ineffective. For example, if a memory controller using Elastic Refresh issues a batch of pending refresh commands immediately after the DRAM becomes idle, then the DRAM would need to be kept in the highest power active mode until all the pending refreshes have been completed, thereby limiting the effectiveness of low power mode switching. Conversely, if the rank is immediately switched to SR mode upon becoming idle, then Elastic Refresh would be unable to service any pending refreshes, thereby rendering Elastic Refresh scheme ineffective. The main reason for the interference between intelligent refresh scheduling and low power mode switching is that these mechanisms work in isolation with each other.

In this paper, we make the novel observation that coordinating the operation of these two mechanisms can improve both the performance and energy efficiency of the DRAM subsystem. We propose a new set of techniques, collectively referred to as "Coordinated Refresh". The key idea behind these techniques is to coordinate the scheduling of low power mode transitions and refresh commands such that most of the required refreshes are scheduled when the DRAM rank is in the lowest power SR mode.

Our two techniques, *Coordinated FAST refreshes in SR (CO-FAST)* and *Coordinated FLUSH refreshes in SR (CO-FLUSH)*, utilize the full flexibility of refresh scheduling by postponing refreshes when the memory is busy and servicing them during periods of idleness. The key difference between our techniques and Elastic Refresh is as follows: instead of the memory controller issuing all the pending refresh commands, the coordinated techniques first transition DRAM to the SR mode and then service the pending refreshes in the SR mode, thereby saving background power *and* mitigating the impact of refresh on performance *at the same time*. CO-FAST satisfies the timing constraints for pending refreshes by doubling the refresh rate during SR mode, whereas CO-FLUSH simply flushes all the pending refreshes immediately upon entering the SR mode.

While operating in SR mode saves DRAM background power, there is a performance cost associated with the latency of switching back to active mode. Therefore, frequent transitions between SR and active modes could degrade performance and energy efficiency. Thus, the effective use of SR mode requires accurate and quick detection of long idle periods as well as the capability to issue more refreshes in SR mode. To that end, we add two more optimization techniques. First, we augment the history based prediction scheme proposed in [3], which tracks the length of previous idle periods to *accurately predict* the length of current idle period. We use this prediction to guide the thresholds for switching to low power modes in our coordinated refresh techniques. Second, we utilize the advance refresh option, which issues multiple refresh operations ahead of time during an idle period, so that the latency penalty of these refreshes during the subsequent active period is avoided. We enhance the effectiveness of CO-FAST and CO-FLUSH by using advance refresh, in addition to the pending refreshes used in Elastic Refresh.

The key contributions of this work are as follows:

- This is the first paper that addresses the need for coordinating the scheduling of low power mode transitions and refresh operations during idle DRAM periods
- We propose CO-FAST and CO-FLUSH, two novel techniques together referred as *Coordinated Refresh*, which save DRAM background power by carrying out most of the refreshes during the lowest power SR mode.
- Our proposed solutions improve the DRAM energy efficiency by 10% on average (up to 25%), as compared to baseline technique across the entire SPEC CPU 2006 benchmark suite.

II. BACKGROUND AND MOTIVATION

A. DRAM Power Consumption

DRAM power consumption can be divided into three categories (1) active power, (2) background power, and (3) refresh power. Active power represents the energy required to activate and pre-charge the rows, and to service read and write requests, including I/O transfers. Active power is consumed only when the DRAM is servicing memory requests. Background power, on the other hand, consists of the energy consumed by the peripheral circuitry, irrespective of whether the DRAM is servicing requests or is sitting idle. Finally, since DRAM cells lose charge over time, they are required to be refreshed periodically and thereby dissipating refresh power.

Background power is reduced substantially by switching to a low power mode. Current DRAM devices have the following three operational modes: (1) Active, (2) Power-Down (PD), and (3) Self Refresh (SR). Active mode is the normal operating mode in which the rank can immediately service requests. In the PD mode, some I/O signals and peripheral logic is disabled, resulting in lower power consumption.

In SR mode, the entire DRAM clocked circuitry and the DLL are turned off. Therefore, no power is consumed except by refresh operations, which are triggered internally by a built-in timer. The DDR3 switching time from SR to active mode is specified as the maximum of the following two parameters: (i) t_{RFC}: the time required to service a refresh command, (ii) t_{DLLK}: the DLL lock period. t_{RFC} increases with the size of the DRAM device; whereas t_{DLLK} remains constant (e.g. 512 clock cycles in DDR3 devices) irrespective of device size and speed.

B. Refresh Penalty

In DDR devices, scheduling of refresh operations is dictated by two timing parameters. The first parameter, t_{RFC}, represents the time required to complete one refresh operation, and the second parameter, t_{REFI}, specifies the average time period between two refresh operations. The value of t_{RFC} depends upon the number of rows refreshed with one refresh operation, whereas t_{REFI} depends on t_{RFC} and the total number of rows to be refreshed. As device density increases, we either have to refresh more rows per refresh operation (increase t_{RFC}) or service refreshes more frequently (decrease t_{REFI}). DDR3 devices are specified to keep t_{REFI} constant at 7.8µs. Consequently, t_{RFC} increases with increasing device density.

When the memory controller issues a refresh command (also called *Auto-refresh*) to a rank, each device in that rank simultaneously starts to refresh; therefore the entire rank becomes unavailable to service any memory requests for t_{RFC} period. Furthermore, Auto-refresh commands can be issued only when the rank is in active mode. If the rank happens to be in PD mode, the memory controller must first transition it to the active mode, and then schedule an Auto-refresh command. Consequently, while servicing Auto-refreshes, DRAM devices not only consume refresh power but also high background power.

C. Prior Art

The most prevalent refresh approach in current-day memory controllers is Demand Refresh (DR), in which an Auto-refresh command is issued immediately after every t_{REFI} time period (shown in Figure 1(a)). However, DR does not address the increasing refresh penalty in high density devices. Recently proposed Elastic Refresh [9] postpones up to eight refresh commands during a high memory activity phase, and then compensates by servicing those pending refreshes during a subsequent idle memory phase (shown in Figure 1(b)). To satisfy the average refresh rate constraint specified by t_{REFI}, pending refreshes have to be issued at a rate faster than $1/t_{REFI}$. The *Elastic* memory controller satisfies this constraint by adjusting the Auto-refresh command issue rate based on the number of pending refreshes. If the number of pending refreshes is high, Auto-refresh commands are issued at a faster rate, and vice versa. Therefore, by scheduling most of the refreshes during idle periods, *Elastic* can mitigate the performance impact of refreshes. However, since Auto-refresh commands require the DRAM to stay in active mode, which consumes more background power, *Elastic* mitigates only the performance impact of refreshes and does not address the background power consumption during refresh operations.

D. Taking Advantage of SR Mode

When a DRAM rank is in the SR mode, the memory controller does not need to issue any external Auto-refresh commands as the device internally issues refreshes. Since all the clocked circuitry during the SR mode is turned off, background power is reduced when refresh is issued internally in SR mode. Table 1 shows the currents drawn during refresh when DRAM is in active mode versus in the SR mode for Micron's 4Gb DDR3 devices running at different speed grades [10]. The last row in the table is for 3200Mbps bandwidth devices, which corresponds to upcoming DDR4 devices. The parameter values for 3200Mbps bandwidth are extrapolated from current DDR3 device trends.

The current drawn during Auto-refresh command (I_{DD5B}), increase with clock speed for the same DRAM device size. This is mainly due to the clocked peripheral circuitry, which consumes more power at higher clock speeds. In contrast, I_{DD6}, which is the current drawn during SR mode remains constant for same density device, even for device with higher speed operations. This is because, in SR mode, the external clock is disabled, and the refresh is generated by a built-in timer. Furthermore, I_{DD6ET} is the current drawn when the refresh rate in SR mode is doubled, which is intended for DRAM cells to operate in the higher extended temperature range. The difference between I_{DD6} and I_{DD6ET} represents the average current drawn by a refresh command when it is issued internally in SR mode. This value remains constant (6 mA) for all speed grades of same density DRAMs.

The last column in Table 1 shows the power savings achieved during refresh operations by serving refreshes in SR mode instead of through Auto-refresh commands in active mode. For instance, in 4Gb devices running at 1333Mbps, 26% of the power is saved by issuing a refresh command in SR mode. Further, this savings increases to 50% in 3200Mbps devices. For devices with density 8Gb and higher, these power savings will be more substantial, since refresh operations will take longer and the overall contribution of refresh energy to the total memory system energy would become more significant.

Table 1: Refresh currents in 4Gb DRAMs. Avg. Auto-refresh current in second last column is calculated as "$I_{DD5B}*(t_{RFC}/t_{REFI})$".

Speed	I_{DD5B} (mA)	I_{DD6} (mA)	I_{DD6ET} (mA)	t_{RFC} (Cycles)	t_{DLLK} (Cycles)	Refresh Current in SR (mA)	Avg. Auto-refresh Current (mA)	Savings: SR vs Auto-refresh
DDR3-1333	210	22	28	200	512	6	8.07	26%
DDR3-1600	220	22	28	240	512	6	8.46	29%
DDR3-1866	230	22	28	280	512	6	8.84	32%
DDR4-3200	300	22	28	480	512	6	11.53	48%

III. COORDINATED REFRESH

The key towards reducing the background power consumption during refresh operations is to coordinate the scheduling of low power mode transitions and refresh commands in such a way that most of the required refresh operations are scheduled when the DRAM rank is in the SR mode. Furthermore, this rescheduling of refresh operations must not violate retention time constraints for DRAM cells. Below, we present two techniques, collectively referred to as "Coordinated Refresh", which achieve these goals:

A. *Coordinated Fast Refreshes in SR (CO-FAST)*

In current DDR3 devices, there is an option to double the refresh rate in SR mode [11]. This is configured by a mode register, which could be changed any time before switching to SR mode. When the faster rate is enabled, one refresh command is scheduled internally every 3.9µs rather than the usual 7.8µs (t_{REFI}) period. This option is provided for DRAM to work in the extended high temperature range. However, we observe that one can also use this option in the regular temperature range to artificially increase the refresh rate. Our first technique, called Coordinated Fast Refreshes in SR mode (CO-FAST), leverages this option to service more refreshes in the SR mode, thereby reducing the number of refreshes issued in the active mode.

Figure 1(c) explains the workings of CO-FAST. When a DRAM rank is busy, CO-FAST postpones any periodic refresh commands (up to a maximum of eight refreshes) and waits till the next idle period opportunity to issue extra refreshes to compensate for pending ones. The key difference between CO-FAST and *Elastic* is that unlike *Elastic*, CO-FAST attempts to coordinate the scheduling of pending refreshes with low power mode transitions. Specifically, for long idle periods, CO-FAST switches to SR mode *before* servicing the pending refreshes. Furthermore, for long enough idle periods, CO-FAST issues up to eight advance refreshes (According to JEDEC standard for DDR3 device [11], up to eight refresh commands can be either issued in advance or can be postponed). The issuance of advance refreshes is based on the prediction that in the next active phase, this rank will receive high memory traffic, and carrying out some of the refreshes in advance could avoid the latency penalty of refresh commands. However, in case of short idle periods, CO-FAST falls back to an approach similar to *Elastic*, where refreshes are flushed in the active mode in short idle periods, so that the performance penalties of switching from SR to active mode are avoided. Finally, in the worst case, when there are no idle periods at all, refreshes are issued like the demand refresh scheme, since pending refresh count will reach to its maximum of eight.

A scenario could arise in which CO-FAST may switch to SR mode with a faster refresh rate, and the idle period may prolong to the extent where all the pending refreshes and the maximum allowed advance refresh commands have already been issued. In such a scenario, the rank refresh rate needs to be reduced to its usual 7.8µs value in order to avoid the energy overhead of faster refresh rate. To enable this change, the rank is first transitioned to active mode, the mode register is re-written to decrease the refresh rate, and then the rank is switched back to SR mode.

The main advantage of CO-FAST is that it does not require any change to the DDR3 device. However this advantage also becomes a limitation, since the maximum increase in refresh rate during the SR mode cannot be more than 2x of the usual refresh rate. Consequently, for short idle periods, CO-FAST can only issue a small number of extra refreshes (for example, one extra refresh during a 7.8µs idle period). To mitigate this limitation, the DRAM device must provision for higher refresh rates beyond 2x of usual refresh rates during the SR mode.

B. *Coordinated Flush Refreshes in SR (CO-FLUSH)*

In order to transition a DRAM rank into the SR mode, the memory controller issues a "self-refresh" command. In existing DDR3 devices, the self-refresh command does not need any other attribute, since the DRAM rank internally tracks the address of the next row to be refreshed and uses an internal timer to schedule the required refreshes. Our second technique, Coordinated Flush refreshes in SR mode (CO-FLUSH), requires a minor modification in the DRAM device, wherein a specified number of refreshes could be flushed (initiated as a batch), just after switching to the SR mode. To make this modification, we introduce a new command called "self-refresh-flush", wherein a few of the address bits would be used to specify the number of immediate refreshes to be initiated. After entering SR mode, the device would first finish these many refreshes as shown in Figure 1(d), and then only it would resume the normal refresh rate.

With this small change in the DRAM device, CO-FLUSH can flush many refresh commands in SR mode, which otherwise would have been issued in active mode. This change enables CO-FLUSH to be more effective than CO-FAST in situations where the idle periods are too short such that the simpler approach of doubling the refresh rate is insufficient to issue extra refreshes.

A scenario could arise where the memory controller may transition the DRAM device from SR mode to the active mode before all the immediate refresh commands have been issued. In such a scenario, the memory controller must account for the remaining refreshes and service them in the active mode. This functionality can be implemented by adding a timer to track the

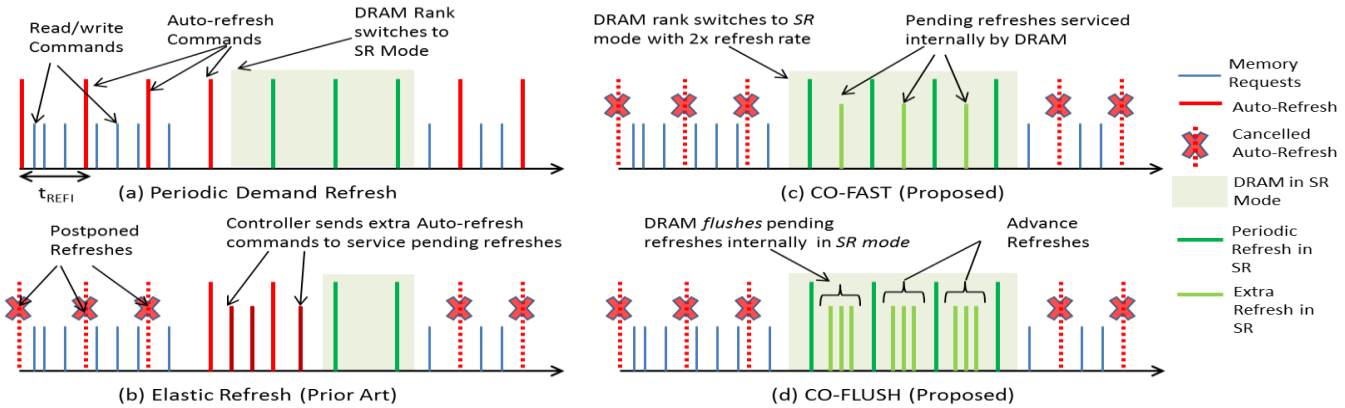

Figure 1: An illustration of prior art and proposed Coordinated Refresh techniques. *Elastic* (b) postpones refreshes during high memory activity and schedules Auto-refresh when device is idle. However in coordinated techniques (c & d), first the device is switched to self refresh (SR) mode and then only extra refreshes are serviced. Therefore, background energy is saved in proposed schemes.

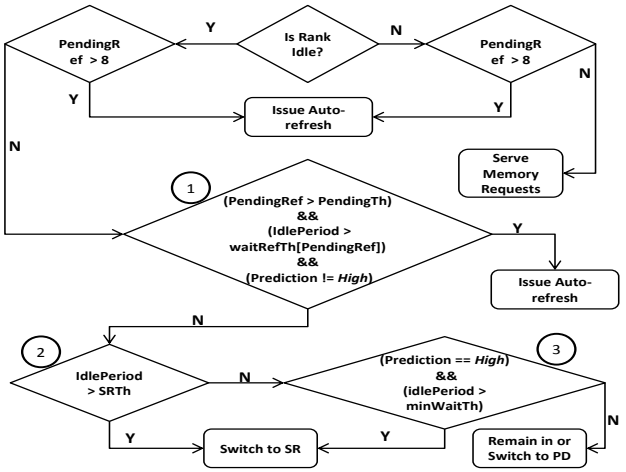

Figure 2: Implementation details of Coordinated Refresh integrated with idle period predictions.

number of cycles in SR mode. Based on the timer value, memory controller would decide the number of unfinished refreshes. Furthermore, an extra 3 bit counter is required in the DRAM device, which stores the number of refreshes to be initiated immediately in SR mode. This counter is decremented for each refresh issued, and reset when the SR mode exits.

Similar to CO-FAST, CO-FLUSH postpones refreshes in a high activity phase and then finds the appropriate idle period for switching to SR mode, wherein those pending refreshes are internally serviced by the DRAM. Also, like CO-FAST, CO-FLUSH may schedule some refreshes in advance depending on the length of the idle period. Since CO-FLUSH could use smaller gaps to flush extra refresh commands; it needs smaller threshold values to switch to SR mode if there is scope for issuing extra refreshes. Consequently, short gaps in activity could sometimes be utilized to transition into SR mode quickly and flushing extra refreshes.

C. Implementation Details

A simple history-based prediction (HBP) has been proposed by Delaluz et al. [3] to predict the next inter-access time based on the previous inter-access time value. We observe that HBP's approach of relying *only* on *one* previous idle period length makes it incapable of capturing common patterns present in many programs, like alternating low and high activity phases.

We instead implemented a more sophisticated prediction mechanism, which categorizes previous idle periods in ranges, based on period lengths. The number of previous idle periods stored for history (n) and the number of levels used for idle period ranges (m) are controlled by configurable parameters for a DRAM rank. In our simulations, we used *m*=3 which categorized idle period lengths as: Low (0 to 0.67*t_{REFI}), Medium (0.67*t_{REFI} to 1.5*t_{REFI}) and High (longer than 1.5*t_{REFI}). We found that storing three previous idle periods (n= 3) was sufficient to predict stable as well as alternating memory patterns with an overall accuracy of 84% in the simulated workloads. In our predictor, alternating idle period pattern is captured when a sequence of {Low, High, and Low} is seen, while a stable long idle period is predicted after observing previous two High periods.

Figure 2 shows the implementation details of coordinated techniques when integrated with our prediction mechanism. When a rank becomes idle, the memory controller first checks the pending refresh count. If the pending refresh count exceeds eight then the memory controller immediately issues an Auto-refresh. Otherwise, an Auto-refresh command is issued only if all the following three criterions have been satisfied (①in Figure 2): First, the number of pending refreshes has exceeded a threshold (*PendingTh*); we set *PendingTh* to 4 in CO-FAST and 5 in CO-FLUSH. Second, the rank has been idle for more than a threshold (*waitRefTh*); we set this threshold as a function of the number of pending refreshes. Third, this idle period is predicted as a short idle period. Together, these three criteria enable our techniques to be conservative in scheduling Auto-Refresh commands for servicing pending refreshes.

Both CO-FAST and CO-FLUSH switch to SR mode under two scenarios: (a) Regular switching: If the idle period exceeds a threshold (*SRTh*), we switch to SR mode, irrespective of the prediction made by HMAP (②in Figure 2), (b) Eager switching: If prediction is a *long* idle period, then we wait for a much shorter threshold (*minWaitTh*) before switching to SR-mode (③ in Figure 2). CO-FAST characterizes only High idle period predictions as *long* idle periods, whereas CO-FLUSH characterizes both Medium and High predictions as *long* idle periods. We have experimented with different values of switching thresholds, and found that, *SRTh* = t_{REFI} and *minWaitTh* = 2 * t_{RFC} works best for our techniques. Once in SR mode, both CO-FAST and CO-FLUSH issue advance refreshes if idle periods are long enough and all the pending refreshes have been serviced.

Table 2: Composition of heterogeneous workload mixes

workload	benchmarks	workload	benchmarks	workload	Benchmarks
low_1	povray,h264ref, namd,calculix	med_1	gcc,milc,astar, cactusADM	high_1	mcf,libquantum, lbm,leslie3d
low_2	gamess,hmmer, h264ref,dealII	med_2	milc,gromacs, wrf,sjeng	high_1	GemsFDTD,mcf, lbm, libquantum
low_3	povray,namd, calculix,tonto	med_3	gobmk,sjeng, sphinx3, leslie3d	high_1	omnetpp,leslie3d, GemsFDTD,libquantum
low_4	h264ref,gamess , namd,povray	med_4	soplex,hmmer, bwaves,cactusADM	high_1	mcf,omnetpp, leslie3d,lbm
mix_1	namd,h264ref, gobmk,mcf	mix_2	hmmer,GemsFDTD, gamess,sjeng	mix_3	GemsFDTD,libquantum, gromacs,namd

IV. SIMULATION METHODOLOGY

To evaluate our proposed techniques, we use MARSSx86 [12], a full-system x86 simulator, configured as shown in Table 3 for single and multi-core experiments. We integrate MARSSx86 with a modified version of DRAMSim2 [13] to model refresh and low power mode timings, compliant with DDR3 standard. The DRAM parameters used in our simulations are listed in Table 3. We calculate DRAM energy from the device's I_{DD} numbers, using the methodology described in [14]. For refresh timing parameters (t_{REFI} and t_{RFC}), we use the same values as those used in [9].

Table 3: CPU and memory parameter settings used in the simulations

	Single Core	Multi-core
Processor	2 GHz, out-of-order, 4-issue per core	4 cores, 2 GHz, out-of-order, 4-issue per core
L2 Cache	2MB, 8-way, 64B Block Size, 5 cycles latency	Shared, 8MB, 8-way, 64B Block Size, 8 cycles latency
Main Memory	1 Channel, 64 bit width, 8GB, 2 Ranks	2 Channels, 2 Ranks per channel, 16GB, 64 bit width
L1 Cache (per core)	128 KB, 8-way associativity, 64B Block Size, 2 cycle latency	
Memory controller	Open row closes after 4 access or queue is empty, FR-FCFS, "row:bank:rank:channel:column" mapping, 64-entry queue	
DRAM devices	8Gb, x16, speed 3200Mbps, t_{RP}=15ns, t_{RCD}=15ns, t_{RFC}=550ns, t_{REFI} = 7.8µs,	
I_{DD} Currents (mA)	I_{DD0}=150, I_{DD2P0}=35, I_{DD2P1}=71, I_{DD3N}=113, I_{DD5B}=360, I_{DD6}=35, I_{DD6ET}=45	

We use the SPEC CPU2006 benchmark suite [15] for both single- and multi-core experiments. For single core runs, each program executes 1 billion instructions in its region-of-interest (RoI) determined using SimPoint 3.0 [16]. We characterize programs into three categories based on their main memory bandwidth requirements: (1) LOW (< 100MBps), (2) MEDIUM (>100MBps and < 1500MBps) and (3) HIGH (> 1500 MBps). For our multi-core runs, we simulate a total of 1 billion instructions, where each program starts from its RoI. We use total instructions-per cycle (IPC) for performance results. Further, we construct heterogeneous multi-core workload mixes as shown in Table 2.

Our baseline scheme uses fixed switching thresholds to transition idle DRAM ranks into low power modes. A rank switches to PD slow exit immediately after the request queue for that rank becomes empty, as proposed in [17]. If the rank remains idle for a time period equal to t_{REFI}, then the rank switches to SR mode.

We compare our techniques against Elastic Refresh. Note that the *Elastic* implementation in [9] does not employ any low power modes. Our evaluation showed that such an implementation consumes on average more than twice the energy as compared to the baseline. To enable a fair comparison of our techniques against *Elastic*, we implemented a modified version of elastic refresh, which switches idle ranks to low power modes based on the same thresholds as the baseline. Our experiments show that this modified *Elastic* implementation performs only 3% slower on average, compared to *Elastic* without using low power modes, while consuming less than half of the DRAM energy. We use this modified *Elastic* scheme in all our evaluations.

V. RESULTS

In this section, we present the energy and performance results of single and multi-core systems using the near-future DRAM devices of 8Gb density and 3.2Gbps speeds.

A. Single Core Evaluations

Figure 3 shows the energy reduction and performance improvement for Coordinated and *Elastic* techniques normalized to the baseline in a single-core CPU. We arrange benchmarks from LOW to HIGH categories and show average results in the rightmost set of bars.

Both CO-FAST and CO-FLUSH achieve significant energy savings and performance improvements, in particular for the MEDIUM and HIGH categories. CO-FAST reduces DRAM energy by up to 17% and increases performance by up to 13%, whereas CO-FLUSH provides up to 25% energy reduction and up to 14% IPC improvements.

Table 4: Percentage of Refresh operations scheduled in SR

Technique	% of Refresh in SR mode for each category		
	LOW	MEDIUM	HIGH
Baseline	97.3	40.4	8.1
Elastic	97.3	40.2	7.8
CO-FAST	99.5	58.7	13.9
CO-FLUSH	99.6	66.6	24

The energy reductions achieved by the coordinated techniques are primarily due to the higher fraction of refreshes serviced in the SR mode. To quantify this benefit, Table 4 shows the percentage (over the total number of refreshes) of refreshes issued during the SR mode for different techniques. In the LOW category, the baseline already issues most of the refreshes (97%) in SR mode; therefore coordinated techniques do not provide substantial extra benefit. However, in the MEDIUM category, which contains 15 of our 29 benchmarks, coordinated techniques are particularly effective in increasing the percentage of SR mode refreshes from 40% in the baseline to 59% and 67% for CO-FAST and CO-FLUSH, respectively. Consequently, in the MEDIUM category, CO-FAST and CO-FLUSH reduce DRAM energy on average by 10% and 13% as compared to the baseline, and 9% and 12% as compared to *Elastic*, respectively. For the HIGH category, most of the refreshes have to be issued in the active mode due to the shorter idle periods. In these programs, coordinated and *Elastic* technique provide similar performance improvements as compared to the baseline.

B. Multi-core Evaluations

For multi-core experiments, we use two types of workloads: (i) SPECRate-type homogeneous workloads (ii) heterogeneous workloads composed of program mixes from Table 2.

Figure 4 shows the energy and performance results for our multi-core workloads, when using Coordinated and *Elastic* techniques. For homogeneous workloads, we show only average results for each category in the interest of space. Compared with the baseline, *Elastic*, CO-FAST and CO-FLUSH achieve energy reductions of 2.0%, 8.2% and 10.1%, and performance improvements of 3.7%, 3.5% and 3.5% respectively, over all the workloads

Most of the trends observed in the single program workloads (Section V.A) repeat in the multi-core scenarios. In homogeneous multi-core workloads, coordinated techniques provide higher energy benefits in MEDIUM and HIGH workload categories. The results for heterogeneous workload mixes demonstrate significant benefits for coordinated techniques, even if they have fairly random memory request patterns generated by characteristics of constituent programs.

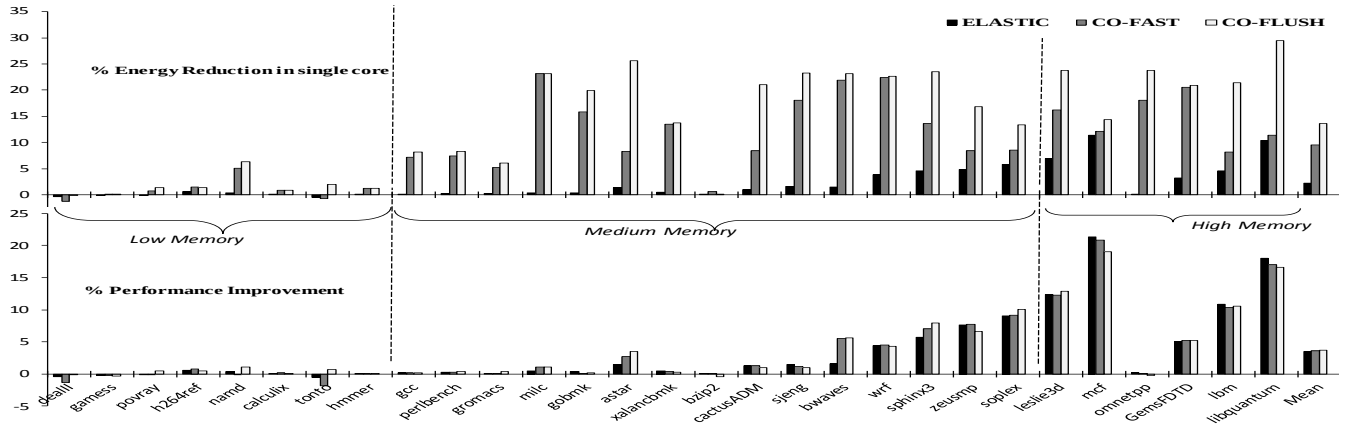

Figure 3: DRAM energy and performance improvements for our proposed coordinated techniques in 8 Gb devices. X-axis common

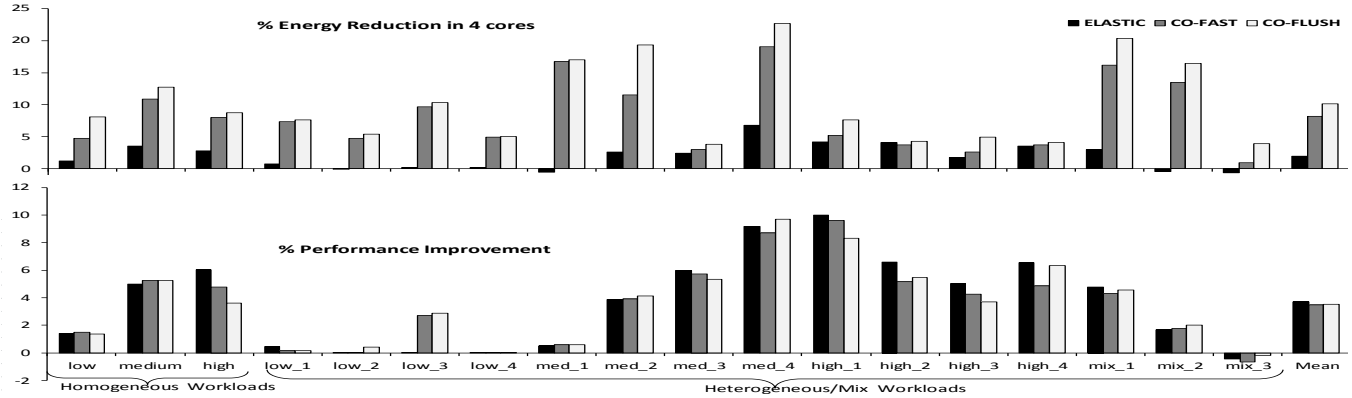

Figure 4: DRAM energy savings and Performance improvements in 4 cores. Heterogeneous workloads composition shown in Table 2

C. *High Speed Device Termination*

Multi-DIMM DDR3 memory systems have signal integrity challenges at high speeds due to interference between multiple DRAM ranks sharing the same multi-drop bus. In addition, when using low power modes (slow exit PD or SR), DRAM ranks turn-off their on-die termination (ODT), worsening the interference from non-target ranks in a multi-rank system. We validated our techniques in a *single*-rank-per channel configuration. For this configuration, CO-FLUSH achieves 9% memory energy savings and 3% IPC improvement across MEDIUM and HIGH categories, relative to the baseline.

VI. RELATED WORK

Early works on reducing DRAM background power propose hardware and software policies for switching to low power DRAM modes [3–6], [18]. Our techniques further reduce DRAM background by servicing most of the required refreshes in the lowest power mode.

Huang et al. [7] observed that the PD mode switching can be done immediately when the request queue for a rank is empty. Delaluz et al. [3] proposed a simple history based predictor (HBP) for switching thresholds based on the length of previous idle interval on that device.

Recent work in [13, 19, 20] have either throttled or reshaped the main memory traffic to create longer idle periods, thereby increasing the opportunity to switch to low power modes. Bi et al. [20] use the file I/O and system calls information to predict the DRAM activity for memory used as buffer cache. Our proposed techniques do not actively reshape or throttle the memory requests; therefore these techniques are complementary and can co-exist.

Flikker [21], RAPID [22], and RAIDR [1] techniques use the information about variability in DRAM cell retention times to reduce the required refresh operations. Our approach of intelligently coordinating DRAM refreshes and low power mode transitions is orthogonal to these schemes.

VII. CONCLUSIONS

In order to satisfy the ever-increasing memory capacity and performance requirements for computer systems, the speed and density of DRAM devices has increased in successive technology generations. These trends have resulted in two main scalability concerns relating to the energy efficiency of future DRAM subsystems, namely, the increase in background power consumption of the DRAM peripheral circuitry and the growing performance and energy penalties of DRAM refresh operations. To address these concerns, we have proposed a set of novel techniques, called coordinated refresh. Our techniques are based on the key idea that coordinating the scheduling of low power mode transitions and refresh operations during idle memory periods can provide both energy savings and mitigate the performance penalties of refresh operations. Our proposed techniques increase DRAM energy efficiency by 8% as compared to the state-of-the-art *Elastic* technique (10% compared to baseline), averaged across all the SPEC 2006 programs. As energy efficiency quickly becomes a key design constraint, techniques like coordinated refresh will become a key driver for energy-efficient operation of future computer systems.

References

[1] J. Liu, et al. “RAIDR: Retention-aware intelligent DRAM refresh,” in *ISCA*, Jun. 2012.
[2] L. Minas and B. Ellison, “The problem of power consumption in servers,” Intel Press Report, 2009.
[3] V. Delaluz, et al.“DRAM energy management using software and hardware directed power mode control,” in *HPCA*, 2001.
[4] X. Fan, et al. “Memory controller policies for DRAM power management,” in *ISLPED*, 2001.
[5] V. Delaluz, et al.“Scheduler-based DRAM energy management,” in *Design Automation Conference*, 2002.
[6] V. Pandey and R. Bianchini, “DMA-Aware Memory Energy Management,” *in HPCA*, 2006.
[7] H. Huang, et al. “Improving energy efficiency by making DRAM less randomly accessed,” in *ISLPED*, 2005.
[8] B. Diniz, et al. “Limiting the power consumption of main memory,” in *ISCA*, 2007.
[9] J. Stuecheli, et al.“Elastic Refresh: Techniques to Mitigate Refresh Penalties in High Density Memory,” *in MICRO*, 2010.
[10] Micron Technology, “4Gb DDR3 SDRAM Datasheet,” 2009.
[11] JEDEC, “JEDEC DDR3 Standard,” 2010.
[12] A. Patel, et al. “MARSS: a full system simulator for multicore x86 CPUs,” in *DAC*, 2011.
[13] P. Rosenfeld, et al. “DRAMSim2: A Cycle Accurate Memory System Simulator,” *Computer Architecture Letters*, 2011.
[14] Micron “Calculating Memory System Power for DDR3,” 2007.
[15] J. L. Henning, “SPEC CPU2006 benchmark descriptions,” *SIGARCH Comput. Archit. News*.
[16] G. Hamerly, et al.“Simpoint 3.0: Faster and more flexible program phase analysis,” in *JILP*, vol. 7, no. 4, pp. 1–28, 2005.
[17] I. Hur and C. Lin, “A comprehensive approach to DRAM power management,”*in HPCA*, 2008.
[18] A. R. Lebeck et al. “Power Aware Page Allocation” in *In ASPLOS*, 2000.
[19] A. M. Amin and Z. A. Chishti, “Rank-Aware Cache Replacement and Write Buffering to Improve DRAM Energy Efficiency,” *ISLPED*, 2010.
[20] M. Bi, et al.“Delay-Hiding energy management mechanisms for DRAM,” in *HPCA*, 2010.
[21] S. Liu, et al.“Flikker: saving DRAM refresh-power through critical data partitioning,” in *ASPLOS* , 2011.
[22] R. K. Venkatesan, et al., “Retention-Aware Placement in DRAM (RAPID): Software Methods for Quasi-Non-Volatile DRAM,” in *HPCA* 2006.