DRAM Refresh Mechanisms, Penalties, and Trade-Offs

Ishwar Bhati, Mu-Tien Chang, Zeshan Chishti, *Member, IEEE*,
Shih-Lien Lu, *Senior Member, IEEE*, and Bruce Jacob, *Member, IEEE*

Abstract—Ever-growing application data footprints demand faster main memory with larger capacity. DRAM has been the technology choice for main memory due to its low latency and high density. However, DRAM cells must be refreshed periodically to preserve their content. Refresh operations negatively affect performance and power. Traditionally, the performance and power overhead of refresh have been insignificant. But as the size and speed of DRAM chips continue to increase, refresh becomes a dominating factor of DRAM performance and power dissipation. In this paper, we conduct a comprehensive study of the issues related to refresh operations in modern DRAMs. Specifically, we describe the difference in refresh operations between modern synchronous DRAM and traditional asynchronous DRAM; the refresh modes and timings; and variations in data retention time. Moreover, we quantify refresh penalties versus device speed, size, and total memory capacity. We also categorize refresh mechanisms based on command granularity, and summarize refresh techniques proposed in research papers. Finally, based on our experiments and observations, we propose guidelines for mitigating DRAM refresh penalties.

Index Terms—Multicore processor, DRAM Refresh, power, performance

1 Introduction

Growing memory footprints, data intensive applications, the increasing number of cores on a single chip, and higher speed I/O capabilities have all led to higher bandwidth and capacity requirements for main memories. Most computing systems use DRAM (dynamic random access memory) as the technology of choice to implement main memory due to DRAM's higher density compared to SRAM (static random access memory), and due to its lower latency and higher bandwidth compared to nonvolatile memory technologies such as PCM (phase change memory), Flash, and magnetic disks.

A DRAM cell is composed of an access transistor and a capacitor. Data is stored in the capacitor as electrical charge, but electrical charge leaks over time. Therefore, DRAM must be refreshed periodically to preserve the stored data. Refresh negatively impacts DRAM performance and power dissipation. First, the memory controller stalls normal read and write requests to the part of the memory that is being refreshed. Second, refresh operations consume energy because refreshing a DRAM row involves operations such as reading and restoring data. As the speed and size of DRAM devices continue to increase with each new technology generation (Fig. 1), the performance and power overheads of refresh are increasing in significance (Fig. 2). For instance, our study shows that when using high-density 32Gb devices, refresh contributes to more than 20 percent of the DRAM energy consumption (Fig. 2a), and degrades the system performance by more than 30 percent (Fig. 2b).

- *I. Bhati is with Oracle, Santa Clara, CA, USA.
E-mail: ishwar.bhati02@gmail.com.*
- *M.-T. Chang is with Samsung, San Jose, CA, USA.
E-mail: mutien27@gmail.com.*
- *B. Jacob is with Department of Electrical and Computer Engineering, University Of Maryland, College Park, MD 20742, USA.
E-mail: blj@umd.edu.*
- *Z. Chishti and S.-L. Lu are with Intel Labs, Hillsboro, OR, USA.
E-mail: {zeshan.a.chishti, shih-lien.l.lu}@intel.com.*

*Manuscript received 2 May 2013; revised 13 Feb. 2015; accepted 3 Mar. 2015. Date of publication 0 . 0000; date of current version 0 . 0000.
Recommended for acceptance by E. L. Miller.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TC.2015.2417540*

In this paper, we evaluate and summarize the refresh mechanisms, trade-offs, and penalties in modern DRAMs, making the following main contributions:

1) We clarify the refresh command sequence for modern synchronous DRAM (SDRAM). In particular, since the traditional asynchronous interface is completely replaced, earlier refresh categorization specified in [1] such as RAS-only, CAS-before-RAS, and hidden refresh are no longer available in SDRAMs.
2) We summarize currently available refresh modes and refresh timings. We also review the characteristics of DRAM data retention time.
3) Based on full-system simulations, we demonstrate the refresh penalties versus device speed, device size, and total memory size. We also show that as the total memory capacity increases, background energy becomes more significant.
4) We categorize refresh scheduling mechanisms based on command granularity (i.e., rank, bank, and row). We also survey previously proposed refresh techniques and summarize their applicability to current and future memory systems.

Based on our experiments and observations, we provide general guidelines for designing techniques to mitigate refresh penalties.

2 DRAM Refresh: Status Quo

In the mid-1990s, DRAM architecture evolved rapidly from conventional asynchronous DRAM to Fast Page Mode

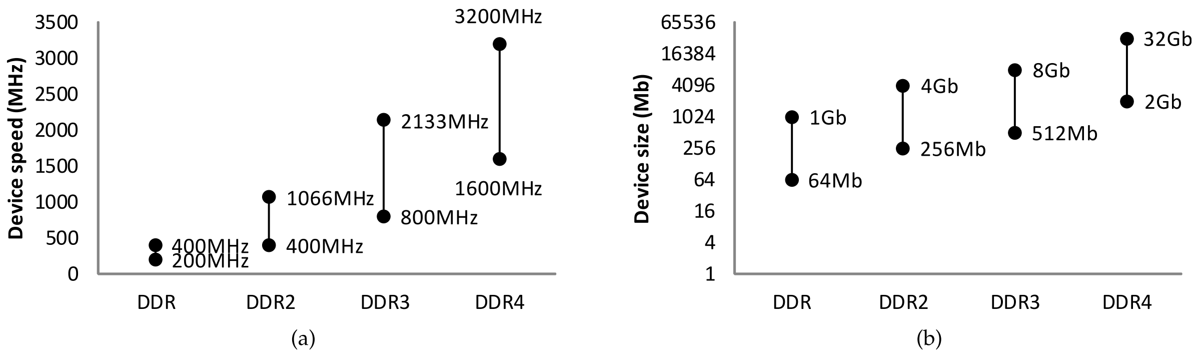


Fig. 1. DRAM device trends. Both speed and size increase with each DDR generation.

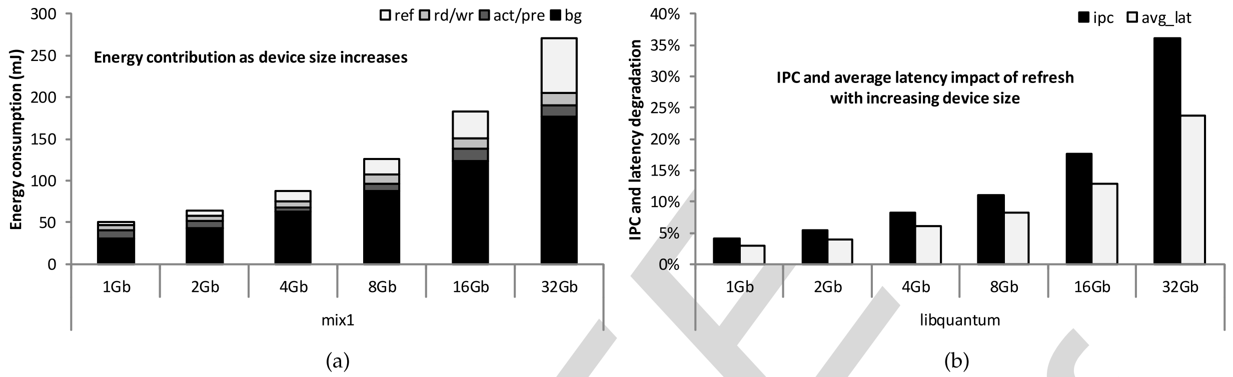


Fig. 2. Impact of refresh on energy and performance. (a) Energy breakdown (refresh, read/write, activate/pre-charge, and background). As device size increases, refresh and background energy become higher. (b) IPC (instruction per cycle) and average memory access latency degradation. The performance penalty due to refresh increases with increasing device size.

(FPM), then Burst Extended Data Out (BEDO), and finally SDRAM. SDRAM requires commands asserted on edges of the system clock. Significant structural modifications to the DRAM device have been explored to increase throughput, and memory vendors adopted different paths to reduce memory latency [2]. Subsequently, JEDEC formed a committee involving major players in the memory ecosystem to unify DRAM architecture and standardize its interface. Consequently, JEDEC specified the Double Data Rate (DDR) SDRAM architecture, which became highly successful and was widely adopted in the last decade. Currently, in addition to specifying newer generations of commodity DRAM devices (DDRx), JEDEC frequently publishes versions of low power (LPDDRx) and high bandwidth graphics (GDDRx) DRAM standards, all based on the original DDR architecture. The basic structure of the DRAM cell remains the same since its invention, which consists of a capacitor-transistor pair, as illustrated in Fig. 3.

A JEDEC-style SDRAM is organized into banks, rows, and columns, as shown in Fig. 3. Each bank has dedicated sense amps and peripheral circuitry, so multiple banks can process memory requests in parallel with some timing restrictions to avoid contentions on common internal and external buses. Multiple banks are essential to achieve sustained high bandwidth. According to the latest DDR4 standard [3], the banks in a device are further partitioned into bank groups. The banks in a group share some resources, therefore consecutive accesses to the banks in the same group require longer time. Internal accesses to DRAM arrays, such as refresh and activation, function at the row granularity. After an activation command, the entire row is read and buffered into the sense amplifiers. Subsequent column commands on the row can therefore use faster buffered data, rather than going to the DRAM array. Table 1 shows the organization parameters of DDRx and LPDDR3 devices.

Furthermore, multiple DRAM chips are wired together to build a memory rank, with a wider data bus. All the devices in a rank share address, command, and control signals. They all receive and serve the same requests, but each DRAM device owns its own portion of data on the bus. Typically, commodity DRAM chips are arranged on a

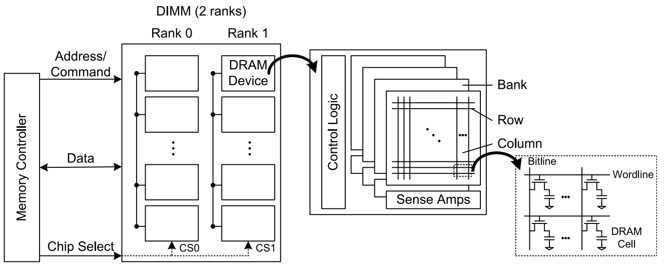


Fig. 3. DRAM memory system organization.

TABLE 1
Specified DDR Device Organization Parameters

Parameter	DDR2	DDR3	DDR4	LPDDR3
Bank groups	1	1	2/4	1
Banks	4/8	8	8/16	8
Rows per bank	4K–64K	4K–64K	16K–256K	16K–32K
Columns per row	512–2K	1K–4K	1K	1K–4K
I/O (bits)	4/8/16	4/8/16	4/8/16	16/32

Dual Inline Memory Module (DIMM), which could have one or more ranks. Finally, a memory channel is formed using one or more DIMMs, therefore potentially having several ranks. The ranks in a channel share all signals except the chip select signal, which is used to distinguish between multiple ranks.

2.1 Asynchronous vs. Synchronous DRAMs

As DRAM architecture evolved from “asynchronous” DRAM to SDRAM, refresh parameters and options were also modified accordingly. The following changed from earlier asynchronous devices [1] to current SDRAM.

- *Refresh Rate.* In traditional “asynchronous” DRAM, there are two types of devices, one with standard refresh rate (15.6 us), and the other with extended refresh rate (125 us). In current SDRAM devices, the required refresh rate only changes with temperature, regardless of device organization. For example, all DDR3 devices require refresh rate of 7.8 us at normal temperature range (0–85°C), and 3.9 us at extended temperature range (up to 95°C).
- *Distributed and Burst Refresh.* In traditional “asynchronous” DRAM, the memory controller can decide to complete all required refreshes in a burst or to distribute evenly the refreshes over the retention time. In modern DDRx devices, only the distributed refresh option is supported, to keep refresh management simple. LPDDRx devices, on the other hand, also support burst refresh which can be used to meet the deadlines of real-time applications.
- *RAS-Only Refresh.* In traditional “asynchronous” DRAM, RAS-only refresh is performed by asserting RAS with a row address to be refreshed, and CAS remains de-asserted. The controller is responsible for managing the rows to be refreshed. There is no equivalent command in modern SDRAM. To accomplish something similar to RAS-only refresh, one could issue an explicit activate command followed by a precharge to the bank. As we show in later sections, this has higher energy and performance costs. It would also require higher management burden on the memory controller.
- *CAS-Before-RAS (CBR) Refresh.* In traditional “asynchronous” DRAM, CBR refresh starts by first asserting CAS and then asserting only RAS. There is no requirement of sending a row address, because a device has an internal counter that increments with each CBR command. In modern SDRAMs, a variation of CBR is adopted with two important changes. First, both RAS and CAS are asserted simultaneously on the clock edge, rather than one before the other. Second, instead of internally refreshing only one row, SDRAM devices can refresh more rows depending upon the total number of rows in a device. This command is referred to as *auto-refresh* in JEDEC-based SDRAM devices.
- *Hidden Refresh.* Hidden refresh is referred to as an immediate CBR command after a read or write operation by keeping CAS asserted, while RAS is de-asserted once and then asserted again. This means that the data on the DQ lines is valid while performing refresh. There is no timing advantage when compared to a read/write followed by an explicit CBR command. Hidden refresh is implemented in “asynchronous” DRAMs but not in SDRAMs.

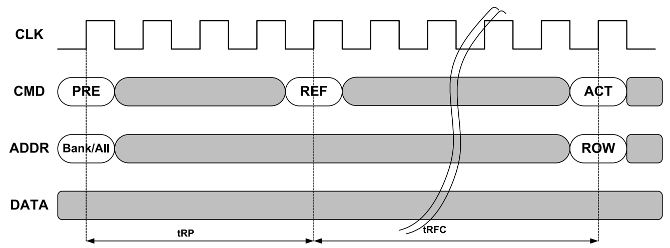


Fig. 4. Auto-refresh in SDRAM devices. The opened rows are precharged before issuing an AR command. Subsequent operations need to wait until tRFC for refresh to complete.

2.2 SDRAM Refresh Modes

SDRAM devices use auto-refresh (AR) and self-refresh (SR) to perform refresh. In general, SR is used when idle for power saving, while AR is used when busy.

2.2.1 *Auto-Refresh (AR)*

The shift from asynchronous to synchronous DRAM devices changed the refresh command interface and protocols. In SDRAM devices, the command and address signals are sampled on the edges of a clock. Fig. 4 illustrates a typical AR scenario where the device is brought to the idle state by precharging all the opened rows, and then AR is issued. When signaling an AR command in DDRx, the memory controller asserts both row access strobe (RAS) and column access strobe (CAS), along with selecting the device by chip select (CS) [4].

To simplify refresh management, each DRAM device has an internal refresh counter that tracks the rows to be refreshed during the next refresh operation. The controller is responsible for issuing AR commands at a specified rate to refresh a certain number of rows in all the banks (referred to as *all-bank auto-refresh*). Normal memory operations resume only after the completion of an AR.

LPDDRx devices use double data rate even on the command/address (CA) bus to reduce pin count. An AR is initiated when the CA0 and CA1 pins are driven LOW while keeping CA2 HIGH on the rising edge of the clock [5]. Unlike DDRx devices, LPDDRs have the flexibility to schedule AR at the bank granularity (referred to as *per-bank auto-refresh*), which only requires the bank to be refreshed to be idle, while other banks could service memory requests. Note that per-bank AR cannot specify the bank address to be refreshed, i.e., the DRAM maintains the target bank number internally, and with each command the target bank number is incremented sequentially starting from bank 0. Therefore, the memory controller must ensure that its notion of target bank number is in sync with the LPDDR device’s notion of the target bank number by using all-bank refresh. It is also worth noting that whether an AR is per-bank or all-bank can be dynamically decided based on the CA3 signal (LOW for per-bank and HIGH for all-bank).

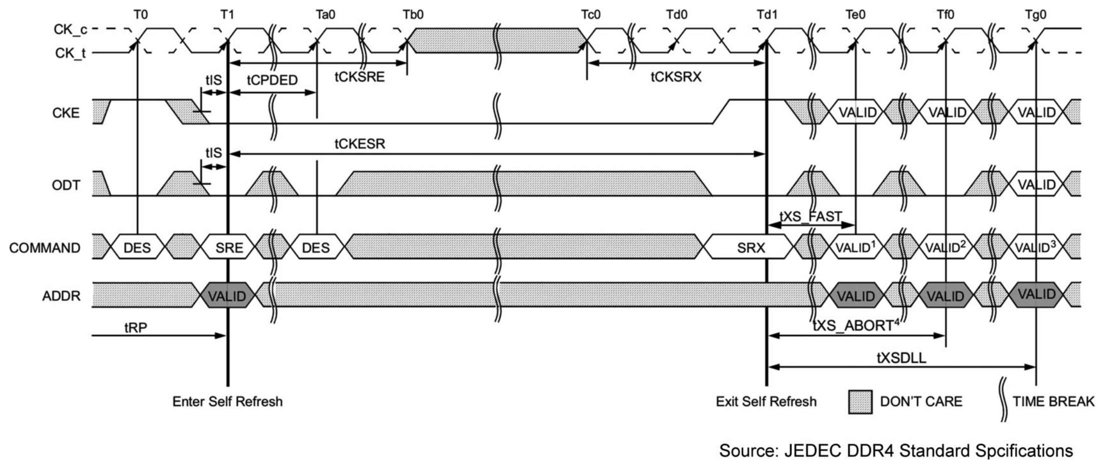


Fig. 5. Self-refresh entry/exit timings for DDR4 [3].

2.2.2 Self-Refresh (SR)

Auto-refresh dissipates substantial power since all the clocked circuitry in an SDRAM remains active during the entire refresh period. As a result, in addition to the power required for refresh, background power is consumed due to the delay locked loop (DLL) and peripheral logic. To save background power, a DRAM device has an option to enter SR mode, where the device internally generates refresh pulses using a built-in analog timer. In other words, when a device is in SR mode, all external I/O pins and clocks are disabled, the DLL is turned off, and the device preserves data without any intervention from the memory controller. SR is the lowest power mode for a DRAM device without losing data.

Fig. 5 shows the entry and exit timing of SR for DDR4 devices [3]. First, same as in AR, all the banks should be precharged before entering SR. The device enters SR mode when the clock enable (CKE) signal is sampled low while the command is decoded as refresh (RAS = LOW, CAS = LOW, WE = HIGH, and CS = LOW). Additionally, commands on the previous and the next clock cycle should be deselected (CS=HIGH). Furthermore, the DRAM device should remain in SR mode for at least a time period specified by tCKESR. The device should also internally schedule a refresh command within tCKE period upon entering SR mode. Once the device is in SR mode for tCKSRE, external clocks can be disabled.

When exiting SR, a specified time is required to ensure that the ongoing refresh command is finished and the DLL is locked properly. The specified time is the maximum of the following two timing parameters: (i) tRFC, the time required to service a refresh command, and (ii) tXSDLLK, the DLL lock period. It is worth noting that DDR4 devices support an option to abort an ongoing refresh command, making exiting SR faster (tXS_FAST and tXS_ABORT). Nonetheless, subsequent commands that require locked DLL still need to wait until tDLLK is complete. Since LPDDRx devices do not have a DLL, the time to exit SR only depends on tRFC. Finally, before re-entering SR mode, at least one AR must be issued.

LPDDRx devices dedicate more resources to reduce the background power during SR. Specifically, two important techniques are used in LPDDRs: (i) temperature compensated refresh rate guided by on-chip temperature sensors, and (ii) the partial array self-refresh (PASR) option, where the controller can program the device to refresh only a certain portion of the memory. These techniques could substantially reduce the energy consumption while in the SR mode. For example, Fig. 6 shows how the current drawn during SR changes with temperature and with PASR [6].

2.3 Refresh Timings

Modern DRAM devices contain built-in refresh counters; thus the only requirement of the memory controller is to issue refresh commands at appropriate times. The fundamental requirement is that each DRAM cell should be refreshed or accessed at least once within its retention period. Most of the commodity DRAM devices have either 32 or 64 ms retention time, also called the refresh window (tREFW). The retention time usually decreases with increasing temperature. Additionally, on average one AR command should be issued within a refresh interval time (tREFI). Therefore, the memory controller should issue at least $\frac{tREFW}{tREFI}$ number of AR commands within a refresh window to ensure that every DRAM cell is refreshed before the retention time expires.

Each AR command refreshes a number of rows in each bank depending on the total number of rows in the DRAM device. For instance, a DDR3 device has a tREFI of 7.8 us and a tREFW of 64 ms. Therefore, 8,192 refresh commands

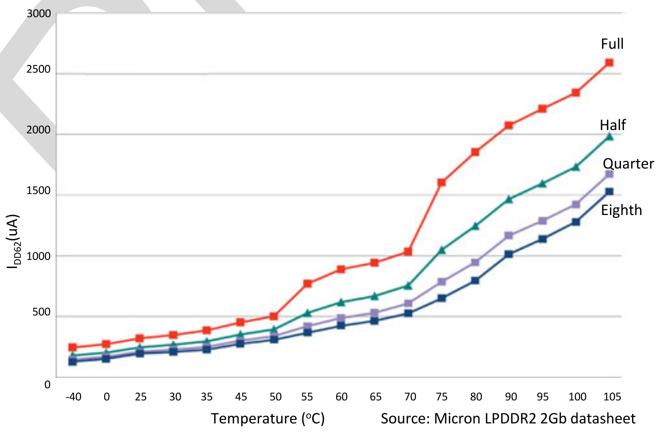


Fig. 6. Refresh power reduction in LPDDRx when employing temperature compensation and partial array self-refresh (PASR) [6]. As the temperature increases, the refresh current (IDD62) becomes higher, and PASR shows more benefits.

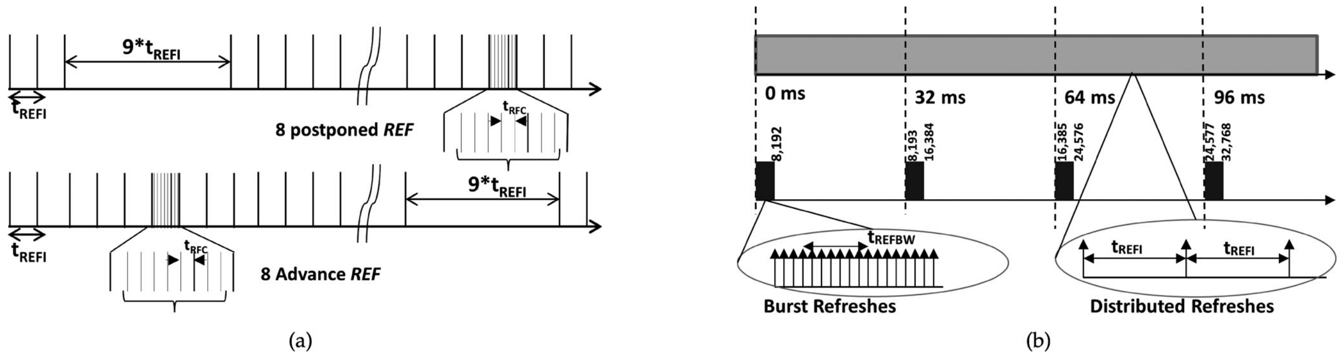


Fig. 7. Available refresh scheduling flexibility in (a) DDRx and (b) LPDDRx devices. (a) In DDRx, up to eight AR commands can be postponed and later compensated for, by issuing extra AR. Similarly, up to eight AR can be launched in advance and later may be skipped. (b) In LPDDRx, the scheduling ranges from distributed refresh where only one AR is scheduled every tREFI, to a burst mode in which all required AR are completed in a burst in the beginning of a refresh window.

are issued in a tREFW period. For a 512 Mb $\times$ 8 device, there are 8,192 rows per bank, and hence each AR needs to refresh only one row in each bank, and the internal refresh counter is only incremented by one. However, for a 4 Gb $\times$ 8 device, there are 65,536 rows per bank; therefore one AR command should refresh 8 rows in each bank and increment the internal counter by eight. As the number of rows to be refreshed by a single AR increases, the refresh completion time also increases. The time for a refresh command is the refresh cycle (tRFC).

In DDRx devices, the refresh flexibility allows up to eight AR commands to be postponed or issued in advance, as shown in Fig. 7a. The JEDEC standard allows for a debit scheme, in which up to eight refreshes are postponed during the high memory activity phase, and later on in the idle period these extra AR commands are be issued. Alternatively, a credit scheme can be devised by first issuing extra refreshes and then later on skipping them. However, the rate of refresh should meet two constraints: (i) at least one AR must be issued in 9*tREFI time period, and (ii) no more than 16 AR commands are issued in a 2*tREFI interval.

LPDDRx devices provide more flexibility in scheduling AR, as shown in Fig. 7b. These devices support both distributed and burst refresh mode and anything in between. The requirement is that, in a sliding window of tREFW, 8,192 all-bank-AR operations should be issued. Moreover, a maximum of 8 refreshes may be issued in a burst window called tREFBW, of duration 32*tRFC.

DDR2 and DDR3 devices are specified to keep tREFI constant (7.8 us), but with different tRFC period according to the device density. Due to this, tRFC becomes prohibitively long for high-density devices. In response to growing tRFC, DDR4 has introduced a fine-granularity refresh mode that allows tREFI to be programmed [3]. This allows users to enable 2x or 4x mode, where tREFI is divided by 2 or 4, respectively. Consequently, the number of rows refreshed for a single refresh command is decreased by 2x or 4x, which in turn shortens tRFC. With an on-the-fly setting, one could change the refresh rate dynamically to suit the memory demand.

Table 2 shows the tREFI and tRFC timing values for several DRAM generations and several device sizes. Note that for a given DRAM architecture (e.g., DDR3), tRFC is not constant and can vary significantly.

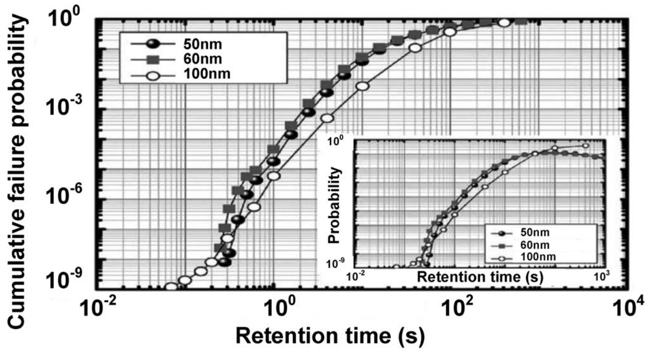
2.4 DRAM Retention Time

Due to junction leakage, gate-induced drain leakage, off-leakage, field transistor leakage, and capacitor dielectric leakage, a DRAM cell loses charge over time [7]. Therefore, those cells storing useful data need to be refreshed periodically to preserve data integrity. The primary timing parameter for refresh is retention time, the time between storing data and the first erroneous readout. Note that in a DRAM device, cells do not have the same retention time because of process variations. This phenomenon is referred to as "inter-cell" distributed retention time. The cells could be broadly divided into two categories: leaky and normal cells. The leaky cells draw order of magnitude higher leakage currents than the normal cells. As shown in Fig. 8a, most of the cells are normal cells, which have retention time more than 1 second [8], [9]. However, to accommodate the worst-case scenario, the retention time of a DRAM device is determined by the retention time of the leakiest cell.

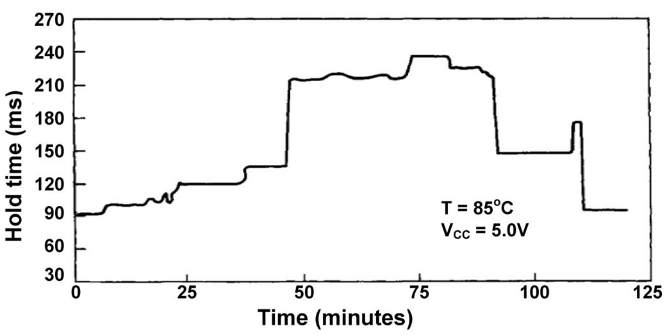
TABLE 2
Refresh Timing Different Values for different DDR Device Generations

Device	Timing Parameter	1 Gb	2 Gb	4 Gb	8 Gb	16 Gb	32 Gb
DDR2 (tREFI = 7.8us)	tRFC (ns)	127.5	197.5	327.5	—	—	—
DDR3 (tREFI = 7.8us)	tRFC (ns)	110	160	300	350	—	—
DDR4 1x (tREFI = 7.8us)	tRFC (ns)	—	160	260	350	TBD	—
DDR4 2x (tREFI = 3.9us)	tRFC (ns)	—	110	160	260	TBD	—
DDR4 4x (tREFI = 1.95us)	tRFC (ns)	—	90	110	160	TBD	—
LPDDR3 (tREFI = 3.9us, tREFW = 32ms)	tRFCab (ns)	—	—	130	210	TBD	TBD
	tRFCpb (ns)	—	—	60	90	TBD	TBD

DDR4 has an optional feature to shorten tREFI, by either 2x or 4x. LPDDR3 has per-bank (pb) and all-bank (ab) auto-refresh commands, while DDRx has only all-bank.



(a)



(b)

Fig. 8. DRAM retention time characteristics. (a) Inter-cell retention time distribution (source: [9]). Most of the cells have higher retention time, while very few cells are leaky and therefore exhibit low retention time. (b) Intra-cell variable retention time (source: [10]). The retention time of a single cell vary with temperature as well as time.

TABLE 3
Processor and Memory Configurations

Processor	4 cores, 2GHz, out-of-order, 4-issue per core
L1 Cache	Private, 128 KB, 8-way associative, 64 B block size, 2 cycle latency
L2 Cache	Shared, 8 MB, 8-way associativity, 64 B block size, 8 cycle latency
Memory controller	Open page, first ready first come first serve, "RW:BK:RK:CH:CL" address mapping, 64-entry queue, 1 channel
Total memory size	8 GB–64 GB (default: 8 GB)
DRAM device	Size 1 Gb–32 Gb (default: 8 Gb); speed 1066 Mbps–3200 Mbps (default: 1333 Mbps)

TABLE 4
Workload Composition

Memory Bandwidth	Workloads (4 copies of same program or mixed instances)
LOW	hmmer; namd; mix1 (games, namd, hmmer, povray)
MEDIUM	milc; gromacs; GemsFDTD
HIGH	libquantum; mcf; mix2 (mcf, libquantum, milc, GemsFDTD)

Another phenomenon is "intra-cell" variable retention time. Variable retention time corresponds to two or multiple meta-states in which a DRAM cell can stay [10]. Since each of the states have different leakage characteristics, the retention time of a DRAM cell varies from state to state. Additionally, the switching frequency between different states increases at higher temperature. For instance, a DRAM retention time state can switch as frequently as 15 times in an hour at 85°C. Fig. 8b shows an example of variable retention time [10].

Finally, retention time has high sensitivity to temperature. As the temperature increases, leakage also increases and therefore shortens the retention time. As a result, at extended temperatures (i.e., 85–95°C), DDRx devices must increase the refresh rate. LPDDRx devices also have on-device temperature sensors that adjust the refresh rate according to the temperature.

3 Refresh Penalties

In this section, we quantify the impact of refresh operations on energy and performance by simulating various DRAM device and memory system configurations.

3.1 Experimental Setup

We use DRAMSim2 [11], a cycle-accurate memory-system simulator. It is integrated with MARSSx86 [12], a full-system x86 simulator based on QEMU [13] and an out-of-order superscalar multicore processor model [14]. Table 3 shows the baseline system configuration. We also model accurate timings for low power mode switching overheads and refresh constraints, compliant with the DDR3 standard. The DRAM parameters used in our simulations are taken from vendor datasheets [15]. For the device sizes and speed grades not currently available, we extrapolate from existing DDR3 devices based on recent scaling trends. We calculate DRAM energy from the device's IDD numbers, using the methodology described in [16]. In all our experiments, the memory controller closes an open row if the queue for that rank is empty or after four accesses to that row, as suggested in [17]. The address mapping configuration used by the memory controller ensures that each channel and rank receives uniform memory traffic. A rank switches to slow exit power down mode immediately after the request queue for that rank becomes empty, as proposed in [18].

We use the SPEC CPU2006 benchmark suite [19]. For each benchmark, we determine the region of interest using SimPoint 3.0 [20] and simulate a total of 1 billion instructions, starting from the region of interest. The workloads are categorized into *LOW*, *MEDIUM* and *HIGH*, depending on their memory bandwidth requirements. We pick some workloads in each category, as shown in Table 4, to simulate and analyze

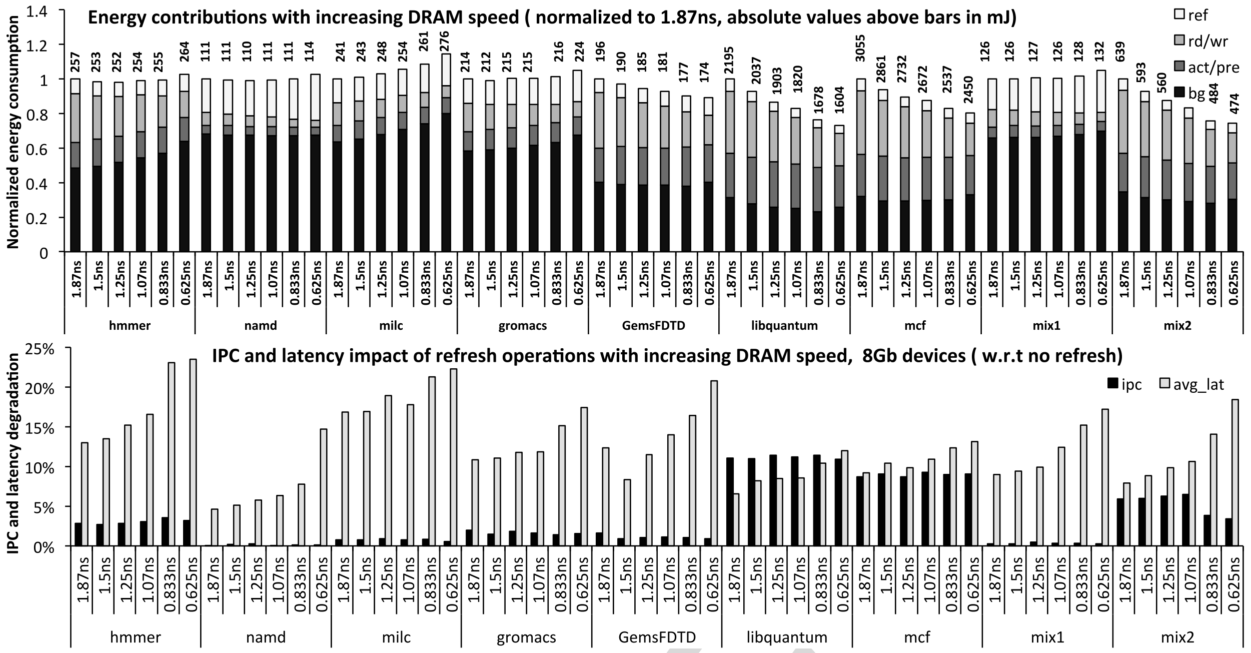


Fig. 9. Refresh penalty vs. device speed. The device speed increases from 1.87ns (1066Mbps) to 0.625ns (3200Mbps). The background energy increases due to faster switching peripherals. The performance penalty due to refresh operations in *HIGH* bandwidth workloads is substantial, but does not change much with speed.

different aspects of the memory system. These workloads show representative results for the purpose of our study.

We note that our power model follows the trends observed on real memory systems. For instance, in our simulations, *LOW* workloads consume 0.5–1W/GB, *MEDIUM* consume 0.8-1.2W/GB, and *HIGH* consume 1–2.8W/GB. These power numbers are within the range of DRAM power consumed in recent HP [21] and Intel [22] servers.

3.2 Refresh Penalty vs. Device Speed

Fig. 9 shows the impact of refresh on memory energy consumption and performance with various device speed configurations. Note that for all the energy bar graphs in Section 3, we show both the normalized energy breakdown and the absolute energy values, in milli-Joules. The energy consumption of each workload is normalized to the first configuration. In the performance bar graphs, we show the negative impact of refresh on the system's instructions per cycle (IPC) and average memory latency. We obtain the percentage degradation results by comparing memory systems with refresh against ones with no refresh, using the same configuration.

In the *LOW* bandwidth workloads of Fig. 9, background and refresh energy increase with device speed, due to fast switching of I/O circuits in the DRAM device. However, in the *HIGH* bandwidth workloads, higher speeds result in better performance and therefore less overall energy consumption. The performance penalty of refresh in the *HIGH* bandwidth workloads is substantial and results in up to 11.4 percent IPC loss. With varying device speed, there is not much change in the IPC loss; however, the penalty on the average DRAM latency increases with device speed. This indicates that memory requests are either not in the critical path of the program, expected in compute-bound applications; or in the case of memory-bound applications, there is enough Memory Level Parallelism (MLP) to hide the increase in memory latency. Moreover, latency degradation in *LOW* bandwidth programs varies the most with speed (e.g., from 13 to 23.5 percent). These programs have few memory requests, which magnifies the refresh penalty.

3.3 Refresh Penalty vs. Device Density

Fig. 10 shows the effect of refresh when DRAM device density increases. Both the refresh and background energy increase substantially with device density. For instance, refresh represents 25-30 percent of DRAM energy for 32 Gb device in *LOW* bandwidth programs. In *HIGH* bandwidth workloads, most of the energy is due to memory accesses. Furthermore, the performance penalty is severe in high density devices for *HIGH* bandwidth programs—e.g., when using 32 Gb devices, the IPC degradation is more than 30 percent for *libquantum* and *mcf*.

As device density increases, *LOW* and *MEDIUM* bandwidth workloads show substantial increase in energy consumption as well as noticeable performance drop. To reduce energy and performance penalties, these workloads require intelligent refresh-scheduling schemes to effectively utilize idle periods when memory can either switch to low power SR mode or issue more refresh operations in advance. As detailed in Section 4.2, several recently proposed refresh schemes use techniques to track access patterns to predict idleness and then use the prediction to schedule energy- or performance-efficient operations [23], [24], [25], [26], [27].

3.4 Refresh Penalty vs. System Memory Size

We increase the total memory size from 8 GB to 64 GB, keeping the device size and speed constant at 8 Gb and 1333 Mbps. Note that the number of ranks in a channel increases with increasing memory capacity, i.e., 1, 2, 4 and 8 ranks for 8 GB, 16 GB, 32 GB and 64 GB memory.

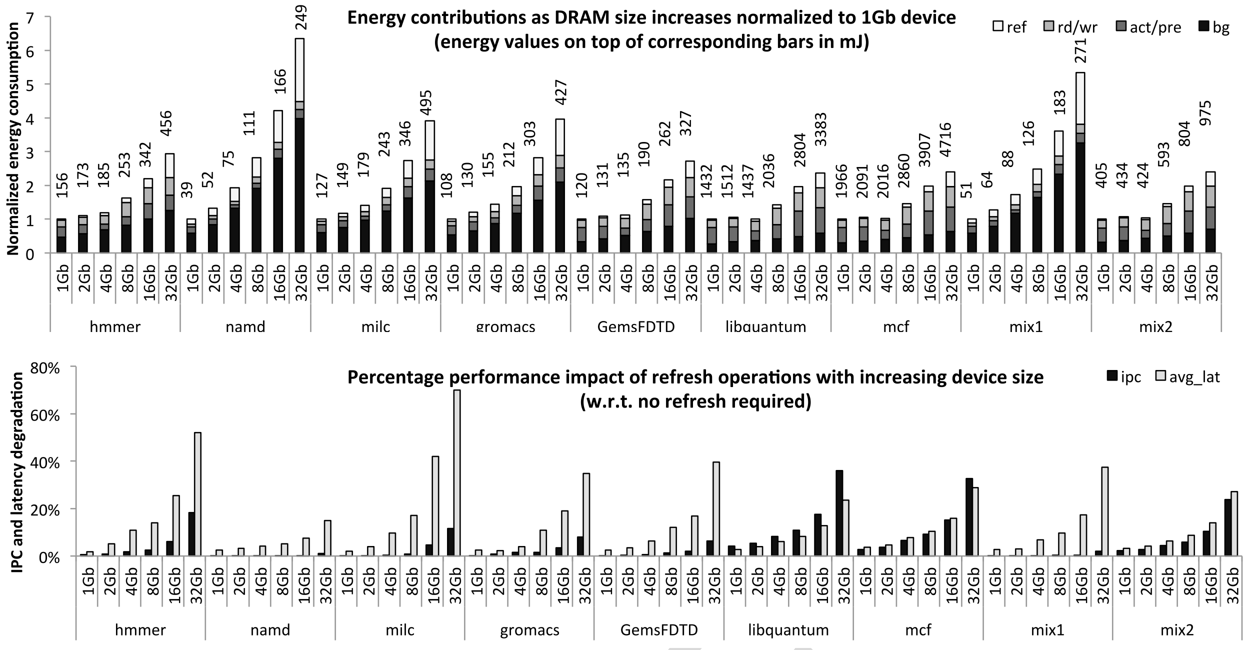


Fig. 10. Refresh penalty vs. device density. The memory subsystem is one channel, one rank. Device size increases from 1 Gb to 32 Gb. Refresh energy increases with size, more substantial in *LOW* bandwidth workloads. Background energy also increases for two reasons: (i) more peripherals as size increases; (ii) longer active mode refresh commands.

Fig. 11 shows the energy breakdown and performance penalty as the system memory size increases. Systems with larger total memory capacity dissipate more background power because more 8 Gb devices are utilized. For a system with 64 GB memory, refresh and background power are the major sources of DRAM energy consumption, even when running *HIGH* bandwidth programs. Finally, we observe that refresh has greater IPC impact on *HIGH* bandwidth programs, while affecting average latency more on *LOW* bandwidth workloads.

As system memory increases, some workloads (*mix2* and *hmmer*) show less IPC drop, while others (*libquantum* and *mcf*) show more. This behavior depends on the program's memory-access pattern, e.g., whether memory requests are evenly distributed on each rank, or some ranks are accessed more often than others. If memory accesses are distributed evenly, and as the number of ranks increases with system memory, then when one rank is refreshing, the remaining ranks can serve requests in parallel. Therefore, in this scenario, the refresh penalty decreases with higher memory

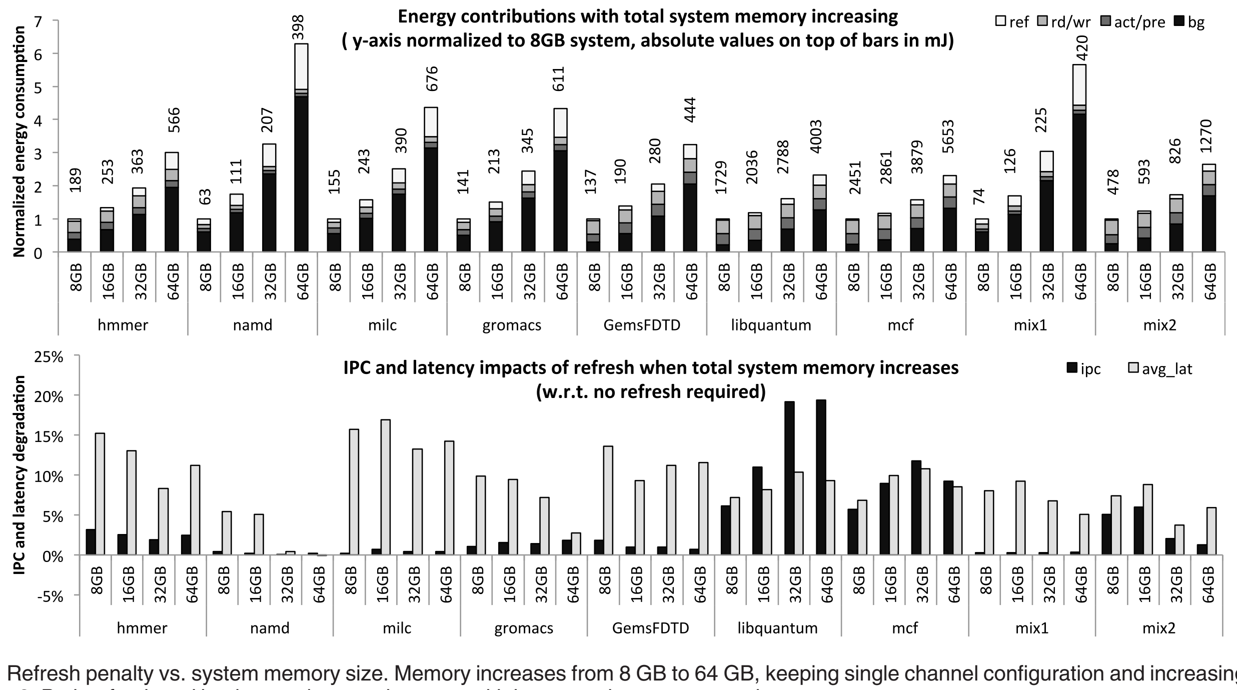


Fig. 11. Refresh penalty vs. system memory size. Memory increases from 8 GB to 64 GB, keeping single channel configuration and increasing ranks from 1 to 8. Both refresh and background energy increase with larger total memory capacity.

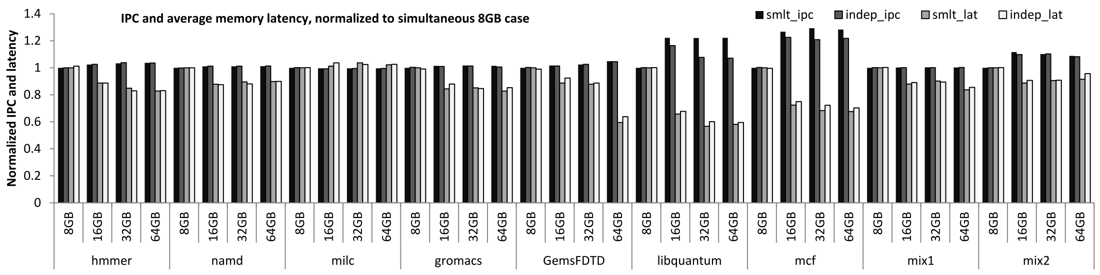


Fig. 12. All-rank (simultaneous) refresh vs. per-rank (independent) refresh. The *X*-axis shows the total system memory capacity varying from 8 GB to 64 GB, and increasing the number of ranks on a channel.

size. On the other hand, if the program does not access the ranks evenly, and since ranks on the same channel share command and data buses, the increased number of refreshes will have an adverse effect on memory bandwidth as the system memory increases.

4 Refresh Options and Techniques

In this section, we first categorize refresh options based on the command granularity: rank, bank, and row level. Furthermore, we survey various refresh techniques and discuss their applicability in modern DRAM systems.

4.1 Refresh Granularity

Refresh commands can be processed at the rank, bank, or row level.

Rank-Level: All-Rank (Simultaneous) and Per-Rank (Independent) Refresh. At the system level, the memory controller can either schedule AR to all ranks simultaneously (simultaneous refresh), or it can schedule AR commands to each rank independently (independent refresh). In simultaneous refresh, the entire system is unavailable during the refresh completion period, while in the independent refresh case, some ranks in a multi-rank memory system are still available to service requests. Depending upon the number of processes and their address mappings, either simultaneous or independent refresh could result in better performance.

In Fig. 12, we compare the effects of choosing either simultaneous or independent refresh options when changing the number of ranks from 1 to 8 with 8 Gb DRAM devices. In most of the cases, there is not much difference between simultaneous or independent refresh options. However, some *HIGH* memory programs show slightly better performance in the case of simultaneous refresh option due to the overall reduced refresh down time.

Bank-Level: All-Bank and Per-Bank Refresh. General purpose DDRx devices only have the AR commands at the granularity of the entire device (i.e., all the banks in the device are unavailable when an AR command is issued). Therefore, an AR is given to all the devices in a rank, and none of the banks is allowed to service any requests until refresh is complete. This command is referred to as *all-bank refresh*. On the other hand, in addition to all-bank AR commands, LPDDRx devices have the option of per-bank AR, where only one bank is down when an AR is issued, while other banks could still serve normal memory requests [5]. Eight sequential per-bank refreshes are equivalent to one all-bank refresh, assuming eight banks.

Row-Level. A row-level refresh can be accomplished by either adding a new command to refresh a certain row in a given bank, or by explicitly activating a row and then precharging it. The former requires changes to SDRAM devices; the latter does not require device changes but requires more command bandwidth.

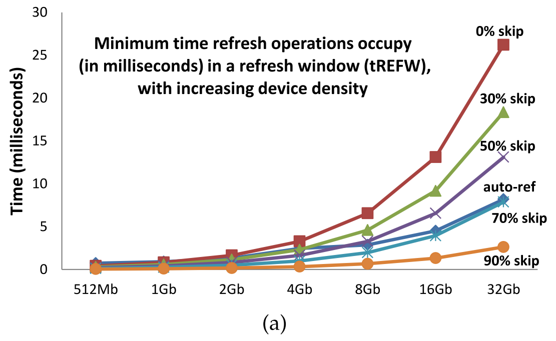
The advantage of row-level refresh is that the memory controller can skip redundant refresh operations based on the status of each row. For example, if a row has a longer retention time than tREFW (e.g., 128 ms), then using the normal refresh rate (e.g., assuming 64 ms retention time) results in redundant refreshes. Alternatively, if a row is read or written more frequently than the refresh rate, then refreshes to that row become redundant and can be skipped. However, as shown in Fig. 13a, for higher density devices, the time required for refresh operations using row-level refreshes gets longer as compared to AR. For instance, even if the controller can skip 70 percent of the rows to be refreshed in a refresh window, the time is still comparable to AR. The main reason for this performance difference is that DRAM vendors have optimized AR to refresh rows in the same bank in parallel. Furthermore, AR internally utilizes aggressive bank-level parallelism by activating rows faster than the tRRD (row-to-row activation delay) and tFAW (four-bank activation window) constraints, since device organizations and power surge are exactly known and optimized for AR. However, external activate commands required for row-level refresh need to follow tRRD and tFAW to meet DRAM power constraints, as shown in Fig. 14.

Moreover, issuing explicit activate/precharge commands to refresh each row consumes substantial command bandwidth in high density devices. As shown in Fig. 13b, the overall command bandwidth for refresh commands in a 4-rank system approaches 100 percent of the total bandwidth (assuming 64 ms refresh window). This not only degrades performance, it also eliminates many opportunities to switch to power-down modes. Finally, using SR mode for row-level refresh poses a difficulty, since the device's internal refresh counters need to be synchronized to the appropriate rows before entering SR.

4.2 Refresh Schemes

4.2.1 Based on Row-Level Refresh

Ohsawa et al. [28] analyzed the increasing impact of refresh operations on system energy and performance in merged DRAM/logic chips. The study proposed two DRAM refresh architectures that eliminate unnecessary refreshes. One



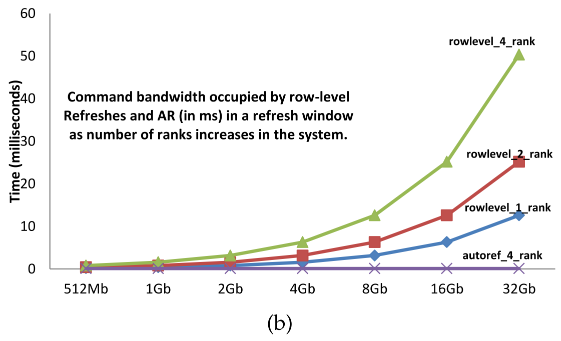


Fig. 13. Row-level refresh vs. auto-refresh. (a) The minimum time to satisfy refresh requirement in a bank vs. device density. The percentage skip corresponds to the number of rows which need not be refreshed in tREFW (i.e., refreshes to a row can be bypassed if its explicitly accessed). The advantage of row-level refresh decreases with each DRAM generation. (b) Command bandwidth consumed by refresh operations based on activate/precharge commands. The command bandwidth increases with increasing device density and the number of ranks.

technique, Selective Refresh Architecture (SRA), performs refresh operations at fine granularity and can either select or skip refresh to a row. In particular, SRA can reduce refreshes if the controller has knowledge of whether the data stored in the rows are going to be used in the future. To implement SRA, one option is to add per-row flags in DRAM to indicate whether a row should be refreshed or not. These flags can be programmed by the memory controller using customized commands. Another option is to implement row-level refresh, which the memory controller can selectively issue. The former option requires extra flags for each row in a DRAM device, while the latter option introduces refresh scheduling complexity and storage overhead to the memory controller. As the number of rows increases, both options create substantial overheads.

Smart Refresh [29] also adopts the selective row-refresh mechanism: refreshes are skipped for rows accessed in the last refresh period, and the memory controller sends row-level refresh commands to the remaining rows. The memory controller requires a large SRAM array for the state information of each row in the entire memory system. Although the authors proposed schemes based on CBR and RAS-only, modern DDRx SDRAM devices do not support per-row refresh commands. The option for RAS-only is to send an explicit activate command followed by a precharge command for each row, as opposed to using the AR command. Since AR is usually optimized for performance as well as energy by DRAM vendors, some of the benefits of Smart Refresh will be nullified in high density devices.

ESKIMO [30] uses semantic refresh and the concept from the row-selective approach to avoid refreshing the rows storing data that have been freed. The authors proposed to use SRA so that fine-grained row-level flags are used to skip some of the unwanted refreshes.

4.2.2 Using Retention Time Awareness

Also proposed in Ref. [28] is Variable Refresh Period Architecture (VRA), wherein the refresh interval for each row is chosen from a set of periods. Since most DRAM cells exhibit higher retention times, few rows require the worst-case rate. Therefore, VRA reduces a significant number of refreshes by setting an appropriate refresh period for each row. However, the hardware overhead for maintaining refresh interval tables in DRAM devices or in the controller becomes significant, as the number of rows has increased rapidly in recent devices.

Flikker [31] and RAPID [32] use the distribution of DRAM cell retention periods to reduce the number of refresh operations. Flikker requires the application to partition data into critical and non-critical sections, then it uses the sections with regular refresh rate for critical data and the sections with slow refresh rate for non-critical data. This means that the non-critical regions can tolerate some degree of data loss. In RAPID, the operating system (OS) uses the retention time of the pages to prefer pages with longer retention time. This allows RAPID to choose the shortest-retention period among only the populated pages, rather than all memory pages. This mechanism involves only software but requires the OS to determine each page's retention period.

Liu et al. proposed RAIDR [33], which optimizes the refresh rate based on the retention time distribution of DRAM cells. DRAM rows are first binned based on the

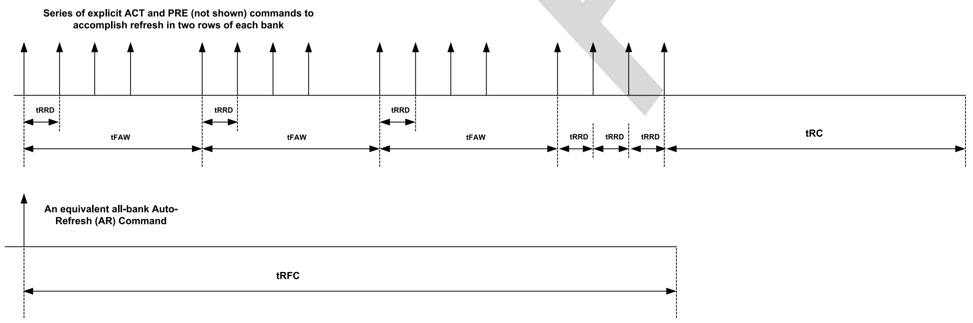

Series of explicit ACT and PRE (not shown) commands to accomplish refresh in two rows of each bank
tRRD
tFAW
tRRD
tFAW
tRRD
tFAW
tRRD
tRRD
tRRD
tRC
An equivalent all-bank Auto-Refresh (AR) Command
tRFC

Fig. 14. Timing constraints and row-level refreshes (sets of activate/precharge commands) required to accomplish refresh operation equivalent to an AR. In this example, an AR command refreshes two rows in each of the 8 banks (corresponds to 1 Gb device). Timing parameters are taken from DDR3 specs [4] (tRC = 50 ns, tRRD = 6 ns, tFAW = 30 ns, and tRFC = 110 ns).

retention time of the row's leakiest DRAM cell. Since leaky cells occur infrequently, most of the rows require lower refresh rates. Furthermore, RAIDR uses bloom filters to encode the required refresh rate for each bin. RAIDR does not use auto-refresh but sends explicit activate and precharge sequences to each row.

Recently, experimental refresh studies [34], [35] characterize retention periods, and their results confirm the normal distribution of retention time in modern DRAMs. However, profiling and accounting for retention period variability still remains an unsettled topic. In addition to retention period characterization, Baek et al. [35] propose two software based schemes to account for weak cells, either discarding them through OS based memory management, or skipping unused refresh regions under the control of system software. Both schemes need RAS-only refresh command support from DRAM devices.

4.2.3 Using Refresh Scheduling Flexibility

Many previous papers proposed mechanisms to mitigate the impact of long refresh periods on performance. Stuecheli et al. [23] proposed Elastic Refresh which relies on re-scheduling the refresh commands so that they overlap with periods of DRAM inactivity. Elastic Refresh postpones up to eight refresh commands in high-memory request phases of programs, and then issues the pending refreshes during idle memory phases at a faster rate, to maintain the average refresh rate. Based on the number of pending refreshes and the memory request patterns, the thresholds for refresh scheduling in a rank are dynamically changed. However, all the refresh commands are issued in the active power mode, which consumes more background power. The increase in energy consumption due to long refresh period in high density DRAM devices was not evaluated.

Another recent technique, Adaptive Refresh [25], uses finer-granularity refresh modes introduced in Ref. [3] to reduce the refresh performance penalty. Adaptive Refresh decides the appropriate refresh granularity (between normal 1x and finer-grained 4x) by a heuristic based on dynamically monitoring the serviced memory bandwidth. This technique shows benefits only when the command queues in the memory controller are limited in size and are shared between all the ranks of a channel.

Lastly, Coordinated Refresh [26] focuses on both performance and energy consumption of refresh operations. This mechanism relies on the ability to re-schedule refresh commands to overlap with periods of DRAM inactivity while utilizing full-flexibility of refresh commands as in Elastic Refresh. Coordinated Refresh co-schedules the refresh commands and the low power mode switching such that most of the refreshes are issued energy efficiently, in SR mode.

4.2.4 For DRAM-Based Caches

DRAM is also used as a cache for main memory; for example the IBM Power7 uses eDRAM as its last-level cache [36], and hybrid memory systems use small DRAMs to cache the non-volatile main memory [37]. It is worth noting that the data retention time for eDRAMs is much shorter than commodity DRAMs. Also, DRAM caches are usually not required to follow the protocols for DRAM main memory; therefore there is more flexibility in designing refresh-reduction mechanisms.

For on-chip DRAM-based caches, an effective refresh-reduction technique is the use of error correcting codes [38], [39], [40]. This approach reduces the refresh rate by disassociating failure rate from single effects of the leakiest cells. Another promising approach exploits memory access behaviors; for instance, if a cache line is intensively read or written, refresh operations to that cache line are postponed [41], [42], [43]. On the other hand, if a cache line holds useless data (i.e., dead cache blocks), refresh operations can be bypassed [44].

For off-chip DRAM caches, the OS can be effective in assisting refresh reduction. For instance, Moshnyaga et al. [45] proposed to reduce the refresh energy based on the OS's knowledge in DRAM/Flash memory systems. They divide the active and non-refreshed banks based on the access patterns of data present in these banks. Refresh operations to a bank are disabled if the bank is not referenced in a given time-window and contains only unmodified pages. Since the OS knows which pages are being referenced, it can decide which banks to disable. Dirty pages in non-referenced banks are put into the swap cache, which are then written back to Flash.

4.3 Applicability Matrix

Table 5 summarizes the refresh techniques discussed in this section. The table shows the following: first, we characterize the power and performance improvements achieved using these schemes. Second, we categorize the schemes according to their refresh command granularity, to understand their feasibility in general-purpose DRAM devices. Third, the modifications required to implement the schemes are considered. This is important because the cost of changing at the device level is higher than modifying the memory controller, and software-level changes are relatively easier to accept than hardware modifications. Furthermore, we evaluate the difficulty of schemes to co-exist with the SR mode, since SR is very important for energy efficiency. We also consider how well these schemes will scale in future large memory systems built with high-density devices. Finally, since some techniques allow data in portions of the memory to get corrupted, we evaluate memory-system reliability.

4.4 Potential Refresh Improvements

We compared various data retention times against ideal memory without refresh. Fig. 15 shows the energy and performance impact when changing the refresh timing parameters. Our results illustrate the potential refresh improvement achieved when utilizing retention awareness. We expect refresh awareness to be even more effective when applied to future high density devices, but, to fully utilize its potential, trade-offs such as command bandwidth should be considered.

Fig. 15 shows the effects of decreasing the normal AR interval by 2x and 4x finer granularities, i.e, tREFI is decreased from 7.8 us to 3.9 us and 1.95 us, respectively. The corresponding tRFC values are chosen from the DDR4 specification, as shown in Table 2. For most of the

TABLE 5
Applicability Matrix of Refresh Schemes Discussed

Category	Scheme	Benefits		Refresh granularity			Modifications			SR support	Scalable	Reliable
		Energy	Performance	Row	Bank	Rank	Device	Controller	Software			
Row selective	SRA [28]	✓	×	✓	×	×	✓	✓	×	?	×	✓
	Smart Ref. [29]	✓	×	✓	×	×	✓	✓	×	×	×	✓
	ESKIMO [30]	✓	×	✓	×	×	✓	✓	✓	×	×	✓
Retention aware	RAIDR [33]	✓	✓	✓	×	×	×	✓	×	×	?	✓
	VRA [28]	✓	✓	✓	×	×	✓	✓	×	×	×	✓
	Flikker [31]	✓	?	×	×	✓	×	✓	✓	✓	✓	×
	RAPID [32]	✓	?	×	×	✓	×	×	✓	✓	✓	?
Refresh scheduling	Elastic [23]	×	✓	×	×	✓	×	✓	×	✓	✓	✓
	Pausing [24]	×	?	×	×	✓	✓	✓	×	✓	✓	✓
	Adaptive [25]	×	✓	×	×	✓	×	✓	×	✓	✓	✓
	Coordinated [26]	✓	✓	×	×	✓	×	✓	×	✓	✓	✓
DRAM as cache	Access behavior [38], [41], [42], [43], [44]	✓	×	✓	×	×	N/A	✓	×	N/A	✓	✓
	ECC-based [38], [39], [40]	✓	×	✓	×	×	N/A	✓	×	N/A	✓	?
	OS-control [45]	✓	×	×	✓	×	✓	✓	✓	×	×	✓

Symbols: ✓ → Yes; × → No; ? → Difficult to say Yes or No.

workloads, the finer grained options increase energy and degrade performance. However, for the *milc* benchmark, using 4x granularity improves performance. This indicates that finer granularities can potentially mitigate refresh penalties, but rather than constantly employing these options, one should use them intelligently.

5 Refresh Trends and Conclusions

Based on the recently proposed refresh techniques, DRAM scaling trends, and our refresh exploration results, we summarize our observations below:

1) Some refresh techniques based on old DRAM devices and asynchronous interfaces are useful for DRAM caches, but, for general purpose DDR SDRAM devices, they are less practical.
2) With increasing device density, JEDEC has provided multiple refresh scheduling and timing options. Understanding design trade-offs such as refresh management complexity, device level changes for refresh, and available refresh scheduling flexibility, will be important for designing practical refresh optimization schemes.
3) Techniques using DDR's built-in mechanisms to reduce the refresh penalty are desirable, for example: techniques exploiting Self Refresh mode, techniques exploiting refresh flexibility with auto-refresh commands, and techniques using the finer-granularity options in DDR4 devices.
4) Auto-refresh is optimized by DRAM vendors for power and performance. Therefore, schemes using explicit row-level commands to fulfill refresh requirements lose out on this efficiency. Moreover, their management overhead grows with the increasing number of rows in high density DRAM devices. Analytically, we have shown that, unless more than 70 percent of rows can skip refresh, there is no benefit to using row-level refresh for high capacity memory systems.

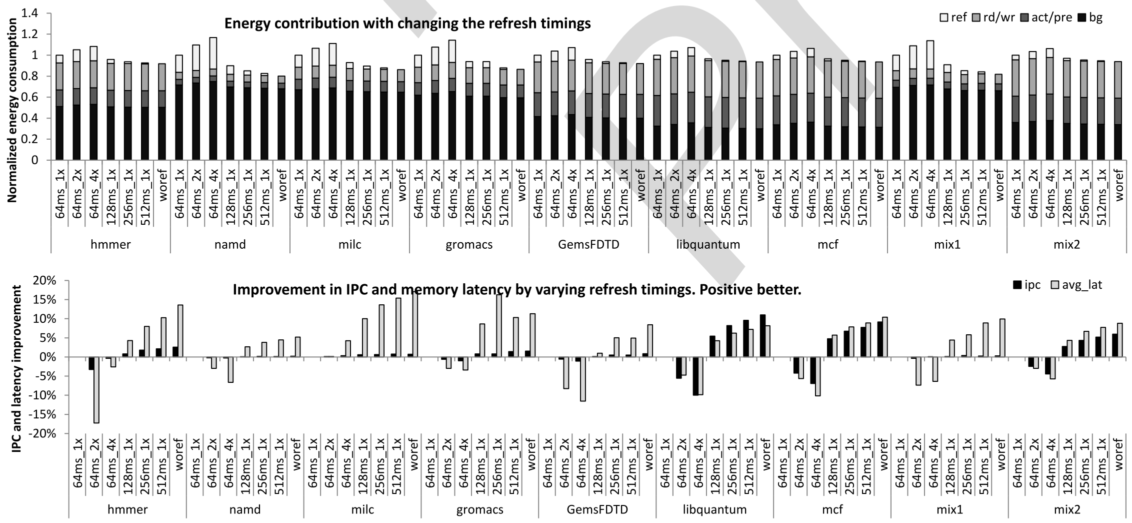


Fig. 15. Refresh penalty vs. refresh timings. We investigate the following: (i) tREFIs values decreased by (1x, 2x, 4x); (ii) various retention times (64 ms, 128 ms, 256 ms, 512 ms).

Additionally, the controller complexity, command bandwidth, and energy overhead make row-level refresh even less attractive.

5) The performance and energy penalties of refresh increase to 35 percent in near future 32Gb devices. The background energy in high density devices also increases substantially. Therefore, refresh and background power management become key design considerations. Future designs can and should use techniques available in LPDDRs (PASR, Temperature Compensated Self Refresh, etc.) without sacrificing too much performance.
6) Exploiting retention awareness of DRAM cells to reduce refresh operations can reduce refresh overhead. However, these schemes should use auto-refresh and self-refresh modes effectively, otherwise the gains obtained by retention awareness will be lost by issuing row-selective refresh commands, especially in future high density DRAMs.

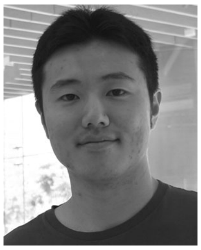
ACKNOWLEDGMENTS

The authors would like to thank David Wang and the reviewers for their valuable inputs. The research was funded in part by Intel Corporate Research Council's University Research Office, the United State Department of Energy, Sandia National Laboratories, and the United States Department of Defense.

REFERENCES

[1] Micron Technology, "Various methods of DRAM refresh," Tech. Note. TN-04-30, 1999.
[2] B. Jacob, S. W. Ng, and D. T. Wang, *Memory Systems: Cache, DRAM, Disk*. San Mateo, CA, USA: Morgan Kaufmann, 2007.
[3] JEDEC, "DDR4 standard specifications," Tech. Rep. JESD79-4, 2012.
[4] JEDEC, "JEDEC DDR3 Standard," Tech. Rep. JESD79-3E, 2010.
[5] JEDEC, "Low power double data rate 3 (LPDRR3) standard specifications," Tech. Rep. JESD209-3, 2012.
[6] Micron Technology, "Mobile LPDDR2 SDRAM," Tech. Rep. MT42L256M16D1, 2010.
[7] M. Joodaki, *Selected Advances in Nanoelectronic Devices: Logic, Memory and RF*. New York, NY, USA: Springer, 2013.
[8] T. Hamamoto, S. Sugiura, and S. Sawada, "On the retention time distribution of dynamic random access memory (DRAM)," *IEEE Trans. Electron Devices*, vol. 45, no. 6, pp. 1300–1309, Jun. 1998.
[9] K. Kim and J. Lee, "A new investigation of data retention time in truly nanoscaled DRAMs," *IEEE Electron Device Letters*, vol. 30, no. 8, pp. 846–848, Aug. 2009.
[10] D. S. Yaney, C. Y. Lu, R. A. Kohler, M. J. Kelly, and J. T. Nelson, "A meta-stable leakage phenomenon in DRAM charge storage variable hold time," in *Proc. IEDM*, 1987.
[11] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "DRAMSim2: A cycle accurate memory system simulator," *IEEE Comput. Archit. Lett.*, vol. 10, no. 1, pp. 16–19, Jan. 2011.
[12] A. Patel, F. Afram, S. Chen, and K. Ghose, "MARSSx86: A full system simulator for x86 CPUs," presented at the *Proc. 48th Design Autom. Conf.*, DAC 2011, San Diego, CA, USA, 2011.
[13] F. Bellard, "QEMU, a fast and portable dynamic translator," in *Proc. ATEC*, 2005, p. 41.
[14] M. T. Yourst, "PTLsim: A cycle accurate full system x86-64 microarchitectural simulator," in *Proc. ISPASS*, 2007, pp. 23–34.
[15] Micron Technology, "4Gb DDR3 SDRAM Datasheet," Tech. Rep. MT41J1G4, 2009.
[16] Micron Technology, "Calculating memory system power for DDR3," Tech. Note. TN-41-01, 2007.
[17] D. Kaseridis, J. Stuecheli, and L. K. John, "Minimalist open-page: A DRAM page-mode scheduling policy for the many-core era," in *Proc. MICRO*, 2011, pp. 24–35.
[18] I. Hur and C. Lin, "A comprehensive approach to DRAM power management," in *Proc. HPCA*, 2008, pp. 305–316.
[19] SPEC CPU 2006. [Online]. Available: http://spec.org/cpu2006/, 2006.
[20] G. Hamerly, E. Perelman, J. Lau, and B. Calder, "SimPoint 3.0: Faster and more flexible program analysis," in *Proc. MoBS*, 2005.
[21] Hewlett-Packard, "DDR3 memory technology," *Technology Brief TC100202TB, Hewlett-Packard*, 2010.
[22] H. David, C. Fallin, E. Gorbatov, U. R. Hanebutte, and O. Mutlu, "Memory power management via dynamic voltage/frequency scaling," in *Proc. ICAC*, 2011.
[23] J. Stuecheli, D. Kaseridis, H. Hunter, and L. K. John, "Elastic refresh: Techniques to mitigate refresh penalties in high density memory," in *Proc. MICRO*, 2010.
[24] P. Nair, C. C. Chou, and M. K. Qureshi, "A case for refresh pausing in DRAM memory systems," in *Proc. HPCA*, 2013.
[25] J. Mukundan, H. C. Hunter, K. hyoun Kim, J. Stuecheli, and J. F. Martnez, in *ISCA*, A. Mendelson, Ed. ACM, pp. 48–59.
[26] I. Bhati, Z. Chishti, and B. Jacob, "Coordinated refresh: Energy efficient techniques for DRAM refresh scheduling," in *Proc. ISLPED*, 2013.
[27] I. Bhati, "Scalable and energy-efficient DRAM refesh techniques," PhD Thesis, Dept. Electr. Comput. Eng., Univ. Maryland, College Park, MD, USA, 2014.
[28] T. Ohsawa, K. Kai, and K. Murakami, "Optimizing the DRAM refresh count for merged DRAM/logic LSIs," in *Proc. ISLPED*, 1998.
[29] M. Ghosh and H.-H. Lee, "Smart refresh: An enhanced memory controller design for reducing energy in conventional and 3d die-stacked DRAMs," in *Proc. MICRO*, 2007.
[30] C. Isen and L. K. John, "ESKIMO–energy savings using semantic knowledge of inconsequential memory occupancy for DRAM subsystem," in *Proc. MICRO*, 2009.
[31] S. Liu, K. Pattabiraman, T. Moscibroda, and B. G. Zorn, "Flikker: Saving DRAM refresh-power through critical data partitioning," in *Proc. ASPLOS*, 2011.
[32] R. K. Venkatesan, S. Herr, and E. Rotenberg, "Retention-aware placement in DRAM (RAPID): Software methods for quasi-non-volatile DRAM," in *Proc. HPCA*, 2006.
[33] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu, "RAIDR: Retention-aware intelligent DRAM refresh," in *Proc. ISCA*, 2012.
[34] J. Liu, B. Jaiyen, Y. Kim, C. Wilkerson, and O. Mutlu, "An experimental study of data retention behavior in modern DRAM devices: Implications for retention time profiling mechanisms," in *Proc. ISCA*, 2013.
[35] S. Baek, S. Cho, and R. Melhem, "Refresh now and then," *IEEE Trans. Comput.*, vol. 30, no. 2, pp. 7–15, Mar./ Apr. 2010.
[36] R. Kalla, B. Sinharoy, W. J. Starke, and M. Floyd, "Power7: IBM's next-generation server processor," *IEEE Micro*, vol. 63, no. 12, pp. 3114–3126, Dec. 2014.
[37] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable high performance main memory system using phase-change memory technology," in *Proc. ISCA*, 2009.
[38] P. G. Emma, W. R. Reohr, and M. Meterelliyoz, "Rethinking refresh: Increasing availability and reducing power in DRAM for cache applications," *IEEE Micro*, vol. 28, no. 6, pp. 47–56, Nov./Dec. 2008.
[39] W. Yun, K. Kang, and C. M. Kyung, "Thermal-aware energy minimization of 3D-stacked L3 cache with error rate limitation," in *Proc. ISCAS*, 2011.
[40] C. Wilkerson, A. R. Alameldeen, Z. Chishti, W. W. D. Somasekhar, and S.-L. Lu, "Reducing cache power with low-cost, multi-bit error-correcting codes," in *Proc. ISCA*, 2010.
[41] W. R. Reohr, "Memories: Exploiting them and developing them," in *Proc. SOCC*, 2006.
[42] X. Liang, R. Canal, G. Y. Wei, and D. Brooks, "Process variation tolerant 3T1D-based cache architectures," in *Proc. MICRO*, 2007.
[43] A. Agrawal, P. Jain, A. Ansari, and J. Torrellas, "Refrint: Intelligent refresh to minimize power in on-chip multiprocessor cache hierarchies," in *Proc. HPCA*, 2013.
[44] M.-T. Chang, P. Rosenfeld, S.-L. Lu, and B. Jacob, "Technology comparison for large last-level caches (L3Cs): Low-leakage SRAM, low write-energy STT-RAM, and refresh-optimized eDRAM," in *Proc. HPCFA*, 2013.
[45] V. G. Moshnyaga, H. Vo, G. Reinman, and M. Potkonjak, "Reducing energy of DRAM/Flash memory system by OS-controlled data refresh," in *Proc. ISCAS*, 2007.

Ishwar Bhati received BTech degree in electronics and communication engineering from Indian Institute of Technology, Guwahati, India, in 2005, and the MS and PhD degrees in computer engineering from University of Maryland, College Park, MD, in 2013 and 2014, respectively. He is currently working as a Senior Hardware engineer at Oracle, Santa Clara, CA. His research interests include energy efficient memory systems and high performance computing.



Mu-Tien Chang received the BS and the MS degrees in electronics engineering from National Chiao Tung University, Hsinchu, Taiwan, in 2006 and 2008, respectively, and the PhD degree in computer engineering from University of Maryland, College Park, MD, in 2013. He is currently a senior system architecture engineer at Memory Solutions Lab, Samsung, San Jose, CA. His research interests include memory circuit and system architecture design.

Zeshan Chishti received the BSc (Hons) degree in electrical engineering from the University of Engineering and Technology, Lahore, Pakistan, in 2001, and the PhD degree in computer engineering from Purdue University, West Lafayette, IN, in 2007. He is a research scientist at Intel Labs, Hillsboro, OR. His current research focuses on energy-efficient memory and cache design. He is a member of the IEEE.

Shih-Lien Lu received the BS degree in electrical engineering and computer science from UC Berkeley, Berkeley, CA, and the MS and PhD degrees in computer science and engineering from UCLA, Los Angeles, CA. He is currently a principal researcher and directs the Memory Architecture Lab in Intel Labs. Prior to Intel, he served on the faculty of the Electrical and Computer Engineering Department at Oregon State University from 1991 to 1999. From 1984 to 1991 he worked for the MOSIS Service as a VLSI System Manager. He is a senior member of the IEEE.

Bruce Jacob received the AB degree *cum laude* in mathematics from Harvard University, Cambridge, MA, in 1988, and the MS and PhD degrees in computer science and engineering from the University of Michigan, Ann Arbor, MI, in 1995 and 1997, respectively. At the University of Michigan, he was part of a design team building high-performance, high-clock-rate microprocessors. He has also worked as a software engineer for two successful startup companies: Boston Technology and Priority Call Management. At Boston Technology, he worked as a distributed systems developer and, at Priority Call Management, he was the initial system architect and chief engineer. He is currently on the faculty of the University of Maryland, College Park, where he is currently an associate professor of electrical and computer engineering. His present research covers memory-system design, DRAM architectures, virtual memory systems, and microarchitectural support for real-time embedded systems. He is a recipient of a US National Science Foundation CAREER award for his work on DRAM architectures and systems. He is a member of the IEEE and the ACM.