Improving DRAM Latency with Dynamic Asymmetric Subarray

Shih-Lien Lu
Intel Corp.
Hillsboro, Oregon 97124 USA
sllu@jf.intel.com

Ying-Chen Lin
National Taiwan University
Taipei, Taiwan ROC
lemur.yin@gmail.com

Chia-Lin Yang
National Taiwan University
Taipei, Taiwan ROC
yangc@csie.ntu.edu.tw

ABSTRACT

The evolution of DRAM technology has been driven by capacity and bandwidth during the last decade. In contrast, DRAM access latency stays relatively constant and is trending to increase. Much efforts have been devoted to tolerate memory access latency but these techniques have reached the point of diminishing returns. Having shorter bitline and wordline length in a DRAM device will reduce the access latency. However by doing so it will impact the array efficiency. In the mainstream market, manufacturers are not willing to trade capacity for latency. Prior works had proposed hybrid-bitline DRAM design to overcome this problem. However, those methods are either intrusive to the circuit and layout of the DRAM design, or there is no direct way to migrate data between the fast and slow levels.

In this paper, we proposed a novel asymmetric DRAM with capability to perform low cost data migration between subarrays. Having this design we determined a simple management mechanism and explored many management related policies. We showed that with this new design and our simple management technique we could achieve 7.25% and 11.77% performance improvement in single- and multi-programming workloads, respectively, over a system with traditional homogeneous DRAM. This gain is above 80% of the potential performance gain of a system based on a hypothetical DRAM which is made out of short bitlines entirely.

1. INTRODUCTION

In the evolution of DRAM technology, the demand for lower-price and larger-capacity products has long been an important driving force behind the trend. Therefore, much effort has been put on optimizing DRAM technology scaling and design in order to pack more bits onto a single chip. Within the last decade, this trend has pushed more than eight times growth in single chip DRAM capacity. The current state of the art single chip DRAM has reached 8 giga-bits capacity [1].

During the same period, we observed two very different growth rates in DRAM bandwidth and latency. DRAM bandwidth increased approximately 100 times over the last decade [2]. This incredible number is achieved by the introduction of synchronous interface, higher frequency and increased parallelism (e.g. number of memory channels and banks). On the other hand, DRAM latency didn nott improve much. There are many factors that hold back the reduction in DRAM latency. One that will be the focus of this paper is the trade-off with DRAM chip capacity.

To mitigate the growing memory latency, designers have resorted to latency tolerating micro-architectural techniques such as speculative out-of-order execution and average latency reduction methods of multiple levels of caching. However, these approaches have reached a point of diminishing return and they tend to increase memory bandwidth leading to higher power. Caching is effective in reducing the average memory access time. But it costs silicon area and standby power. Caching also creates design complexity and overhead when multiple cores need to synchronize their cache contents. Lowering the DRAM latency directly will tend to alleviate some of the pressures at the processor end.

It is well known that DRAM stores data in charge form and reading of the data is done by sensing the voltage difference created by charge sharing. As a result, a big portion of the latency is attributed to how fast the voltage can be developed after charge sharing. With cell capacitance being relative constant, the bitline capacitance determined the initial voltage difference after access transistor is activated. Both the initial voltage and the bitline capacitance contribute to how fast the sense amplifier can reach the final result. In order to get high cell density, the number of cells on a bitline has been increasing thus limiting the DRAM access latency from going lower. There are niche DRAMs such as RLDRAM [3] which has lower latency but it pays with a large area penalty. It is, thus, desirable to investigate means to reduce DRAM latency without sacrificing much density.

Previous work [4, 5] has proposed two possible hybrid-

MICRO-48, December 05–09, 2015, Waikiki, HI, USA

DOI: http://dx.doi.org/10.1145/2830772.2830827

bitline architectures to reduce the latency of commodity DRAM. The basic idea behind them is to mix both low latency low density subarrays with long latency but high density subarrays on the same DRAM chip. The goal is to maintain reasonable density while reducing average latency.

In asymmetric-subarray DRAM [4], DRAM subarrays are rigidly arranged into two levels. A smaller level adopts the low-latency design in order to speed up accesses of frequently accessed data, while not imposing much impact on the total chip capacity. The shortcoming of this method is that there is no direct migration mechanism between fast and slow levels. Its performance gain depends heavily on a static policy of data arrangement. Without comprehensive information about a workload it is unlikely to achieve its full potential.

In TL-DRAM [5], each DRAM subarray is separated into fast and slow segments by a row of switches. When switches are turned off, only the shorter segments are seen by sense amplifiers. Shorter bitlines imply lighter load resulting in shorter access latency. The challenge of this method is that transistors used as switches to segment bitlines cannot be the same as the transistors used to access cell capacitors which are much denser. Inserting these segmenting transistors in the core array may result in the violation of layer density rules. Moreover they will add large resistance to bitlines which will increase restore time and may impact the functionality of cells on the long segments.

Our main goal is to provide a way to migrate data between fast and slow levels in asymmetric-subarray DRAM. Previous work, RowClone [6], provides a mean to move rows around within a subarray only. However moving data between subarrays using the narrow global datapath is expensive. For example, the datapath width for a modern DRAM chip with 8b I/O width (x8) is 64 while each row has 8K bits. To transfer all bits in a row across a 64b width path will take 128 cycles.

In this paper, we propose a hybrid-bitline DRAM design which provides a low overhead row migration mechanism. This row migration mechanism could be used to support other usages such a partial power down. Employing our design we can perform direct row migration between fast and slow levels in asymmetric-subarray DRAM. With our lightweight row migration mechanism, we could apply dynamic data management method to make the asymmetric-subarray DRAM adaptive to a variety of different workloads. The following are our main contributions.

1. We propose a novel hybrid-bitline DRAM design supporting lightweight row migration. The row migration mechanism exploits shared sense amplifiers between subarrays, and requires only minimum modification to current DRAM structures with insignificant area overhead. Our design features large fast level capacity, 1/8 of total DRAM capacity, and only costs 6.6% area overhead.

2. We propose a hardware based exclusive cache management solution utilizing the new hybrid bitline DRAM with low cost swapping capability. Exclusive caching avoids capacity loss (data duplication) and is more favorable in large fast level situations. Our lightweight row migration mechanism allows frequently used data to stay in the fast level longer. We also include in our solution a caching scheme for the translation information since it is large. We further show that the large fast level simplifies the caching policy, e.g., replacement and promotion policies, which makes the proposed DRAM design more suitable for practical use.

3. Our proposed design and management achieves up to 28% performance improvement for single programming workloads. This gain is over 95% of the hypothetical potential of a reduced latency DRAM.

2. BACKGROUND

In this section, we describe the modern DRAM architecture to gain better understanding of design constraints of a DRAM chip. It will also explain some of the timing parameters for row migration describe in later sections.

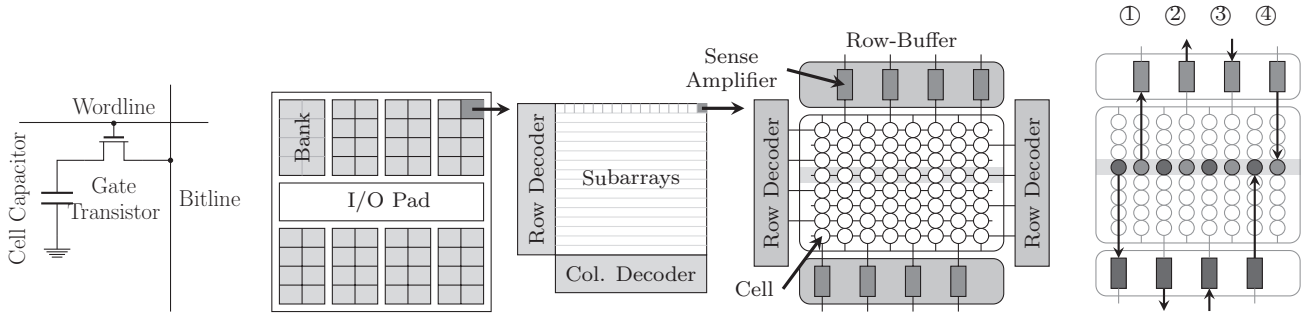
2.1 DRAM Device Organization

Figure 1a shows the circuit diagram of a DRAM cell. Each DRAM cell consists of a capacitor and a transistor. The capacitor is used to store the charge difference and the transistor acts as the switch between the capacitor and the bitline. The transistor is controlled by the row logic through a wordline. When the voltage of a wordline raises, it turns on the transistor causing the voltage difference signal stored in the capacitor to release into the bitline. On each bitline, there is a circuit called a sense amplifier that amplifies this small voltage difference that is diluted by the capacitance of a bitline, and drives it back to the correct voltage level. Each sense amplifier is associated with a column address, and could be accessed by the column logic of the bank.

Figure 1b shows the structure of a modern DRAM device (a DRAM chip). For each DRAM device, there is a group of centralized I/O pads that receive or transmit command/data signals through the external memory bus. Then, the commands and data are decoded and forwarded to the corresponding bank unit according to the address signal. Each bank consists of a two dimensional array of DRAM cells, and peripheral logic that controls the cell-array. The peripheral logic could be separated into row and column circuits. The function of row logic is to select a row in the two dimensional cell-array, and then column logic could further choose cells in the selected row to access.

2.2 Subarray Structure

Instead of building a DRAM bank with a single 2-dimensional array, it is constructed with many subarrays due to wire length's limit on performance. Figure 1c shows the structure of a subarray in a modern



(a) DRAM Cell (b) DRAM Organization (c) Subarray Structure (d) DRAM Operations

Figure 1: Subarray Structure and DRAM Operations

DRAM device which has cells, sense amplifiers (which are called row buffers) and decoders. An important fact is that all modern DRAM has an open bitline circuit architecture [7]as shown in Figure 1c. Open bitline architecture helps a single DRAM cell area to go from $8F^2$ to $6F^2$ [8]. This density increase is important but it also creates challenge for row migration. When a row is selected, two row buffers (top and bottom) are in action. This is an important fact to understand as we will show how migration can be achieved in a modern open bitline DRAM architecture.

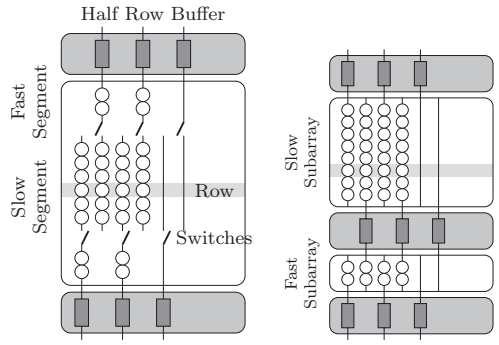
2.3 DRAM Operations

Each complete access of DRAM can be broken down into 3 stages: ACTIVATE, RD/WR, and PRECHARGE. The transitioning between different stages is controlled by DRAM commands, and there is a specific order to carry out correct data operation. Figure 1d shows the time order and subarray operations in these 3 stages (ACTIVATE: ①, RD/WR: ②/③, PRECHARGE: ④). The reason for this design is because DRAM accesses are destructive. All data contents being read out are actually destroyed in the cells, and need to be fully restored before the next access. Due to this, a complete access cycle is lengthy, DRAM leverages memory access locality by reading out a whole row of cells into a row buffer to amortize access latency.

First, in the ACTIVATE stage, a bank opens up a row of access transistors allowing the charge stored in cells to flow into bitlines. After a short time, voltages will reach equilibrium among bitlines (voltage difference is fairly diluted). Then, sense amplifiers sense the small voltage change on bitlines. Finally sense amplifiers restore the voltage value back to cells indicate bit 1 or bit 0. These sense amplifiers are also called row buffers because column logic could only access data in it, not directly to the bit cells. After data in row buffers become available, READ/WRITE commands could start to access it through the I/O circuit between the bank and external memory channel. At the final stage, after the data signals are fully restored to the cell, a PRECHARGE command closes the transistors of the row of cells. Then it neutralizes the voltage signal on the bitline and prepares it for the next ACTIVATE command.

3. HYBRID-BITLINE DRAM

Traditional DRAM chips are homogeneous in that all subarrays are of the same type. Hybrid-bitline DRAM chips have non-uniform subarrays. For example, we can have two types of subarrays. One type has short access latency and the other aims to maintain density. By having heterogeneity, we can amortize silicon area overhead of the fast subarrays with the rest of the subarrays. It also gives us an opportunity to achieve better overall latency if we can manage the accesses and hit on fast subarrays most of the time. In this section, we first summarize two types of hybrid-bitline DRAM architectures proposed previously [4, 5] and discuss the challenges in those designs in detail. We then propose a more practical and truly dynamic hybrid-bitline DRAM architecture design. We also describe and suggest management policies based on the new design. Let us examine the timing path of a DRAM access. There are three major parts - (1) I/O transfer; (2) peripheral logic; and (3) cell-array operations. I/O transfer is the time needed to transfer data between row buffers and the memory controller. It is primarily determined by DRAM internal datapath and the parameters related to the external memory bus. This paper will focus only on the delay internal to the DRAM device (chip). If the chip area of a DRAM device is not changed drastically then the delay for internal datapath should stay roughly the same. The delay due to peripheral logic is referring to the time needed for a DRAM device to decode commands into array access signals. Again we assume there are no changes to the command and interface therefore this part of the delay should be unchanged. Lastly, the delay of cell-array operations includes timing paths that are related to charge sharing (tRCD), sense amp operation and restoring (tRAS), and precharging the bitline (tRP). It has been identified by prior works that tRAS is the main contributor of overall DRAM device access latency. Reducing the length of bitline is an effective way to reduce the cell-array operation delay. There are both commercial products and prior research based on this approach such as FCRAM [9], RLDRAM [3], Hybrid Memory Cube (HMC) [10], CHARM [4] and TL-DRAM [5]. For each bitline, there is a parasitic ca-



(a) TL-DRAM [5] (b) CHARM [4]

Figure 2: Hybrid-Bitline DRAMs

pacitance associated with its length. This capacitance determines the charge sharing resulting voltage. This initial voltage delta contributes to sensing delay by the sense amplifier. Bitline resistance also contributes to the data restoring time necessary for DRAM since it is destructive-read. All of these factor into DRAM timing parameters - tRCD, tRAS and tRP. As mentioned shortening bitline will impact the area overhead. Existing commercial products all utilize subarrays with uniform bitline lengths. We discuss trade-offs of two research proposes utilizing non-uniform bitline lengths.

3.1 TL-DRAM

Figure 2a shows the internal array structure of TL-DRAM [5] with segmented bitline design. In this design, an isolation transistor is used to divide a single bitline into two segments. The segment connected directly to the sense amplifier is called the near segment, and the other segment is called the far segment. The role of the isolation transistor is to decouple the far segment from the sense amplifier to reduce the access latency of the near segment. When accessing the cells located in near segments, the isolation transistor is turned off, and it shortens the length of a bitline seen by the sense amplifier. Therefore, the time and energy needed to drive the bitline is reduced. On the other hand, when accessing the cells located in the far segment, the isolation transistor is turned on. The time and energy needed to drive this longer bitline is more. One additional delay that may need to be accounted for is the time needed to restore cell data after sensing. Since DRAM is destructive read cell data needs to be written back from sensing results. Adding a transistor in series with the bitline will increase the RC constant and prolong the data restore time.

The silicon area overhead of TL-DRAM [5] can be accounted for in two parts. The first part is caused by the isolation transistors and it is about 11.5X of a row. This is because the isolation transistor will be different from the cell access transistor. The second part is caused by the lowering of cell density of the near segment. Since all modern DRAM chips employ the open bitline architecture, a sense amplifier has the bit line and the bit-bar line in opposite sides. In the same fashion, the near segments will also need to reside on both ends of a subarray (Figure 2(a)). As a result, half of the near segment region will be empty or not used. Therefore, the cell density of the fast-segment is only one half of a normal cell array. With 128 rows in the fast section, the total area overhead would be around 24% assuming the sense amp height is 108 rows. Clearly this kind of overhead is too high.

The main challenge with TL-DRAM [5] is the intrusive nature of the isolation transistors inside the cell array. In modern DRAM, cell access device is specially designed to reduce leakage and provide cell density [11]. It cannot be used to isolate a bitline. Isolation must use transistors for sense amp and logic gates. Introducing these transistor in the middle of the cell array will tend to impact layer density rules and can degrade yield. Moreover adding a transistor in series with a bitline will increase the bitline RC constant. Any variation or weakness of the isolation transistor will impact the cell data restore time prolonging tRAS greatly. Given that DRAM cell array are highly optimized for manufacturing yield, this approach will have a high barrier for actual adoption by commercial DRAM vendors.

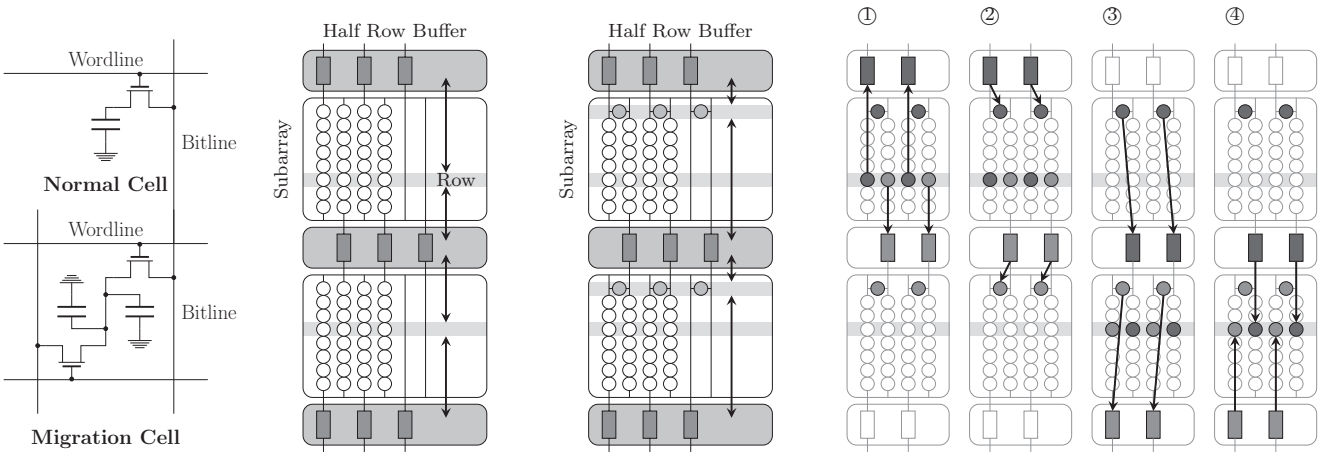
3.2 Asymmetric-subarray DRAM

Figure 2b shows the DRAM internal structure of the asymmetric-subarray design. Basically, there are two types of subarrays. Fast subarrays have short bitlines to save latency and energy. This single hybrid-bitline design was proposed by CHARM [4]. The area overhead of the asymmetric-subarray design comes from the higher cell-to-sense amplifier ratio due to fast subarrays. There will also be more peripheral circuits such as decoders and column muxes. Besides shorter bitline length, asymmetric-subarray design has the same cell-array as traditional DRAM. There is much less yield impact and is a more practical implementation.

The main shortcoming of the asymmetric-subarray design is that there is no low cost data migration path between fast and slow levels. In the current asymmetric-subarray design, data must go through the narrow global data bus after column select. As a result any dynamic data management polices will tend to have high latency. Without a low cost direct row data migration mechanism, the unit of each migration will tend to be smaller. This may create a substantial amount of address translation tags. In this paper, we propose a simple circuit design that achieves direct row migration in an asymmetric-subarray DRAM. Based on this architecture, we also introduce a simple management policy.

4. PROPOSED DESIGN

As mentioned previously, the main challenge of asymmetric subarray DRAM is how to maximize the usage of fast subarrays. Static assignment based on profiling is the easiest way and it has shown good performance upside. However, it is not completely practical to know program access patterns in advance. In order to keep data in the right place, lightweight data migration between fast and slow subarrays becomes essential.



(a) Circuit Diagrams (b) Normal Subarrays (c) With Migration Row (d) Row Migration Procedure

Figure 3: Migration Circuit Design

We will introduce our lightweight row migration mechanism in this section. We call this Dynamic Asymmetric-Subarray DRAM or DAS-DRAM.

4.1 Migration Circuit Design

Row migration exchanges data between fast and slow levels. We will first introduce a design which provides a mechanism for row migration between two neighboring subarrays. This mechanism is based on the fact that the row buffer is a shared resource between two neighboring subarrays. As shown in Figure 3b, a buffer of a row in a conventional open bitline DRAM is actually made out of two "half row buffers", each located on one opposite edge of the cell array. These two row buffers are shared by two neighboring subarrays providing an existing channel for moving half rows between neighboring subarrays. Through this channel, a "half row" could be transferred across the shared "half row buffers". However, this existing feature only provides migration route between neighboring subarrays sharing the same "half row buffer". There is no path to move to another subarray beyond it. We propose a novel design to mitigate this issue. A new "migration cell" is added to each bitline. This cell has two access ports sharing a single storage capacitor as depicted in Figure 3a. This cell allows two unassociated bitlines to access the shared storage cell. It acts as the temporary storage for row data transfer (see Figure 3c which we will describe in the next section. This is the key idea which this paper is based upon. It is worth to note that a simple transistor connecting the two adjacent bitlines would not work due to reasons described above about TL-DRAM. Figure 4 shows the abstract layout of part of a DRAM subarray with normal cells and a row of migration cells [12, 13]. Wordlines and bitlines are shown in horizontal and vertical direction, respectively. Below bitlines and wordlines, there are tilted islands connected to the bitlines through contacts in the middle, shown as crossed squares. Each of the sides of an island has

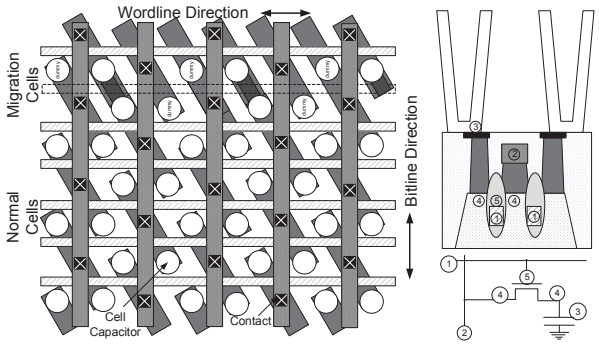


Figure 4: Layout of Subarray and Chip Cross-Section with Corresponding Circuit

a storage capacitor shown as circles. Access transistors are formed where wordlines cross islands and can be seen in the figure to the right of the layout where gate is buried. Migration cells differ from normal cells only slightly and follows the dense layout rules of a modern DRAM cell-array. More importantly circuit properties of a migration cell should be similar to a normal cell if not faster due to a larger cell capacitance.

4.2 Row Migration Procedure

Figure 3d shows the procedure of a full row migration from a row in the upper subarray to a row in the lower subarray. Logically it can be thought of as copying the source row to the temporary migration row and then transferring from there to the destination row. Since a row is made out off two "half rows" there are two identical series of operations done in parallel. We will only trace the path of the half row residing in odd columns. First in ①, a row is opened, and half of the bits (odd) are sensed by one side of the half row buffer. Then in ②, rather than restoring data back only to its origin row, the migration row will also be activated to hold the

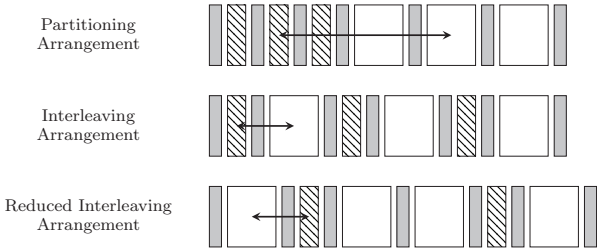


Figure 5: Arrangements of subarrays

data temporally. After the row buffer is precharged, in ③, the migration row is opened to the other part of row buffer, the other part of the row buffer can start to sense signal on its bitlines. Finally in ④, the data is restored to the specified destination row. A half row migration is completed at this point.

The latency of a row migration could also be estimated from the above described procedure. First, if we look at the steps in Figure 3d and assign a DRAM operation to each one, then the first and the third step will be row activations and the second and the fourth step will be row buffer . A simple estimation of the total latency will be 2 tRC. We know that most of tRC is contributed by tRAS (around 70%), which is used to ensure that voltage stored in cells will be high enough for correct operation given a retention time (i.e. 64ms). However, in the case of row migration, the signal stored in the migration row will be read out right away. Therefore, we could tighten up the tRAS timing parameter. For our experimental study described in later sections, we set a row migration (not fully a row promotion yet) time to be 1.5 tRC.

4.3 Area Overhead and Migration Path

The silicon area overhead is proportional to the bitline length of the fast subarray and the ratio of fast and slow subarrays. From [4], the improvement of speed drops significantly after bitline length is reduced below 128 cells (512 cells for common subarrays). Therefore, we adopt this bitline length for our fast subarrays.

Another factor affecting the area overhead is the ratio of fast and slow subarrays. We could not choose this number freely since it is associated with the length of migration path. Figure 5 shows three different arrangements of subarrays. For partitioning, there is no limitation to the ratio of subarrays, but the average row migration path will be lengthy. Interleaving solves the problem but it locks the ratio to 1:1. In this paper, we adopted the reduced interleaving and set the ratio to 1:2 for fast to slow subarrays. Under the assumption that row buffer is 1/6 of a subarray [4], it will only produce 6.6% area overhead.

5. DATA MANAGEMENT

There are two different approaches to manage the asymmetric-subarray DRAM. First approach treats the fast subarrays as a hardware-managed inclusive cache. The most appealing feature of this approach is its simplicity. However we lose in total capacity. The second approach incorporates both fast and slow subarrays to form an uniform memory space. We can think of this approach as an exclusive caching scheme. Even though this approach shares many features in common with virtual memory managed by the operating system of an existing system, however the time to swap a page (row) needs to be much shorter. Thus we would need to adopt a hardware-based page management design. In summary there are a few criteria for comparing

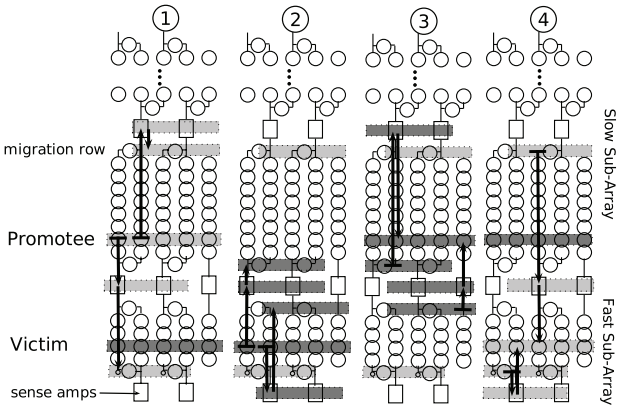


Figure 6: Procedure of a row promotion

these two approaches: 1) total capacity; 2) complexity and overhead related to address translation; and 3) cache replacement time. In terms of total capacity, the exclusive-cache approach is more cost effective. The inclusive-cache approach must duplicate data in both slow and fast subarrays. As we will discuss later, the capacity ratio of fast-to-slow is expected to be 1/8. This means at least 1/8 in capacity is lost for an inclusive-cache. For address-translation complexity and overhead, inclusive-cache approach needs a smaller translation table (cache tags) because only the addresses in the fast level are dynamic while both the fast and slow levels' addresses are dynamic for exclusive-cache. Looking up an address table will increase access latency. We will discuss later on how to leverage tag-caching to bring down the latency overhead. For cache replacement, the inclusive-cache approach should be faster in response time. In the case of a clean victim row, we only need to bring one row from the slow level to the fast level. While in the exclusive-cache approach, there will always be a row-swapping between the fast and slow levels. In this paper, we adopt the exclusive-cache approach mainly because of the total capacity concern.

5.1 Row Swapping

For the exclusive-cache approach, a row promotion will involve swapping of two rows. The most naive way to do this is to adopt the common software swapping algorithm. This involves three row migrations and needs a temporary buffer. It is worth to note that for inclusive-cache, even though a row promotion with no write back (clean line) is not a swap. But if the victim row is dirty, it will also need a swap.

Our proposed DRAM design allows us to complete row migration using the two migration rows in each subarray. Whenever possible two movement may happen in parallel as long as there is no conflicts. As shown in Figure 6, we can complete the swap in four steps. First, we move the promotee to two half rows in ①. We then move the victim row also to two migration rows (each is half row) as in ②. In ③ we have two movements in parallel since there is no conflicts in row buffers as well as migration rows. After step ③ the victim row has moved to where the promotee row was completely. Step ④ also has two movements in parallel and completes the movement of the promotee row to the row where the victim was.

5.2 Translation Table

Treating fast subarrays as an exclusive-cache for the whole array means the translation table (cache tags) needs to cover both the fast level and the slow level and will be large. Therefore, the translation table would need to be stored in memory. Each memory access then becomes two accesses. We adopt a translation cache (tag cache) in the memory controller to cache only the most recently used part of the translation table to reduce the time we need to do memory look-up. We propose to utilize the last level cache to cache the translation table as well. When a memory request misses the last-level cache (LLC), it is sent to the memory controller. In the memory controller, the address of a memory request needs first to be translated. The memory controller looks up the translation cache for the corresponding entry. If the entry is not there, then the memory controller checks the last-level cache. This allows us to utilize part of the last-level cache capacity for the translation table. If the entry is still not found, then the look-up process continue to read the memory for finding the corresponding address mapping and then data.

There is a way to hide the translation cache look-up latency. To hide the translation cache look-up latency, we could do the translation table look-up concurrently with LLC accesses. Since the translation cache capacity is much smaller than LLC, its latency can be completely hidden. Therefore, if the corresponding address translation of a memory request can be found in the address translation cache, our design does not cause extra delay.

In order to put the limited capacity translation cache to good use, we adopt a way to increase the number of entries while maintaining the total capacity. This is achieved through limiting the migration freedom resulting in the reduction of the size of each entry. In contrast to a page table where each virtual page can be mapped to any physical page, we restrict the mapping to only a sub-group of the total memory. If we make the size of each migration group no larger than 256, then each entry will only cost 1 byte instead of the multiple bytes needed to cover the entire memory space. However less migration freedom also means more opportunity for replacement conflicts. In the result section, we evaluate the trade-offs of migration group sizes as well.

5.3 Management Mechanism

The proposed management mechanism is a hardware approach. We show, in the result section, that the number of row migrations could be more than 3% of the total memory accesses. At this ratio using software interrupt to handle migration will have too much overhead. Another advantage of our approach is that the management of asymmetric-subarray DRAM is completely transparent to operating system and application software. We separate the management mechanism for our proposed asymmetric-subarray DRAM into three parts: 1) address translation for memory requests; 2) trigger and filtering mechanisms for row promotion; and 3) the choice of replacement policies.

Address translation There are two main concerns for address translation: 1) the response time; and 2) the silicon area overhead. To achieve fast response time, we try to maximize the hit ratio of the translation cache. To limit the translation cache capacity, we only cache the translation entries for the fast level. As mentioned in Section 4.3., we group memory sections into migrations groups to reduce the size of each entry of the translation cache. We present the trade-offs between the size of the migration group and the translation cache capacity in the results section.

Filtering policy for row promotion We propose two possible policies. The first policy focuses on achieving the highest utilization of the fast level without any consideration on the row promotion rate. Every hit on the slow level triggers a row promotion in this policy. The policy is designed assuming memory accesses exhibit temporal locality. The implementation for this policy is simple. The second policy adds a threshold to filter out the promotion for the rows that are not accessed frequently. The implementation of this policy is slightly more complex than the first one. It needs a set of hardware counters to count the number accesses in a row before it is promoted. When the memory access pattern is highly concentrated on a small region of its total footprint, this policy shows benefits.

Replacement policy We thought replacement policies would impact performance when there are contentions at the fast level initially. We discovered, contrary to our belief, they have very little effect. There are two reasons. First, the latency of each row promotion is only a few tRC cycles. Second, the capacity of of fast level is large (one eighth of the total memory). We evaluated a few replacement policies include LRU algorithm, sequential, random, and a pseudo-random replacement policy using a global increasing counter and reported these results in the later section.

6. EXPERIMENTAL SETUP

We modeled the new asymmetric subarray DRAM as part of a detailed cycle accurate full system simulation environment-Marss86 [14]. The system simulated includes multiple detailed Out-Of-Order CPUs and a memory subsystem as shown in Table 1. We selected a subset of memory bound benchmarks from the SPEC CPU2006 suite based on information obtained from [15].

Table 2 lists single-programming workloads and multi-programming workloads tested. We also evaluated our designs against our proposals in a multi-programming environment. This is done by combining several SPEC CPU2006 workloads with divergent memory access patterns to exercise the memory subsystem.

Processor	3GHz, 4-wide issue, 192-entry ROB
Cache	64KB 8-way private L1 (4 cycles), 256KB 8-way private L2 (12 cycles), 4MB 8-way shared LLC (20 cycles)
Memory Contrller	32-entry request queue, open-page policy, FR-FCFS
DRAM	Two 4GB DDR3-1600 DIMMs, 2 channels, 2 ranks for each channel, tRCD: 13.75ns, tRC: 48.75ns Detailed timing parameters [16]
Asym. DRAM	Fast-level Capacity Ratio: 1/8, Migration group size: 32 rows, Migration latency: 146.25ns, tRCD (fast/slow): 8.75/13.75ns, tRC (fast/slow): 25/48.75ns [4]

Table 1: System Configuration

Single-programming Workloads	
astar/BigLakes2048, cactusADM/benchADM, GemsFDTD/ref,lbm/lbm,leslie3d/leslie3d, libquantum/ref, mcf/ref, milc/su3imp, omnetpp/omnetpp, soplex/pds-50	
Set	**Multi-programming Workloads**
M1	cactusADM, mcf, milc, omnetpp
M2	cactusADM, GemsFDTD, lbm, mcf
M3	cactusADM, lbm, leslie3d, omnetpp
M4	astar, cactusADM, lbm, milc
M5	astar, libquantum, omnetpp, soplex
M6	GemsFDTD, leslie3d, libquantum, soplex
M7	leslie3d, libquantum, milc, soplex
M8	lbm, libquantum, mcf, soplex

Table 2: Target Workloads

In order to achieve more deterministic and reliable results we employed a couple evaluation methods. First we bound each program to a dedicated core to reduce system level non-determinism. Second we sampled four points of each workload to get a more balanced view of each program. However, for multi-programming workloads, we only sample one point (at the 4-minute mark after launching the program) since each benchmark will appear in multiple samples at difference phases. For each experiment, we started from the same checkpoint of every sampling points to ensure coherent simulation results. Our simulation length is 100M instructions for single-programming workloads and 400M instructions for multi-programming workloads. The first 20% of the simulation was used for warm up and only the rest are used in statistics calculation.

7. RESULTS AND DISCUSSION

We evaluate several DRAM designs using both single and multi-programming workloads to quantify the performance improvement over the system with standard homogeneous DRAM chip. These designs are listed as:

1. *SAS-DRAM*: Static Asymmetric-Subarray DRAM with no row migration. Each workload is profiled first and the most-frequently-used portion of its footprint is pre-assigned to the fast level.
2. *CHARM [4]*: This is SAS-DRAM with optimized column access latency for the fast level. Again all workloads are profiled first and we pre-assign most frequently used addresses to the fast level. In practice this is not possible.
3. *DAS-DRAM*: Stands for Dynamic Asymmetric Subarray DRAM. It is the hybrid bitline DRAM with lightweight row migration and dynamic management policy as proposed by this paper.
4. *DAS-DRAM (FM)*: Ideal DAS-DRAM with zero row migration latency which is used to help us understand the overhead of DAS-DRAM.
5. *FS-DRAM*: Fast-subarray DRAM. Ideal low latency DRAM made out of fast subarrays entirely.

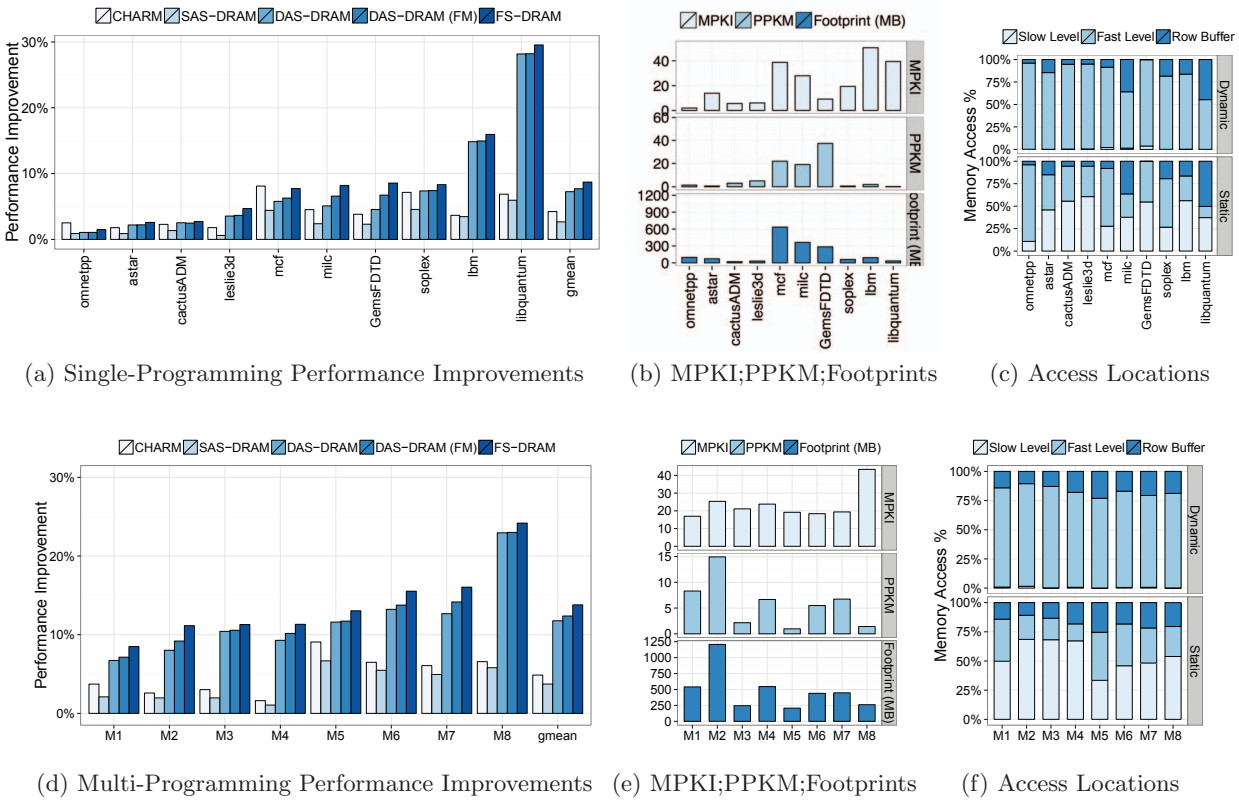
7.1 Single-Programming Evaluation

Figure 7 shows the simulation results using single-programming workloads. This figure includes (7a) the performance improvement over standard DRAM, (7b) MPKI (misses per kilo-instructions), PPKM (promotion per kilo-misses), and memory footprint during the captured episodes, (7c) the distribution of memory access location. A statically managed DRAM cache, like CHARM, capture the hottest data in a program lifetime, while a dynamic approach captures the hottest data in program phases. Even though the number of migrations is small with respect to total accesses (PPMK) but the effect of migration is large because data in promoted rows are accessed frequently. Figure 7(c) shows the percentage of accesses landing on the fast subarrays increases significcantly with dynamic migration.

First, Figure 7a shows that our proposed design (DAS-DRAM) comes very close to the upper bound in most cases (FS-DRAM). On average, DAS-DRAM achieves 7.25% performance improvement, while FS-DRAM is at 8.71% when compared with standard DRAM.

Second, DAS-DRAM improves performance significantly over static methods, including CHARM. Figure 7c shows by performing dynamic migration the hit ratio to the fast level increase significantly over static methods. Therefore, DAS-DRAM fully utilizes the fast level. On average, SAS-DRAM and CHARM achieve 2.66% and 4.23% respectively over regular DRAM only.

Third, the performance loss due to row migration overhead is small, even less significant than the overhead caused by address translation. Figure 7a shows the row migration overhead is 0.45%, which is the gap between DAS-DRAM (FM) and DAS-DRAM, and the



(a) Single-Programming Performance Improvements (b) MPKI;PPKM;Footprints (c) Access Locations

(d) Multi-Programming Performance Improvements (e) MPKI;PPKM;Footprints (f) Access Locations

Figure 7: Single-Programming and Multi-Programming Workloads

address translation overhead is 0.99%, the gap between FS-DRAM and DAS-DRAM (FM). We believe it is due to the low migration time of our proposed novel row migration design (1.5 tRC).

Lastly, the improvement of some benchmarks (GemsFDTD and MILC) are smaller for DAS-DRAM. Figure 7b shows that the PPKM and footprint numbers for those benchmarks are higher. This means that we are spending more time on overheads. A filtering mechanism for row promotion could help to reduce the overhead caused by row migration, but it could also reduce the benefits coming from higher fast-level hit ratio. We discuss trade-offs in more detail later.

7.2 Multi-Programming Evaluation

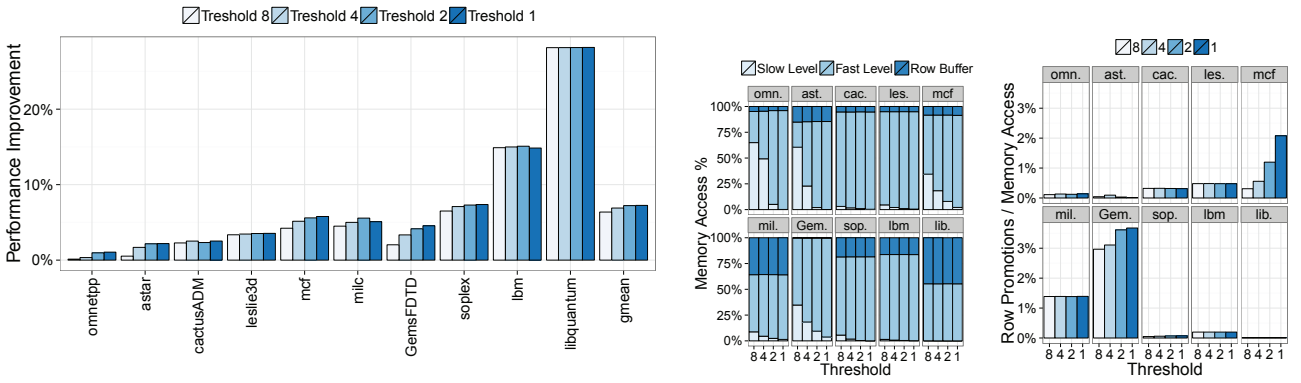
We also evaluate the same five DRAM designs for a multi-core system with multi-programming workloads. Figure 7d shows the performance improvements over a system with standard DRAM. On average, DAS-DRAM achieve 11.77% performance improvement, while the upper bound (FS-DRAM) is at 13.79%. SAS-DRAM and CHARM achieve only 3.72% and 4.87%, respectively. In general the performance improvement is better than single-programming system. Examining the memory behavior of the workloads from Figure 7e, we observe that multiple workloads have higher MPKI. As a result improvement to average memory latency will have higher impact on overall performance.

Furthermore, dynamic methods (e.g. DAS-DRAM) improve performance more than static methods in multi-programming situations. Although cache interference misses do not create new footprint, it degrades the locality of memory accesses. Figure 7f shows that more memory accesses fall in the slow region for static profiling methods in comparison with dynamic methods. This is one of the reason dynamic migration have better performance than static profiling methods.

Lastly, Figure 7e shows that PPKM numbers of multi-programming workloads tend to be lower. This makes sense because the misses created by cache interferences are more likely to fall inside the rows accessed previously which is in the fast section already. As the row promotion overhead being amortized by more memory accesses, it explains why we do not see large performance gaps between DAS-DRAM and DAS-DRAM (FM).

7.3 Filtering Policy for Row Promotion

Figure 8 shows the simulation results of filtering policies with different row-promotion thresholds. This figure has three groups of information - (8a) performance improvement, (8b) miss ratio of the fast level and (8c) ratio of row promotion to memory access. Filtering mechanism requires additional hardware to gather information of memory accesses. In this experiment, a set of 1024 counters is arranged to obtain the hit numbers of the most recently used rows.



(a) Performance improvement (b) Access Locations (c) Promotion/Access Ratio

Figure 8: Filtering Policies

Employment of a filtering mechanism can have both positive and negative effects on performance. It reduces row promotion frequency, thus lowers the migration overhead. But, it also limits row promotion and degrades the utilization of fast level. Figure 8c shows that filtering mechanism is not very effective at reducing row promotion frequency. The row promotion to memory access ratio is already small in most cases (less than 1%). As a result, we rarely see any performance improvement in Figure 8a. On the other hand, Figure 8b shows that filtering mechanism could decrease the utilization of fast level arrays significantly. Therefore, we see a general trend in performance reduction as the filtering threshold number increases in Figure 8a. At the end, we implemented no filtering mechanism (or a threshold 1 of DAS-DRAM).

7.4 Translation Cache Capacity Sensitivity

Figure 9a shows the performance improvement of different translation cache capacities. Our design only caches the translation to the fast level. The reason for this approach is to maximize the hit ratio in the translation cache. As shown previously, the hit ratio of the fast level is generally over 90%. Therefore, it is reasonable to not cache the translation entries for the slow level. Results show that an 128KB on-chip translation cache could achieve good performance. Since translation cache look-up could be in parallel with last-level cache look-up, there is no latency overhead for accessing the translation cache.

7.5 Migration Group Size Sensitivity

Figure 9b shows the performance improvement of different migration group sizes. The smaller the size of a migration group is, the less bits are needed for mapping information. On the other hand, smaller size of a migration group could also increase the chance of contention. Results show that this effect is subtle to the overall performance.

7.6 Fast Level Capacity and Replacement

Figure 9c shows the performance improvement of different capacity ratios of the fast level to the total capacity. The silicon area overheads are 6.6% and 11.3% for ratios of 1/8 and 1/4 respectively. Reducing the ratio further will result in some performance impact to a couple benchmarks (mcf and MILC) due to their large memory footprints. Our DAS-DRAM adopts a fast level capacity ratio of 1/8 to maximize perfromance gain and to reduce area overhead. As for replacement policy impact, figure 9d shows overall performance for two replacement policies - LRU and random. The difference between them is negligible. The small difference is due to large fast level ratio.

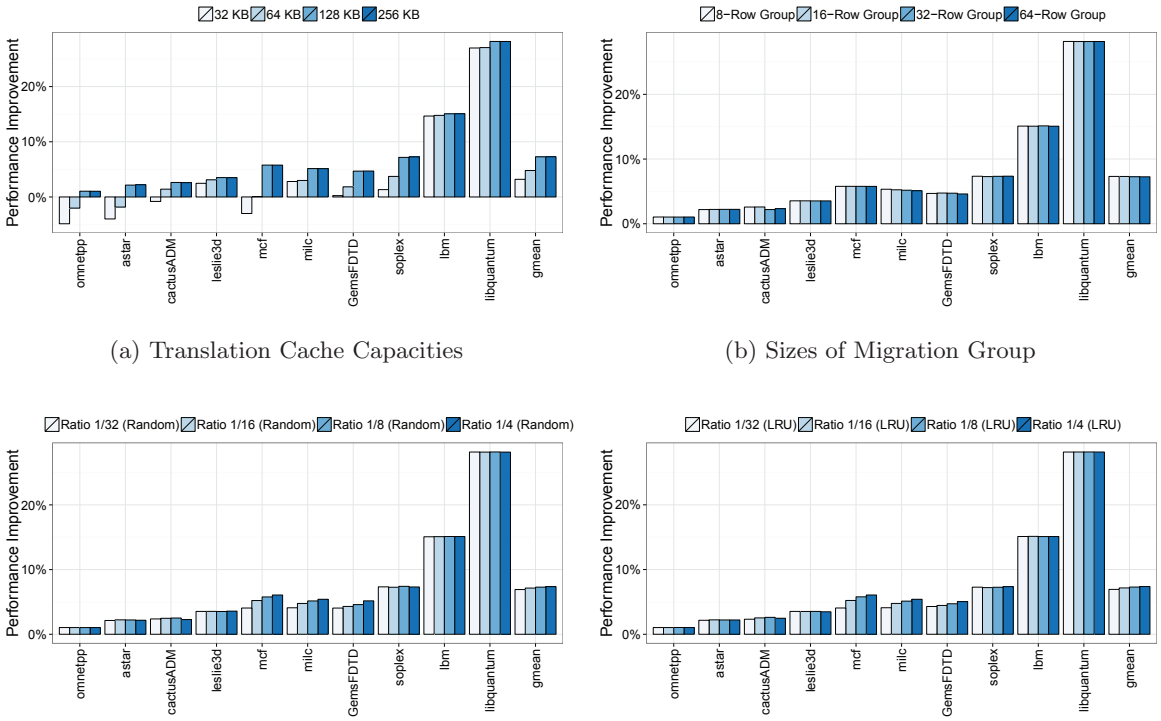
7.7 Power Implications

There are two factors contribute to power consumption difference. One is the percentage of accesses to the fast level. The other is the migration power overhead. From figure 7c and figure 7f we observe that there is very little time our proposed design will access the slower sections. Moreover, from figure 8c we see that the percentage of migration to total access is low. This implies that this design will have lower power consumption than the static asymmetric DRAM design.

8. RELATED WORKS

Low Latency DRAM Reducing bitline length is an effective way to reduce DRAM latency. However, a homogeneous design comes with high silicon area overhead. Nevertheless, in some high-performance applications, this area overhead may be tolerable. FCRAM from Fujitsu [9], RLDRAM from Micron [3], and Low Latency DRAM from Renesas [17] are all examples of homogeneous design. Although their tRC is reduced by 40% to 70%, their capacity is only 15% to 30% of the comparable mainstream products [18, 3]. As shown in the results section, our hybrid-bitline design could achieve 80% of the performance on average with a modest 6.6% of capacity loss.

High Parallelism DRAM Parallelism opens up another dimension of DRAM research. Large body of works have been done on this subject. For subarray



(a) Translation Cache Capacities

(b) Sizes of Migration Group

(c) Ratios of Fast Level with Random Replacement

(d) Ratios of Fast Level with LRU Replacement

Figure 9: Fast Level Capacities and Replacement Policies

level parallelism, Udipi et al. proposed Single Subarray Access (SSA) to allow the matrices in a row buffer to be independent of each other [19]. Kim et al. proposed Subarray Level Parallelism (SALP) to let multiple local row buffers of a bank to be opened at the same time [20]. For inter-chip level parallelism, many works have proposed to divide up a rank to increase rank level parallelism at the cost of data burst latency [21, 22, 23, 24]. Those methods are generally compatible with low latency designs, and also could be conjugated with the hybrid designs.

Hybrid Memory System With advancement in packaging technologies, hybrid memory system becomes a popular area of research. Black et al. and Loh explored the design space of 3D-stacked DRAM [25, 26]. Dhiman et al. proposed a hybrid PRAM and DRAM memory system to reduce standby power of all DRAM memory system [27].

Since hybrid memory systems are built with heterogeneous memory chips, data transfer between chips needs to go through the narrow memory bus. Frequent data transfers will increase memory traffic significantly. Therefore, sophisticated management mechanisms are needed for this approach. Zhao et al. proposed a tag management method for DRAM cache [28]. Jiang et al. proposed CHOP, a filtering mechanism, to reduce the data migration traffic [29]. In contrast, data transfer in hybrid-bitline DRAMs would not increase external memory bandwidth. Moreover, the migration latency is much smaller. It allows simple management policy to be used and still provides good performance.

Hybrid-Bitline DRAM Hybrid bitline DRAM appears to be yet another hybrid memory system. But, its ability to migrate data on-chip distinguishes it from the other hybrid memory systems. Lee et al. proposed TL-DRAM, a segmented bitline design, but with some manufacturing obstacles [5]. Son et al. proposed CHARM, a asymmetric bitline design, but without migration mechanism [4]. In this paper, we proposed a novel hybrid-bitline design, which is more friendly to manufacturing. Moreover, we studied the management policies based on the characteristic of the proposed design.

9. CONCLUSION

During the last decade, DRAM evolution has focused on capacity and bandwidth. Access latency has not improve much over this period of time. Recent studies proposed DRAM with asymmetric bitline length to overcome this latency stagnation. However both of those designs have some limitations. In this paper, we proposed a novel asymmetric bitline DRAM design which supports light-weight dynamic row migration. Using this design we propose a simple management mechanism and studied several trade-offs involved with migration. This newly proposed dynamic asymmetric DRAM achieves similar performance of a homogeneous fast-subarray DRAM without the silicon area overhead.

10. ACKNOWLEDGEMENTS

This work is supported in part by research grants from the Ministry of Science and Technology of Taiwan (MOST-103-2220-E-002-005 and MOST-104-2220-E-002-005) and the Excellent Research Projects of National Taiwan University (104R890822).

11. REFERENCES

[1] T.-Y. Oh, H. Chung, Y.-C. Cho, J.-W. Ryu, K. Lee, C. Lee, J.-I. Lee, H.-J. Kim, M. S. Jang, G.-H. Han, K. Kim, D. Moon, S. Bae, J.-Y. Park, K.-S. Ha, J. Lee, S.-Y. Doo, J.-B. Shin, C.-H. Shin, K. Oh, D. Hwang, T. Jang, C. Park, K. Park, J.-B. Lee, and J. S. Choi, "A 3.2Gb/s/pin 8Gb 1.0V LPDDR4 SDRAM with integrated ECC engine for sub-1V DRAM core operation," in *Proceedings of Int. Solid-State Circuits Conf.*, 2014.

[2] R. D. Williams, T. Sze, D. Huang, S. Pannala, and C. Fang, "Server memory road map," 2012. `http://www.jedec.org/sites/default/files/Ricki_Dee_Williams-Final_0.pdf`.

[3] Micron, "RLDRAM 2 SIO," 2004. `http://www.micron.com/-/media/documents/products/data%20sheet/dram/576mb_rldram_2_sio.pdf`.

[4] Y. H. Son, O. Seongil, Y. Ro, J. W. Lee, and J. H. Ahn, "Reducing memory access latency with asymmetric dram bank organizations," in *Proceedings of the 40th International Symposium on Computer Architecture*, 2013.

[5] D. Lee, Y. Kim, V. Seshadri, J. Liu, L. Subramanian, and O. Mutlu, "Tiered-latency dram: A low latency and low cost dram architecture," in *Proceedings of the 19th Symp. on High Performance Computer Architecture*, 2013.

[6] V. Seshadri, Y. Kim, C. Fallin, D. Lee, R. Ausavarungnirun, G. Pekhimenko, Y. Luo, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "RowClone: Fast and Energy-efficient in-DRAM Bulk Data Copy and Initialization," in *Proceedings of the 46th International Symposium on Microarchitecture*, 2013.

[7] T. Vogelsang, "Understanding the energy consumption of dynamic random access memories," in *Proceedings of the 43rd Int. Symp. on Microarchitecture*, 2010.

[8] T. Takahashi, T. Sekiguchi, R. Takemura, S. Narui, H. Fujisawa, S. Miyatake, M. Morino, K. Arai, S. Yamada, S. Shukuri, M. Nakamura, Y. Tadaki, K. Kajigaya, K. Kimura, and B. Kiyoo Itoh, "A multigigabit dram technology with 6f2 open-bitline cell, distributed overdriven sensing, and stacked-flash fuse," *IEEE JSSCC*, Nov 2001.

[9] Y. Sato, T. Suzuki, T. Aikawa, S. Fujioka, W. Fujieda, H. Kobayashi, H. Ikeda, T. Nagasawa, A. Funyu, Y. Fuji, K. Kawasaki, M. Yamazaki, and M. Taguchi, "Fast cycle ram (fcram); a 20-ns random row access, pipe-lined operating dram," in *Proceedings of the Symp. on VLSI Circuits*, 1998.

[10] J. T. Pawlowski, "Hybrid memory cube (HMC)," in *Hotchips 2011*. `http://www.hotchips.org/wp-content/uploads/hc_archives/hc23/HC23.18.3-memory-FPGA/HC23.18.320-HybridCube-Pawlowski-Micron.pdf`.

[11] T. Schloesser, F. Jakubowski, J. v.Kluge, A. Graham, S. Slesazeck, M. Popp, P. Baars, K. Muemmler, P. Moll, K. Wilson, A. Buerke, D. Koehler, J. Radecker, E. Erben, U. Zimmermann, T. Vorrath, B. Fischer, G. Aichmayr, R. Agaiby, W. Pamler, T. Schuster, W. Bergner, and W. Mueller, "6f2 buried wordline dram cell for 40nm and beyond," in *Proceedings of IEDM*, pp. 1–4, Dec 2008.

[12] A. Kotabe, Y. Yanagawa, R. Takemura, T. Sekiguchi, and B. Kiyoo Itoh, "Asymmetric cross-coupled sense amplifier for small-sized 0.5-v gigabit-dram arrays," in *Proceedings of the Asian Solid State Circuits Conference (A-SSCC)*, 2010.

[13] T. Schloesser, F. Jakubowski, J. v.Kluge, A. Graham, S. Slesazeck, M. Popp, P. Baars, K. Muemmler, P. Moll, K. Wilson, A. Buerke, D. Koehler, J. Radecker, E. Erben, U. Zimmermann, T. Vorrath, B. Fischer, G. Aichmayr, R. Agaiby, W. Pamler, T. Schuster, W. Bergner, and W. Mueller, "6f2 buried wordline dram cell for 40nm and beyond," in *Proceedings of Int. Elec. Devices Meeting*, 2008.

[14] A. Patel, F. Afram, S. Chen, and K. Ghose, "MARSS: A Full System Simulator for Multicore x86 CPUs," in *Proceedings of the 48th DAC*, 2011.

[15] A. Jaleel, "Memory characterization of workloads using instrumentation-driven simulation," 2010. `http://www.glue.umd.edu/ajaleel/workload`.

[16] Samsung, "2Gb D-die DDR3 SDRAM," 2011. `http://www.samsung.com/global/business/semiconductor/file/2011/product/2011/8/29/729200ds_k4b2gxx46d_rev113.pdf`.

[17] Renesas, "1.1G-BIT Low Latency DRAM-III," 2013. `http://documentation.renesas.com/doc/products/memory/r10ds0012ej0200_memory.pdf`.

[18] Micron, "2gb: x4, x8, x16 ddr2 sdram," 2006. `http://www.micron.com/-/media/documents/products/data%20sheet/dram/ddr2/2gb_ddr2.pdf`.

[19] A. N. Udipi, N. Muralimanohar, N. Chatterjee, R. Balasubramonian, A. Davis, and N. P. Jouppi, "Rethinking dram design and organization for energy-constrained multi-cores," in *Proceedings of the 37th Annual Int. Symposium on Computer Architecture*, 2010.

[20] Y. Kim, V. Seshadri, D. Lee, J. Liu, and O. Mutlu, "A case for exploiting subarray-level parallelism (salp) in dram," in *Proceedings of the 39th International Symposium on Computer Architecture*, 2012.

[21] F. Ware and C. Hampel, "Improving power and data efficiency with threaded memory modules," in *Proceeding of the Int. Conf. on Computer Design (ICCD)*, 2006.

[22] H. Zheng, J. Lin, Z. Zhang, E. Gorbatov, H. David, and Z. Zhu, "Mini-rank: Adaptive dram architecture for improving memory power efficiency," in *Proceedings of the 41th International Symposium on Microarchitecture*, 2008.

[23] D. H. Yoon, M. K. Jeong, and M. Erez, "Adaptive granularity memory systems: A tradeoff between storage efficiency and throughput," *SIGARCH Computer Architecture News 6/2011*.

[24] M. K. Jeong, D. H. Yoon, D. Sunwoo, M. Sullivan, I. Lee, and M. Erez, "Balancing dram locality and parallelism in shared memory cmp systems," in *Proceedings on the 18th High Performance Computer Architecture*, 2012.

[25] B. Black, M. Annavaram, N. Brekelbaum, J. DeVale, L. Jiang, G. Loh, D. McCauley, P. Morrow, D. Nelson, D. Pantuso, P. Reed, J. Rupley, S. Shankar, J. Shen, and C. Webb, "Die Stacking (3D) Microarchitecture," in *Proceedings of the 39th Int. Symposium on Microarchitecture*, 2006.

[26] G. H. Loh, "3D-Stacked Memory Architectures for Multi Core Processors," *SIGARCH Computer Architecture News*, 2008.

[27] G. Dhiman, R. Ayoub, and T. Rosing, "Pdram: A hybrid pram and dram main memory system," in *Proceddings of the 46th Design Automation Conference (DAC)*, pp. 664–669, July 2009.

[28] L. Zhao, R. Iyer, R. Illikkal, and D. Newell, "Exploring dram cache architectures for cmp server platforms," in *Proceedings of the 25th ICCD*, 2007.

[29] X. Jiang, N. Madan, L. Zhao, M. Upton, R. Iyer, S. Makineni, D. Newell, D. Solihin, and R. Balasubramonian, "Chop: Adaptive filter-based dram caching for cmp server platforms," in *Proceedings of the 16th High Performance Computer Architecture*, 2010.