2016 29th International Conference on VLSI Design and 2016 15th International Conference on Embedded Systems

Massed Refresh: An Energy-Efficient Technique to Reduce Refresh Overhead in Hybrid Memory Cube Architectures

Ishan G Thakkar, Sudeep Pasricha
Department of Electrical and Computer Engineering
Colorado State University, Fort Collins, CO, U.S.A.
{ishan.thakkar, sudeep}@colostate.edu

***Abstract*— This paper presents a novel, energy-efficient DRAM refresh technique called *massed refresh* that simultaneously leverages bank-level and subarray-level concurrency to reduce the overhead of distributed refresh operations in the Hybrid Memory Cube (HMC). In *massed refresh*, a bundle of DRAM rows in a refresh operation is composed of two subgroups mapped to two different banks, with the rows of each subgroup mapped to different subarrays within the corresponding bank. Both subgroups of DRAM rows are refreshed concurrently during a refresh command, which greatly reduces the refresh cycle time and improves bandwidth and energy efficiency of the HMC. Our experimental analysis shows that the proposed *massed refresh* technique achieves up to 6.3% and 5.8% improvements in throughput and energy-delay product on average over JEDEC standardized distributed per-bank refresh and state-of-the-art scattered refresh techniques.**

***Index Terms*—Hybrid Memory Cube, DRAM, energy-efficiency, distributed refresh, bank-level parallelism**

I. Introduction

Since the emergence of DRAM technology, reducing the overhead of refresh operations has been one of the major challenges for DRAM designers. Due to the volatile nature of DRAM cells, it is necessary to perform periodic refresh operations on them to retain the stored data bits over time. One of the downsides of periodic refresh operations is that they contribute significantly to the memory power, e.g., up to 30% of total power consumption [2]-[3]. Refresh operations also require stalling waiting memory requests until the refresh finishes, which reduces DRAM performance, e.g., as explained in [9], the overall performance for a DDR4 system degrades by 18.8% for a variety of workloads due to periodic refresh.

In recent years, many techniques have been proposed to minimize the performance overhead of refresh operations in modern DRAMs [3]-[10], [16]. Although these techniques are mainly focused on 2D DRAMs, they can also be applied to emerging 3D-stacked DRAMs. However, the smaller bank-size and increased bank-level parallelism of 3D-stacked DRAMs results in higher power density and die temperature [1][24][25], which reduces data retention time, requiring more frequent refreshes (thus increasing refresh overhead) in 3D-stacked DRAMs. The greater power density of 3D DRAMs also increases the amount of current drawn from the power delivery networks (PDNs) of memory modules [21], which raises the noise level in PDNs resulting in degraded memory reliability and performance [18][21]. Recent work has shown that the power density of 3D-stacked DRAMs can be decreased by decreasing refresh power [23]. *This provides a strong motivation for 3D-stacked DRAM designers to explore new techniques that can reduce refresh overheads, thereby ensuring reliability and high performance.*

In this paper, we propose *massed refresh*, a novel energy-efficient DRAM refresh technique that intelligently leverages bank-level as well as subarray-level parallelism to reduce refresh cycle time overhead in the 3D-stacked Hybrid Memory Cube (HMC) DRAM architecture. We have observed that due to the increased power density and resulting high operating temperatures, the effects of periodic refresh commands on refresh cycle time and energy-efficiency of 3D-stacked DRAMs, such as the emerging HMC [1], are significantly exacerbated. Therefore, we select the HMC for our study, even though the concept of *massed refresh* can be applied to any other 2D or 3D-stacked DRAM architecture as well. Our novel contributions in this paper are summarized below:

- We demonstrate how minor changes in the DRAM bank access control logic of the HMC can be leveraged to increase bank-level parallelism during a distributed refresh operation, as part of our proposed *massed refresh* technique. We also quantify the extra overhead of area and power incurred due to these changes;
- Unlike the JEDEC standardized *all-bank refresh* technique [22] that refreshes all the banks of a rank concurrently, we restrict our proposed *massed refresh* technique to refresh only a few selected banks in parallel. This enhancement limits the PDN noise and peak refresh current to acceptable levels resulting in increased operation efficiency and performance;
- We investigate, for the first time, the effect of state-of-the-art *distributed per-bank refresh* [19][20] and *scattered refresh* [9] techniques on DRAM energy-efficiency and compare these techniques with our *massed refresh* technique in terms of memory throughput and energy delay product (EDP) for the HMC DRAM architecture.

II. Related Work

During the initial stages of DRAM development, the *burst refresh* scheme was used to refresh the whole DRAM device with a single refresh command in every retention cycle of 64ms under normal temperature conditions (<85^{0}C) or every 32ms under high temperature conditions (>85^{0}C). During a burst refresh operation, the entire DRAM device, including all the banks in all the ranks, becomes unavailable to non-refreshing memory tasks such as reading/writing data from/into memory for the amount of time it takes to perform the refresh, during which no productive tasks are performed.

To reduce the length of memory pauses and to increase the productivity of DRAM systems while still supporting refresh operations, the Joint Electron Device Engineering Council (JEDEC) standardized a new scheme called *distributed/ interleaved refresh* [16], wherein a memory controller sends out multiple refresh commands in one retention cycle. In this scheme, a subset of DRAM rows, called "refresh bundle" are refreshed for

978-1-4673-8700-2/16 $31.00 © 2016 IEEE
DOI 10.1109/VLSID.2016.13

every distributed refresh command issued. The time taken in refreshing a refresh bundle is termed as refresh cycle time (tRFC). The distributed refresh operation can be implemented in a DRAM rank at one of two granularities – (1) per-bank or (2) all-bank. In the *all-bank refresh* scheme that is used by all general purpose DDRx memory standards including the DDR4 standard [22], the refresh bundle is distributed among all banks of a rank and refresh operations of all the banks are completely overlapped in time. In the *per-bank refresh* scheme that is supported by some new memory standards such as LPDDR4 [19] and high bandwidth memory (HBM) [20], all rows of the refresh bundle map to a single bank and their refresh operations are performed sequentially. In state-of-the-art 2D-DRAMs, depending on the workload characteristics, either *per-bank refresh* or *all-bank refresh* is favored to minimize the refresh overhead. But, as discussed in [23], the selection of an appropriate refresh method that can reduce the refresh overhead for 3D-stacked DRAMs requires much more complex deliberation and analysis of a number of architecture-level and design-level tradeoffs. We provide a brief discussion of these tradeoffs in Section IV.

In recent years, many research efforts have focused on reducing DRAM refresh overhead [3]-[10]. Based on their method to minimize refresh overhead, these techniques can be classified into four different categories: *(i)* cell retention time aware methods [4][5], *(ii)* methods that eliminate unnecessary refreshes [3][6], *(iii)* refresh-aware scheduling methods [7][8], and *(iv)* refresh cycle time reduction methods [9][10].

Cell retention time aware methods exploit DRAM inter-cell variation in retention time either to minimize refresh operations [4] or to adaptively select a refresh period for a particular refresh operation [5]. But, these methods make DRAM cells more prone to errors which harms data integrity, because they operate DRAM cells at a refresh interval beyond the specification range that DRAM manufacturer's guarantee. The methods that eliminate unnecessary refreshes [3][6] require extensive data profiling at design time and power hungry run-time decision making. The refresh-aware scheduling methods [7][8] schedule refresh commands so that they do not collide with read or write commands, which improves utilization of resources and DRAM performance. But, these methods complicate memory controller design and also heavily depend on data access patterns of the workloads, providing benefits only if a large number of rows are activated. Refresh cycle time reduction methods [9][10] identify refresh command cycle time (tRFC) as the main factor that limits its DRAM performance. Among these, methods such as [10] intelligently choose refresh granularity to selectively reduce tRFC, which in turn reduces the queuing delay of memory access requests for a given application. On the other hand, *scattered refresh* [9] reduces refresh cycle time by exploiting subarray level parallelism, wherein the rows of a refresh bundle are scattered to different subarrays and their refresh operations are overlapped in time. Such refresh cycle time reducing methods have been shown to have more potential to overcome the DRAM refresh problem without the drawbacks of other methods. Besides, refresh cycle time reducing methods are orthogonal to other methods and can be applied in any combinations of the other methods.

In this paper, we propose *massed refresh*, a novel energy-efficient distributed refresh technique that reduces refresh cycle time overhead by leveraging bank-level as well as subarray-level parallelism in the hybrid memory cube (HMC). The results of our analysis indicate that *massed refresh* can significantly reduce the overhead of distributed refresh operations in the HMC, thus improving memory throughput and energy-efficiency while keeping the PDN noise level to within acceptable limits for reliable operation.

III. Background: Hybrid Memory Cube (HMC)

This section provides a brief overview of the HMC architecture. The reader is directed to [1], [11] for more details on the HMC specification and operating mode configurations.

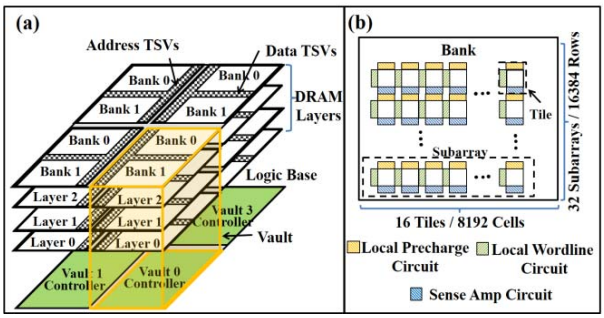

Figure 1: (a) Schematic of 4Gb hybrid memory cube (HMC) quad unit; (b) schematic of an HMC bank.

An HMC architecture consists of a single power efficient package containing multiple memory dies and one base die, stacked together using through-silicon-via (TSV) technology. The base die layer is generally configured as the logic base, or LoB. The logic layer consists of multiple components that provide both external link access to the HMC device as well as internal routing and transaction logic. The data storage in an HMC module is organized in a three dimensional manner into multiple quad units. Figure 1(a) shows a schematic of a 4Gb HMC quad unit. As shown in the figure, one quad unit consists of four vault units. A vault is a major organization unit of data storage in an HMC that vertically spans each of the memory layers (a total of four memory layers in the example shown) using TSVs. Each vault has two 128Mb sized banks on each memory layer. Each bank has 16384 rows with each row of size 8Kb (Figure 1(b)). The rows in a bank are grouped into 32 subarrays. To reduce the capacitance of the memory access path. The subarrays are further divided into 16 tiles with each tile having a 512×512 array of cells. The physical layout of subarrays and tiles are similar to that found in commodity 2D DRAMs [9]. As shown in Figure 1(a), each vault has a vault controller on the logic base layer that manages all memory reference operations within that vault. A vault controller is analogous to a DIMM controller. The refresh operations of each vault are managed by its corresponding refresh controller.

IV. Minimizing Refresh Overhead in 3D DRAMs

The main idea behind minimizing DRAM refresh overhead is to increase DRAM availability for non-refresh operations, so that memory performance can be improved. All existing DRAM architectures support *all-bank refresh* [16][22] where the corre-

sponding rows in all banks within a rank are refreshed concurrently. This technique increases DRAM availability by reducing the refresh cycle time, and may increase DRAM performance, depending on the workload characteristics. But it is inefficient and less reliable for 3D-stacked DRAMs, because as discussed in [18] and [21], the smaller size and increased physical density of banks increases the noise level of the power delivery network (PDN) when concurrently refreshing the corresponding rows in all the banks. The increased noise level in the PDN often leads to malfunctions that result in erroneous DRAM operation, making the *all-bank refresh* method highly undesirable for 3D-stacked DRAMs.

In contrast, in *per-bank refresh* only one bank per rank is refreshed during a refresh command. All other non-refreshing banks are available during per-bank refresh. Therefore, this technique allows the non-refresh (i.e., read/write) operations targeted at non-refreshing banks to be overlapped with refresh operation of the refreshing bank. In 3D-stacked DRAMs such as the HMC, there exists greater bank-level parallelism than 2D DRAMs because of the fine-grained structure of their data array, creating even more options for overlapping refresh and non-refresh operations. For example, in HMC, the data array is divided into multiple vaults, with each vault having multiple ranks and partitions; and it is possible to overlap refresh and non-refresh operations across multiple vaults, across ranks within a vault, and across banks within a rank. However, in traditional *per-bank refresh*, all rows in the refresh bundle are refreshed sequentially, which increases refresh cycle time by a factor equal to the refresh bundle size, and leads to inefficiencies. *Moreover, reducing the energy overhead of refresh in HMC still remains a critical problem to improve the energy-efficiency of the memory subsystem*, due to the greater power density of HMC, as discussed in Section I. The next section describes our approach to overcome these challenges.

V. Massed Refresh: Overview

A. Concept and Implementation

In this subsection, our proposed *massed refresh* technique is explained, including a description of how the control logic and vault peripherals required to enable this technique are implemented. The following descriptions of the concept, implementation, and experimental results are presented for a 1Gb vault of the example HMC design explained in Section III. For clarity, it is imperative to first briefly discuss the function of the refresh controller for *distributed per-bank refresh* [19][20], *distributed all-bank refresh* [16][22], and *scattered refresh* [9], before going into the details of how it functions for our proposed *massed refresh* technique.

An HMC has a low data cell retention period of 32ms, due to the increased power density and operating temperature of its 3D-stacked memory dies. A total of 8192 refresh commands must be issued by a refresh controller in 32ms under a distributed refresh scheme, which sets the refresh command interval (tREFI) for HMC to be 3.9µs. Also, for a vault capacity of 1Gb and an 8Kb row size, refresh bundle size is 16. Typically, a refresh controller consumes the row address supplied by an internal address counter and identifies a refresh bundle based on the implemented refresh scheme. For every refresh command, the starting value of the internal address counter is the number of refresh commands sent so far in the current refresh/retention period. In the *distributed per-bank refresh* scheme, for each refresh command, the refresh controller determines the address of the first row to be refreshed from the starting value of the internal address counter. Then, the address calculator increments the row address by one every time to determine the address of the next row to be refreshed. As an example, suppose that the first row to be refreshed in a distributed refresh command is row R-5632 in subarray SA-12 of bank B-0 on layer L-0. Therefore, the next row to refresh would be row R-5633 in the same subarray SA-12. Hence, in *distributed per-bank refresh*, as shown in Figure 2(a), all rows of a refresh bundle map to a single subarray. For brevity, Figure 2 considers a smaller refresh bundle of 4 rows for a DRAM system with 4 banks, but a similar trend holds for a refresh bundle size of 16 and a memory system with more than 4 banks as well.

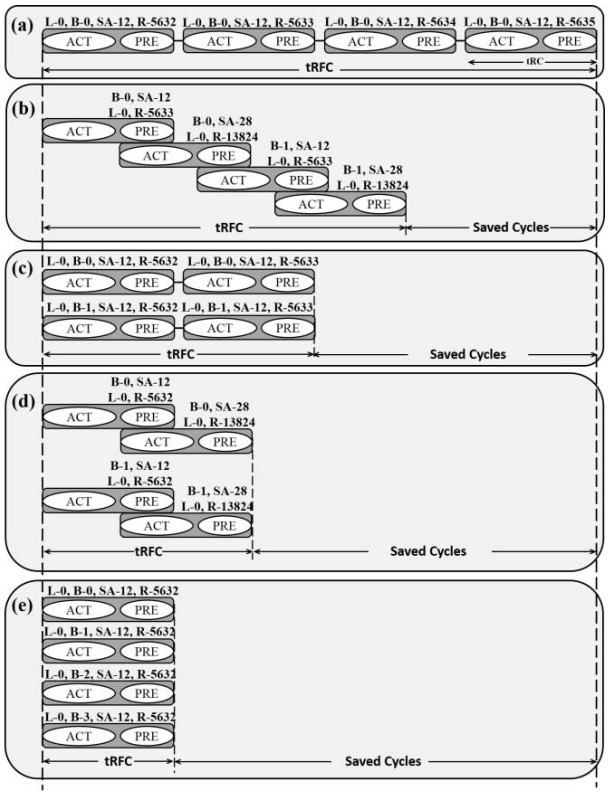

Figure 2: Refresh cycle for (a) distributed per-bank refresh (b) scattered refresh (c) crammed refresh (d) massed refresh (e) distributed all-bank refresh.

In the *distributed all-bank refresh* scheme, as shown in Figure 2(e), the corresponding rows (R-5632) in all the banks (B-0 to B-3) are refreshed simultaneously, which reduces the refresh cycle time (tRFC) by several cycles. For this scheme, if the refresh bundle size (k) is equal to the number of banks (n), each bank will have only one row to refresh. Similarly, if k is greater than n, each bank will have k/n rows to refresh. Therefore, in the case of *distributed all-bank refresh*, the row address is incremented only when $k>n$.

On the other hand, in the case of *scattered refresh* [9], the address calculator increments the address counter by 8192 so that every other row of the refresh bundle is mapped to a different subarray [9]. This arrangement in conjunction with the provision of a dedicated address latch to each individual subarray ends up improving subarray-level parallelism. Hence, as shown in Figure 2(b), for *scattered refresh* the refresh operations of rows that are mapped to different subarrays are overlapped in time, which reduces tRFC by several cycles. In general, for *distributed per-bank refresh* and *scattered refresh*, the address calculator increments the row address counter by 1 and by 8192 respectively for 16 times during a refresh command before moving on to the next refresh command after the tREFI interval. More information on these refresh techniques can be found in [9], [16], [19], [20], and [22].

Our proposed *massed refresh* technique builds on the concept of subarray-level parallelism in *scattered refresh* by leveraging additional bank-level parallelism during a refresh command in the DRAM device. We note that as the global row address decoder is shared among all the subarrays of a bank, the refresh operations of rows that are mapped to different subarrays of a single bank are not completely overlapped in time in *scattered refresh,* which reduces efficiency.

Figure 3 illustrates how bank-level parallelism is exploited in our proposed scheme to improve upon *scattered refresh*. As shown in the physical address latch of the figure, the physical row address is divided into three parts: a 14-bit row address, a 2-bit layer address, and a 1-bit bank address. Before routing the address bits to individual memory layers using TSVs, the layer address and bank address are decoded using a physical address decoder to obtain layerID (LID) and bankID (BID) respectively. The figure also shows how the physical address latch and the address decoder are shared among the read/write command scheduler and the refresh controller.

During a regular read/write command operation, the LID and BID values direct the row address bits to a particular bank on a particular memory layer. But, during a refresh command operation, the BID is masked by the refresh controller so that the row address is directed to both the banks on the target memory layer. *This arrangement simultaneously refreshes two rows in two banks on the selected layer for each physical address generated by the address calculator.* In other words, a refresh bundle is divided into two equal subgroups and both subgroups are refreshed concurrently. The refresh operations of these two subgroups are completely overlapped in time. Due to this bank-level parallelism exploited in our approach, the address counter value needs to be incremented only for 8 times to refresh a bundle of 16 rows.

The bank-level parallelism obtained by masking the BID bits can be used in two ways: one with the added subarray-level parallelism and the other without it. For clarity, we refer to our refresh technique with the added subarray-level parallelism as the *massed refresh* technique and our technique without the added subarray-level parallelism is referred to as *crammed refresh*. As shown in Figure 2(c), in *crammed refresh*, the example refresh bundle of 4 rows is divided in two subgroups with two rows (R-5632 and R-5633) in each subgroup. These two subgroups are mapped on bank B-0 and bank B-1 on the memory layer L-0, and their refresh is completely overlapped in time, which reduces the refresh cycle time (tRFC) by a large number of cycles. However, rows R-5632 and R-5633 are refreshed sequentially in both subgroups, because they are mapped on the same subarray SA-12.

In contrast, as shown in Figure 2(d), in *massed refresh*, the constituent rows R-5632 and R-13824 of both the subgroups (which are mapped on bank B-0 and bank B-1 on layer L-0) belong to two different subarrays, subarray SA-12 and SA-28 respectively. Therefore, refresh operations of the constituent rows of both subgroups are further overlapped in time. Hence, *massed refresh* exploits the subarray-level parallelism in addition to the bank-level parallelism of *crammed refresh* to reduce tRFC even further. The main difference between *all-bank refresh* and *massed/crammed refresh* is that the *massed/ crammed refresh* methods are restricted to refresh only two banks in parallel. The smaller degree of bank-level parallelism in *crammed refresh* and *massed refresh* sets the PDN noise levels to more acceptable level, which makes them more efficient and desirable than the *all-bank refresh* method.

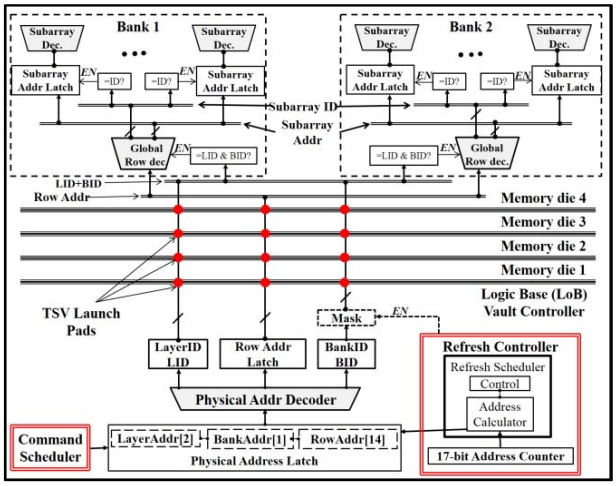

Figure 3: Schematic implementation of control logic and peripheral circuits for the bank-level and subarray-level parallelism of our proposed *massed refresh* technique.

B. Refresh Cycle Time and Overhead Analysis

In this subsection, we present refresh cycle time calculations for the *crammed refresh* and *massed refresh* techniques along with those for the state-of-the-art *distributed per-bank refresh, distributed all-bank refresh,* and *scattered refresh* schemes, before analyzing their overheads.

In general, refresh cycle time (tRFC) is equal to the product of refresh bundle size (k) and row cycle time (tRC) with some additional recovery time (tREC) [10]. Consequently, tRFC for *distributed per-bank refresh* is (tRC*16) + tREC (as k=16). Typically, tRC is a sum of row access cycle time (tRAS) and precharge delay (tRP). As explained in [9], the subarray-level parallelism of the *scattered refresh* scheme hides tRP time from the tRC time of each row whose refresh operation is overlapped with another row's refresh during a refresh command. For *scattered refresh*, in a refresh bundle of 16 rows, refresh operations of only 15 rows are overlapped. Hence, tRFC for *scattered refresh* is (tRAS*15) + tRC + tREC. For *all-bank refresh*, each of

the total 8 banks (n=8) has two rows to refresh (as k=16). So, tRFC for *all-bank refresh* is (tRC*2) + tREC = 80ns. In a similar manner, we can calculate tRFC for our *crammed refresh* and *massed refresh* techniques.

The calculated tRFC values for all the aforementioned refresh techniques are shown in Table 1. We consider a row cycle time (tRC) of 35ns for the example 1Gb vault of HMC (shown in Figure 1) by modeling the vault using CACTI-3DD [12] with 50nm technology parameters. Our choice of this technology node is motivated by Micron demonstrating a 4Gb HMC quad fabricated in 50nm technology [1]. We assume a recovery time (tREC) of 10ns [10] in our calculations of tRFC. It should be noted that performing refresh operations on more than one row simultaneously draws extra current from the PDN. So, it is important to ensure that the proposed *crammed refresh* and *massed refresh* techniques do not overshoot the current delivering capacity of the PDN. To investigate this aspect, we also calculated the required peak refresh current for the various refresh techniques using their 50nm technology parameter based architectural models developed in CACTI-3DD [12]. These values are shown in Table 1.

Table 1: Refresh cycle time (tRFC) and peak refresh current for state-of-the-art and proposed DRAM refresh techniques

	Per-bank	Scattered	Crammed	Massed	All-bank
tRFC (ns)	570	420	290	220	80
Refresh current (mA)	130.4	178.1	260.7	347.6	1290.5

The peak current capacity of the 4Gb HMC quad design demonstrated by Micron was shown to be 9.2A [1], which implies that the peak current capacity of a 1Gb HMC vault can be reasonably estimated to be 2.3A. Table 1 shows that *all-bank refresh* yields the smallest tRFC. However it also has a prohibitively high peak refresh current of 1290.5mA, along with a propensity to increase PDN noise when concurrently refreshing the corresponding rows in all the banks [18], [21]. Due to these practical limitations, *all-bank refresh* is not well suited for 3D-stacked DRAMs such as HMC.

The peak refresh current for the remaining refresh techniques is significantly lower than for *all-bank refresh*. Our *massed refresh* technique has a peak refresh current of 347.6mA, which is well below the current delivering capacity of an HMC vault. Therefore, the proposed *crammed refresh* and *massed refresh* techniques can be easily implemented without overshooting the current delivery capacity in HMC or creating PDN noise issues. Moreover, our proposed crammed refresh and massed refresh techniques reduce refresh cycle time much more effectively compared to the other refresh techniques, as shown in Table 1. Also, the HMC specification [17] defines separate pins to externally supply the DRAM wordline boost voltage V_{PP}, which relaxes the need for on-chip charge pumps for the V_{PP} supply. Dedicated pins for V_{PP} supply increase charge supplying capacity of an HMC vault, easily allowing parallel activation of two or more rows in a vault, as required in our proposed refresh techniques.

Similar to *scattered refresh*, our proposed *crammed refresh* and *massed refresh* techniques would need a 13-bit counter to keep track of the number of refresh commands in a refresh period for the example 1Gb HMC vault, along with a 17-bit internal address counter. These counters, when implemented at the 50nm technology node, consume 170μm^2 and 230μm^2 area respectively. Thus, the area overhead of the changes in the peripherals required for subarray-level and bank-level parallelism is negligible, similar to [9].

VI. Experimental Results

We performed trace-driven simulation analysis to compare our proposed *crammed refresh* and *massed refresh* techniques with the state-of-the-art *distributed per-bank refresh* [19], [20] and *scattered refresh* [9] techniques. We do not compare our proposed refresh techniques against the *all-bank refresh* technique because it is impractical to implement for 3D DRAMs as discussed in Section IV and V.B.

Table 2: Gem5 simulation configuration

#Cores	4 ARM	L2 Coherence	MOESI
L1 I Cache	16KB	Frequency	5 GHz
L1 D Cache	16KB	Issue Policy	In-order
L2 Cache	128KB	# Memory Controllers	1

Memory access traces for the PARSEC benchmark suite [13] were extracted from detailed cycle-accurate simulations using the gem5 full-system simulator [14]. Table 2 gives the configuration of the gem5 simulator that was used for this study. We considered twelve different applications from the PARSEC benchmark suite: *Blackscholes (BS), Bodytrack (BT), Canneal (CN), Dedup (DD), Facesim (FS), Ferret (FR), Fluidanimate (FA), Freqmine (FM), Streamcluster (SC), Swaptions (SW), Vips (VP)*, and *x264 (X2)*. We ran each PARSEC benchmark for a "warm-up" period of 1 billion instructions and captured memory access traces from the subsequent 1 billion instructions executed. These memory traces were then provided as inputs to the DRAM simulator DRAMSim2 [15]. We heavily modified DRAMSim2 to model *crammed refresh*, *massed refresh,* and the refresh techniques from prior work for comparison. Throughput and energy-delay product (EDP) values for the memory subsystem for all of these refresh techniques considered were obtained from this DRAMSim2 simulator. We performed timing and energy analysis by adapting the code for CACTI-3DD [12] to model the 1Gb HMC vault at the 50nm technology node. These parameters for the HMC architecture are summarized in Table 3. All TSVs in this study were modeled based on ITRS projections for intermediate interconnect level TSVs.

Table 3: Row access time (tRAS), row cycle time (tRC), activation-pre-charge energy (ActPreE), read energy (ReadE), and background power (BGP) parameter values for HMC

tRAS (ns)	tRC (ns)	ActPreE (nJ)	ReadE (nJ)	BGP (mW)	Data bus width
25	35	1.8	2.7	11	128b

Figure 4 shows throughput values for the various refresh techniques across the PARSEC benchmarks. A rank-based round-robin scheduling scheme, rank:row:col:bank address mapping scheme, and open page policy were used for all simulations. It can be observed that *massed refresh* achieves 6.3% more memory throughput on average over *per-bank refresh* and *scattered refresh*. More specifically, the proposed *massed refresh* technique achieves 8.4%, 4.3%, and 1.4% more memory throughput on average over *per-bank refresh*, *scattered refresh* and *crammed refresh* respectively. *Massed refresh* has 62.5%, 47.6% and 24.1% less refresh cycle time over *per-bank refresh*, *scattered refresh* and *crammed refresh* respectively, which

translated into the highest throughput for *massed refresh* over other refresh schemes.

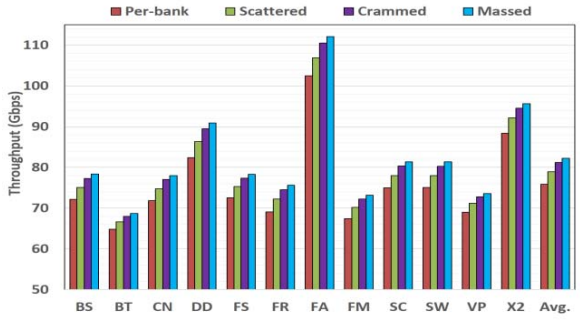

Figure 4: Memory throughput results for various DRAM refresh schemes across PARSEC benchmarks.

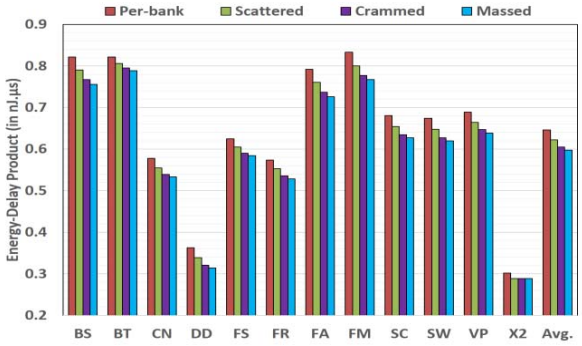

Figure 5: Energy-delay product (EDP) results for various DRAM refresh methods across PARSEC benchmarks.

Figure 5 shows the energy-delay product (EDP) values for the various DRAM refresh techniques across the PARSEC benchmarks. EDP values were calculated by multiplying the average-energy with average-latency for memory accesses. Average-energy in each case was calculated by dividing the total energy by the total number of memory transactions, and the average-latency was calculated by dividing the total latency by the total number of memory transactions. It can be seen that *massed refresh* achieves 5.8% less EDP on average over *per-bank refresh* and *scattered refresh*. More specifically, our proposed *massed refresh* technique achieves 7.5%, 3.9% and 1.2% less EDP on average over *per-bank refresh*, *scattered refresh*, and *crammed refresh*, respectively. As was shown in Table 1 earlier, even though the peak refresh current (power) required for *massed refresh* is higher than all the other refresh techniques (except *all-bank refresh*), the total energy spent in refresh operations during a given amount of time remains the same for all the refresh techniques (except *all-bank refresh*). Moreover, the increased throughput of *massed refresh* translates into the least average-energy. The least average-energy and the highest throughput result in the best improvements in EDP for *massed refresh* among all of the DRAM refresh techniques, making it the most energy-efficient refresh technique compared to the other techniques.

VII. Conclusions

In this paper we proposed a new refresh technique for the 3D-stacked Hybrid Memory Cube (HMC) DRAM called *massed refresh* that yields 6.3% and 5.8% improvements in throughput and EDP on average over the JEDEC standardized *distributed per-bank refresh* and the state-of-the-art *scattered refresh* schemes. These promising results indicate that our *massed refresh* technique can significantly reduce the overhead of distributed refresh operations in the HMC (and other DRAMs) by improving their throughput and energy-efficiency.

ACKNOWLEDGMENT

This research is supported by grants from SRC, NSF (CCF-1252500, CCF-1302693), and AFOSR (FA9550-13-1-0110).

REFERENCES

[1] J. Pawlowski, "Hybrid Memory Cube (HMC)," in *Hot Chips*, 2011.
[2] Y. Han, Y. Wang, H. Li, X. Li, "Data-aware DRAM refresh to squeeze the margin of retention time in hybrid memory cube," in I*CCAD*, 2014.
[3] 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,* 2007.
[4] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu, "RAIDR: Retention-aware intelligent DRAM refresh," in *ISCA*, 2012.
[5] R. K. Venkatesan, S. Herr, and E. Rotenberg, "Retention-aware placement in DRAM (RAPID): software methods for quasi-non-volatile DRAM," in *HPCA,* 2006.
[6] S. Liu, K. Pattabiraman, T. Moscibroda, and B. G. Zorn, "Flikker: Saving DRAM Refresh-power Through Critical Data Partitioning," in *ASPLOS*, 2011.
[7] J. Stuecheli, D. Kaseridis, H. C. Hunter, and L. K. John, "Elastic Refresh: Techniques to Mitigate Refresh Penalties in High Density Memory," in *MICRO*, 2010.
[8] P. Nair, C.-C. Chou, and M. K. Qureshi, "A case for Refresh Pausing in DRAM memory systems," in *HPCA*, 2013.
[9] T. V. Kalyan, K. Ravi, M. Mutyam, "Scattered refresh: An alternative refresh mechanism to reduce refresh cycle time," in *ASPDAC,* 2014.
[10] J. Mukundan, H. Hunter, K. Kim, J. Stuecheli, and J. F. Martínez, "Understanding and Mitigating Refresh Overheads in High-density DDR4 DRAM Systems," in *ISCA*, 2013.
[11] J. D. Leidel and Y. Chen, "HMC-Sim: A Simulation Framework for Hybrid Memory Cube Devices," in *IPDPSW,* 2014.
[12] K. Chen, S. Li, N. Muralimanohar, J.-H. Ahn, J. B. Brockman, and N. P. Jouppi, "CACTI-3DD: Architecture-level modeling for 3D die-stacked DRAM main memory," in *DATE,* 2012.
[13] C. Bienia et al., "The PARSEC Benchmark Suite: Characterization and Architectural Implications," in *PACT*, 2008.
[14] N. Binkert et al., "The Gem5 Simulator," *in Comp. Arch. News*, 2011.
[15] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "DRAMSim2: A Cycle Accurate Memory System Simulator," in *Comp. Arch. Letters*, 2011.
[16] JEDEC DDR3 SDRAM Standard, 2009.
[17] Hybrid Memory Cube Consortium. "Hybrid Memory Cube Specification 2.0", 2014.
[18] M. Shevgoor, et al., "Quantifying the Relationship between the Power Delivery Network and Architectural Policies in a 3D-Stacked Memory Device", in *MICRO*, 2013.
[19] "JEDEC Standard: Low Power Double Data Rate 4 (LPDDR4)", JEDEC Solid State Technology Association, 2014.
[20] "JEDEC Standard: High Bandwidth Memory (HBM) DRAM", JEDEC Solid State Technology Association, 2013.
[21] U. Kang et al., "8 Gb 3-D DDR3 DRAM Using Through-Silicon-Via Technology", *IEEE JSSC*, 2010.
[22] "JEDEC Standard: DDR4 SDRAM Specification", JEDEC Solid State Technology Association, 2013.
[23] I. S. Bhati, "Scalable and Energy Efficient DRAM Refresh Techniques", *PhD Thesis*, 2014.
[24] I. Thakkar and S. Pasricha, "3-D WiRED: A Novel WIDE I/O DRAM With Energy-Efficient 3-D Bank Organization", *IEEE D&T*, 2015
[25] I. Thakkar and S. Pasricha, "3D-Wiz: A novel high bandwidth, optically interfaced 3D DRAM architecture with reduced random access time", in *ICCD*, 2014.