Smart Refresh: An Enhanced Memory Controller Design for Reducing Energy in Conventional and 3D Die-Stacked DRAMs

Mrinmoy Ghosh
School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332
mrinmoy@ece.gatech.edu

Hsien-Hsin S. Lee
School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332
leehs@gatech.edu

ABSTRACT

DRAMs require periodic refresh for preserving data stored in them. The refresh interval for DRAMs depends on the vendor and the design technology they use. For each refresh in a DRAM row, the stored information in each cell is read out and then written back to itself as each DRAM bit read is self-destructive. The refresh process is inevitable for maintaining data correctness, unfortunately, at the expense of power and bandwidth overhead. The future trend to integrate layers of 3D die-stacked DRAMs on top of a processor further exacerbates the situation as accesses to these DRAMs will be more frequent and hiding refresh cycles in the available slack becomes increasingly difficult. Moreover, due to the implication of temperature increase, the refresh interval of 3D die-stacked DRAMs will become shorter than those of conventional ones.

This paper proposes an innovative scheme to alleviate the energy consumed in DRAMs. By employing a time-out counter for each memory row of a DRAM module, all the unnecessary periodic refresh operations can be eliminated. The basic concept behind our scheme is that a DRAM row that was recently read or written to by the processor (or other devices that share the same DRAM) does not need to be refreshed again by the periodic refresh operation, thereby eliminating excessive refreshes and the energy dissipated. Based on this concept, we propose a low-cost technique in the memory controller for DRAM power reduction. The simulation results show that our technique can reduce up to 86% of all refresh operations and 59.3% on the average for a 2GB DRAM. This in turn results in a 52.6% energy savings for refresh operations. The overall energy saving in the DRAM is up to 25.7% with an average of 12.13% obtained for SPLASH-2, SPECint2000, and Biobench benchmark programs simulated on a 2GB DRAM. For a 64MB 3D DRAM, the energy saving is up to 21% and 9.37% on an average when the refresh rate is 64 ms. For a faster 32ms refresh rate the maximum and average savings are 12% and 6.8% respectively.

1. INTRODUCTION

Dynamic Random Access Memory (DRAM) is used as the bulk of the main memory in computing systems for its high density, high capacity and low cost. Due to the dynamic, leaky nature of a DRAM cell, periodic refresh operations are required for retaining the data. Such regular refreshes account for a large energy consumption in DRAMs even in the *standby* mode. For instance, a detailed power analysis of the ITSY computer [25] shows that even in the lowest power mode, the refresh power needed accounts for about one third of the total DRAM power dissipated. The refresh rate for DRAMs depends on the memory vendor and the design technology they use. A typical refresh interval is 64ms [6, 8, 10]. The refresh intervals in embedded DRAMs are an order of magnitude shorter. A typical refresh interval for an NEC eDRAM is 4ms [2], and for an IBM eDRAM implementation is 64μs [21]. During each refresh operation, the data of every DRAM bit cell is read out and then written back. This refresh can incur large power and bandwidth overhead, nonetheless, it is inevitable for the sake of data correctness.

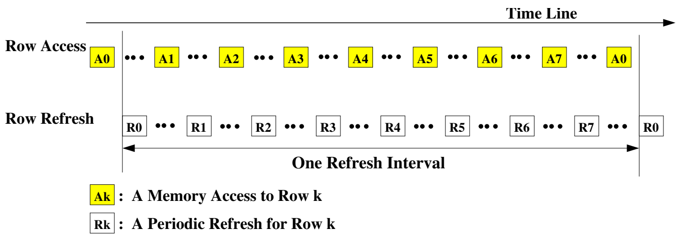


Figure 1: Best Case for Smart Refresh

As processor designers moving toward the direction of integrating 3D die-stacked DRAM (or 3D DRAM) on a package to alleviate memory latency and bandwidth issues [9, 11, 28], the overhead of the refresh operations will increase. There are two reasons behind this increase. First, a 3D DRAM could be used either as a cache between the last level SRAM-based cache and the system memory or to replace the last level cache entirely.[1] Thus, the refresh operation will become a significant overhead relatively. Second, since the 3D DRAM is bonded directly on top of the processor using die-to-die vias, the heat dissipated from the processor will be conducted across the DRAM layers, leading to a much higher temperature operation environments for the DRAM. Annavaram *et al.* [14] showed that the operating temperature of a 64MB 3D DRAM will be 90.27°C. Furthermore, the leakage will also increase exponentially with an escalating operating temperature. According to the datasheet of Micron DRAM [23], the refresh rate must be doubled if the operating temperature exceeds 85°C. Therefore, a 3D DRAM will require double (or more) refreshes, increasing the relative energy overhead substantially.

To address these issues, in this paper, we propose a novel technique called *Smart Refresh* to eliminate all the unnecessary DRAM refresh overheads. This technique uses a simple time-out counter for each row in a memory module, tracks the normal memory transactions, and eliminates the excessive refresh operations. The basic concept behind our scheme is that a memory row that has been recently read out or written to does not need to be refreshed again by the periodic refresh mechanism. By simply exploiting such property dynamically, the number of regular row-sweeping refresh operations in both conventional DRAMs and 3D DRAMs can be substantially reduced.

2. MOTIVATION

To motivate the case for our Smart Refresh technique, a conjured memory access pattern in Figure 1 is used to demonstrate the requirement for refresh operations. To simplify our illustration, we

[1] A tag array is still needed for such 3D DRAM caches for data lookup and storage. For brevity, we simply call this 3D DRAM cache a 3D DRAM hereafter.

assume that there are only 8 rows in the DRAM.

In this example, we assume that the DRAM is accessed by the processor with a regular access pattern such that each memory row is accessed right before the row is to be refreshed. For a normal, periodic refresh policy, all the memory rows will be, anyhow, refreshed by the memory controller without the knowledge of these recent accesses. Note that each access to a memory row initiated by the processor, in fact, performs an operation equivalent to a regular refresh from the standpoint of data preservation. In other words, if a row has been recently read or written to, there is no need to refresh the row immediately as shown in this figure. For the above example, in an ideal situation, there is no need to perform refresh at all since these regular memory accesses have already accomplished the same effect.

Our Smart Refresh technique exploits such energy savings opportunities by keeping a time-out counter for each row in the memory controller to minimize the required refresh cycles. Basically, the time-out counters of those rows being accessed will be reset to a default value (e.g. the refresh interval) and any following periodic refresh operation before the counter counts down to zero will be aborted. When applying such mechanism to the access pattern shown in Figure 1, the DRAM will not be refreshed at all by the default periodic refresh, without affecting the correctness. Thus we will be eliminating half of the refresh operations on the DRAM using this technique. So in theory, the best possible energy savings that can be achieved by using Smart Refresh is 50% of the entire DRAM, in which all the periodic refreshes are avoided.

The rest of the paper is arranged as follows. Section 3 describes different schemes of refreshing a DRAM. Section 4 explains how we will apply our "Smart Refresh" scheme to reduce the refresh overheads and estimates the overhead. Section 5 discusses the implementation details of "Smart Refresh" and the evaluation methodology. Section 7 analyzes the results and Section 8 discusses related work. Finally, Section 9 concludes the paper.

3. DRAM REFRESH TECHNIQUES

There are two common refresh modes in commodity DRAMs:

- Burst Refresh: In this scheme, the entire refresh operation of all the rows are done sequentially in a bursty fashion. The scheme is less desirable as it increases the peak power consumption of the DRAM. Moreover, during the time of the refresh operations, the DRAM module cannot handle normal access requests, causing potential performance degradation.
- Distributed Refresh: In distributed refresh, the memory controller spreads out the refresh cycles for different rows evenly across the refresh interval. This method is more favorable as it refreshes each DRAM row in a timely manner, enables accesses to rows that are not being refreshed, and minimizes the delay of normal memory requests.

In addition, a DRAM refresh cycle can be implemented in two distinct ways [24]. Note that a refresh cycle can be executed in either the distributed mode or the burst mode explained above.

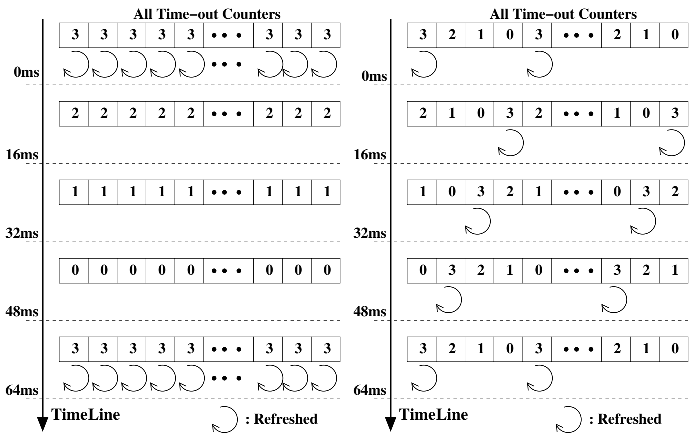
- RAS-only refresh: To perform a RAS-only refresh, a row address is put on the address lines and then the RAS (Row Address Strobe) signal is asserted LOW. When the RAS falls, that row will be refreshed as long as the CAS (Column Address Strobe) signal is held HIGH. It is the DRAM controller's function to provide the addresses to be refreshed and make sure that all rows are being refreshed at the appropriate times. It is important to note that for refresh operations the row order of refreshing does not matter; however, each row must be refreshed before the data stored by the cell is destroyed.
- CAS before RAS refresh: This is often referred to as CBR refresh, and is a frequently used method for refresh because it is easy to use and provides the advantage of lower power. A CBR refresh cycle is performed by setting the CAS signal to LOW (active) before the RAS signal is switched from HIGH to LOW. One refresh cycle will be performed each time the RAS signal falls. The Write Enable (WE) signal must be held HIGH during the period when the RAS signal is falling. The memory module contains an internal address counter which is initialized to a preset value when the device is powered up. Each time a CBR refresh is performed, the device refreshes a row based on the counter value, and then the counter is incremented. When CBR refresh is performed again, the next row indicated by the counter is refreshed followed by an increment in the counter. The counter is wrapped around automatically when it reaches the maximum allowable value equivalent to the number of rows. There is no way to reset the counter once set after initializing. Conventionally, CBR refresh is a more favorable refresh policy as it consumes lower power because the address does not have to be put on the bus. In this paper we will show that our Smart Refresh technique is suited to RAS-only Refresh, and despite the overhead over CBR, RAS-only refresh method with our Smart Refresh technique shows significant energy savings over a CBR refresh policy.

4. SMART REFRESH

4.1 Basic Operation

Inspired by the Cache Decay work [19], our *Smart Refresh* technique applies the idea of using time-out counters in the context of the refresh operation of a DRAM to reduce dynamic energy consumption. Before we discuss Smart Refresh, we will discuss the basic operation of a DRAM access in more detail. Any DRAM read or write operation initiated by a bus agent (e.g., the processor) starts with the memory controller selecting a bank and asserting the RAS signal to LOW to be active. It simultaneously posts the row address on the address bus. This causes the corresponding memory module to activate the sense amplifiers for the entire row, and the data from the given row is brought into the sense amplifiers. Note that this read operation essentially destroys the data present in the DRAM cells. Subsequently, the CAS signal is set from HIGH to LOW (active) and the column address is placed on the address bus, which causes the column decoder to multiplex the data out for a read operation. In the case of a write operation, the data on the data bus is written to the correct set of the sense amplifiers. The data for the open row stays in the sense amplifiers until there is an access to another bank or a different row. In either case the data in the sense amps is written back to the original cells and the new row is pre-charged. We know that the refresh operation of a DRAM also involves reading from the cells and writing back to them. Thus we can see that a read or a write to a given row in the DRAM is actually the same as a refresh to that row for data retention purposes. To summarize, whenever a row is accessed, it does not need to be refreshed before another refresh interval is due. If the memory controller can keep track of the rows that have been accessed, then it can potentially delay the refresh of rows that have been recently accessed. This brings us to the concept of Smart Refresh.

The basic idea of our technique is to associate a time-out counter for each *(bank, row) pair* of a memory module. The proposed array of time-out counters is stored and updated in the memory controller. Each time-out counter is simply a 2-bit or 3-bit binary down counter. The time-out counters uniformly count down from its maximum value to zero within the refresh interval of the DRAM. If the value of a counter reaches zero, it indicates that the particular row must be refreshed. The counter is reset to its maximum value whenever the corresponding bank and row in memory is accessed and the row is opened. Since we assume an open page policy in this work, the counter corresponding to an open row is reset again when the page is closed with a precharge operation. This is because during the closing of a page, the values in the DRAM cells of the page are automatically refreshed. The memory controller does not refresh rows whose corresponding counters have a non-zero value. Hence that particular row for the accessed bank will not be refreshed during the regular refresh period. This means that



(a) Timeout counters decremented together

(b) Timeout counters initially staggered

Figure 2: Down-counting Timeout counters

whenever a row is accessed for a normal memory operation (e.g., one induced by a cache miss), the refresh operation for that row is delayed. In the best case, if every row happens to be accessed right before it needs to be refreshed, there will be no need for a separate, default refresh operation.

4.2 Staggered Countdown

In this section we analyze the potential problems of accessing all time-out counters simultaneously. Let us consider a Smart Refresh memory controller that has a 2-bit time-out counter for each row of the DRAM. The array of counters is illustrated in Figure 2(a) horizontally. The refresh cycle in this example is assumed 64ms. For simplicity we assume that there is no access to the DRAM in these examples. The figure shows the counter value for each row of the DRAM as it is being updated by the memory controller. The timeline flows from top to bottom. The 2-bit counter is designed to down-count from 3 to 0 within 64 ms to ensure refresh to all rows are done timely to retain correct data values. If all counters are decremented simultaneously as shown in Figure 2(a), then they will be decremented at times 16ms, 32ms and 48ms respectively. At 48ms, all the counters reach 0 and when the memory controller accesses them again at 64ms, all the rows must be refreshed at that time, similar to a burst refresh condition that adversely reduces memory system performance. We should note that even though all the rows need to be refreshed at the same time, they can only be refreshed in a sequential order.

One solution to partially take care of this unwanted burst refresh situation is shown in Figure 2(b). In this figure, the initialization of the time-out counters is staggered.In this case, one quarter of all the counters will decrement to zero at 16ms, another quarter become zero at 32ms, and so on. We have a situation similar to burst refresh where many memory rows need to be refreshed one after another. This staggering at the beginning also incurs some power overhead, because at the beginning even all rows have been refreshed, but 1/4 of the counters are initialized to 0. Therefore they are refreshed again within the first 64ms. This however does not solve our problem. When the rows are accessed during normal processor reads and writes, their corresponding counters are reset to its maximum value. This could lead to burst refresh like conditions as potentially a large number of counters may have the same value and since they are decremented together, they will all count down to zero at the same time. This problem can be solved only if the decrement to the counters is also staggered along with the initialization.

The solution used in our design is shown in Figure 3. In this scheme, the counters are evenly hashed into N logical segments where $N = 4$ in this illustration. The selection of N segments is based on the size of the *pending refresh request queue* to be explained in Section 5.[2] The major difference of this technique with previous techniques is that in this solution all the counters will not be accessed by the memory controller simultaneously.

In this new staggered scheme, refresh or counter decrement by the memory controller are only allowed for those four *indexed* counters (with arrows shown on top of the counter in the figure) at a given time. As a result of the hashing function, only N counters (4 in this case) are active at the same time. The goal of this scheme is to index each counter exactly once within a so-called *counter access period* which is defined as the refresh interval (i.e. 64ms in our example) divided by the size of the counter (= 2^{2bit} = 4). In Figure 3, the counter access period is 16ms. The index is advanced to the next counter by a clock period equal to the counter access period divided by the number of time-out counters (i.e. memory rows) within each segment. For example, if there are 16 memory rows for each segment and the refresh period is 16ms, then the counter index will advance by one every 1ms. The update of the counter is the same as previously described. When the value of the indexed counter is zero, at the next time it is indexed again, the counter will be reset back to the maximum value followed by a refresh request for the corresponding memory row; otherwise, the indexed counter value simply decrements by one. The refresh request is immediately sent to the pending refresh request queue for dispatching a refresh operation. Without any memory accesses issued by the memory controller, the refresh policy is similar to a distributed refresh policy, with each refresh operation performing a burst refresh for N memory rows, the same size of pending refresh request queue.

The above solution ensures that the number of counters accessed simultaneously is equal to the number of segments (N) chosen. This makes sure that we do not have more than N refreshes pending in the pending refresh queue simultaneously. In Section 5 we show that the for DRAMs with refresh time of 32ms, the time interval between accessing counters is enough for completing the refreshes in the pending refresh request queue.[3] This staggering algorithm also ensures accesses to counters at regular intervals and thus the staggering will not reduce over time, avoiding any possible situation where burst refresh may occur.

Now let us assume there are normal accesses intersperse with refresh operations. Whenever a memory row is accessed by normal reads or writes, the counter corresponding to the row will be reset to the maximum value. Thus refresh operation to the counter will be delayed until it counts down to zero. Our staggered countdown mechanism guarantees that another refresh only takes place 64ms after the row has been accessed instead of a regular refresh period. This delaying of refresh of memory rows that are being accessed enables Smart Refresh to save significant amount of refresh energy if enough rows are accessed.

The size of the counter is chosen to be 2 bits for our explanation for the technique. We actually used a 3-bit counter for our simulations. The size of the counter determines the granularity with which the refresh operations can be controlled. A larger sized counter will need more steps to count down and allow more finer grained control over how much time the refresh operation can be delayed once the corresponding row is accessed. This will lead to potentially greater power savings at the cost of maintaining and accessing a bigger counter array.

4.3 Smart Refresh Correctness

We prove that for an arbitrary access pattern the Smart Refresh scheme always refreshes the data within the refresh interval deadline. The proof for this is pictorially described in Figure 4. For the example shown in the figure the refresh interval chosen is 64 ms and the counter is 2 bits wide. The figure just shows the Smart

[2]All simulations were done using 8 entry pending refresh queue and 8 segments.

[3]Proof for the 32ms case automatically proves the 64ms case.

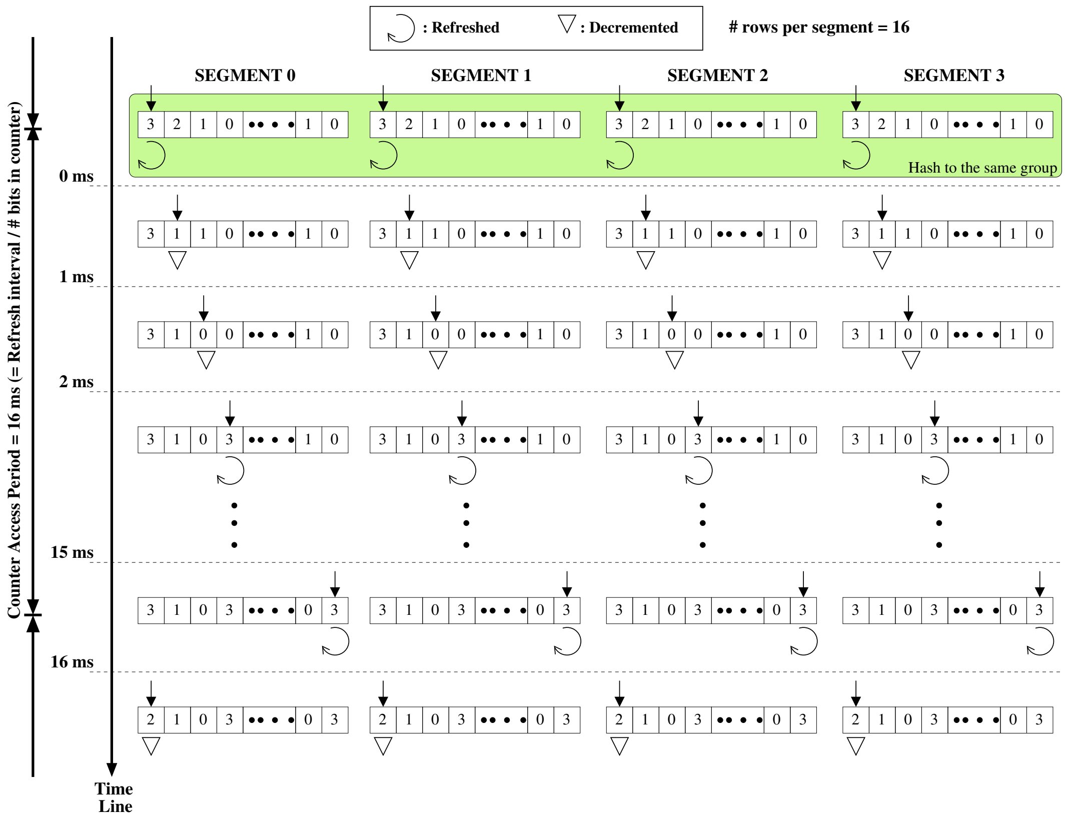


Figure 3: Countdown counters divided into logical segments and countdown is staggered

Refresh technique is applied to one particular memory row and its associated counter. The inverted triangles show the times when the counter is decremented. The number above the triangle represent the counter value after it was decremented. As explained earlier, the counter is decremented exactly once within 16ms. We can have only two possible cases for an access to this row. The figure shows that in both cases the row will guarantee be refreshed within 64ms.

In the first case on the left-hand side, the row is accessed *D ms* before it is decremented. The access is denoted by an upward arrow. Note that $D < 16ms$. An access to the row resets the counter to its maximum value 3. After *D ms* the counter is indexed and decremented to 2. From the timeline progression the counter becomes *0* in *D + 32 ms*. Thus when the counter is accessed again at *D + 48 ms*, the memory controller sees a *0* and refreshes the row. Therefore, in this case the row is refreshed after *D + 48 ms* after it is accessed and meets the deadline of 64ms.

The second possible case is that the row is accessed *D ms* after the counter is decremented as shown in the right-hand side. The counter value becomes 3 on the access. It gets decremented to 2 at *16ms -D*, and *0* at *48ms - D* after its access. Finally, it is refreshed at *64ms -D* after it was accessed. Since this is less than 64ms, the refresh is effective.

For a D greater than 16ms, the scenario can be reduced to either Case 1 or Case 2 by subtracting 16ms from D repeatedly until D is smaller than 16ms. Therefore, we show that in all possible access patterns, the row will always be refreshed before its data retention deadline.

4.4 Optimality of Smart Refresh

We define optimality for refresh as a metric of how close a DRAM row is refreshed to the data retention deadline. Thus an ideal scheme where each row is refreshed exactly after 64ms is said to be 100% optimal. In the Smart Refresh case, the optimality of the scheme depends on the number of bits we use for each counter. We can easily see from Section 4.3, if we use a two-bit counter for every row, the least optimal case will be when all the rows are refreshed at *48ms + D* where D is close to zero. Thus the optimality of Smart Refresh for a 2 bit counter is *48/64* = 75 %. Similarly for a 3 bit counter the worst case comes when each row is refreshed at *56ms*. Thus the optimality of Smart Refresh for a 3 bit counter is 87.5%. The general optimality formula is a function of the counting granularity and can be given by:

$$Optimality = [1 - \frac{1}{2^{N_{bits\_per\_counter}}}] * 100\%$$

4.5 Smart Refresh Technique for 3D DRAM

3D die stacking is an emerging technology that vertically integrates two or more die with inter-die vias [9, 11, 14, 15, 29]. These vias serve both as a fast communication interface and a stability providing mechanism to the stacked die structure. 3D die stacking reduces wire length and provides tight, high-speed coupling of die designed and manufactured with incompatible technologies. One immediate application is to integrate 3D die-stacked DRAMs with processor cores to alleviate the memory latencies and global wire power consumption by replacing long on-board wires with short, fast inter-die vias [14].

The refresh operation will be a major overhead for 3D DRAMs. The operating temperature of the 3D DRAM will likely be much higher than their conventional DRAM counterpart. As shown in [14], a 64MB 3D DRAM will raise its operating temperature to 90.27°C. According to [23], the refresh rate must be doubled after the temperature exceeds 85°C in order to retain data. Therefore, 3D DRAMs will have a higher refresh rate than conventional DRAMs, increasing the power consumption and potentially the access latency. On the other hand, another design trend could increase the number of accesses to the 3D DRAM compared to a conventional DRAM. As multi-layer DRAM is made possible to be integrated on the same package, it reduces the requirement of having a large L2 on the processor core for area/cost efficiency. The constraints on the size of

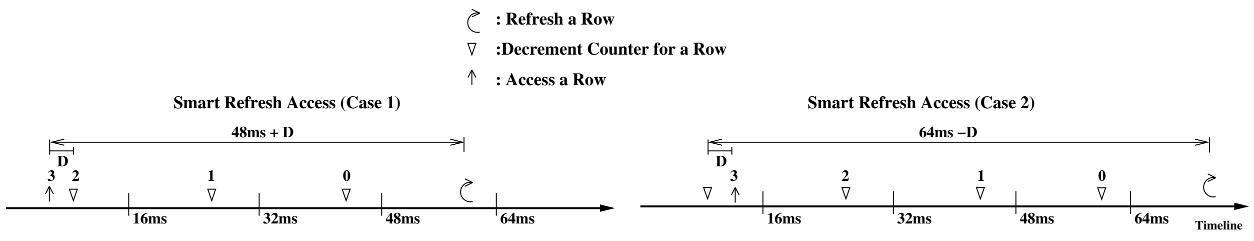


Figure 4: Smart Refresh Correctness

the 3D DRAM is mainly the number of DRAM cells that can be fitted into the available die layers and the number SRAM tags that can be fitted into the processor die for accessing the 3D DRAM.

Another interesting aspect of 3D DRAMs is that it will be more frequently accessed. Thus the refresh operation will also have a noticeable performance overhead. Our Smart Refresh technique, in fact, uses the more frequent accesses to its advantage to significantly reduce the amount of refreshes required for a 3D DRAM implementation. In Section 7.2 we will discuss in more detail the performance and energy benefits of using a 3D DRAM.

4.6 Disabling Smart Refresh

By no means will Smart Refresh always reduce refresh operations in the memory. This happens when the entire data working set fit into L1 (and L2) caches with very infrequent accesses to the DRAM. In this case the refreshing action of Smart Refresh will be degenerated to that of the CBR policy. However, Smart Refresh will consume some extra energy in maintaining those counters and also using the address bus for the RAS-only refresh operation. To avoid this situation we add a simple circuitry that can disable Smart Refresh policy and configure the memory controller to perform a regular CBR policy if accesses to memory is found to be below 1% of the number of rows over the total refresh interval (64ms or 32ms). The same circuitry will also be responsible for turning Smart Refresh on autonomously, if the accesses to DRAM exceed 2% of the number of rows in it. This turn-off is especially useful for the conventional DRAM below a large 3D DRAM cache, whose size is of the order of 32MB or 64MB as studied in [14]. Also, for the conventional DRAM we checked this policy by simulating an idle OS for 1 billion instructions. We observed that even for the idle OS we got savings of around 10% in refresh energy consumption. With such self-configurability, this feature will exploit dynamic data working set behavior for achieving the best energy management.

4.7 Area Overhead

We now explain the storage overhead for maintaining the time-out counters. In our design, we refresh one row for a specific bank and rank in a single refresh operation. Hence, we need to maintain a counter for each row and each bank in each rank. First we use the configuration shown in Table 1 for a 2GB DRAM module. The number of banks, ranks, and rows for the module is 4, 2, and 16384 respectively. Since we need a counter for every bank, rank and row the number of counters needed are 4 * 2 * 16384 = 131,072. Each of these counters has 3 bits. Thus the area overhead is (131,072 * 3)/(8 X 1024) = 48 KB. If we assume that the memory controller can support up to 32 GB, the counter space needed will be 768 KB. A simple formula is derived below for calculating the area overhead in the counters. In our experiments in Section 7, the energy overheads caused by these extra counters were all accounted for.

$$Area = \frac{N_{banks} * N_{ranks} * N_{rows} * N_{bits\_per\_counter}}{8 * 1024}(KB)$$

5. SMART REFRESH IMPLEMENTATION

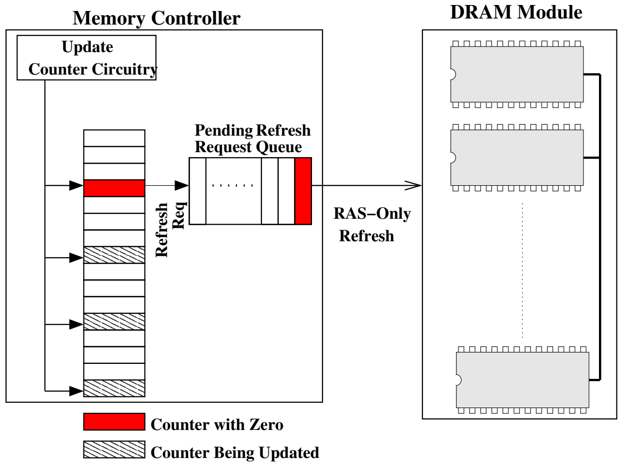


Figure 5: Smart Refresh Control Schematic

The schematic of the circuitry controlling refresh operations in the memory controller is illustrated in Figure 5. We can see that the counter update circuitry updates a specific number of time-out counters in a memory controller cycle. If one of the counters that needs to be updated has counted down to zero, then the row address and bank address corresponding to the counter are inserted into the *pending refresh requests queue*. The memory controller reads the addresses in the pending queue and puts the least recent row address on the data bus and issues an RAS only refresh command. This requires neither changes in the existing DRAM module itself nor the interface between the DRAM module and the memory controller. The only changes are inside the memory controller, making Smart Refresh a highly viable and cost effective technique.

One potential issue of having a queue to store pending refresh operations is to find out any possible case of DRAM access patterns where the queue may overflow. We show that this is not possible. A typical time taken to refresh a row is 70ns [10]. As explained earlier, if the refresh interval is 32ms and there are 8192 rows in the device, the counters are accessed every 4μs. Now if we choose the size of the refresh pending queue to be 8 entries then we will divide the row into 8 segments. This will guarantee that at most 8 refresh operations are triggered at a time. To avoid overflow for a queue having eight entries, it is essential that all the eight refresh transactions are handled till the next time the counters are accessed. Since refreshing a row takes 70ns and the counters are accessed every 4μs, if there is no normal DRAM access, the number of rows that may be refreshed between successive counter accesses will be 57. Nevertheless, in the worst case, we only need to refresh 8 rows in that deadline. Thus a queue of length 8 is sufficient for the purpose and it will never overflow. In the worst case, normal DRAM accesses may get delayed due to at most 8 refresh requests coming one after another. However, our experiments show that for all the benchmark programs considered, since we reduce the refresh operations considerably, any interference refreshes may have with normal accesses is reduced and we always have a slight performance improvement.

Parameter	**Value**
Type	DDR2
Size	2 GB and 4GB
Rows	16384
Frequency	667 MHz
Number of Banks	4 and 8
Number of Ranks	2
Number of Columns	2048
Data Width	72 bits (64 data + 8 ECC)
Row Buffer Policy	Open Page
Refresh Interval	64ms
L2 Cache Size	1 MB
Number of L2 Port	1
L2 Cache Assoc	8 ways

Table 1: DRAM Module and L2 Cache Configuration

Parameter	**Value**
Type	DDR2
Size	64 MB
Rows	16384
Frequency	667 MHz
Number of Banks	4
Number of Ranks	1
Number of Columns	128
Data Width	72 bits
Row Buffer Policy	Open Page
Refresh Interval	32ms
Ports	1
Associativity	Direct Mapped

Table 2: 3D DRAM Cache Configuration

We would like to emphasize that Smart Refresh for RAS only refresh does not change the interface between the Memory Controller and the DRAM module. Although CBR refresh is often chosen as the refresh policy for modern DRAMs, we use it as a baseline in our results to show that Smart Refresh provides significant savings even after considering the additional overhead of RAS only refresh.

Another potential issue with "Smart Refresh" is the design of the memory controller with requisite number of counters, as the size of DRAM is not known when the memory controller is designed. This problem can be handled by the memory controller having multiple banks of count-down counters. The total number of counters would be the number of rows for the maximum permissible size supported by the memory controller. The BIOS will turn on requisite number of banks on startup of the system, based on the memory size and configuration.

6. EVALUATION METHODOLOGY

Our simulation infrastructure consists of three portions: *Simics* [12], *Ruby* [22] and *DRAMsim* [33]. Firstly, we used Virtutech Simics to execute the benchmark applications. Simics is a full system emulator that can run unmodified production software like full blown operating systems. This infrastructure was used to emulate a "Sun" virtual machine called "sarek" running a version of Solaris 8. We used three different benchmark suites — SPLASH2 [34], SPECint2000 and Biobench [13] for their different memory behaviors. All programs were compiled for the Solaris machine and installed in the virtual disk.[4] Except for SPLASH2 that was executed on a 2-processor emulated CMP system sharing a 1MB conventional L2 cache, all other benchmark programs were run on a uni-processor system. We also run a set of experiments where we selectively pair off any two SPECint benchmark programs and run them together to emulate a multi-workload execution environment.

[4]Although we could successfully compile all programs, not every benchmark ran for sufficient number of instructions due to limitations and incompatibilities of the simulation infrastructure. We report results for those applications that successfully ran on the simulated system.

These experiments were performed to observe the effect of more frequent memory look-up's for our technique. Although Simics is a full system emulator, we only use it for functional simulation. To simulate memory and cache behavior in details, the Ruby module developed at University of Wisconsin was loaded into Simics. Ruby leverages the full system infrastructure of Simics and provides timing simulation for the memory hierarchy. However, Ruby does not faithfully simulate the DRAM behavior. The characteristics of DRAM were, on the other hand, simulated using a third simulator called DRAMsim [33] from University of Maryland. DRAMsim can be used either as a standalone trace-driven simulator or as a module that can be integrated into Ruby. The complete implementation of our Smart Refresh technique was done in DRAMsim. Table 1 shows the DRAM module and the L2 cache size used in our simulation. We used the conventional DDR2 memory rather than the latest FB-DIMM for our simulation because the conventional DDR2 performs better for benchmarks that are not limited by bandwidth [18], which is the case for our benchmarks. The module configurations were based on actual DRAM specification from [7]. The 3D DRAM Cache configuration is shown in Table 2. The DRAM refresh command policy is one-channel, one-rank, one-bank for all configurations. Each benchmark was simulated for 1 billion instructions after fast-forwarding the first billion instructions.

The calculation of power involves two distinct components: the power consumption of the DRAM module, and the power overhead of the newly proposed time-out counters. To calculate power consumption for the DRAM module we used the power model provided by DRAMsim [33]. For the time-out counters we assume a design consisting of an array of SRAM bits storing the counter values and a logic circuit for the decrement operation. The SRAM bit array has entries equal to the number of rows in the DRAM which is of the order of thousands. Accessing such an array will need a large decoder and very long bit-lines to transfer the data out. In contrast, the counter logic will have tens of gates. Therefore the energy consumption of storing and accessing the array of SRAM bits will be an order of magnitude larger than the energy consumed by the logic circuitry to decrement the respective values. Thus the energy consumption of the logic circuitry was neglected in our energy calculations. The SRAM array was designed using the *Artisan* 90nm SRAM library [1] to get an estimate on the dynamic energy required to access it. The Artisan SRAM generator is capable of generating synthesizable Verilog code for SRAMs using 90nm process technology. The generated datasheet gives an estimate of the read and write power of the generated SRAM. The counter arrays may be accessed in two different situations. First, when a specific row is accessed and its corresponding time-out counter needs to be reset. This is considered as a write operation to the SRAM array. The second case is that when a counter is checked against zero value for triggering a refresh. When the value is positive, it is decremented. As explained in Section 4.2, whenever the counters are accessed for decrementing, eight counters are decremented at the same time. Therefore, in our design we count eight reads and eight writes for each such counter access operation. The results of the simulation will be presented in the next section.

Since Smart Refresh uses RAS-only refresh that consumes relatively more energy than CBR refresh due to the requirement of posting the row address, we assume that the baseline DRAM uses CBR refresh (a lower power baseline) in our experiments, while the Smart Refresh DRAM is based on RAS-only refresh. The extra power for RAS-only refresh is mainly consumed in putting the row address to be refreshed on the bus. To model the power consumption of the bus we use the elementary model explained in [16]. The energy consumption of the bus is given by:

$$\text{Energy} = C * V_{DD}^2 * Width\_of\_Bus * Num\_Accesses$$

Here "C" is the average capacitance of one wire of the bus and is given by:

$$\text{C} = C_{load} + C_{driver}$$

Parameter	Value
On Chip Length	36 mm
Off Chip Length	102 mm
On Chip Wire Capacitance	.21 pF/mm
Off Chip Wire Capacitance	0.1 pF/mm
Input Capacitance of Memory Modules	3 pF

Table 3: Parameter Values Used in Bus Energy Calculation

Now for proper impedance matching according to [16], C_{driver} is chosen to be 30 % of C_{load}. Thus $C = 1.3 * C_{load}$.

$$\begin{aligned} C_{load} = & \quad L_{onchip} * C_{permmonchip} \\ & + \quad L_{offchip} * C_{permmoffchip} + \sum_{m \in M} C_{in}(m) \end{aligned}$$

where "M" is the number of memory modules (ranks) in the DRAM system, and $C_{in}(m)$ is the input capacitance of each module.

The estimation of wire length on the chip is done using the widely used *"semi perimeter"* method [30]. The on-chip length (L_{onchip}) is taken as double the length of one side of the Intel 855PM Chipset MCH die [3]. Typical values of off-chip length was determined from Intel 855PM Chipset design guide [4]. Values of the on-chip capacitance per unit length was obtained from the ITRS Roadmap [5]. The input capacitance of a memory module was obtained from Micron datasheet [6]. The actual values used for these equations have been summarized in Table 3.

Apart from evaluating our technique for conventional DRAMs we also performed experiments to evaluate the effectiveness of Smart Refresh for the emerging 3D DRAMs. We extend DRAMsim functionality to simulate the processor using 3D DRAM as another level of cache between L2 and the on-board DRAM. Note that, to access and allocate data on the 3D DRAM cache, an SRAM tag array is still needed on the processor. We implemented Smart Refresh for such a configuration and ran the same benchmark programs for two different sized 3D DRAM (32MB and 64MB). Practically, the 3D DRAMs are level 3 caches integrated directly on top of a processor core with a 256KB Level 2 SRAM cache. The capacity of the 3D DRAM is limited by the core area and the number of DRAM die layers available. We chose the sizes in accordance with the results of the feasibility study given in [14]. According to [14], 3D DRAMs will operate at much higher temperatures (90.27°C) than conventional DRAMs, we performed our experiments using two different refresh intervals (32ms, and 64ms). The following section discusses our results.

7. EXPERIMENTAL RESULTS

7.1 Conventional DRAM

Figure 6 shows the number of refresh operations per second taking place for each benchmark program. To show the effectiveness of our technique, we mark the baseline number of refreshes per second required for a memory module with the same configuration.

From Figure 6 we can observe that, though the relative reduction in refreshes heavily depends on the memory behavior of an application, the Smart Refresh technique is very effective in reducing the number of regular refresh operations. The reductions in refresh operations per second range from around 26% for fasta to as high as 85.7% in water-spatial. On average, our technique can reduce more than 59.3% of regular refresh operations over all the benchmark programs.

Figure 7 shows the relative energy consumption for Smart Refresh for refresh operations. We can see that Smart Refresh is successful in saving a significant percentage of energy consumed in refreshing the DRAM. The savings range from 25% in gcc to as much as 79% for radix. On an average Smart Refresh saves 52.57% of energy consumed in DRAM refresh. We should note that there does not exist a linear relationship between the percentage reduction in the number of refresh operations and the relative reduction in refresh energy. This is because the energy consumed in refreshing a row depends on the state of the bank where the row is being refreshed. For example, if the row of sense amplifiers is in the *precharge* state for the bank where a row is being refreshed, the refresh operation involves bringing the row of data being refreshed to the sense amps, restoring their charge, writing them back to the row and precharging the sense-amps for the next operation. However, if the row of sense amps already has an open page, the refresh operation will involve writing the present open page back to the DRAM cells, precharging the sense amps and then refreshing the row as above. This clearly consumes more energy than the case when the sense-amps were already precharged.

Figure 8 shows the relative energy consumption for the DRAM. We took into account the energy overhead of maintaining the time-out counters in the memory controller. We can see that benchmarks that have high refresh energy savings also have large savings in the total energy. Thus benchmark programs such as perl_twolf whose relative refresh energy savings is high, show high total energy savings of 25%. On average, the total savings of DRAM energy is around 12.13%. However, we must also note that there is no exact linear relationship between the relative refresh energy savings and the total DRAM energy savings. This is because the total energy savings depend heavily on what percentage of the total DRAM energy is contributed by refresh energy. This depends on the number of memory references of a benchmark.

Figure 9, Figure 10, and Figure 11 show the number of refresh operations per second, relative refresh energy and relative DRAM energy for a 4GB conventional DRAM. For the baseline CBR refresh technique, we find that the number of refresh operations per second for every benchmark with the 4GB DRAM module is doubled compared to the 2GB module. This is expected because the 4GB DRAM module has double the number of banks. Since the refresh command policy used in all experiments is one channel/one bank/one rank, the number of rows that need to be refreshed for a 4GB module is twice the number of rows of the 2GB module. As all benchmarks require similar number of cycles to complete in both the 2GB and 4GB configurations, the number of refreshes for the 4GB module is doubled. We observe that the relative reduction in refresh operations is around 40% for a 4GB DRAM. The average reduction in refresh energy is 23.76% and total energy reduction is 9.10%. The energy savings for a 4GB DRAM is generally lower because all the benchmarks simulated have a memory footprint less than 2GB. So on using a 4GB DRAM we increase both the base DRAM energy consumption and also the energy required to maintain double the number of time-out counters. This reduces the savings that can be obtained using the Smart Refresh technique. For example, phylip from Biobench had about 13.3% total energy savings as shown in Figure 8 while the savings dropped down to almost 7.3% as shown in Figure 11. Also, as in the case of the 2GB DRAM module, there is no linear relationship between relative reduction in refresh operations, the refresh energy savings, and the total energy savings.

7.2 3D Die-stacked DRAM

To study the benefit of Smart Refresh for emerging 3D integration technology, we performed experiments by assuming a limited size DRAM bounded with a processor through die-to-die vias. The results for the 3D die-stacked DRAM with a capacity of 64MB total and a 64ms refresh period are shown in Figure 12, Figure 13 and Figure 14. As in the case of the conventional DRAM, the reduction in the number of refresh operations per second shown in Figure 12 highly depend on the benchmark considered. The reduction in refresh operations is significant. It ranges from 42% in mummer to 4% in fasta. Figure 13 shows the relative energy savings using Smart Refresh for 3D DRAMs with 64ms refresh rate. Though there is no linear relationship, we observe that the refresh energy savings follow the number of refresh operations reduced per sec-

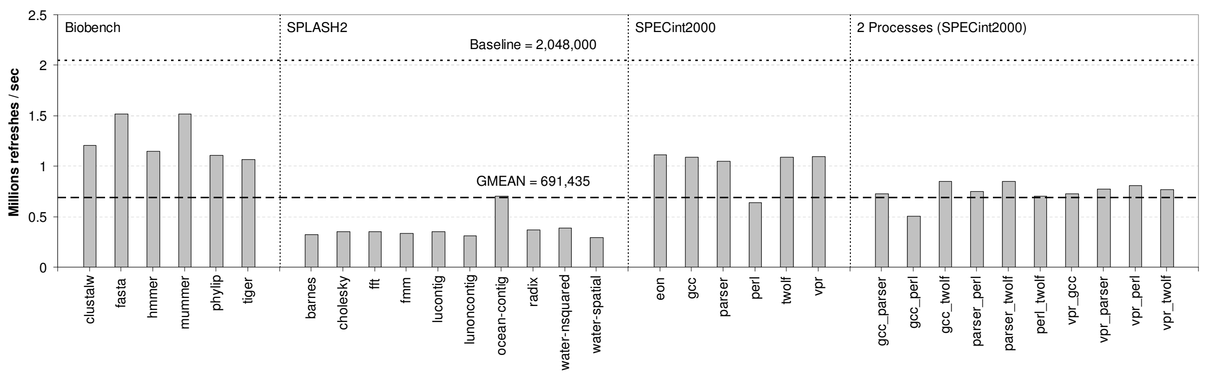


Figure 6: Comparison of Number of Refreshes per second for a 2GB DRAM

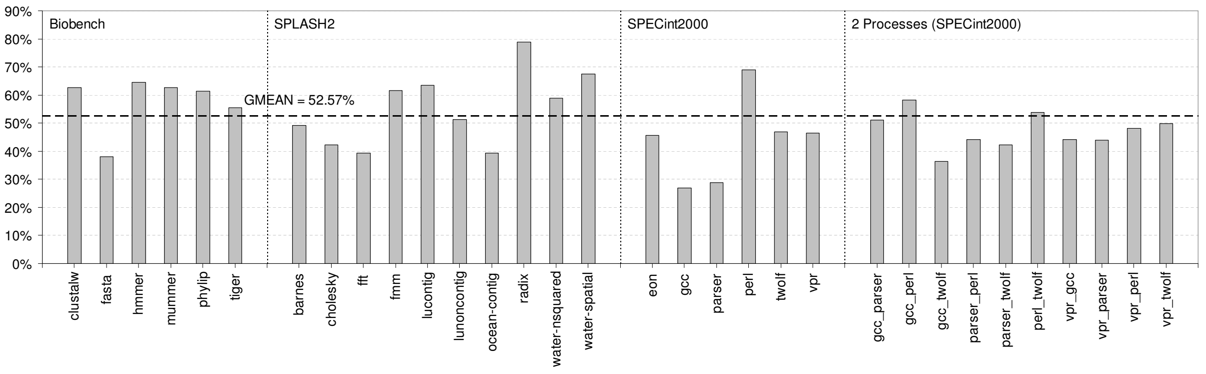


Figure 7: Relative Refresh Energy Savings for a 2GB DRAM

ond. The savings range from 42% in the Biobench benchmarks like clustalw, mummer, to as low as 7% in fasta. The geometric mean of refresh energy savings is 21.91%.

Figure 14 shows the total energy savings for the same 3D DRAM configuration. These savings numbers consider that the baseline 3D DRAM cache uses CBR refresh. Thus the power consumption in the wires and vias connecting the memory controller in the processor die and DRAM in the stacked die have been modeled and added as an overhead for the Smart Refresh technique. It can be seen from the savings that refreshes are a significant overhead in 3D DRAMs. We obtain savings of up to 21.5% when running gcc and twolf together. The geometric mean of the savings are 9.37%. A general trend that can be observed from these simulations is that the savings continue to increase for those systems running two processes. One reason for this is that dual process benchmark runs contain less spatial locality of accesses than a single benchmark. So it is more likely for a 2-process benchmark to access different rows rather than having a row buffer hit all the time. Since different rows are accessed, fewer number of rows need to be refreshed and this helps in saving energy. For the 3D DRAM cases, we also had a 2GB conventional DRAM that back up the 3D DRAM which is essentially used as a Level 3 DRAM cache. Since these benchmark programs fit into the 64MB cache and accesses to main memory was negligible, in the order of a few thousands over more than 2 billion cycles for all the benchmarks, thus we did not observe Smart Refresh shows any energy savings for the conventional DRAM with a 64MB 3D DRAM integrated on top of a processor face-to-face. But since Smart Refresh can effectively switch off all the counters and go to CBR refresh mode when less than 1% of the rows are accessed over a whole refresh interval as described in Section 4.6, we did not detect any energy loss in the conventional DRAM.

Since the 3D Cache will operate at a temperatures of 90.27°C, the refresh rate required will likely be doubled from the 64ms refresh rate. Therefore, we conducted experiments on the same 64MB 3D DRAM with a faster 32ms refresh rate.

Figure 15 compares the number of refresh operations per second using Smart Refresh with a conventional CBR refresh for the 64MB DRAM with the doubled 32ms refresh rate. As expected, the trend in number of refreshes is similar to the 64ms case, but the baseline number of refreshes is scaled up to twice the number of refreshes in the 64ms case. But since the number of accesses is constant, the number of refreshes eliminated is reduced. This is better illustrated in Figure 16, which shows the relative refresh energy savings using Smart Refresh for 3D DRAMs with 32ms refresh rate. Although the trends are similar to the 64ms case, the relative refresh energy savings is in general less than the 64ms case. The geometric mean of refresh energy savings is 15.79%. The total 3D DRAM cache energy savings is shown in Figure 17. We can see that even though the refresh energy savings are modest, we get a decent saving in total energy. The geometric mean of the total energy saved across benchmark suites is 6.87%. One reason for this is even though relatively refresh savings have reduced, refresh energy for the 32ms case accounts for a large part of the total energy. Thus the net energy savings for the 32ms is not much different than the 64ms case.

7.3 Performance Implication

While the objective of Smart Refresh is aimed at reducing the number of periodic refresh operations, it also shortens the potential memory access delays caused by these redundant refresh operations. Figure 18 shows the performance benefit of using Smart Refresh applied to a 3D DRAM (32ms refresh rate) over a conventional CBR refresh policy. It can be seen that for all the benchmarks we have very slight (less than 1%) improvement in performance. For all the other DRAM and 3D DRAM configurations with Smart Refresh, we found very similar performance results. This shows

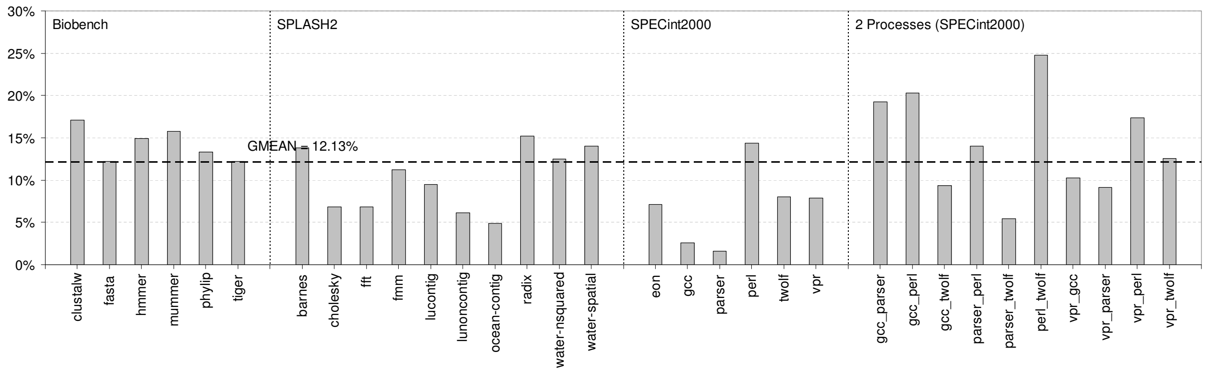


Figure 8: Relative Total Energy Savings for a 2GB DRAM

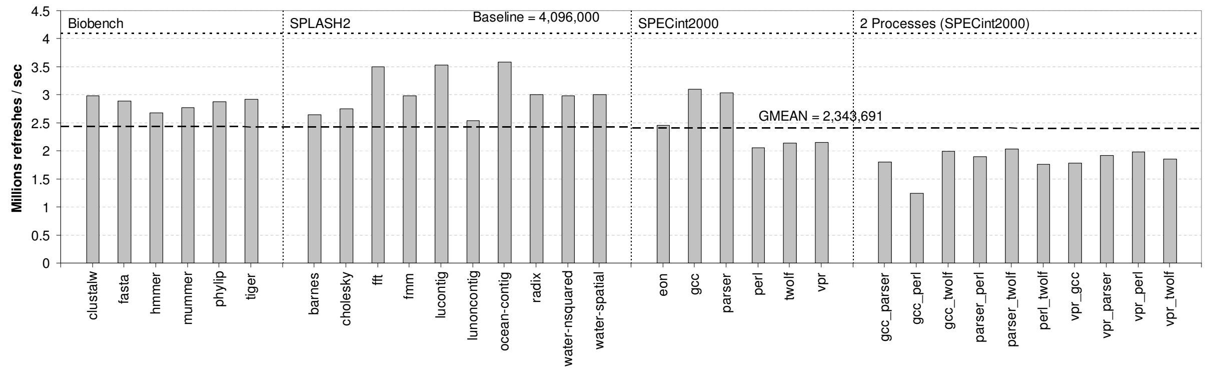


Figure 9: Comparison of Number of Refreshes for a 4GB DRAM

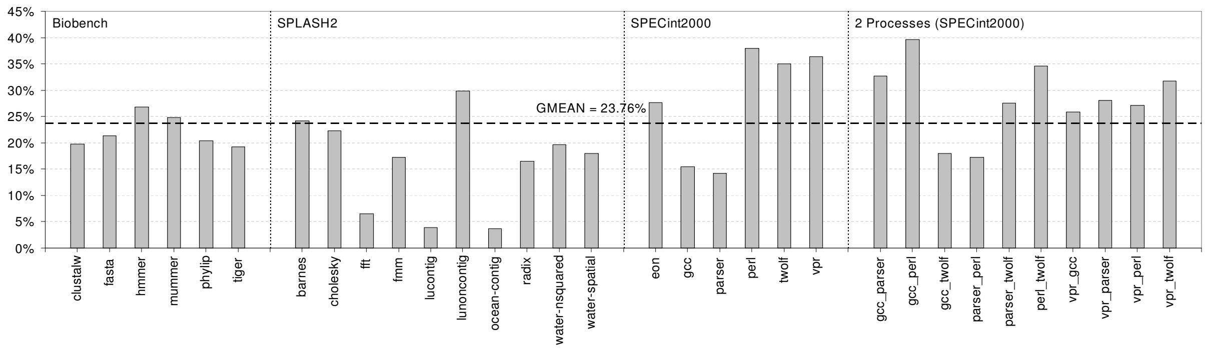


Figure 10: Relative Refresh Energy Savings for a 4GB DRAM

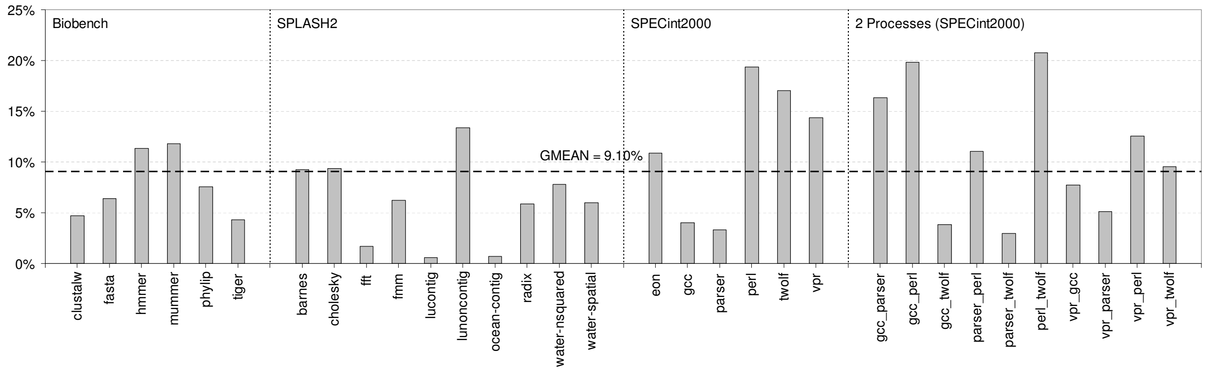


Figure 11: Relative Total Energy Savings for a 4GB DRAM

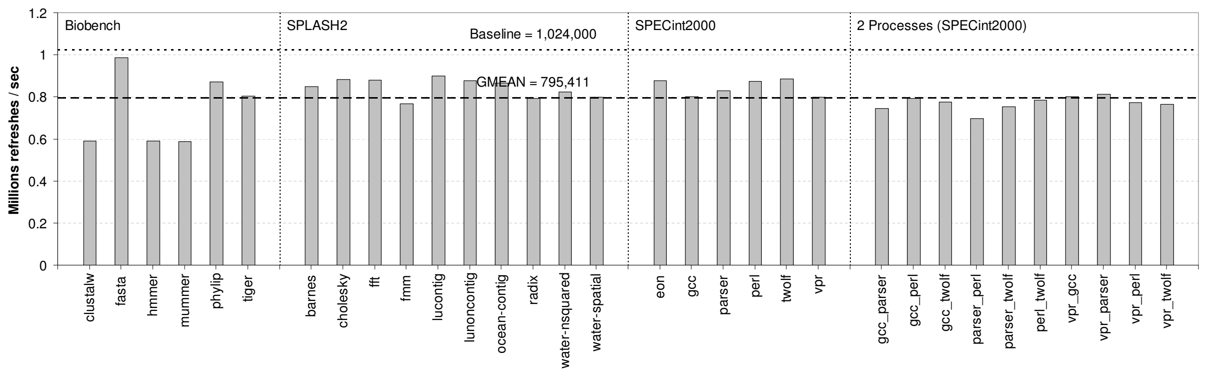


Figure 12: Comparison of Number of Refreshes for a 64MB 3D DRAM Cache with 64ms refresh rate

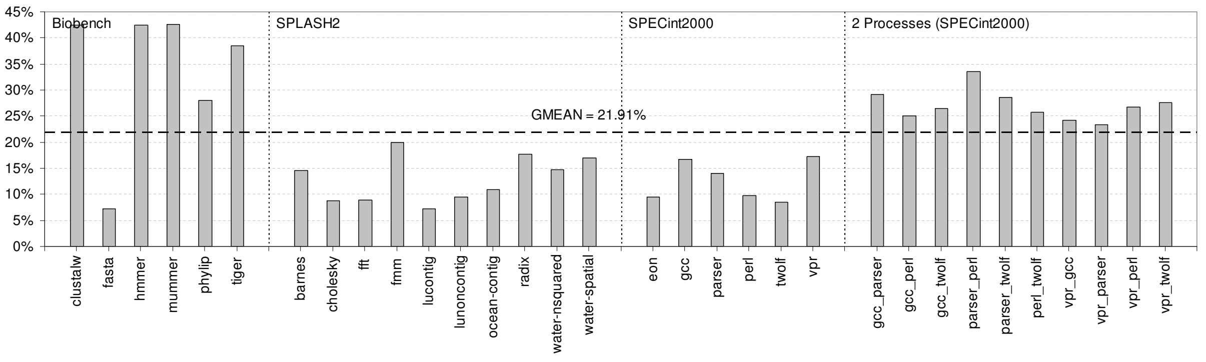


Figure 13: Relative Refresh Energy Savings for a 64MB 3D DRAM Cache with 64ms refresh rate

that our Smart Refresh technique does not incur any performance degradation, but sometimes has performance improvement.

8. RELATED WORK

Using countdown timers for tracking DRAM refresh was proposed in a patent disclosure [17]. This patent describes a timer based circuitry to reduce the number of refresh operations in a DRAM based cache. This paper is significantly different from the patent as they have different objectives. The major contribution of our work is to save DRAM power by achieving near-optimal periods of refresh for all DRAM rows with our Smart-Refresh algorithm. The patent's objective was to invalidate lines (via decay) that have not been accessed for a given time interval in the context of DRAM Caches. Our technique does not invalidate any DRAM row and thus can be used for any DRAM system adn not just caches. In addition, our paper details a novel staggering mechanism that prevents a burst-refresh situation. The broadly described patent does not include sufficient discussion of how to handle a large number of simultaneous refreshes. Furthermore, our paper illustrates the usefulness of our intelligent refresh scheme in both conventional and the emerging 3D DRAM technology.

Another similar idea of using counters to reduce refreshes is described in [27]. However, as in the case of [17], the method described in the patent is far from optimal and does not have any technique to solve the burst refresh situation. The patent disclosed by Song *et al.* [31] also described a technique to selectively refresh DRAM rows based on their access pattern. However, the technique based on the limited explanation in the patent can lead to situations where the data of a row may be destroyed because it is not refreshed in time. Our technique guarantees integrity of the data as explained in Section 4.3.

Venkatesan *et al.* in [32] introduced RAPID, a retention-aware placement algorithm. This work tries to reduce refresh operations to the DRAM by experimentally identifying that different rows require different refresh times. Our technique is orthogonal to this technique and can be applied on top of the retention-aware DRAM technique. Kim *et al.* in [20] exploits multiple DRAM refresh times and ECC codes to reduce the number of refresh operations. As in the case of [32], our technique is orthogonal to this technique and thus may be used on top of it. Ohsawa *et al.* used several techniques in [26] to reduce refresh operations required. One of the techniques used by [26] is to statically declare a line to be dead. This may also be done with the help of the OS. The lines marked as dead in the DRAM are not refreshed. Another scheme is called VRA where counters are used to handle variable data refresh times. We should point out that VRA is different from our scheme as it is done only in the context of handling different refresh times and not to optimize refreshes based on access patterns.

9. CONCLUSION

This paper presented a simple, low cost technique using time-out counters to save power in DRAMs. This technique does not involve any change in the interface between the memory controller and the DRAM, making it highly feasible. All additional hardware goes in the memory controller that controls and issues the needed refresh operations. The paper demonstrates that many refresh transactions are indeed not needed for their corresponding rows were recently accessed due to cache misses. This technique saved up to 25% and on an average 12.13% of the energy consumed in DRAMs. Modern computing systems like CMP, CMT, SMP and SMT would try to exploit MLP and would have increasing number of threads trying to access memory. In this case, the Smart Refresh technique will be instrumental in saving energy as it is very light weight and would increase the bandwidth availability and reduce energy consumption for refresh operations in DRAMs. The emerging 3D stacked ICs will enable the accesses to the DRAM at a much lower latency. Also, AMD's licensing of ZRAM technology indicates that future AMD processors may use DRAM type memory using SOI technology for their caches. This paper clearly demonstrated that the Smart Refresh technique is very useful for such DRAM type caches. Energy savings of up to 21.5% and 9.4% on an average was obtained when Smart Refresh was used in a 3D Die-Stacked DRAM Cache.

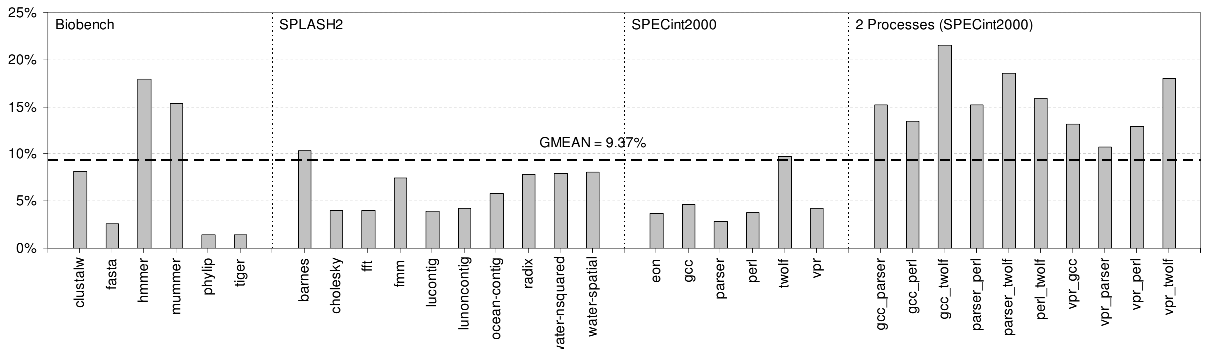


Figure 14: Relative Total Energy Savings for a 64MB 3D DRAM Cache with 64ms refresh rate

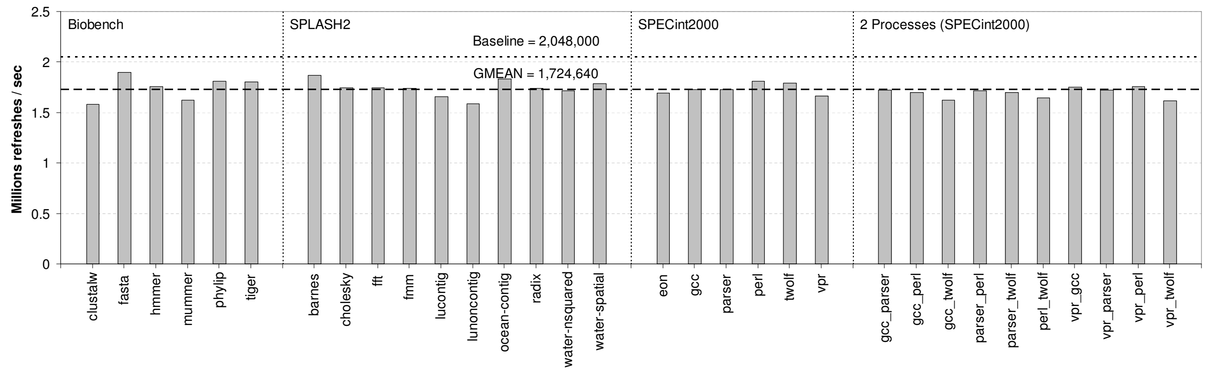


Figure 15: Comparison of Number of Refreshes for a 64MB 3D DRAM Cache with 32ms refresh rate

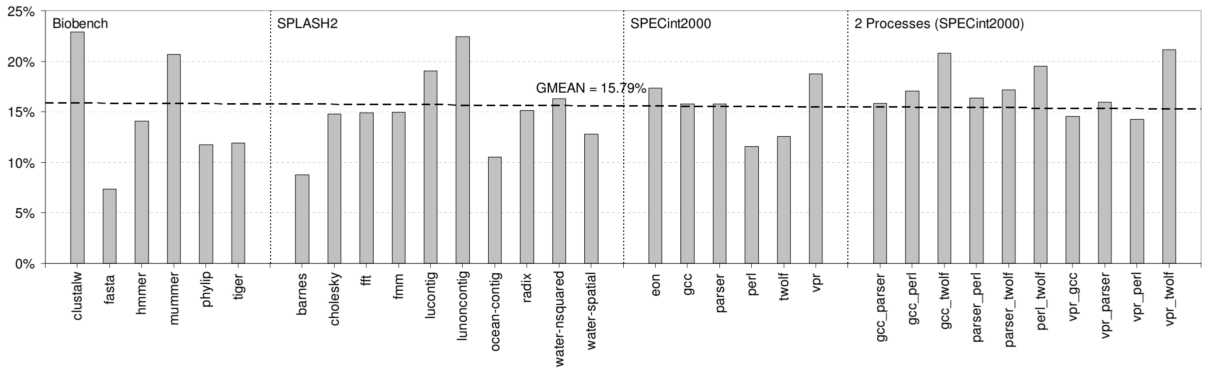


Figure 16: Relative Refresh Energy Savings for a 64MB 3D DRAM Cache with 32ms refresh rate

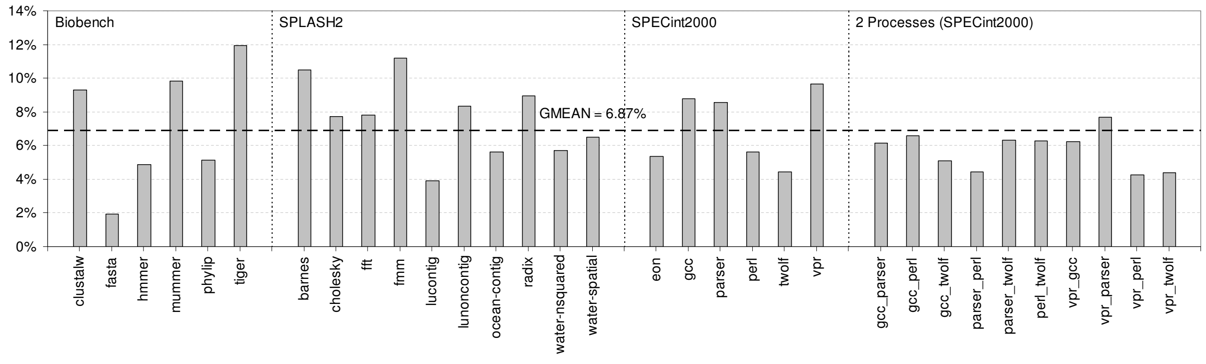


Figure 17: Relative Total Energy Savings for a 64MB 3D DRAM Cache with 32ms refresh rate

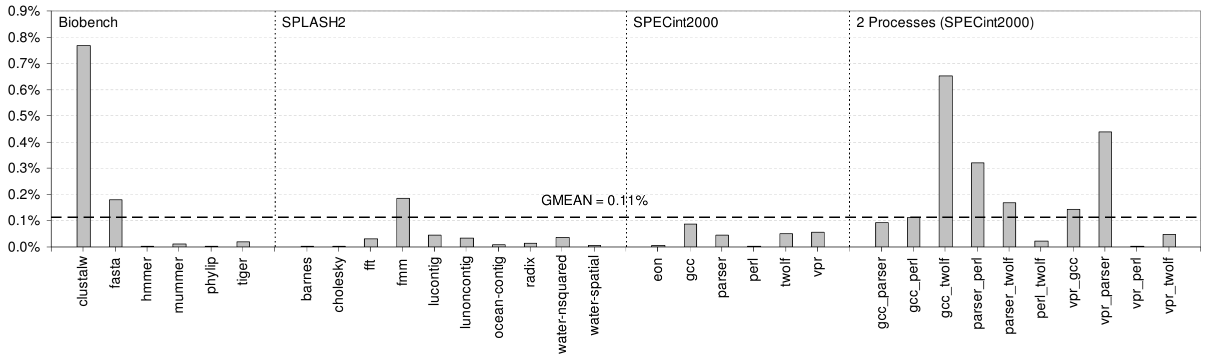


Figure 18: Performance improvement using Smart Refresh for a 64MB 3D DRAM Cache with 32ms refresh rate

10. ACKNOWLEDGMENTS

This research was supported by NSF Grant CNS-0325536, an NSF CAREER Award (CNS-0644096) and the C2S2 center of the Focus Center Research Program. We would like to thank William Reohr who first pointed out the similarity between this research and an IBM patent [17] after we presented our work the first time at IBM T. J. Watson Research Center. We also thank Onur Mutlu and anonymous reviewers for their valuable feedback to this work.

11. REFERENCES

[1] Artisan sram generator. http://www.artisan.com.

[2] Easy Integration of Embedded DRAMs. http://www.am.necel.com/process/edramsequences.html.

[3] Intel 855PM Chipset Memory Controller Hub (MCH) DDR datasheet. ftp://download.intel.com/design/chipsets/datashts/25261303.pdf.

[4] Intel 855PM Chipset Platform Design Guide. http://download.intel.com/design/mobile/desguide/25261403.pdf.

[5] ITRS Roadmap 2006 Interconnect Update. http://www.itrs.net/Links/2006Update/FinalToPost/09_Interconnect2006Update.pdf.

[6] Micron 128Mb: x32 SDRAM data sheet. http://download.micron.com/pdf/datasheets/ dram/sdram/128MbSDRAMx32.pdf.

[7] Micron DDR2 SDRAM Registererd DIMM 2GB and 4GB data sheet. http://download.micron.com/pdf/datasheets/modules/ddr2/HTF36C256_512x72.pdf.

[8] NEC 16 M-Word by 64-Bit DDR Synchronous Dynamic RAM Module Unbuffered Type Specification. http://www.nec.com.

[9] Samgsung Develops 3D Memory Package that Greatly Improves Performance Using Less Space. http://www.samsung.com/PressCenter/PressRelease/PressRelease.asp?seq=20060413_0000246668.

[10] Samsung 512Mb D-Die DDR SDRAM Specification. http://www.samsung.com.

[11] Tezzaron Semiconductor, FaStack Technology. http://www.tezzaron.com/technology/FaStack.htm.

[12] Virtutech Simics. http://www.simics.net.

[13] K. Albayraktaroglu, A. Jaleel, X. Wu, M. Franklin, B. Jacob, C.-W. Tseng, and D. Yeung. Biobench: A benchmark suite of bioinformatics applications. In *Proceedings of the International Symposium on Performance Analysis of Systems and Software*, 2005.

[14] B. Black, M. Annavaram, N. Brekelbaum, J. DeVale, L. Jiang, G. H. Loh, D. McCaule, P. Morrow, D. W. Nelson, D. Pantuso, P. Reed, J. Rupley, S. Shankar, J. Shen, and C. Webb. Die stacking (3d) microarchitecture. In *Proceedings of the 39th International Symposium on Microarchitecture*, pages 469–479, 2006.

[15] B. Black, D. W. Nelson, C. Webb, and N. Samra. 3D Processing Technology and Its Impact on iA32 Microprocessors. In *Proceeding of International Conference on Computer Design*, 2004.

[16] F. Catthoor. *Custom Memory Management Methodology: Exploration of Memory Organisation for Embedded Multimedia System Design*. Kluwer Academic Publishers, 1998.

[17] P. G. Emma, W. R. Reohr, and L.-K. Wang. Restore Tracking System for DRAM, U.S Patent No 6,839,505 B1, 2002.

[18] B. Ganesh, A. Jaleel, D. Wang, and B. Jacob. Fully-Buffered DIMM Memory Architectures: Understanding Mechanisms, Overheads and Scaling. *Proceedings of the 13th International Symposium on High Performance Computer Architecture*, 2007.

[19] S. Kaxiras, Z. Hu, and M. Martonosi. Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power. In *Proceedings of the 28th International Symposium on Computer Architecture*, pages 240–251, 2001.

[20] J. Kim and M. C. Papaefthymiou. Dynamic Memory Design for Low Data-Retention Power. In *Proceedings of the International Workshop on Integrated Circuit Design, Power and Timing Modeling, Optimization and Simulation*, 2000.

[21] T. Kirihata, P. Parries, D. Hanson, H. Kim, J. Golz, G. Fredeman, R. Rajeevakumar, J. Griesemer, N. Robson, A. Cestero, et al. An 800-MHz Embedded DRAM With a Concurrent Refresh Mode. *IEEE Journal of Solid-State circuits*, 40(6):1377, 2005.

[22] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood. Multifacet's General Execution-Driven Multiprocessor Simulator (GEMS) Toolset. *SIGARCH Comput. Archit. News*, 33(4), 2005.

[23] Micron. DDR2 SDRAM SODIMM 1GB 2GB Data Sheet. http://download.micron.com/pdf/datasheets/modules/ddr2/HTF16C64_128_256x64HG.pdf.

[24] Micron. Various Methods of DRAM Refresh. http://download.micron.com/pdf/technotes/DT30.pdf.

[25] M.Viredaz and D. Wallach. Power Evaluation of a Handheld Computer: A Case Study. Technical report, Compaq WRL, 2001.

[26] T. Ohsawa, K. Kai, and K. Murakami. Optimizing the DRAM refresh count for merged DRAM/logic LSIs. In *Proceedings of the International Symposium on Low Power Electronics and Design*, 1998.

[27] J. T. Pawlowski. Intelligent refresh controller for dynamic memory devices, U.S Patent No 5,890,198 B1, 1999.

[28] L. A. Polka, H. Kalyanam, G. Hu, and S. Krishnamoorthy. Package Technology to Address the Memory Bandwidth Challenge for Tera-Scale Computing. *Intel Technology Journal*, 11(03), 2007.

[29] A. Rahman and R. Reif. System Level Performance Evaluation of Three-Dimensional Integrated Circuits. *IEEE Tranctions on VLSI*, 8(6):671–678, 2000.

[30] S. Sait and H. Youssef. *VLSI Physical Design Automation: Theory and Practice*. McGraw-Hill, 1995.

[31] S. P. Song. Method and system for selective DRAM refresh to reduce power consumption, U.S Patent No 6,094,705 B1, 2000.

[32] R. Venkatesan, S.Herr, and E. Rotenberg. Retention-Aware Placement in DRAM (RAPID):Software Methods for Quasi-Non-Volatile DRAM. In *Proceedings of the Twelfth Annual Symposium on High Performance Computer Architecture*, pages 155–165, Nov. 2006.

[33] D. Wang, B. Ganesh, N. Tuaycharoen, K. Baynes, A. Jaleel, and B. Jacob. DRAMsim: a memory system simulator. *SIGARCH Comput. Archit. News*, 33(4):100–107, 2005.

[34] S. Woo, M. Ohara, E. Torrie, J. Singh, and A. Gupta. The SPLASH–2 Programs: Characterization and Methodological Considerations. In *Proceedings of the 22nd International Symposium on Computer Architecture*, pages 24–36, June 1995.