CREAM: a Concurrent-Refresh-Aware DRAM Memory Architecture*

†Tao Zhang, †Matt Poremba, †Cong Xu, §Guangyu Sun, †Yuan Xie
†The Department of Computer Science and Engineering, Pennsylvania State University
§The School of Electronics Engineering and Computer Science, Peking University
Email: {tzz106, czx102, yuanxie}@cse.psu.edu, mrp5060@psu.edu, gsun@pku.edu.cn

Abstract

As DRAM density keeps increasing, more rows need to be protected in a single refresh with the constant refresh number. Since no memory access is allowed during a refresh, the refresh penalty is no longer trivial and can result in significant performance degradation. To mitigate the refresh penalty, a Concurrent-REfresh-Aware Memory system (CREAM) is proposed in this work so that memory access and refresh can be served in parallel. The proposed CREAM architecture distinguishes itself with the following key contributions: (1) Under a given DRAM power budget, ***sub-rank-level refresh (SRLR)*** *is developed to reduce refresh power and the saved power is used to enable concurrent memory access; (2)* ***sub-array-level refresh (SALR)*** *is also devised to effectively lower the probability of the conflict between memory access and refresh; (3) In addition, novel* ***sub-array level refresh scheduling*** *schemes, such as sub-array round-robin and dynamic scheduling, are designed to further improve the performance. A quasi-ROR interface protocol is proposed so that CREAM is fully compatible with JEDEC-DDR standard with negligible hardware overhead and no extra pin-out. The experimental results show that CREAM can improve the performance by 12.9% and 7.1% over the conventional DRAM and the Elastic-Refresh DRAM memory, respectively.*

1. Introduction

In recent years, DRAM density has been improved dramatically as DRAM technology evolves along with CMOS process scaling. Recently, 16Gb DRAM chip was defined in DDR4 specification [1]. The increasing density, especially the increasing row number, introduces significant refresh penalty because more rows are required to be refreshed at a time. As a result, it takes longer time and more power to complete a refresh. The larger refresh penalty is no longer trivial as it can result in negative impact on memory performance and power. For example, significant performance degradation has been observed [2]. Consequently, the DRAM system should be carefully designed to mitigate the refresh penalty. In this work, we propose a Concurrent-REfresh-Aware Memory (CREAM) architecture that allows memory access and refresh to be executed *concurrently* under the pre-defined power constraint. The contributions of our work are summarized as follows:

- **Trade-off between power constraint and performance.** The capability of concurrent refresh in today's commodity DRAM is limited due to power (current) constraint. To relax the constraint, sub-rank-level refresh (SRLR) is proposed so that fewer banks are refreshed simultaneously to reduce the refresh power consumption. The saved power can in turn be used for memory accesses to improve performance.
- **Sub-array-level refresh.** With SRLR alone, the whole sub-rank is still locked during a refresh. To further improve the performance, a novel sub-array-level refresh (SALR) is proposed to increase the probability of memory concurrency between normal access and refresh. To our best knowledge, this is the first work to deploy SALR for the reduction of refresh penalty.
- **Concurrent refresh design.** By combining SRLR and SALR, CREAM can issue a memory access even when a refresh operation is ongoing, which means that the access and refresh can be served in parallel. In this way, CREAM can effectively hide the refresh latency overhead and meanwhile improve the performance. Moreover, a quasi-ROR interface is designed to make CREAM compatible with JEDEC-DDR standard, with negligible hardware overhead.
- **Refresh-aware memory optimization.** Two mechanisms, sub-array round-robin and dynamic refresh scheduling, are devised for further performance improvement. Sub-array round-robin evenly distributes the fine-grained refresh in a bank to avoid conflict between memory access and refresh. In addition, the concurrent refresh can be scheduled dynamically according to the status of a sub-rank.

2. Background and Motivation

To better understand the enabling techniques proposed in Section 3, the conventional DRAM architecture is reviewed

*This work is supported in part by NSF 1213052/1218867, DoE under Award DE-SC0005026, NSF China 61202072, National High-tech R&D Program of China 2013AA013201, and AMD Gift Grant.

978-1-4799-3097-5/14/$31.00 ©2014 IEEE

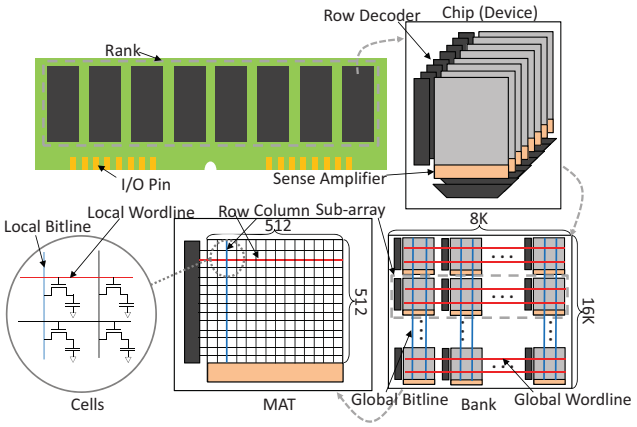


Figure 1: DRAM hierarchy – a 1Gb-8bank×8 example

in this section. In addition, the source of refresh penalty and the power constraint on refresh are briefly introduced as the motivation of this work. The ineffectiveness of two existing refresh solutions are also evaluated, which also motivates us to propose CREAM.

2.1. Conventional DRAM Architecture

Without loss of generality, the DRAM memory structure has a pyramid-like hierarchy, which, from top to bottom, consists of rank, chip, bank, sub-array, MAT, row/column and cell, as shown in Figure 1. A rank is composed of multiple memory chips (a.k.a. device) that operate in *lockstep* to feed the data bus. Inside one chip, several banks are employed as cell arrays and can be accessed independently. Usually, bank interleaving is applied to improve memory concurrency and data bandwidth. The bank is an "atomic" unit for memory access though it can be further divided into many sub-arrays. All sub-arrays share the output of the global row address decoder so that only one sub-array is allowed to be active at any time. In one sub-array, there are many MATs and each of them has its own sense amplifier array as the local row buffer. As the example shown in the figure, eight chips compose a rank to provide 64-bit data and each chip delivers 8-bit data (so-called ×8 chip). Every chip contains eight banks and each bank has 16K rows and 8K columns. Therefore, the capacity of one bank is 128Mb and one chip is 1Gb size in total.

2.2. The Source of Refresh Penalty

Due to the leakage current, the charge in each storage cell gradually dissipates and eventually the stored data can be lost as the state becomes unrecognizable. The lifetime of the data is noted as *retention time*. To prevent the data loss, refresh is required to recharge the storage cell within the retention time. In theory, a refresh scheme at any granularity is valid as long as the cell can be refreshed in time. The refresh in modern DRAM memory, however, is usually organized at rank level, which means *all chips in the rank and all banks in the chip are refreshed in lockstep*. No memory access is permitted to a rank where a refresh is undergoing. In other words, refresh and memory access are mutually exclusive to each other in the scale of rank, which causes the primary refresh penalty.

Table 1: Refresh related parameters with various DRAM capacities (based on Micron DDR3-1333 data sheet)

Capacity	1Gb	2Gb	4Gb	8Gb	16Gb
Row Num.	16K	32K	64K	128K	256K
Refresh Num.	8K	8K	8K	8K	8K
Rows/REF	2	4	8	16	32
tREFW(ms)	64/32	64/32	64/32	64/32	64/32
tREFI(µs)	7.8/3.9	7.8/3.9	7.8/3.9	7.8/3.9	7.8/3.9
tRFC(ns)	110	160	260	350	450
tRFC/tREFI	1.41% 2.82%	2.05% 4.11%	3.34% 6.68%	4.34% 8.68%	5.77% 11.54%
tRRD(ns)	6	6	6	6	6
tFAW(ns)	30	30	30	30	30
tRC(ns)	51	51	51	51	51
IDD5B(mA)	165	111	148	230	260
IDD0(mA)	65	41	47	85	95
Version	G[4]	K[5]	E[6]	D[7]	N/A

Due to the mutual exclusion between refresh and memory access, one bank can only serve one out of four memory operations at a time: activation (RAS or ACT), precharge (PRE), read/write (CAS), and refresh (REF). Consequently, all memory operations can only be issued in sequence, which we call *intrabank-zero-parallelism*. The intrabank-zero-parallelism requires memory operations to wait if a refresh is being processed. Moreover, since all banks are refreshed simultaneously, refresh further suppresses bank-level parallelism, which we denote as *interbank-zero-parallelism*. The combined effect of intrabank- and interbank-zero-parallelism mandates all memory accesses to be delayed until the refresh completes. Considering the refresh should be taken periodically, an application may experience a long, periodic delay, which can incur significant performance degradation.

In JEDEC-DDR standard [1, 3], three timing parameters are used to define the refresh characteristics of a DRAM chip.

tREFW is the refresh window that refers to the cell retention time.

tREFI is the time interval of two REF commands. Since distributed refresh is applied, the value of tREFI is determined by tREFW and the refresh count in a tREFW.

tRFC is the refresh latency of one REF command, which is also known as refresh cycle.

Table 1 shows the values of these three parameters as the chip capacity ranges from 1Gb to 16Gb[1]. According

[1]The values of 16Gb DRAM has not been defined in JEDEC DDR4 specification. The data listed in the table is based on our projection ac-

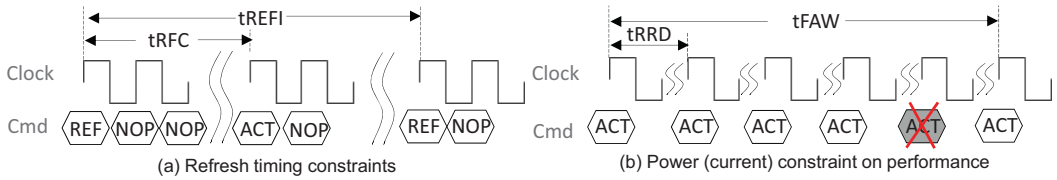


Figure 2: Refresh basics. (a) Refresh command and related timing constraints. (b) impact of power constraint on performance.

to the operating temperature, tREFW can be either 64ms (Temp≤85°C) or 32ms (85°C<Temp<95°C). From the table, the refresh count remains at 8,192 even though the memory capacity increases. Therefore, tREFI sustains at 7.8μs (3.9μs) regardless of the DRAM evolution. Figure 2a shows the basic timing constraints of tREFI and tRFC.

From Table 1, it is clear that the refresh cycle tRFC dramatically increases as the chip capacity keeps growing. Correspondingly, the refresh overhead that is defined as tRFC/tREFI becomes larger, which means the following memory accesses should wait for a longer time. The large performance degradation has been observed in [2] and we also see the similar result. Figure 3a illustrates the potential performance gain if there is no refresh in DRAM. The average speedup for all benchmarks and memory-intensive benchmarks is 6.3% and 11.2% with 3.9μs refresh rate, respectively. An interesting observation from Figure 3a is that some benchmarks, such as `gobmk` and `libquantum`, have larger performance drop (15.7% and 14.3%) than the maximum refresh overhead (11.54%), which indicates the refresh generates accumulative effect on an application and therefore aggravates the performance degradation.

2.3. Power Constraint on Refresh

In addition to the aforementioned refresh penalty caused by intrabank- and interbank-zero-parallelism, power (current) constraint is the secondary factor that necessitates the mutual exclusion between refresh and memory access. Even though the refresh power constraint is not detailed in JEDEC standard and DRAM vendors' data sheet, we leverage the well-known power constraint on activation to help the explanation.

To ensure that the peak current consumption does not exceed the pre-defined threshold, a DRAM chip prohibits very frequent activations. Correspondingly, two timing parameters are used to limit the `ACT` frequency. tRRD is known as activation-to-activation delay, which determines the minimum interval between two successive `ACT`s. Moreover, the four-activation window constraint tFAW only allows four `ACT`s to be issued in any tFAW cycles. Figure 2b shows how tRRD and tFAW limit DRAM performance. Once the MC issues an `ACT` to open a row, the next `ACT` can only be issued after tRRD cycles. Since tFAW is usually greater than the sum of four tRRDs (see Table 1), the fifth `ACT` (highlighted in gray) can only be issued after tFAW cycles even if it satisfies tRRD constraint. Therefore, tFAW constrains the performance in a DRAM system that the *close-page* row buffer management policy is applied [8]. As CREAM aims at the server memory that close-page policy is commonly deployed, tFAW effectively limits CREAM's performance as well.

Since a refresh can be simply considered as a pair of activation and precharge (see Section 3), it should abide with the same power constraints too. From Table 1, a refresh consumes about $3X$ currents (IDD5B) than an activation (IDD0), which means it almost reaches the peak current. As a result, no activation can be issued during a refresh.

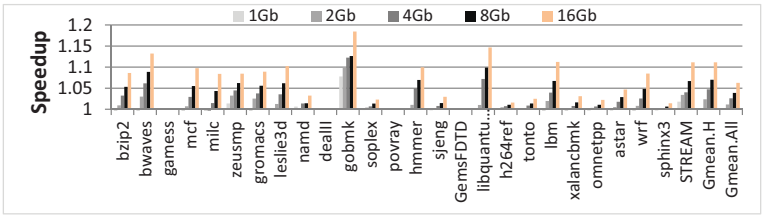
2.4. The Ineffectiveness of Existing Solutions

To mitigate the refresh penalty, the new DDR4 standard [1] proposes a fine-grained refresh scheme, in which refresh frequency can be two (2× mode) or four times (4× mode) higher than the original refresh rate (1× mode). Since more refreshes are involved, fewer rows need to be refreshed during a single refresh. In this way, the fine-grained refresh can reduce tRFC so that the waiting memory accesses can be served earlier to improve performance. Table 2 lists the timing changes when fine-grained refresh is applied[2]. We implement this fine-grained refresh in DRAMSim2 [10] and select some memory-intensive benchmarks from SPEC2006 suite to evaluate its effectiveness (see Section 5 for the detail of simulation platform). As shown in Figure 3b, on average 2× and 4× mode can only achieve 3.9% and 6.4% performance gain, respectively, which is consistent with JEDEC report [9]. Therefore, the proposed refresh scheme is ineffective to eliminate the refresh penalty, still leaving a big performance gap to be filled.

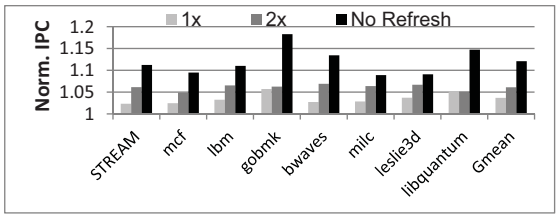
The main problem of such fine-grained refresh is that it only reduces the waiting time caused by intrabank- and interbank-zero-parallelism rather than eliminate either of them. As a result, all banks are still locked during the refresh even though the refresh cycle becomes smaller. Moreover, the higher refresh rate adversely affects the performance as well. Compared to the 1× mode where an application only needs to wait for one tRFC, the application

cording to the trend from 1Gb to 8Gb.

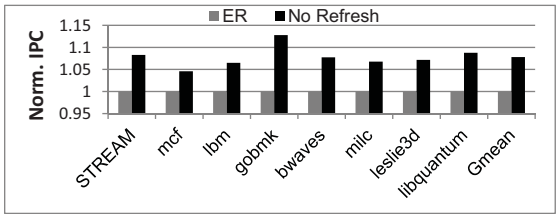
[2] 8× mode is mentioned in [9] but not defined in [1].



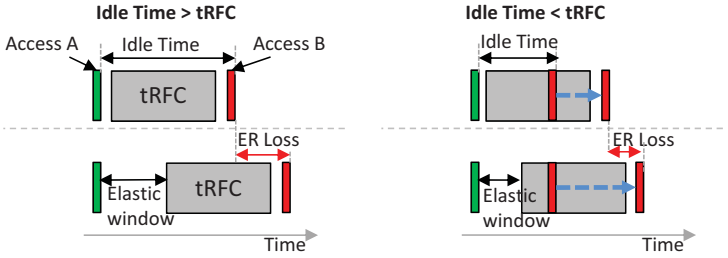
(a) Performance gain when no refresh is applied



(b) Ineffectiveness of fine-grained refresh defined in DDR4 [1]



(c) Ineffectiveness of elastic refresh in [2]



(d) The cause of ER's failure

Figure 3: The motivation experimental results. (a) performance gain without refresh as chip size ranges from 1Gb to 16Gb; (b) the ineffectiveness of fine-grained refresh in DDR4; both 2× and 4× modes are shown by the normalized IPC to the baseline 1× mode; (c) the ineffectiveness of ER; (d) cause of ER's failure.

running with the fine-grained refresh scheme can experience multiple refreshes in the same tREFI. The accumulative refresh penalty, which is calculated as $N_{stall} \times tRFC_{2\times}$ ($tRFC_{4\times}$) where N_{stall} is the maximum stall number, could be larger than the single refresh cycle $tRFC_{1\times}$ and thus even induce performance degradation in ×8 mode as shown in [9].

Table 2: Settings for fine-grained refresh in DDR4 [1]

	1×	2×	4×	8×
tRFC(ns)	350	260	160	75
tREFI(μs)	3.9	1.95	0.975	0.4875

Another famous refresh scheduling policy has been proposed in [2], which is called *elastic refresh* (ER). ER leverages the 8-tREFI refresh scheduling flexibility defined in DDR standard to hide the refresh penalty. Different from prior work that defers a refresh until the memory is idle (DUE) [11], ER leaves an extra elastic window and expects that there are incoming memory accesses in the window. If so, ER further defers the refresh until either no access comes up during the window or it has deferred eight refreshes and thus hits the 9-tREFI refresh limitation. As the window width is critical to ER's performance, ER has capability to adjust the window width in-the-flight based on the average idle time and the number of deferred refreshes. We also implement this approach to measure the performance improvement. Figure 3c illustrates the results. Unfortunately, no result can be comparable to the ideal case where no refresh is applied. On average, there is 7.8% performance gap between ER and the ideal case. In particular, the potential improvement room can be as much as 12.1% (*gobmk*). Similar to the aforementioned fine-grained refresh, ER also renders the ineffectiveness to eliminate the refresh penalty.

The reason of ER's failure can be simply explained as follows. ER uses the average idle cycles as window width for the possible coming access. The average idle cycles calculated by arithmetic mean, however, means only half of accesses that have smaller interval than the idle cycles can benefit from the elastic idle time while the other half can't. Figure 3d shows two examples for the performance loss caused by ER. Firstly, let's assume 1) the elastic window starts after access A (green block) as memory becomes idle; and 2) access B (red block) is too far away from A to catch the window. As a result, as shown in the left part of Figure 3d, a deferred refresh is launched after the window. In this way, ER adversely destroys the potential performance gain since the idle time between A and B is actually long enough to hide a refresh. On the other hand, the right part gives another example where the idle time between A and B is shorter than tRFC. Consequently, B should wait for a while (blue dotted line). With ER, B needs to wait for a longer time because of the late launch of the refresh after the elastic window. As a result, even though ER can allow some memory requests to be served without stall, it may also add additional delay to other accesses, which in turn offsets its effectiveness.

Since the existing refresh policies still leave a big room for further performance improvement, CREAM is proposed as an essential solution to effectively reduce the refresh penalty. The goal of CREAM is to hide the refresh penalty

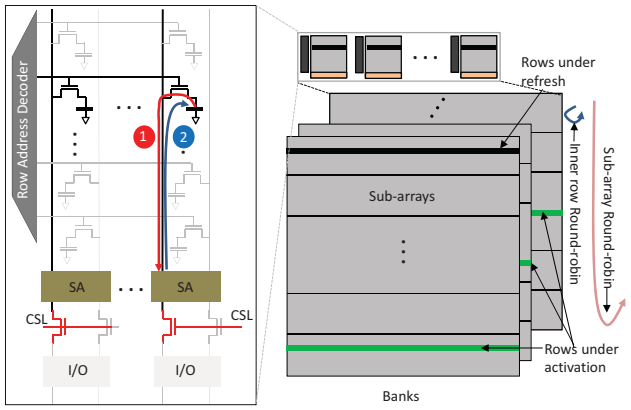


Figure 4: Enabler of concurrent refresh. Left: a refresh in a sub-array is a two-stage operation. The row is firstly open to recharge the cell and then the row is closed by a precharge. Right: a refresh and an activation can be performed concurrently in any banks as long as no sub-array conflict occurs

and thus perform like a "non-refresh" DRAM.

3. The CREAM Architecture

As shown in Figure 4 (left), a basic refresh is a two-stage operation. First, the refresh row is opened to load the data from the cell to the sense amplifier (SA). The cell is then restored to logic '0' or '1'. Once the restoration completes, DRAM moves to the second stage, in which a pre-charge is applied to close the row by resetting the bitline and SA for the following memory accesses. Note that the column select signal CSL is turned off so that the refresh is invisible for the rest of the system. In this way, a refresh can be treated as a pair of activation and pre-charge. The two-stage refresh implies an important feature that CREAM leverages: **the range of data movement during a refresh is limited between a sub-array and the corresponding local SA so that it does not induce resource contention with other memory accesses outside the sub-array** (e.g., the contention on global row buffer, I/O gating logic, or I/O bus). Consequently, it provides an opportunity for a bank to serve a refresh and memory access in parallel as long as they can be isolated from each other.

3.1. Sub-Array-Level Refresh (SALR)

According to Figure 4, one sub-array has a local (dedicated) bitline and local SA array so that it has the potential to complete the refresh without competing for the sharing resources with other sub-arrays. Therefore, we develop a technique called sub-array-level refresh (SALR) to eliminate the intrabank-zero-parallelism. Because of the sharing of row address logic, however, the sub-array-level refresh is not available in conventional DRAM design. To enable SALR, small modification is required and Section 4 will present the design detail.

SALR can significantly reduce refresh penalty due to the low probability of refresh conflict, which in turn indicates the high probability of concurrent refresh that helps hide refresh penalty. For any memory access, the probability of sub-array conflict can be presented as Eq.1, where N_{SA} is the number of sub-arrays in one bank. It is straightforward that the probability is inversely proportional to the number of sub-arrays if the memory access is uniformly distributed in the bank. Note that N_{SA} may not necessarily be equal to the real number of sub-arrays in a bank because multiple sub-arrays can group together as a single sub-array. For instance, N_{SA} can be only eight in a 4Gb bank that has 128 sub-arrays (64K rows with 512 rows per sub-array). According to Eq.1, the probability of a sub-array conflict is only 1.56% as 64 sub-arrays are employed. Thanks to the low probability, sub-array conflict can be effectively reduced to guarantee the high refresh concurrency.

$$P^{1bank}(N_{SA}) = \frac{1}{N_{SA}} \quad (1)$$

3.2. Sub-Rank-Level Refresh (SRLR)

The aforementioned fine-grained refresh does not take into account the limited current budget of DRAM. Since all memory operations must abide with the power constraint, no memory access can be issued if all banks are refreshed simultaneously, which in turn kills the memory concurrency. Therefore, the only way to enable concurrent refresh is to reduce the current consumption by a refresh. Based on our observation from [12], **the total current consumed by a refresh is proportional to the number of banks that are refreshed simultaneously**[3]. As a result, it is straightforward that reducing the number of banks that are refreshed together can save power accordingly. CREAM employs this idea to develop a sub-rank-level refresh scheme (SRLR). The original rank with eight banks are further divided into two (SR2), four (SR4) or eight (SR8) sub-ranks. As a consequence, each sub-rank has four, two or one bank that are refreshed simultaneously. As fewer banks are involved in a refresh, CREAM makes use of the saved current to enable memory access in parallel. In this way, SRLR relaxes or even eliminate the interbank-zero-parallelism because it is no longer necessary to lock all banks during the refresh.

3.3. The Combination of SALR and SRLR

The combination of SALR and SRLR provides CREAM high flexibility of fine-grained refresh due to the significant alleviation of intrabank- and interbank-zero-parallelism limitations. Once a sub-rank is being refreshed, memory accesses can be issued to any sub-ranks as long as there is no sub-array conflict. To obey the power constraint, power

[3]Even though the relation between current consumption and bank number is given by partial-array self refresh (PASR), we believe the trend is also applicable to auto refresh in standard DDR SDRAM.

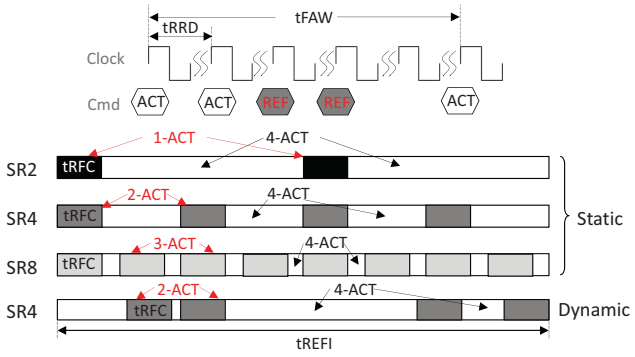


Figure 5: The different capabilities of concurrency in SR2, SR4 and SR8 refresh. A refresh in SR4 occupies two slots in the four-activation window based on current consumption. As a result, only two ACTs are available during a refresh. Similarly, one and three ACTs are available in SR2 and SR8, respectively.

should be reassigned in the new architecture. In this work, we conservatively assume that halving bank number can save current for one activation. Based on the assumption, a refresh in SR2, SR4 and SR8 consumes the same current as three, two, and one normal ACTs[4]. As a result, these three refresh schemes have distinct capabilities to compete with ACT in a window, which is illustrated in Figure 5. For example, as a refresh in SR4 is identical to two ACTs in terms of current consumption, it reserves two slots for refresh and thus leaves another two ACT slots for the memory accesses, which is noted as 2-ACT. Similarly, 1-ACT and 3-ACT are available in SR1 and SR8, respectively.

Nonetheless, fewer banks per refresh require more refreshes in a tREFI. As shown in Figure 5, only two refreshes are required in SR2 while eight refreshes are executed in SR8. The more refreshes may induce performance drop as DDR4 does (see Section 2.4). Thanks to SALP, the probability of sub-array conflict in a sub-rank, which is shown as Eq.2, is low enough to effectively reduce the conflict. In Eq.2, N_{SR} is the number of sub-ranks and N_B is the number of banks in a rank. When there are eight sub-ranks and one bank has 64 sub-arrays, the probability of sub-array conflict is only 0.2%, which implies high possibility of concurrent refresh.

$$P^{subrank}(N_{SR}) = \frac{1}{N_{SR}} \times (1 - (1 - P^{1bank}(N_{SA}))^{\frac{N_B}{N_{SR}}}) \quad (2)$$

3.4. Refresh-aware Optimization

Sub-array round-robin. The probability calculation given in Eq.1 and Eq.2 assumes that the memory accesses are evenly distributed in a bank. In the reality, however, memory accesses are not evenly distributed. Instead, the spatial locality is more or less reflected in the access pattern. In addition, row bits are usually placed in the most significant bits (MSBs) in physical address mapping to augment either row buffer hit rate or bank-(rank-) level parallelism (or both). As a result, it is more likely memory accesses concentrate in a sub-array rather than scatters over the whole bank. To further avoid the sub-rank conflict, CREAM prioritizes the round-robin among sub-arrays over the inner row round-robin within a sub-array (see Figure 4). Once a refresh has been completed in a sub-array, next refresh will move to next sub-array rather than next row in the same sub-array. In this way, CREAM can control the sub-array conflict as a rare case.

Dynamic refresh scheduling. The first three fine-grained refreshes shown in Figure 5 adopt *static* refresh scheduling policy. The meaning of "static" is two-fold. First of all, all sub-ranks are refreshed in order (1-..-N_{SR}). In addition, the refresh is distributed evenly in a tREFI so that the time interval between two sub-rank refreshes can be calculated as tREFI/N_{SR}. The drawback of static refresh scheduling is similar to the baseline refresh scheme because MC initiates a refresh regardless of the memory status. When many memory accesses are queuing in the MC, static refresh may degrade the performance as it wastes the precious ACT bandwidth. Alternatively, a *dynamic* refresh scheduling policy can be employed to address this issue.

According to the status of each sub-rank, the refresh can be executed out of order so that the idle sub-rank is refreshed at first. In this way, the dynamic refresh scheduling policy can further hide the refresh penalty at the sub-rank level. Since all sub-ranks must be refreshed within a tREFI, the sub-rank that is always busy will be forced to be refreshed at the end of a tREFI. In this work, MC simply checks the command queue to obtain the sub-rank status. Once the queue has few memory accesses, MC assumes the sub-rank is idle and starts refreshing it. Section 5 will evaluate the effectiveness of the dynamic refresh scheduling policy.

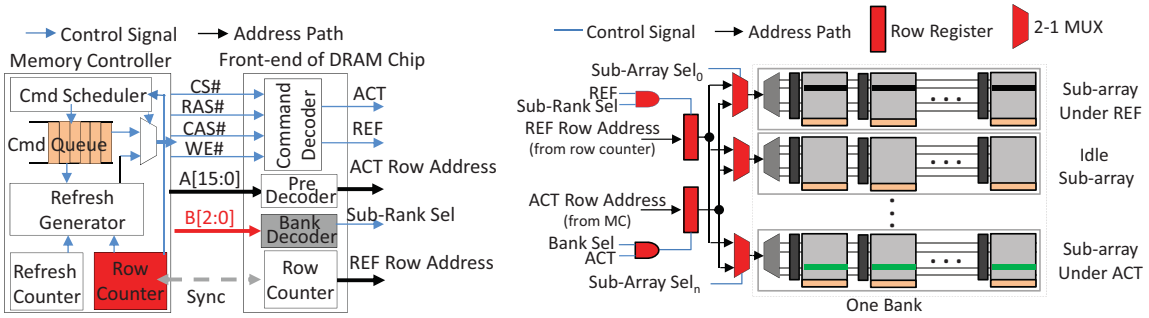
4. The Design of CREAM

We present detail designs for CREAM architecture in this section, with descriptions of protocol changes, hardware modification, and overhead analysis.

4.1. Interface Protocol Change

To support dynamic scheduling, some modification should be made on the memory interface. As shown in Figure 6a, the original auto refresh that is also known as *CBR* (CAS#-Before-RAS# Refresh) should be changed to quasi-*ROR* (RAS#-Only Refresh). In CBR, all address signals, including bank ID (BA[2:0]), are ignored as "Don't Care" bits because the row address is generated by the internal row counter in DRAM chip. In contrast, original ROR

[4]Even though the reassignment is inaccurate, it is close to the truth. The sensitivity study is given in Section 5.4 by relaxing the power assignment.



(a) Interface protocol modification

(b) Row logic modification

Figure 6: Hardware modifications to enable SALR and SRLR. All additional logics are highlighted in red. (a) Quasi-ROR is employed to put sub-rank ID on BA[2:0]; (b) Two registers are used to separate refresh row address and activation row address.

requires the MC to maintain the row counter and explicitly send the refresh row address to DRAM chip. CREAM adopts the mix of ROR and CBR so that sub-rank ID is put on BA[2:0] along with `REF` command but the address bits are still "Don't Care" (so-called quasi-ROR), as illustrated in Figure 6a. DRAM chip reuses the bank decoder to decode the sub-rank ID and then the generated 'sub-rank sel' signal is delivered to the target sub-rank. The row address is still maintained by the internal row counter.

On the other hand, the MC of CREAM needs to know the current refresh row for the avoidance of sub-array conflict. Therefore, a replica of row counter must be deployed in MC and used by the command scheduler. Note that one rank has a dedicated row counter in MC. The row counter keeps counting even if some ranks are in self refresh mode. In this way, the synchronization (highlighted by dotted gray line) between these two row counters is established to make sure no refresh violation occurs[5].

In addition, this quasi-ROR interface can be easily extended to an ROR interface to support the partial array auto refresh (PAAR) At this time, MC puts both bank ID and row address to tell a sub-rank which sub-array should be refreshed.

4.2 The Enabling Technology for SALR

Figure 6b illustrates the modification to conventional DRAM structure to enable SALR. Two row address registers are added to isolate the refreshing row address from the activation address. In addition, a multiplexer array is employed to select the row address for the purpose of either activation or refresh. Once the refresh command `REF` comes in, the dedicated register is enabled to store the refresh row address from the internal row counter. In addition, the corresponding multiplexer is set to select the output of the refresh register. The target sub-array uses the latched refresh row address to complete the refresh in tRFC cycles. During the refresh, the signal CSL (see Figure 4) is turned off so that the refresh is isolated from the rest system. As a result, any other sub-arrays can serve normal `ACT`, `CAS` and `PRE` by using the activation row register.

Note that the proposed architecture is much simpler than the one in SALP [13], where multiple registers are employed for the parallel activation. Instead, only one register is dedicated to latch the row address for an `ACT` command, which means that **at anytime only one sub-array can be activated**. As a result, CREAM can completely reuse the legacy logics in conventional DRAM to serve an `ACT/CAS/PRE` under various timing constraints. Given a register costs much more area than a multiplexer, the simplicity also reduces the area overhead due to the use of less registers. In addition, CREAM has better scalability than SALP. The number of register in CREAM remains at two while the number increases in SALP, if more sub-arrays are parallelized.

4.3. Design Overhead Analysis

Overhead on DRAM Chip. should be carefully considered as DRAM cost is highly sensitive to the area and power. In this work, we implement the additional logics shown in Figure 6b by Verilog HDL and then synthesize the design by Synopsys Design Compiler [14] with TSMC 45nm low-power technology for the area and power analysis. For the comparison, the extra logics introduced in SALP are also synthesized. All implementations are applied to a 2Gb DRAM chip that has 32K rows (or 64 sub-arrays) and the address register is consistent with SALP, which is 40-bit wide.

As shown in Table 3, when both have eight sub-arrays (SA8), CREAM only consumes 786μm^2 area and 427μW power while SALP has to consume 2,325μm^2 area and

[5]A safer but more expensive synchronization scheme is to make the row counter inside DRAM chip visible to MC so that MC can access it immediately after the exit of self refresh, which is the same as MC accesses a mode register.

1,311μW power. On the other hand, as the number of sub-array increases, CREAM presents much better scalability than SALP as well. For example, the area overhead of SALP almost doubles as the sub-array numbers increases to 16. In contrast, CREAM that employs 64 sub-arrays (SA64) has moderate area and power increase due to the simple pass-transistor-based implementation of multiplexor array. Given a 50mm^2 DRAM die with 64 sub-arrays, the overall area overhead can be calculated by 1,975μm^2×8banks, which is only 0.016mm^2 (0.032%). Therefore, the area overhead on the DRAM chip is negligible.

Furthermore, as summarized in [15, 16], it is more costly when the modification is applied to the bitline and sense amplifier while it is less costly when the change is conducted in the row logic and I/O. Considering SALP changes the bitline layout but CREAM only puts additional hardware around row logic and never touches the DRAM core, we believe the architecture of CREAM introduces much smaller hardware overhead than that of SALP.

Table 3: Synthesis Result Comparison for a 2Gb DRAM Chip

	CREAM				**SALP** [13]	
	SA8	SA16	SA32	SA64	SA8	SA16
Power(μW)	427	528	613	**770**	**1,311**	2,609
Area(μm^2)	786	1,007	1,316	**1,975**	**2,325**	4,651

Interface design. The only modification at the interface is the reuse of the link bank ID (BA[2:0]) that was ignored before. No additional pin is introduced in CREAM. As a result, CREAM is completely compatible with the conventional JEDEC-DDR interface protocol. Furthermore, if the dynamic scheduling and PAAR are not employed, no change is even needed to the interface. Also note that the new row counter in MC has fewer bits than the peer in DRAM chip because MC is only aware of sub-array ID. As the maximum number of sub-arrays in a single DRAM chip is 128, the row counter for one bank is a 7-bit register. Given a four-rank DRAM system, the overall overhead is a 40bit register (=4ranks×(7bit row counter + 3bit sub-rank ID)), which is also negligible. In contrast, Smart Refresh [17] needs 768KB storage for the row counters and RAIDR [18] requires at least 1.25KB storage for the bloom filter. As a result, CREAM is compelling in terms of area overhead.

5. Evaluation Results

In this section we describe the evaluation setup and the experimental results for CREAM design.

5.1. Evaluation Environment

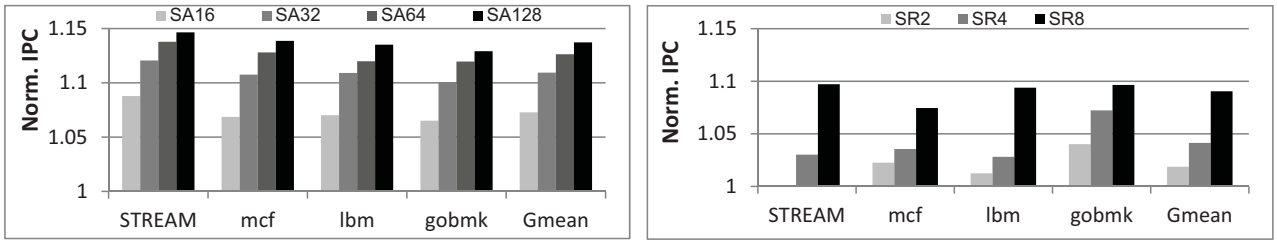
In this work, gem5 [19] is used as our simulation platform. Instead of the time-consuming full-system (FS) simulation, system-call emulation (SE) mode is adopted for the sake of simulation speed. The well-known DRAMSim2 simulator [10] is integrated into gem5 and modified to implement the proposed concurrent refresh schemes. Table 4 shows the gem5 setup. The selected SPEC2006 CPU benchmark with reference input size [20] and STREAM with all functions [21] are evaluated as multi-programmed testbench. We run all benchmarks for 500 million instructions to warm up the cache and then the following 100 million instructions for the statistics. The instructions-per-cycle (IPC) is used as the performance criteria through the evaluation. All timing parameters are excerpted from Micron's data sheet [4, 5, 6, 7] and the refresh-related parameters are listed in Table 1. As CREAM targets at the massive memory system used in server or datacenter where the operating temperature is usually greater than 85°C [22, 23], all simulations are run under extended temperature range, where tREFW=32ms and tREFI=3.9μs.

Table 4: Simulation Platform Configuration

Cores	1/4, ALPHA, out-of-order
CPU Clock Freq.	3 GHz
LDQ/STQ/ROB Size	32 / 32 / 128 entries
Issue/Commit Width	8 / 8
L1-D/L1-I Cache	32kB / 32kB 4-way 2-cycle latency
D-TLB/I-TLB Size	64 / 48 entries
L2 Cache	Shared, Snooping, 1MB, LRU 8-way, 10-cycle latency
Memory	JEDEC-DDR3, 4GB, 64bit I/O bus, 8 banks(×8), 666MHz(DDR-1333), tRCD-tCAS-tRP-tWR 10-10-10-10, address map: row-column-bank-rank,

Two refresh scheduling schemes, **immediate refresh** (IR) and **elastic refresh** (ER) are implemented in DRAMSim2 as the references. IR immediately initiates a refresh command once the refresh counter is timed up. This is the baseline since it simulates the refresh scheduling that is widely used in modern MCs. As mentioned in Section 2.4, ER devised in [2] defers the refresh for an elastic window and expects some memory accesses will come in. The window width is dynamically tuned according to the calculated average idle cycle and number of deferred refreshes. In addition, the memory with **no refresh** is also evaluated as the ideal case.

According to the motivation results shown in Figure 3a, we classify the benchmarks into three categories and select four benchmarks as the representatives for each category, which are symbolized as *H*, *M*, and *L* for high (¿10), medium ([1,10]), and low miss per kilo instructions (MPKI) of last level cache (LLC). Each benchmark is assigned a number for the simplicity as shown in Table 5. Both single-core and four-core simulations are done for the evaluation.



(a) Sub-array (normalized to SA8)

(b) Sub-rank (normalized to no sub-rank)

Figure 7: Design space exploration for sub-rank and sub-array number.

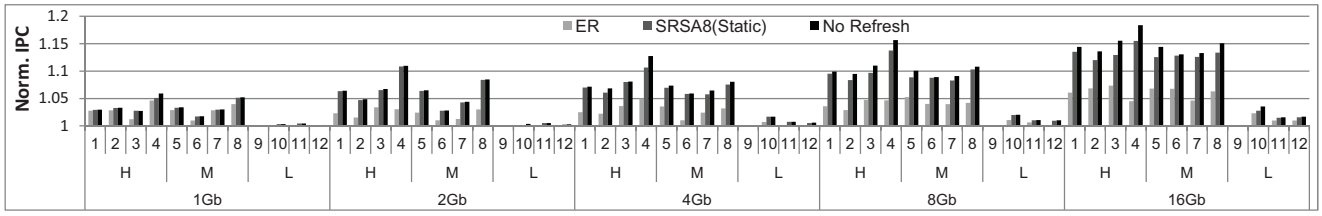


Figure 8: One-core simulation results with 4GB memory. All results are normalized to IR.

Table 5: Benchmark Classification and Model Settings

	#**Benchmarks(MPKI)**
H	[1]STREAM(34.96), [2]mcf(16.26), [3]lbm(31.92), [4]gobmk(38.35)
M	[5]bwaves(6.07), [6]milc(5.13), [7]leslie3d(4.30), [8]libquantum(6.95)
L	[9]gamess(0.02), [10]namd(0.12), [11]h264ref(0.35), [12]sphinx3(0.09)
Memory Models (4GB, DDR3-1333)	
1Gb	Micron MT41J128M8[4], 8B(bank)×8, 4-rank
2Gb	Micron MT41J256M8[5], 8B×8, 2-rank
4Gb	Micron MT41J512M8[6], 8B×8, 1-rank
8Gb	JEDEC-DDR4[1], 8B×8, 1-rank
16Gb	JEDEC DDR4[1], 16B×8, 1-rank
Concurrent Refresh Symbols	
SA*x*	sub-array setting, *x*: 8/16/32/64/128
SRLR*y*	sub-rank setting, *y*: 2/4/8, SRLR only
SRSA*y*	sub-rank setting+SA64, *y*: 2/4/8, SALR+SRLR

We simply duplicate four copies of each benchmark for the four-core simulation. In addition, five 4GB memory models are simulated with various chip capacities. In particular, 1Gb (2/4Gb) memory is the ordinary Micron DDR3-1333 that has four (two/one) ranks and eight banks. 8Gb memory leverages DDR4 timing parameters even though it has only 8 banks. Finally, 16Gb memory refers to DDR4-1333 that has 128K rows and 16 banks[6]. On the other hand, concurrent refresh with different sub-array and sub-rank numbers are noted as **SA*x*** and **SRLR*y***, where *x* stands for the sub-array number that can be 8-128, while *y* is the sub-rank number that can be 2/4/8. Moreover, **SRSA*y*** stands for a memory system that both SRLR and SALR are applied. Table 5 gives the comprehensive information about the memory models and the symbol of concurrent refresh with different settings.

[6]In fact, the data rate 1333MHz is not supported in [1]. We simply double the bank number to simulate an equivalent DDR4 module, such as DDR4-1600.

5.2. Design Space Exploration

Before the performance evaluation, we conduct a design space exploration to determine the optimal sub-rank and sub-array numbers. Only class *H* benchmarks are employed for the evaluation. We first sweep the sub-array number from 8 (SA8) to 128 (SA128) to find the optimal sub-array number. All results in Figure 7a are normalized to SA8. As shown, SA16 and SA32 can achieve 7.2% and 10.4% performance improvement on average than SA8. In addition, SA64 can further outperform SA32 with 3.1% performance gain. However, only 1% benefit is obtained as the sub-array number increases from 64 to 128. As a result, either SA32 or SA64 can be used as the optimal solution. Without special statement, SA64 is applied in all following experiments (SA32 is used in 1Gb simulation because it is the maximum number of sub-array).

With the optional sub-array number, the result of sub-rank exploration is given by Figure 7b. In general, SR8 has the highest speedup (9.7%) over the baseline. SR2 and SR4 have less than 5% speedup due to the insufficient memory concurrency. In particular, SR2 even incurs 1.7% performance drop in `STREAM`. The reason is that SR2 has two refreshes in a tREFI. The benchmark experiences two refresh stalls that can offload the limited performance gain from 1-ACT concurrency.

5.3. Single-Core Simulation Results

Single-core simulation is done with the optional sub-rank and sub-array numbers given above to evaluate the effectiveness of concurrent refresh. Two messages can be

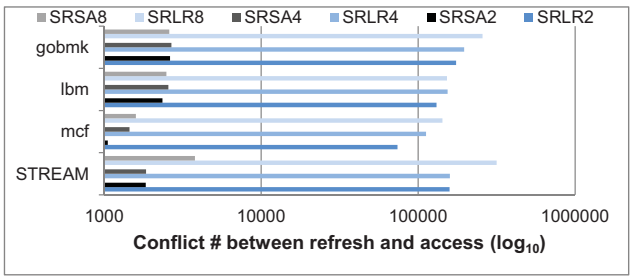


Figure 9: Refresh conflict number comparison between SRLR-only and SRSA (SRLR+SALR)

taken according to the result presented in Figure 8: 1) the three categories render distinct sensitivity to refresh overhead. The refresh has significant impact on *H* and *M* while the *L* class that has less memory intensity is almost immune to the increasing refresh penalty; and 2) concurrent refresh (SRSA8) can be comparable to the ideal case (around 99% of No Refresh). On average, SRSA8 can outperform IR and ER by 9.7% (12.9%) and 6.6% (8.1%) in 8Gb (16Gb) memory system, respectively.

Proof of Effectiveness of SALR. To verify the effectiveness of SALR, we collect and compare the number of conflict between refresh and memory access in both SRLR-only and SRSA models. For the sake of accuracy, only those memory accesses that satisfy the power constraint are counted. Since SRLR-only model only has sub-rank-level parallelism, the value actually reflects the number of sub-rank-level conflict. In contrast, sub-array-level conflict is collected in SRSA models. As shown in Figure 9, SRSA model can achieve 9*X*-90*X* conflict reduction over SRLR-only model with only 0.1% sub-array conflict. An interesting observation is that SRSA can effectively limit the conflict number with modest increase as sub-rank number increases from 2 to 8. Moreover, there is even a small decrease in `lbm`(3) and `gobmk`(4). Alternatively, the conflict in SRLR-only system dramatically increases when more sub-ranks are deployed. Consequently, all benchmarks have performance drops when the sub-rank numbers changes from two to eight. The reason for this phenomenon is similar to the DDR4 fine-grained refresh (see Section 2.4) that more refreshes are invoked when sub-rank number increases. As a result, it is possible for a benchmark to experience multiple stalls due to the more frequent refreshes. Therefore, SRLR-only model cannot work well and SALR is mandatory for the success of concurrent refresh.

Effectiveness of dynamic refresh scheduling Figure 11a presents the effectiveness of dynamic refresh scheduling scheme. The geometric mean value is shown in the figure. Once the dynamic scheduling is applied to SRSA4 (SRSA4(D)), it can obtain 4.3% performance improvement than the other SRSA4 model that employs static scheduling (SRSA4(S)). The result is even close

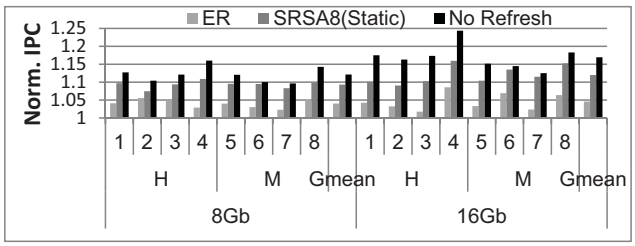


Figure 10: Four-core simulation with 8GB memory. All results are normalized to IR.

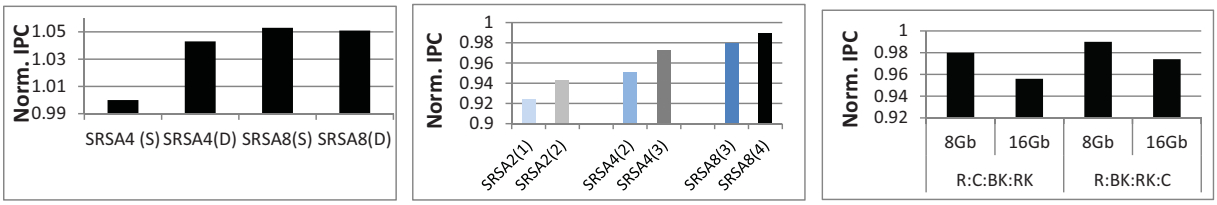
to SRSA8(S). However, no further gain is observed in SRSA8(D). As shown in Figure 5, SRSA8 has eight refreshes so that it leaves little room to dynamically schedule these refreshes.

5.4. Sensitivity Study

Core number and memory size To measure how much gain CREAM can accomplish in a multi-core system, four-core[7] simulations are run under 8GB memory. Only *H* and *M* class along with 8Gb and 16Gb models are used to mimic a heavy-workload environment. The benchmarks in *H* and *M* are simply duplicated for four copies and assigned to each core as multi-programmed simulation. From Figure 10, it is clear that CREAM can achieve more performance gain than that of ER. In general, CREAM has 9.3% and 11.9% improvement than IR while ER can only get 4% and 4.5% improvement, respectively. The gap between CREAM and No Refresh, however, becomes bigger (-2.1% for 8Gb and -4.3% for 16Gb) due to the increased memory intensity (MPKI is about four times larger).

Relax of power constraint. The compromise between performance and power always exists in DRAM system. Even though the battle will continue, we could anticipate the current consumption for sub-rank refresh can be lower than our conservative settings. We relax the power constraint so that two, three, and four `ACT`s can be issued in SRSA2, SRSA4, and SRSA8, respectively. The last configuration that enables concurrent refresh with normal four-activation window constraint is interesting. As long as no sub-array conflict occurs, the refresh can be completely hidden in the background without introducing any refresh penalty. We name it *background refresh* as the solution of concurrent refresh in the near future. Figure 11b illustrates the `ACT` number configuration (in the parenthesis) and simulation result with 8Gb model. The performance can be dramatically improved by relaxing the power constraint for all benchmarks. Specifically, the background refresh (SRSA8(4)) can almost perform the same as No Refresh (98.9%). From this study, we can learn that power constraint is crucial for the success of concurrent refresh.

[7] As gem5 SE mode needs large memory space in the host machine for the simulation, this is the maximum core number we can afford.



(a) Effectiveness of dynamic refresh (b) Impact of relaxing power constraint (c) Impact of address mapping schemes

Figure 11: Simulation results for sensitivity study. In (a), all results are normalized to SRSA4(S); In (b) and (c), all results are normalized to No Refresh.

If the constraint can be relaxed, even SRSA4(3) can have comparable performance (97.3%) to the ideal case.

Address mapping policy. Figure 11c shows the result when CREAM adopts different address mapping policies. In this work, we assume the row bits are placed as MSBs to maximize the row-buffer hit rate, rank- and bank-level interleaving, which is commonly deployed in state-of-the-art MCs. The left part is the result of "R(row):C(column):BK(bank):RK(rank)" that is used as the default address mapping policy through this paper. The other two policies, "R:C:RK:BK" and "R:BK:RK:C" are also evaluated. "R:C:RK:BK" has very similar result as "R:C:BK:RK" so that the result is omitted. Alternatively, "R:BK:RK:C" has even better performance because it remaps more memory accesses into the same bank, which helps the other banks has sufficient time to hide the refresh.

Smaller refresh cycle. All above simulations conservatively sustain tRFC even though SALR and SRLR are employed. In fact, similar to the fine-grained refresh in DDR4, tRFC in CREAM can become smaller if fewer sub-ranks are refreshed together. We rerun simulation with the smaller tRFC given in Table 2. However, the performance enhancement is trivial (result is omitted). Compared to the smaller refresh cycle, the concurrent refresh is more effective to elevate the performance. The reason is straightforward: if memory access can be served during a refresh, the smaller refresh cycle won't change a lot.

6. Related Works

Concurrent Refresh. Kirihata *et al.* [24] proposed a concurrent refresh scheme that can carry on the bank-level refresh, which is similar to our SRLR scheme. However, this concurrent refresh targets at embedded DRAM (eDRAM) with small capacity so that it does not consider the power constraint, which is critical to the standard DRAM. Also, as mentioned in Section 5.3, the bank-level refresh still incurs lots of refresh conflicts without sub-array-level refresh. Our proposed CREAM technique improves beyond their scheme to achieve more performance gain.

Refresh Deferring. To alleviate the increasing refresh penalty, some studies make use of the flexibility of refresh rescheduling in $8 \times tREFI$. The basic idea is straightforward, which defers refresh when memory is busy and hope the deferred refresh can be served later as memory becomes idle. Ipek *et al.* proposed a refresh scheme called DUE to simply defer a refresh until the memory is idle[11]. The drawback of DUE was observed by Stuecheli *et al.* [2]: it can delay the future memory accesses that come out during a refresh. As a result, elastic refresh (ER) was proposed to further defer a refresh for certain cycles even the memory is idle [2]. As mentioned in Section 2.4, ER can adversely introduce extra delay that limits the further performance improvement. Nevertheless, all refresh deferring scheduling methods have a common problem: the number of deferred refresh is limited to eight. Once the deferred refresh number hits the limitation, immediate refresh should be forced. As a consequence, it is ineffective for memory-intensive applications that can quickly accumulate deferred refreshes to reach the constraint.

Refresh Reduction. In addition to the above prior arts, there are also several works to reduce refresh number to improve the power efficiency. Prior works [25, 24] takes advantage of PASR (Partial Array Self-Refresh) [1, 3] to reduce self-refresh counts. By leveraging SALR, PASR can be seamlessly integrated into CREAM to improve the power efficiency. On the other hand, Smart Refresh [17] leverages the fact that a read/write is equivalent to a refresh due to the destructive array access. However, the size of the dedicated counter for each row, which is proportional to the total row number, can be too large to afford (e.g., 1.5MB in a 32GB system). In addition, the effectiveness of smart refresh varies for different memory access patterns.

Recently, RAIDR [18] devises a refresh scheduling policy that differentiates the retention time and correspondingly applies different refresh rate to cells that have different retention times. A bloom filter is deployed to simplify the refresh tracking. However, RAIDR incurs significant area overhead in MC since it requires at least 1.25KB storage to implement the bloom filters. In addition, it may have reliability issue due to the problem of variable retention time (VRT) [8], where the retention time of one cell can suddenly change due to the leakage current. Alternatively, the over-

head of CREAM is negligible and all sub-arrays are still refreshed at a conservative rate.

In addition, refresh prediction has been proposed to hide refresh penalty [26, 27]. These studies are orthogonal to CREAM and can be easily integrated into CREAM.

7. Conclusion

The refresh penalty is no longer negligible as DRAM capacity keeps growing. In this work, a concurrent-refresh-aware memory system, CREAM, is proposed to mitigate the increasing refresh overhead. With the power constraint, CREAM leverages sub-rank-level refresh (SRLR) and sub-array-level refresh (SALR) to make better trade-off between performance and power. In addition, the optimization technologies, such as sub-array refresh round-robin and dynamic refresh scheduling, are designed to help the performance improvement. CREAM does not introduce additional pin-out and only incurs negligible hardware overhead. The experimental results show that CREAM can achieve as much as 12.9% and 7.1% performance gain over the conventional memory and the memory that uses elastic refresh scheduling policy, respectively.

References

[1] JEDEC Solid State Technology Association, "JEDEC Standard: DDR4 SDRAM," http://www.jedec.org/sites/default/files/docs/JESD79-4.pdf, Sep. 2012.

[2] J. Stuecheli, D. Kaseridis, H. C.Hunter, and L. K. John, "Elastic Refresh: Techniques to Mitigate Refresh Penalties in High Density Memory," in *MICRO'43*, Dec. 2010, pp. 375–384.

[3] JEDEC Solid State Technology Association, "JEDEC Standard: DDR3 SDRAM Specification," http://www.jedec.org/standards-documents/docs/jesd-79-3d, Sep. 2009.

[4] Micron, "MT41J128M8JP-15E Data Sheet," http://www.micron.com/products/dram/ddr3-sdram.

[5] Micron, "MT41J256M8HX-15E Data Sheet."

[6] Micron, "MT41J512M8RA-15E Data Sheet."

[7] Micron, "MT41J1G8THE-15E Data Sheet."

[8] B. Jacob, S. W. NG, and D. T. Wang, *Memory Systems: Cache, DRAM, Disk*. Morgan Kaufmann, 2007.

[9] JEDEC Solid State Technology Association, "DDR4 Mini Workshop," http://www.jedec.org/sites/default/files/, Nov. 2011.

[10] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "DRAMSim2: A Cycle Accurate Memory System Simulator," *Computer Architecture Letters*, vol. 10, no. 1, pp. 16–19, Jan.–Jun. 2011.

[11] E. Ipek, O. Mutlu, J. F. Martínez, and R. Caruana, "Self-Optimizing Memory Controllers: A Reinforcement Learning Approach," in *ISCA'35*, 2008, pp. 39–50.

[12] Micron, "TN-41-15: Low-Power Versus Standard DDR SDRAM," http://download.micron.com/pdf/technotes/DDR/tn4615.pdf.

[13] Y. Kim, V. Seshadri, D. Lee, J. Liu, and O. Mutlu, "A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM," in *ISCA'39*, Jun. 2012, pp. 368–379.

[14] Synopsys, "Design Compiler," http://www.synopsys.com.

[15] T. Vogelsang, "Understanding the Energy Consumption of Dynamic Random Access Memories," in *MICRO'43*, Dec. 2010, pp. 363–374.

[16] N. Chatterjee, N. Muralimanohar, Bal, A. Davis, and N. P. Jouppi, "Staged Reads : Mitigating the Impact of DRAM Writes on DRAM Reads," in *HPCA'18*, Feb. 2012, pp. 1–12.

[17] M. Ghosh and H.-H. S. Lee, "Smart Refresh: An Enhanced Memory Controller Design for Reducing Energy in Conventional and 3D Die-Stacked DRAMs," in *MICRO'40*, Dec. 2007, pp. 134–145.

[18] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu, "RAIDR: Retention-Aware Intelligent DRAM Refresh," in *ISCA'39*, Jun. 2012, pp. 1–12.

[19] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi *et al.*, "The gem5 Simulator," *Computer Architecture News*, vol. 39, no. 2, pp. 1–7, Aug. 2011.

[20] Standard Performance Evaluation Corporation, "SPEC2006 CPU," http://www.spec.org/cpu2006.

[21] J. D. McCalpin, "STREAM Benchmark," http://www.cs.virginia.edu/stream.

[22] S. Liu, B. Leung, A. Neckar, S. O. Memik, G. Memik, and N. Hardavellas, "Hardware/Software Techniques for DRAM Thermal Management," in *HPCA'17*, Feb. 2011, pp. 515–525.

[23] Q. Zhu, X. Li, and Y. Wu, "Thermal Management of High Power Memory Module for Server Platforms," in *ITHERM'08*, May 2008, pp. 572–576.

[24] T. Kirihata, P. Parries, D. Hanson, H. Kim, J. Golz, G. Fredeman, R. Rajeevakumar, J. Griesemer, N. Robson, A. Cestero, B. Khan, G. Wang, M. Wordeman, and S. Iyer, "An 800MHz Embedded DRAM With a Concurrent Refresh Mode," *IEEE Solid State Circuits*, vol. 40, no. 6, pp. 1377–1385, Jun. 2005.

[25] S. Liu, K. Pattabiraman, T. Moscibroda, and B. G. Zorn, "Flikker: Saving DRAM Refresh-power through Critical Data Partitioning," in *ASPLOS'11*, Mar. 2011, pp. 213–224.

[26] B. Akesson, K. Goossens, and M. Ringhofer, "Predator: A Predictable SDRAM Memory Controller," in *CODES+ISSS'07*, Oct. 2007, pp. 251–256.

[27] B. Bhat and F. Mueller, "Making DRAM Refresh Predictable," *Real-Time System*, vol. 47, no. 5, pp. 430–453, Sep. 2011.