Simultaneous Multi-Layer Access: Improving 3D-Stacked Memory Bandwidth at Low Cost

DONGHYUK LEE, SAUGATA GHOSE, GENNADY PEKHIMENKO, SAMIRA KHAN, and ONUR MUTLU, Carnegie Mellon University

3D-stacked DRAM alleviates the limited memory bandwidth bottleneck that exists in modern systems by leveraging *through silicon vias* (TSVs) to deliver higher external memory channel bandwidth. Today's systems, however, cannot fully utilize the higher bandwidth offered by TSVs, due to the limited *internal* bandwidth within each layer of the 3D-stacked DRAM. We identify that the bottleneck to enabling higher bandwidth in 3D-stacked DRAM is now the *global bitline interface*, the connection between the DRAM row buffer and the peripheral IO circuits. The global bitline interface consists of a limited and expensive set of wires and structures, called *global bitlines* and *global sense amplifiers*, whose high cost makes it difficult to simply scale up the bandwidth of the interface *within* a single DRAM layer in the 3D stack. We alleviate this bandwidth bottleneck by exploiting the observation that several global bitline interfaces already exist *across* the multiple DRAM layers in current 3D-stacked designs, but only a fraction of them are enabled at the same time.

We propose a new 3D-stacked DRAM architecture, called *Simultaneous Multi-Layer Access* (SMLA), which increases the internal DRAM bandwidth by accessing multiple DRAM layers concurrently, thus making much greater use of the bandwidth that the TSVs offer. To avoid channel contention, the DRAM layers must coordinate with each other when simultaneously transferring data. We propose two approaches to coordination, both of which deliver four times the bandwidth for a four-layer DRAM, over a baseline that accesses only one layer at a time. Our first approach, Dedicated-IO, *statically partitions* the TSVs by assigning each layer to a dedicated set of TSVs that operate at a higher frequency. Unfortunately, Dedicated-IO requires a nonuniform design for each layer (increasing manufacturing costs), and its DRAM energy consumption scales linearly with the number of layers. Our second approach, Cascaded-IO, solves both issues by instead *time multiplexing* all of the TSVs across layers. Cascaded-IO reduces DRAM energy consumption by lowering the operating frequency of higher layers. Our evaluations show that SMLA provides significant performance improvement and energy reduction across a variety of workloads (55%/18% on average for multiprogrammed workloads, respectively) over a baseline 3D-stacked DRAM, with low overhead.

Categories and Subject Descriptors: B.3.2 [**Memory Structure**]: Design Styles

General Terms: Design, Performance, Memory

Additional Key Words and Phrases: 3D-stacked DRAM

ACM Reference Format:
Donghyuk Lee, Saugata Ghose, Gennady Pekhimenko, Samira Khan, and Onur Mutlu. 2015. Simultaneous multi-layer access: Improving 3D-stacked memory bandwidth at low cost. ACM Trans. Archit. Code Optim. 12, 4, Article 63 (December 2015), 29 pages.
DOI: http://dx.doi.org/10.1145/2832911

We acknowledge the support of our industrial partners: Facebook, Google, IBM, Intel, Microsoft, NVIDIA, Samsung, and VMware. This research was partially supported by NSF (grants 0953246, 1212962, 1320531, 1409723), Semiconductor Research Corporation, and the Intel Science and Technology Center for Cloud Computing.

Authors' addresses: D. Lee, S. Ghose, G. Pekhimenko, and O. Mutlu, Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh PA 15213; email: {donghyu1, ghose, gpekhime, onur}@cmu.edu; S. Khan, University of Virginia, 85 Engineer's Way, Charlottesville VA 22904; email: samirakhan@virginia.edu.

1. INTRODUCTION

Main memory, predominantly built using DRAM, is a critical performance bottleneck in modern systems due to its limited bandwidth [Burger et al. 1996; Rogers et al. 2009]. With increasing core counts and more pervasive memory-intensive applications, memory bandwidth is expected to become a greater bottleneck in the future [Dean and Barroso 2013; Mutlu 2013; Mutlu and Subramanian 2014]. For the last few decades, DRAM vendors provided higher bandwidth by using higher IO frequencies, increasing the bandwidth available per pin (improving by 17 times over the last decade, from 400Mbps in DDR2 to 7Gbps in GDDR5 [Smith et al. 2012]). However, further increases in IO frequency are challenging due to higher energy consumption and hardware complexity. Recent developments in 3D integration through *through silicon vias* (TSVs) enable an alternative way of providing higher bandwidth. TSVs enable wider IO interfaces among vertically stacked layers in 3D-stacked DRAM architectures [Kim et al. 2011; JEDEC 2013a, 2014; Hybrid Memory Cube Consortium 2013, 2014; Kang et al. 2009; Lee et al. 2014].

Even though TSV technology enables higher data transfer capacity, existing 3D-stacked DRAMs cannot fully utilize the additional bandwidth offered by TSVs. For example, Wide I/O [Kim et al. 2011] offers a very high external bus width (512 bits, which is 16–64 times wider than conventional DRAM chips), but can only operate at much lower frequencies (200–266MHz) than conventional DRAMs (which operate at as high as 2,133MHz for DDR3). As a result, the effective bandwidth increase can be an order of magnitude lower than the bus width increase. Based on our analysis of the DRAM hierarchy (provided in Section 2), 3D-stacked DRAMs *cannot* fully exploit the wider *external* IO interface due to their limited *internal* bandwidth. On a row activation, a DRAM chip reads a whole row simultaneously from the cell array to a row buffer, but can transfer only a *fraction* of the data in the row buffer through a *limited* set of internal wires (global bitlines) and associated global sense amplifiers. We refer to both of these resources together as the *global bitline interface*. Figure 1(a) shows how the global bitline interface connects the DRAM cell array to the TSVs.

The limited number of global bitlines and sense amplifiers (i.e., the global bitline interface width) thus constrains the internal data transfer capacity of DRAM. One naive way to enable higher bandwidth is to add more global bitlines and sense amplifiers to 3D-stacked DRAMs, as was previously done [JEDEC 2013a, 2014]. However, doing so increases area cost and energy consumption, making this intuitive and simple solution expensive.[1]

Our goal in this work is to enable higher internal bandwidth in 3D-stacked DRAM without incurring the cost of widening the global bitline interface. In order to design such a system, we observe that a large number of global bitlines and sense amplifiers already exist *across the multiple layers* of a 3D-stacked DRAM, but that *only a fraction* of them are enabled at any particular point in time. Figure 1(b) shows that only a single layer of the DRAM stack can transfer data to the external IO interface, while other layers stay idle. We exploit these otherwise idle global bitline interfaces to access multiple DRAM layers *simultaneously*, which can overcome the internal bandwidth constraints of a single layer by delivering enough data to fully utilize the available external bandwidth of the TSV-based IO interface. We call this architecture

[1]We estimate that global sense amplifiers (and their corresponding control logic) consume 5.18% of the total area in a 55nm DDR3 chip [Vogelsang 2010; Rambus 2010]. Considering that 3D-stacked DRAM already contains many more global sense amplifiers than conventional DRAM (see Section 2.4), adding even more sense amplifiers may be quite costly. Each sense amplifier also consumes a large amount of energy, because it is typically implemented as a differential amplifier, whose performance is strongly dependent on standby current [Keeth et al. 2007; Razavi 2000].

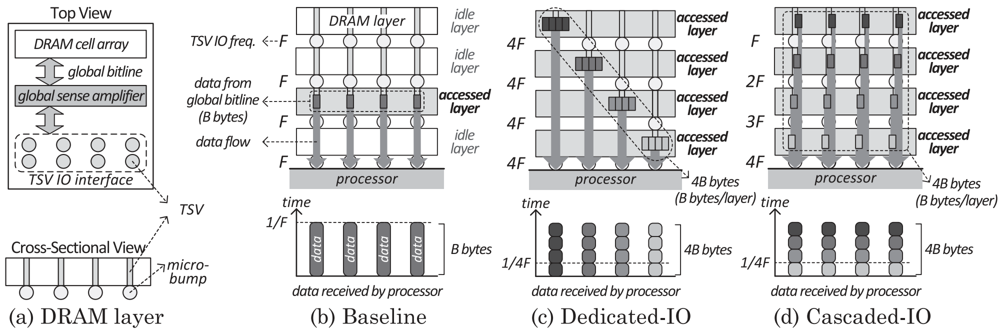


Fig. 1. Single-layer (baseline) versus multi-layer access in 3D-stacked DRAM.

Simultaneous Multi-Layer Access (*SMLA*). Using SMLA, multiple global bitline interfaces in multiple layers supply data to the external IO interface, providing the interface (vertically connected across all stacked layers) with enough data to enable transfers at a much higher frequency than existing 3D-stacked DRAMs (e.g., Wide I/O).

To implement SMLA, simultaneous data transfer from multiple layers through the existing IO interface requires *coordination* across the layers to avoid channel contention. One simple way to enable an SMLA architecture is to restrict the sharing of channels and assign dedicated IOs to each layer. We refer to this simple solution as *Dedicated-IO*, where a subset of TSVs form an *IO group*, with each IO group dedicated only to a single layer (Figure 1(c)). Despite having access to only the smaller number of TSVs in its IO group, each layer can still transfer the same amount of data by operating its IOs at a higher frequency. As each layer transfers data simultaneously at a higher frequency, Dedicated-IO enables bandwidth proportional to the number of layers (e.g., 4 times for a four-layer stacked DRAM) as opposed to the baseline system, which can transfer data from only a single layer (Figure 1(b)).[2] While this architecture enables higher bandwidth, it has two disadvantages. First, as each layer requires dedicated connections to specific TSVs, the design of each layer is not uniform anymore, resulting in higher manufacturing cost. Second, the IO clock frequency scales linearly with the number of layers, resulting in greater dynamic energy consumption.

To solve these problems, we propose *Cascaded-IO*, which exploits the architectural organization of a TSV-based interface, where an upper layer transfers its data through lower layers. Cascaded-IO (Figure 1(d)) enables simultaneous access to each layer by time-multiplexing the IOs in a pipelined fashion. In this design, all layers operate concurrently, with each layer first sending its own data and then sending data transferred from the upper layers. By operating each layer at a frequency proportional to the number of layers above it, Cascaded-IO provides a much higher bandwidth than existing 3D-stacked DRAMs (e.g., 4 times for a four-layer stacked DRAM). While several layers within Cascaded-IO operate at a higher frequency, we observe that only the bottom layer theoretically needs to operate at the highest frequency, as this is the only layer that transfers data from all of the DRAM layers. We propose to reduce the frequency of other layers, optimizing the frequency individually for each layer based on the layer's bandwidth requirements. As a result, Cascaded-IO enables higher overall

[2]A new design from Hynix, High Bandwidth Memory [JEDEC 2013a], which we describe in detail in Section 9, proposes to use exclusive IO interfaces for each layer of 3D-stacked DRAM, but enables higher bandwidth by increasing the internal bandwidth with *additional* global bitlines. Dedicated-IO, in contrast, enables higher bandwidth at a lower cost by leveraging the *existing* global bitline interfaces in DRAM.

DRAM bandwidth at low energy consumption, *without requiring nonuniform layers that have different physical designs*.

Our work makes the following contributions:

(1) We propose a new 3D-stacked DRAM organization, SMLA, which enables higher bandwidth with low cost by leveraging the otherwise idle global bitline interfaces in multiple layers of 3D-stacked memory. SMLA delivers the effective performance of quadrupling the number of global bitlines and global sense amplifiers, without physically requiring any additional global bitlines and global sense amplifiers (and thus without a large area increase), by precisely sequencing how each layer shares the TSVs.

(2) We introduce two low-cost mechanisms to transfer data from multiple layers without conflicting in the shared IO interface of 3D-stacked DRAMs. Our first mechanism, Dedicated-IO, statically partitions the TSVs across layers to avoid channel contention, but increases manufacturing costs and energy consumption. Our second mechanism, Cascaded-IO, time-multiplexes shared IOs so that each layer in 3D-stacked DRAM first transfers data from its own layer and then transfers data from upper layers. Cascaded-IO avoids nonuniformity in the design of each layer, reducing its manufacturing cost with respect to Dedicated-IO.

(3) We introduce a 3D-stacked DRAM architecture that can operate the different DRAM layers at different clock frequencies. By doing so, our Cascaded-IO mechanism enables higher bandwidth at low energy consumption, as it optimizes the IO frequency of each layer based on the layer's bandwidth delivery requirements.

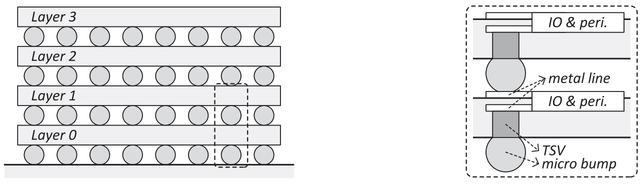
(4) Our extensive evaluation of 3D-stacked DRAMs on 31 applications from the SPEC CPU2006, TPC, and STREAM application suites shows that our proposed mechanisms significantly improve performance and reduce DRAM energy consumption (by 55%/18%, respectively, on average, for our 16-core multiprogrammed workloads) over existing 3D-stacked DRAMs.

2. 3D-STACKED DRAM BANDWIDTH CONSTRAINTS

To understand the internal bandwidth bottleneck of 3D-stacked DRAM, we first describe the unique design of the IO interface in state-of-the-art 3D-stacked DRAM systems (Section 2.1). We then walk through the DRAM architectural components that make up the datapath used for both read and write requests. We analyze the bandwidth of each component along this datapath in Section 2.3 and study the tradeoff between bandwidth and area in Section 2.4.

2.1. Using TSVs to Increase IO Bandwidth

We first explain the detailed organization of a 3D-stacked DRAM, which integrates a much wider external IO bus than conventional DRAM. Figure 2(a) shows a 3D-stacked DRAM consisting of four DRAM layers, which are connected using micro-bumps and TSV interfaces. A TSV interface vertically connects all of the layers. The bottom of the four stacked DRAM layers is either directly placed on top of a processor chip or connected to a processor chip by metal wires (we discuss this in more detail in Section 2.2). Figure 2(b) details the organization of the TSV and micro-bump connections within the DRAM stack. Two metal lines connect each layer to these interfaces. At each layer, the top metal line is connected to the micro-bump of the DRAM layer above, and the bottom metal line is connected to the TSV. At the bottom of the layer, this TSV is exposed and connected to a micro-bump, which then connects to the top metal line of the layer below. These two metal lines are eventually connected with a via or over peripheral circuits. In this way, several such layers can be stacked one on top of another.



(a) Overall (b) Detailed

Fig. 2. TSV interface in 3D-stacked DRAM.

TSVs provide two major benefits to increase memory bandwidth and energy efficiency. First, due to the small feature size of modern TSV technologies (10–35μm pitch for TSVs [Kim et al. 2011; West et al. 2012; Huyghebaert et al. 2010; Harvard and Baker 2011] vs. 90μm pitch for conventional pads), 3D-stacked DRAM can integrate hundreds of TSVs for its connections between layers. Second, the small capacitance of a TSV reduces energy consumption. Compared to conventional DRAM, whose off-chip IO is connected by long bonding wires and metal connections in the package, TSV connections between layers are very short, leading to lower capacitance and energy consumption during data transfer. As a result, 3D-stacked DRAM offers a promising DRAM architecture that can enable both higher bandwidth and energy efficiency.

2.2. Connections Between 3D-Stacked DRAM and Processor

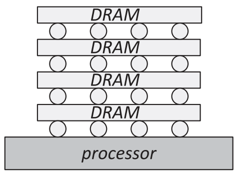
There are two major approaches used today to connect 3D-stacked DRAM with the processor. The first approach is to directly attach the 3D-stacked DRAM onto the processor die, referred to as 3D integration, as shown in Figure 3(a). This approach does not require any additional components (e.g., PCB, interposer) and integrates both the processor and the 3D-stacked DRAM within the minimal feature size (i.e., within a small package). However, this approach reduces the thermal dissipation capacity. During its operation, a modern processor generates a large amount of heat, which requires active thermal dissipation solutions (most commonly, by placing a heat sink plate on top of the processor). If a stacked DRAM sits on top of the processor, it is difficult to attach such heat sinks directly to the processor, which could lead to processor overheating. Furthermore, the heat from the processor propagates to the stacked DRAMs, causing the DRAM cells to leak more charge, which can require more frequent refreshing.

The second approach is to connect the processor and the 3D-stacked DRAM with a *silicon interposer*, referred to as 2.5D integration. As Figure 3(b) shows, both the processor and the 3D-stacked DRAM are located on the silicon interposer, which provides fine-pitch metal wire connections between the two. The major benefits of this approach are (1) not limiting the thermal dissipation capacity of the processor, and (2) decoupling the 3D-stacked DRAM from the processor's heat. The major drawbacks of this approach are that (1) it requires additional cost for producing the interposer, and (2) it could lead to a larger package.

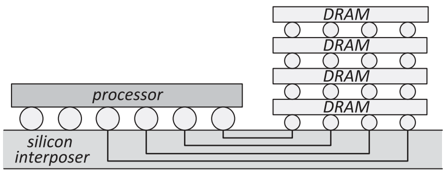
2.3. DRAM Bandwidth Analysis

In this section, we show that the 3D-stacked DRAM bandwidth is bottlenecked by the number of global bitlines and global sense amplifiers, which are costly to increase.

Internal Operation. DRAM has two kinds of accesses (read and write) that transfer data through mostly the same path between DRAM cells and off-chip IO. Figure 4(a) shows the read datapath from the cells to IO. When activating a row, all of the data in the row is transferred to a *row buffer* (a row of *local sense amplifiers*) through bitlines (❶ in Figure 4(a)). Then, when issuing a read command with a column address, only a small fraction of data in the row buffer (corresponding to the issued column

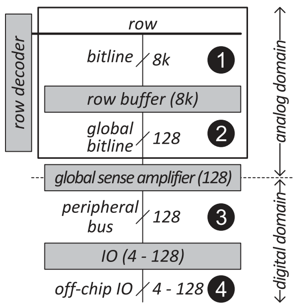


(a) Placing on top of processor (3D)

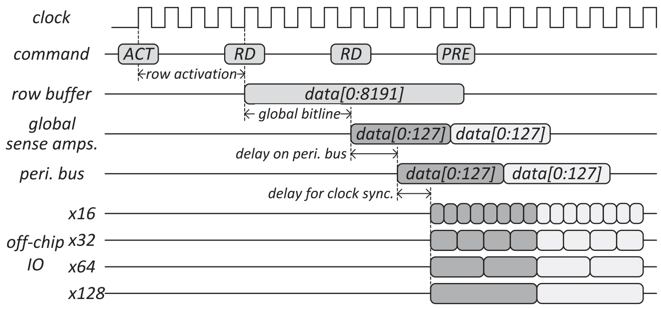


(b) Connecting with a silicon interposer (2.5D)

Fig. 3. Connecting 3D-stacked DRAM to a processor.



(a) Organization



(b) Timing

Fig. 4. Overview of DRAM organization.

address) is read by the *global sense amplifiers* through the *global bitlines* (❷). Peripheral logic then transfers this data to the off-chip IO (❸ and ❹). To write data, commands containing data are issued, and peripheral logic transfers this data to the global sense amplifiers that write the data to the row buffer, which then writes the data to the cells corresponding to the requested address.

Bandwidth Analysis. At each step, the individual structures along the datapath have their own data transfer rate (bandwidth). For a read access, we explore the bandwidth of each step in detail. When activating a row, all data within the row (e.g., 8Kbits [Micron 2014]) is transferred to the row buffer (❶ in Figure 4(a)), which takes about 13ns (based on the t_{RCD} timing constraint [Micron 2014]). Therefore, the bandwidth of the activation step is about 78.8GBps. After migrating data to a row buffer, the global sense amplifiers read 64 to 128 bits of data from the row buffer within 3 to 5ns through the global bitlines (❷). Therefore, the bandwidth of the global bitline interface is about 2 to 4GBps. The data read by global sense amplifiers is transferred to the IO interface (❸), which can operate at frequencies as high as 2GHz. Considering that the off-chip IO bus width can range from 4 to 128 bits (❹), the off-chip bandwidth can be in the range of 1 to 32GBps.

As this analysis shows, there is a bandwidth mismatch in 3D-stacked DRAM, since generally the global bitline interface (2–4GBps) cannot fully saturate (or match) the ample off-chip IO bus bandwidth (1–32GBps). This is in contrast with conventional DRAM, which has a limited off-chip IO bus width (e.g., a DDR3 DRAM chip with an 8-bit bus, and 2GBps bandwidth), and thus can use a much narrower global bitline interface (64-bit width, 2GBps bandwidth) to match the off-chip bandwidth. For 3D-stacked DRAM, where the off-chip bus bandwidth is significantly higher (e.g., the 128-bit TSV interface for Wide I/O), *the limited global bitline interface bandwidth now becomes the dominant bottleneck* of the overall DRAM bandwidth. Currently, there are two alternatives to dealing with this bottleneck: (1) widen the global bitline interface, which incurs a high cost, as we will discuss in Section 2.4, or (2) reduce the operating frequency such that the off-chip bus bandwidth matches the lower bandwidth of the global bitline interface, as is done in existing 3D-stacked DRAMs (e.g., Wide I/O).

To show the relationship between the operating frequency and the bus width of the IO interface, we draw timing diagrams for the data bus during a read access (Figure 4(b)). We divide the DRAM hierarchy into two domains: (1) an analog domain, whose components consist of sensing structures and the data repository (the row buffer and global bitline interface in Figure 4(a)), and (2) a digital domain, whose components (the peripheral bus and off-chip IO in Figure 4(a)) transfer data with full swing of the data value (0 or 1).

In the analog domain, a data repository (row buffer) is connected to a sensing structure (global sense amplifiers) with fixed metal wires. Accessing the row buffer from the global sense amplifiers has a fixed latency corresponding to the global bitlines. Only 1 bit of data can be transferred over a single global bitline. Therefore, the bandwidth of the global bitline interface is dictated by the number of global bitlines and global sense amplifiers (128 bits in this example). Each global sense amplifier is directly connected to a stored bit in the row buffer through a single wire and detects the intermediate voltage value (between 0 and V_{dd}) of the stored bit before finishing the sense amplification of the stored bit in the row buffer [Lee et al. 2013]. Therefore, it is difficult to divide the global bitline interface into multiple stages to implement pipelining, which would increase the frequency and thus the interface bandwidth.

On the other hand, the peripheral interface (from the global sense amplifiers to off-chip IO) and the IO interface (toward off-chip) can be divided into multiple stages and implemented in a pipelined manner, as they handle signals in the digital domain. Therefore, as shown in Figure 4(b), the 128 bits of data from the global sense amplifiers can be transferred through either a narrow IO bus with high frequency (e.g., DDR3 with a 16-bit bus) or a wide IO bus with low frequency (e.g., Wide I/O with four channels, each of which has a 128-bit bus [Kim et al. 2011; JEDEC 2011]). In the ideal case, assuming that the global bitline interface provides enough data to the peripheral interface, the wide bus in the off-chip IO can transfer the data at higher frequency (than global bitlines), enabling much higher external IO bandwidth (i.e., $128bits \times 2GHz = 32GBps$).

Thus, our takeaway from our analysis is that (1) it is relatively easy to increase the external IO bandwidth of DRAM, but (2) the internal bandwidth is bottlenecked by the number of global bitlines and global sense amplifiers, whose bandwidth is much more costly to increase. Based on this analysis, we observe that even though 3D-stacked DRAM can potentially enable much greater external bandwidth, the limited internal bandwidth on the global bitline interface significantly restricts the overall available DRAM bandwidth.

2.4. Bandwidth Versus Global Sense Amplifiers

Figure 5 shows the relationship between the overall bandwidth of DRAM-based memory systems (one chip for conventional DRAM, four layers for 3D-stacked DRAM) and the number of global sense amplifiers (and hence the global bitlines as well) in a DRAM chip. We conservatively plot the minimum number of global sense amplifiers that achieves the corresponding bandwidth in each design. For example, for DDR3 DRAM with an x16 IO bus and a burst length of 8, we assume that it contains only 128 global bitlines and sense amplifiers.

As Figure 5 shows, the bandwidth of DRAM has mostly been a linear function of the number of global sense amplifiers across several generations. In early DRAMs, the number of global sense amplifiers was small (e.g., 64 for a DDR2 chip). Therefore, DRAM vendors scaled the global bitline interface to increase the bandwidth. However, recently proposed 3D-stacked DRAMs [JEDEC 2011, 2013a, 2014] drastically increase

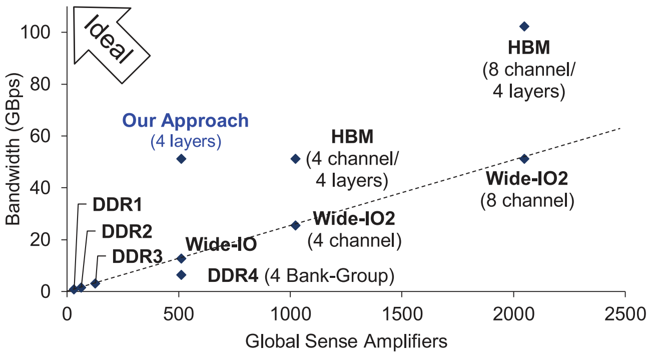


Fig. 5. DRAM bandwidth versus global sense amplifiers (for DRAMs with internal frequency of 200MHz).

the number of global sense amplifiers. For example, HBM [JEDEC 2013a; Lee et al. 2014] offers the highest bandwidth but requires a large number of global sense amplifiers (16 times more than DDR3). This drastic increase comes from introducing TSV technologies that enable a wider, area-efficient bus. In the presence of the wider bus, the dominant bottleneck in achieving higher bandwidth has now shifted to the efficient implementation of global sense amplifiers.

We estimate the area of global sense amplifiers (and their corresponding control logic) to be 5.18% of the total chip area in a 55nm DDR3 DRAM [Vogelsang 2010; Rambus 2010]. Taking into account (1) the large area of global sense amplifiers in DDR3 and (2) the recent increase in the number of sense amplifiers in 3D-stacked DRAMs, we expect that adding more of the expensive global sense amplifiers will become increasingly cost-inefficient. Unfortunately, 3D-stacked DRAM bandwidth is bottlenecked by the number of global sense amplifiers (as we explained in Section 2.3).

Our goal in this work is to enable higher DRAM bandwidth without requiring more global sense amplifiers. As we explain in Section 3, we enable much higher bandwidth at low cost using the existing structures in 3D-stacked DRAM (instead of adding more global sense amplifiers), with the goal of approaching the ideal case of high bandwidth at low cost (the top left corner of Figure 5).

3. OPPORTUNITIES FOR INCREASING BANDWIDTH

Even though 3D-stacked DRAM integrates a much wider external IO bus through the use of TSVs, the limited internal bandwidth within the DRAM chip prevents us from extracting the full data transfer rate of this wider external IO interface, as we analyzed in Section 2.3. While one solution may be to increase the number of global bitlines and sense amplifiers, this can be a costly prospect (as we showed in Section 2.4). Instead, we would like to investigate orthogonal approaches that exploit the *existing* DRAM structures more efficiently to overcome the global bitline bottleneck.

We examine the architectural similarities and differences between 3D-stacked DRAM and conventional DRAM modules and use this comparison to identify challenges and opportunities that are unique to the 3D-stacked DRAM design. We use these observations to drive our low-cost approach. To aid with this comparison, we provide a side-by-side logical view of these two architectures in Figure 6 and focus on three major observations:

(1) *Each layer in the 3D-stacked DRAM acts as a rank, operating independently of each other but sharing the IO interface, which is similar to the rank organization of a conventional DRAM module.*

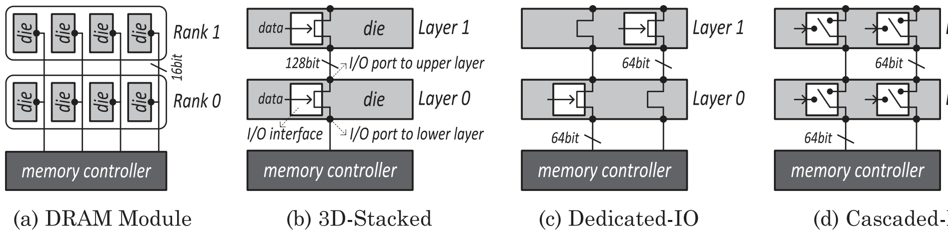


Fig. 6. Off-chip bus connections for existing (a, b) and proposed (c, d) DRAMs.

Figure 6(a) shows a conventional DRAM module that has two ranks, each consisting of four DRAM chips. All DRAM chips in a rank share the control logic and operate in lockstep. The two ranks share IO interfaces, so data from different ranks must be transferred serially to avoid contention. Figure 6(b) shows the organization of a 3D-stacked DRAM that has two stacked layers. Similar to the multi-rank DRAM module, each layer in the 3D-stacked DRAM can operate independently but still shares the IO interface (TSV connections). Therefore, the TSV interface transfers data from different layers serially (i.e., only one layer can send data at a time).

(2) *Only one stacked chip forms a rank in 3D-stacked DRAM, compared to multiple chips in a conventional DRAM module.*
Figure 6(a) shows four conventional x16 DRAM chips forming a rank by dividing the 64-bit IO bus into four smaller 16-bit IO buses, with each smaller bus connected to a different chip. In contrast, each 3D-stacked DRAM layer is made up of a single chip, with the entire TSV bus connected to it (see Figure 6(b)).

(3) *Each layer of 3D-stacked DRAM has two ports that can be easily decoupled by the peripheral logic, whereas conventional DRAM has only a single port.*
As shown in Figure 2(b), each stack layer in 3D-stacked DRAM is connected to its neighboring upper stack layer through the topmost metal layer. Each stack layer is also connected to its lower stack layer through one of its middle metal layers. These two metal layers can be connected within the DRAM stack layer by either fixed vias or peripheral logic. These two different metal layers form two independent IO ports, whose connectivity can be controlled by peripheral logic. Therefore, the two IO ports in each stack layer are connected but can be easily decoupled to support *independent data transfers on each port*.

As we described in Section 2.3, the major bandwidth bottleneck in a 3D-stacked DRAM-based memory system is the internal global bitline interface. Our approach to solve this problem is inspired by the organization of the conventional DRAM module, where multiple DRAM ranks operate simultaneously, thereby increasing the overall bandwidth of the memory system (Figure 6(a)). From our observations, we conclude that the ability to decouple input ports to each layer in the 3D-stacked DRAM can allow us to treat multiple layers similar to the way we treat multiple ranks within a DRAM module. Thus, we would like to make multiple layers operate simultaneously to transfer more data within a given time. We will describe our mechanisms that enable this in the next section.

4. SIMULTANEOUS MULTI-LAYER ACCESS

We propose *Simultaneous Multi-Layer Access*, a new mechanism that enables the concurrent use of multiple stack layers (chips) within 3D-stacked DRAM to fully utilize the TSV-based IO bandwidth. SMLA exploits existing DRAM structures within the chip by altering the IO logic to include multiplexing that supports these concurrent operations. This allows the bandwidth gains obtained by SMLA to be complementary to the gains that could be otherwise obtained from scaling these DRAM structures (e.g., adding more global bitlines and sense amplifiers).

In this section, we describe two implementations for coordinating these multiplexed IOs. The first, *Dedicated-IO*, *statically partitions* the TSV bus into several groups, with each group *dedicated* to the IO of a single layer. The second, *Cascaded-IO*, *time-multiplexes* the TSV bus instead of partitioning it across the layers, by exploiting our observation that we can decouple the input ports to each layer (Section 3). As we will discuss in Section 4.2, the Cascaded-IO implementation allows the circuit design of each stack layer to be identical, reducing design effort (compared to Dedicated-IO), and offers a more energy-efficient approach to implement SMLA.

4.1. Dedicated-IO

Dedicated-IO divides the wide TSV bus into several narrower groups, where each group is statically assigned to a single layer in the 3D stack. In a modern 3D-stacked DRAM, with a total IO width $W = 128$ bits, all layers share all W IO connections, as shown in Figure 6(b). Each layer runs at frequency F, which allows it to transmit W bits per $^1/_F$ time period. In comparison, a two-layer Dedicated-IO DRAM (Figure 6(c)) reserves $^W/_2$ IO connections for each layer, giving the layer exclusive access to those connections, while driving the connections at $2F$ frequency. In general, for a 3D-stacked DRAM with L layers and W IO connections, Dedicated-IO allows each layer to drive W bits over $^W/_L$ TSVs, at a frequency $F \times L$ (which the TSVs can already tolerate). Each layer continues to send W bits per time period $^1/_F$, so we can keep the number of global sense amplifiers and bitlines the same as in the baseline 3D DRAM (W per layer). As each layer has an exclusive set of TSVs, all L layers can now operate in parallel, allowing our mechanism to send $W \times L$ bits (as opposed to only W bits) in aggregate over $^1/_F$ time.

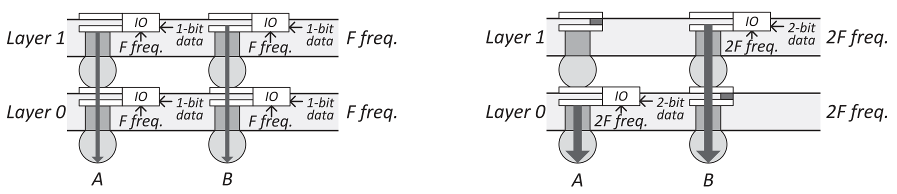
4.1.1. Implementation Details.

Figure 7 compares the detailed organizations of an existing 3D-stacked DRAM (Figure 7(a)), where all layers share all TSVs, and a 3D-stacked DRAM using Dedicated-IO (Figure 7(b)). To clearly explain the access scenarios for both cases, we use a simplified organization of 3D-stacked DRAM consisting of two layers, and we focus only on a 2-bit subset of each layer (i.e., the DRAM contains two TSVs as IO for data transfer, and each layer has two bitlines that can deliver 2 bits every clock period, which is $^1/_F$ for the baseline). In this example, the Dedicated-IO-based 3D-stacked DRAM dedicates TSV A to the lower layer (Layer 0) and dedicates TSV B to the upper layer (Layer 1).

In the baseline 3D-stacked DRAM, only one selected layer (Layer 1 in this example) can transfer data during a single clock period, but can use both TSV A and TSV B. Therefore, Layer 1 transfers 1 bit per TSV (2 bits in total), while Layer 0 remains inactive. On the other hand, for the Dedicated-IO-based 3D-stacked DRAM, Layers 0 and 1 can simultaneously transfer data through TSVs that have been dedicated to each layer. As we discussed in Section 2, it is still possible to increase the IO frequency of 3D-stacked DRAM, as conventional DRAMs have a much higher IO frequency. Therefore, by partitioning the TSVs across the two layers, the Dedicated-IO-based 3D-stacked DRAM can transfer 2 bits of data *from each layer* during one baseline clock period ($^1/_F$) by doubling the IO frequency. As a result, Dedicated-IO enables twice the bandwidth of existing 3D-stacked DRAM by (1) simultaneously accessing both layers (doubling the available data per TSV), and (2) transferring data at double frequency. Note that this does not require any more or higher bandwidth global bitline interfaces within the DRAM chip.

4.1.2. Implementation Overhead and Limitations.

While Dedicated-IO enables much higher bandwidth, there are several drawbacks in this implementation on a real memory



(a) Baseline (b) Dedicated-IO

Fig. 7. Enabling Simultaneous Multi-Layer Access with Dedicated-IO.

system. First, each layer of the 3D-stacked DRAM now requires physically different connections. Therefore, the manufacturing cost of a Dedicated-IO-based 3D-stacked DRAM may be higher than that of the baseline 3D-stacked DRAM. Second, with an increasing number of layers, the IO clock frequency scales linearly, resulting in higher dynamic energy consumption.

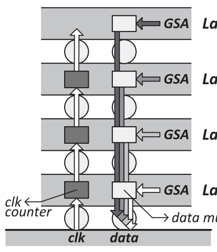
The scalability of Dedicated-IO can be limited by how fast the off-chip IO interface can be clocked (we call this the maximum operating frequency F_{max}), which is a physical limitation of the TSVs. As we mentioned, for L layers and a baseline frequency F, Dedicated-IO clocks the IO interface at frequency $F \times L$. However, the Dedicated-IO frequency can never exceed F_{max} (i.e., $F \times L \leq F_{max}$), which limits the value of L. We discuss approaches to scale the number of layers beyond this limit, though at the cost of bandwidth or additional area, in Section 4.3.

4.2. Cascaded-IO

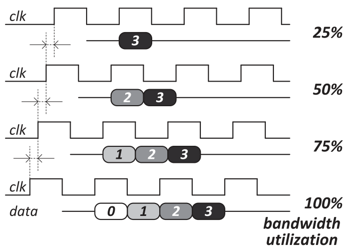
While Dedicated-IO increases bandwidth by *partitioning* the TSV IO interface across the layers (and running each DRAM layer at higher frequency), Cascaded-IO increases bandwidth by *time-slice multiplexing* the TSVs across different layers (and having different frequencies for different layers). Thus, Cascaded-IO keeps the structure of each DRAM layer exactly the same, reducing manufacturing cost over Dedicated-IO.

As Figure 6(d) shows, we add a multiplexer to the IO interface at each layer, to take advantage of the fact that each TSV can be broken into smaller segments. Each TSV segment connects Layer n with Layer $n + 1$, which means that all middle layers are connected to two segments (which we refer to as the upper and lower segments). This multiplexer allows the layer to either transmit W bits of its own data or pass through the W bits of data that are being transmitted by the upper TSV segment onto the lower TSV segment. We can adjust the timing such that within a single clock period $^1/_F$, a layer first sends its own data down, and then sends all of the data from the upper layers down in a pipelined fashion. The lowest layer therefore needs to send the data for all L layers, which requires Cascaded-IO to also transfer data at $F \times L$ frequency at that layer. However, layers further up the stack have to send less data overall, with the uppermost layer responsible for sending only its own data. This mechanism is only possible in a 3D-stacked DRAM that contains two independent ports connected by peripheral circuits, as we discussed in Section 3.

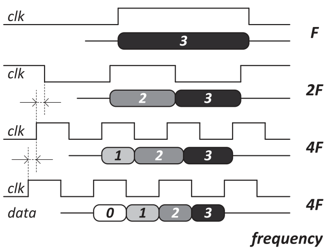
4.2.1. Implementation Details. Figure 8(a) shows the vertical structure of Cascaded-IO on a four-layer 3D-stacked DRAM. We again turn to a simplified example to illustrate the operation sequence, this time focusing on a 1-bit subset of each layer, and with all bits connected to the same TSV. In this setup, the clock frequency of the bottom layer, with respect to the baseline DRAM clock frequency F, is $4F$. For each TSV, the data interface in each layer contains a multiplexer that selects from one of two paths: (1) a connection to the upper TSV, which contains a bit being transmitted from one



(a) Vertical structure



(b) Identical clock frequency



(c) Optimized clock frequency

Fig. 8. Simultaneous Multi-Layer Access with Cascaded-IO.

of the upper stack layers, and (2) a data wire from the layer's global sense amplifiers, connected to its own cell.

In our example Cascaded-IO-based memory system, each layer transmits 4 bits in the time period $^1/_F$. A layer first fetches its own data from its cell. The multiplexer then drives this data as the first data transfer, sending the data to the next layer below. The layer then switches the multiplexer to receive data being transmitted from the layer above, and over the next three cycles sends this data to the layer below. In essence, this approach pipelines the data of all four layers over the four cycles, as shown in Figure 8(b). Layer 1, for example, drives its own data in the first cycle, sending it to Layer 0. In the second cycle, Layer 1 receives Layer 2's data and sends that down to Layer 0. At that same cycle, Layer 2 receives data from Layer 3 and sends that down to Layer 1 so that in cycle three, Layer 1 can send the Layer 3 data down to Layer 0.

One side effect of this pipelined approach is that not all layers contain useful data to send to the lower level every cycle. In our example, Layer 1 has nothing to send in the fourth cycle, while Layer 3, with no layers above it, only sends useful data in the first of the four cycles. As we show in Figure 8(b), when the IO is clocked at $4F$ frequency ($F \times L$), the layer-by-layer TSV bandwidth utilization shrinks from 100% at Layer 0 down to 25% at Layer 3. Since the bandwidth in these upper layers goes unused, it is wasteful to clock them at $4F$ frequency. We exploit this tiered utilization to *lower the energy consumed in the higher layers* by simply running each layer at a lower frequency that matches its actual bandwidth utilization. This frequency reduction at upper layers decreases the energy consumption of a Cascaded-IO-based DRAM (we show this in detail in Section 6).

Existing 3D-stacked DRAMs have a clock signal that typically originates from the memory controller. This signal is then sent to the bottom layer of the 3D-stacked DRAM and propagated to the upper layers via a TSV. In order to match the operating frequency of each layer to its bandwidth utilization, we need to add an efficient way of performing clock division at each layer (since an upper layer is never clocked faster than a lower layer in Cascaded-IO). While phase-locked loops (PLLs [Novof et al. 1995]) are a conventional way to generate clocks of arbitrary frequency, they consume large amounts of energy, making them an infeasible option for our heterogeneous clock frequency domains. Instead, Cascaded-IO adopts simple divide-by-two *clock counters*, which, when enabled, can generate a half-frequency clock. Figure 8(a) shows the organization of the clock path.

In our example, we increase the clock signal from the memory controller to $4F$ frequency (instead of F), and then divide by two in some of the upper layers. One drawback of only using divide-by-two clocks is that some layers do not receive the optimal clock that matches their bandwidth utilization. For example, Layer 1 transmits three bits of data, but since we can only generate clocks at frequencies $4F$, $2F$, F, etc., we have no choice but to continue running Layer 1 at $4F$ frequency. However, we can run Layer 2 at $2F$ frequency, and Layer 3 at F frequency. Generalized to L layers, the lower half of the L layers run at frequency $F \times L$, and then the next $^{L}/_{4}$ layers run at frequency $\frac{F \times L}{2}$, the next $^{L}/_{8}$ layers at frequency $\frac{F \times L}{4}$, and so on. The uppermost layer runs at frequency F. As not every layer performs clock division, we add a division enable control to the clock counters to ensure that the design of each layer remains identical.

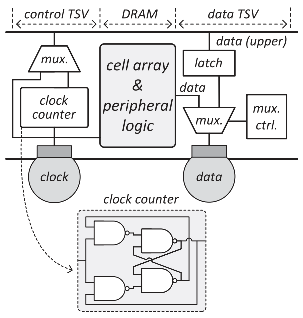
Figure 8(c) illustrates how the timings and data transfer work with our reduced clock mechanism. To avoid issues due to clock skew, the data multiplexer for a layer is actually only responsible for synchronizing its own data. Afterward, when the multiplexer switches to "receive" data from the upper layer, it simply connects a bypass path from the upper layer to the lower layer for the remaining cycles. This allows the upper layer to directly send its data to the bottommost layer. Figure 8(c) depicts the timing of this cut-through mechanism with skew. For example, in Layer 1, the data switches from Layer 2's bit to Layer 3's bit in the middle of the Layer 1 clock cycle, since the multiplexer in Layer 1 directly connects Layer 2 to Layer 0 without synchronization.

Figure 9(a) shows a detailed implementation of Cascaded-IO, focusing on the TSV connections in a single DRAM layer. In each control signal TSV (i.e., clock, commands, and addresses), Cascaded-IO requires a counter that passes these input signals through three additional logic gates. To multiplex the data TSVs between the upper and current layers, each data signal also passes through three additional logic gates (two transistors for the latch, and one transistor for the multiplexer). In total, each request incurs the delay of passing through six logic gates. Note that the DRAM cell array and peripheral logic are still identical to the baseline 3D-stacked DRAM [Kim et al. 2011]. Therefore, Cascaded-IO can be implemented with noninvasive logic modifications. Considering that there is precedence for (1) using simple frequency dividers such as counters to enable multiple high-speed frequency domains [Larsson 1996; Borkar 2007], and (2) using simple transmission gates for data multiplexing [Oh et al. 2014], our additional TSV logic can be easily integrated into 3D-stacked DRAM designs.

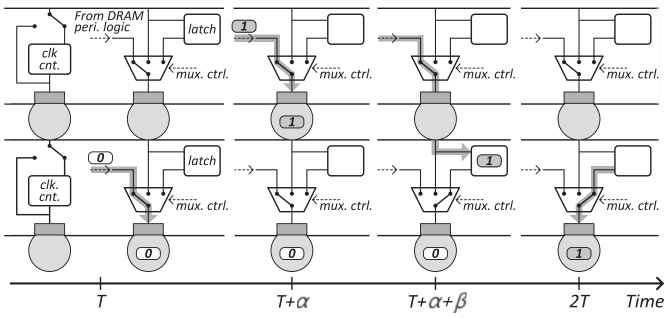
Figure 9(b) shows in detail how data is transferred through two layers in Cascaded-IO. At the start of clock period T, the DRAM periphery in the *lower layer* transfers its data to the data TSV. At $T + \alpha$, the DRAM periphery in the *upper layer* transfers its data to the data TSV. α represents the clock propagation delay for the clock path, including delays due to the clock counter. Then, the data of the upper layer reaches the lower layer data latch after an additional delay β. At the start of the subsequent clock period ($2T$), the upper layer data stored in the latch is transferred to the data TSV at the lower layer. As Figure 9(b) demonstrates, Cascaded-IO first transfers the lower layer's own data, providing *at least one complete clock period* to wait for the upper layer's data to be latched. Therefore, as long as the additional delay ($\alpha + \beta$) for passing through the six logic gates introduced by Cascaded-IO is smaller than one clock period (2.5ns in our evaluation), Cascaded-IO can ensure correct operation. Considering that process technologies from 2007 enable gate delays as low as tens of picoseconds [ITRS 2007], a single clock period (e.g., 2.5ns for an 800MHz DDR interface) is more than enough to hide the additional logic delay of Cascaded-IO.

4.2.2. Implementation Overhead and Limitations. To implement a Cascaded-IO-based TSV interface, each DRAM chip needs to include two additional components:

(1) *Clock control logic*, which consists of a counter to halve the clock signal frequency, and delay logic (for every command and address signal) to synchronize the corresponding signals to match the clock propagation delay.



(a) Detailed Implementation



(b) Detailed Operation

Fig. 9. Detailed implementation and operation of Cascaded-IO.

(2) *Data multiplexing logic*, which consists of a data latch to temporarily preserve the upper layer data, and a multiplexer and associated control logic to switch the TSV connection between two data sources (the data latch with upper layer data, and the data being driven from the current layer).

The counter and multiplexer have very simple designs and only require a small number of transistors, resulting in negligible area overhead. Similar and more sophisticated counters may be needed to implement dynamic frequency scaling in memory chips [David et al. 2011; Deng et al. 2011]. The multiplexer control for each layer also contains a counter and connects its IO drivers to the lower TSVs whenever the counter value is `2'b00`. For all other values, the control switches the multiplexer to connect the input from the upper layer to the lower TSV. The overhead and complexity of the control logic are also negligible.

Like Dedicated-IO, Cascaded-IO has some limitations on the number of layers that can be implemented. For example, the number of layers is limited by the amount of time required for a data signal to traverse over the multiple layers. We discuss how to overcome these limits for larger layer counts in Section 4.3.

4.2.3. Power Consumption Overhead. Due to the different frequencies and data bandwidth at each layer, the bottom layer in Cascaded-IO consumes more power than the upper layers. As the power network is driven from the bottom, power delivery is strongest in the lowest layer of 3D-stacked DRAM and weaker in the upper layers [Shevgoor et al. 2013]. As we discussed in Section 4.2.1, Cascaded-IO reduces power consumption at these upper layers, which as a byproduct allows the layers of Cascaded-IO to align well with the power network strength at each layer. While the bottom layer operates at $4F$ frequency, the upper layers have lower clock frequencies that match their much lower bandwidth requirements (e.g., the topmost layer is clocked at frequency F). Therefore, while the physical design of each layer remains homogeneous, we can minimize power consumption by introducing heterogeneity in the layers' operating frequencies. Such an approach has much lower power consumption than if we were to run all of the layers at $4F$ frequency (as is done in Dedicated-IO).

4.3. Scaling to Large Numbers of Layers

So far, we have discussed a 3D-stacked DRAM organization that has four layers. However, both Dedicated-IO and Cascaded-IO can be applied to larger numbers of stacked layers, with a tradeoff between bandwidth and area cost. To show this, we explain next how our mechanism can be implemented on an eight-layer stack. Note that the number of layers that can be stacked mostly depends on the TSV technology. Our mechanisms (both Dedicated-IO and Cascaded-IO) do not *further* limit 3D-stacked DRAM scaling (i.e., integration of more layers), but they themselves do have some scalability limitations.

For Dedicated-IO, the major limiter for scaling to larger numbers of layers is the maximum operating frequency (F_{max}), as described in Section 4.1.2. There are two possible approaches to implement eight-layer stacking that work around the maximum frequency limitation. Approach 1, which is simpler, provides dedicated TSV IO interfaces for each layer. Compared to four-layer stacks, eight-layer stacks require twice the number of TSVs, and thus provide twice the bandwidth. Approach 2 is to have pairs of layers (i.e., two 4-layer groups for eight-layer stacking) share a single set of TSV IO interfaces, as shown in Figure 10(a). Therefore, the TSV interface in the eight-layer stack is the same as that in the four-layer stack, but this means that the bandwidth also remains unchanged compared to the four-layer stack. In this case, only one of the two-layer groups can be accessed at any given time.

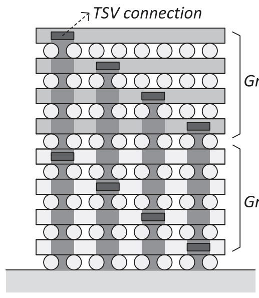
For Cascaded-IO, there are two limiters to increasing the number of layers. The first limiter is burst length. Each layer of Cascaded-IO must provide data to the TSV IO interfaces for every access. Therefore, the maximum number of stacking layers can never exceed the burst length. The second limiter is the amount of time available to hide the multi-layer traversal latency. Approach 1 to avoid these limiters is to implement independent TSV IO interfaces for two groups of four layers each, which essentially combines Dedicated-IO and Cascaded-IO. Such an approach doubles bandwidth at the cost of twice the TSV count. Approach 2 again splits the layers into two groups of four layers each, but the groups now share their TSV interfaces as shown in Figure 10(b). This approach enables eight-layer stacking within the same TSV cost of four-layer stacking but lacks the additional bandwidth. As shown in Figure 10(b), layers operating at the same clock frequency are located adjacently, in order to maintain the same clock division path as the four-layer design.

These approaches can be applied to scale to even larger numbers of layers, but they come with similar tradeoffs to the eight-layer case earlier.

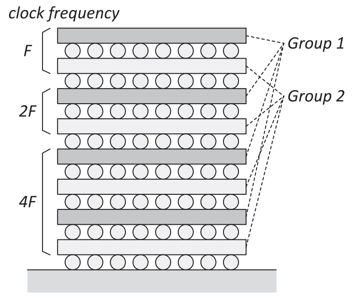
5. RANK ORGANIZATIONS WITH MULTIPLE LAYERS

In DRAM, a rank is a collection of DRAM chips that operate in lockstep. A rank-based organization allows pieces of a row to be split up and distributed across these chips, since they logically operate as a single structure. For example, a read command accesses all chips in the rank, retrieving a portion of the data from each chip. Existing 3D-stacked DRAMs treat each layer as a single rank, as only one of these layers can be accessed at a time. SMLA opens up new possibilities for rank organization, since it enables multiple DRAM layers to be accessed simultaneously. In this section, we explore two such organizations: *Multi-Layer Ranks* (MLR) and *Single-Layer Ranks* (SLR). Figure 11 compares these rank organizations in a two-layer 3D-stacked SMLA DRAM. In MLR, all of the layers work together as a single rank, and the data for a single memory request is distributed across all of the layers, as shown in Figure 11(a). On the other hand, SLR treats each stack layer as an independent rank, as is done in existing 3D-stacked DRAM, which means that for each memory request, all of the data is contained within a single layer, as shown in Figure 11(b).

The two organizations represent a new tradeoff between data transfer latency and rank-level parallelism. Figure 12 shows the timeline of serving requests using MLR and SLR in a two-layer stacked DRAM, for both Dedicated-IO and Cascaded-IO. To show the data flow of Dedicated-IO, the TSVs are divided into two IO groups, with each group connected to only one of the two layers in Dedicated-IO. (To consistently illustrate how data is transmitted in Cascaded-IO and Dedicated-IO, we also show IO groups in the timeline of Cascaded-IO, but Cascaded-IO does not actually divide the

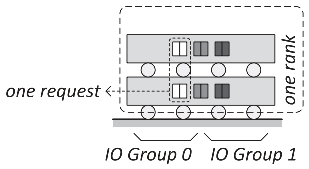


(a) Dedicated-IO (Approach 2)

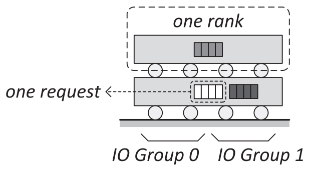


(b) Cascaded-IO (Approach 2)

Fig. 10. Eight-layer 3D-stacked DRAM with simultaneous access to four layers.

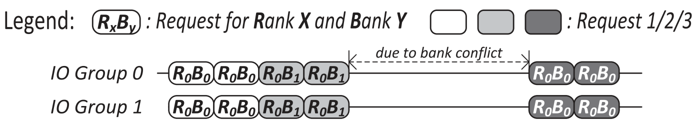


(a) Multi-Layer Ranks (MLR)

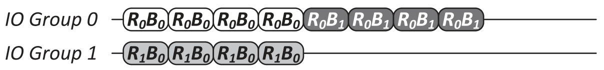


(b) Single-Layer Ranks (SLR)

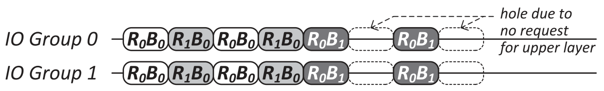
Fig. 11. Rank organization in SMLA.



(a) **SMLA** with **Multi-Layer Ranks** (1 rank, 2 banks/rank)



(b) **Dedicated-IO** with **Single-Layer Ranks** (2 ranks, 2 banks/rank)



(c) **Cascaded-IO** with **Single-Layer Ranks** (2 ranks, 2 banks/rank)

Fig. 12. Request service timeline for SMLA with different rank organizations.

TSVs into groups.) MLR merges both layers into a single rank, while SLR has two ranks.[3] We illustrate the timeline for serving three requests, each of which requires four data transfers through an IO group.

SMLA with Multi-Layer Ranks. MLR is similar to the current rank organization of conventional DRAM, with multiple layers sharing a common command interface. In our example MLR-based memory system (for either Dedicated-IO or Cascaded-IO), the data is transferred through both IO groups, requiring *two* clock cycles to serve a request

[3]In our example, each DRAM layer consists of two banks (four banks in two layers). For MLR, two physical banks in different layers work together, forming a single logical bank (two logical banks in total), while for SLR, the four banks work independently. In the case of MLR, this results in logical banks with double the capacity of the banks in SLR.

(*Request 1* in Figure 12(a)). Note that while both Dedicated-IO and Cascaded-IO have the same timeline, the data being served resides in two different physical locations. In Dedicated-IO, the data for *IO Group 0* comes from one layer (i.e., the bottom layer) and the *IO Group 1* data comes from the other (top) layer. In Cascaded-IO, the first piece of data for both IO groups comes from the bottom layer, while the second piece comes from the top layer. The third request in our sequence has an access to the same bank and rank as the first request. Therefore, there is a delay before serving the third request due to a bank conflict [Kim et al. 2012].

SMLA with Single-Layer Ranks. In a Dedicated-IO-based memory system with SLR (Figure 12(b)), each layer is a rank and has its own IO group. Therefore, a request to a rank transfers data only over its assigned group, requiring four clock cycles to serve a request. On the other hand, Cascaded-IO with SLR (Figure 12(c)) partitions its data transfer cycles in a round-robin fashion, with each layer having an assigned time slot (once every L cycles). In Figure 12(c), *Rank 0* corresponds to the bottom layer, and *Rank 1* to the top layer. We assign the bottom layer (*Rank 0*) requests to the odd time slots, and the top layer (*Rank 1*) requests to the even time slots. In this way, the first and third data bursts represent the first complete request to *Rank 0*. When our third request is serviced, since the Cascaded-IO time slot assignments are fixed, there is a hole in between its data bursts, since *Rank 1* does not have any requests to serve (as our example has only three requests).

Tradeoff: MLR Versus SLR. For the two-layer stack in our example, MLR can fully service an individual request within two cycles, through the use of both IO groups. In contrast, SLR takes up to four cycles to service each request but can deliver *two* requests in that time, since it can overlap latencies for accesses to different ranks. MLR therefore better supports latency-bound applications that do not issue large numbers of memory requests. SLR, on the other hand, is better tuned for memory-level parallelism, as it exposes a greater number of ranks. As we can see in Figure 12, the third request experiences less delay in SLR because the larger number of available ranks reduces the probability of bank conflicts.

While we have considered two extremes of rank organization (one layer per rank vs. using all layers to form a rank), it is possible to pick a design point in the middle (e.g., two ranks in a four-layer memory, with two layers per rank). We leave the evaluation of such organizations as future work.

6. ENERGY CONSUMPTION ANALYSIS

In order to deliver greater memory bandwidth, SMLA increases the memory channel frequency, which can change DRAM energy consumption significantly. In this section, we analyze this energy consumption using energy data from other high-frequency DRAMs. We first determine how each component of the current used within a conventional DRAM (DDR3) scales with clock frequency, and then use these observations to estimate the energy consumption of our proposed designs.[4]

Figure 13 plots the components of current for DDR3 at four different IO frequencies (1066, 1333, 1600, 1866MHz), based on manufacturer specifications [Micron 2014]. Each component is tied to a particular action within DRAM. We group these components into three categories and analyze whether current and IO frequency are coupled for each category. For commands issued in *active mode* (component `IDD1`), including activation, read, and precharge, we observe that cutting the IO frequency in half reduces current (and therefore energy consumption) by 10% to 15%. During *power-down* (components `IDD2P` and `IDD3P`), since the DRAM stops the internal clock to reduce current,

[4]Recent work [David et al. 2011] provides a detailed analysis of the relationship between DRAM energy and frequency.

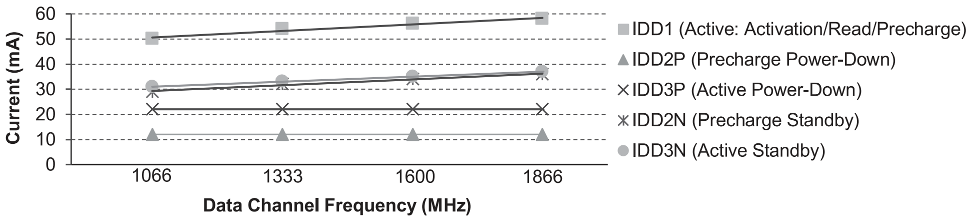


Fig. 13. DDR3 current trend over channel frequency.

we find that the current remains constant despite any change to IO frequency. For *standby mode* (components IDD2N and IDD3N), we find a stronger coupling, as cutting IO frequency in half reduces the current flow by 20% to 30%.

We draw two observations from these results. First, increasing the IO frequency leads to greater current flow (and thus larger energy consumption) within DRAM, except during power-down. Second, the increase in current flow (energy consumption) is approximately linear to the frequency across all of our observed IO frequencies.

We combine our observations with both DRAM energy estimation tools [Rambus 2010; Vogelsang 2010; Micron 2010] and prior measurements of 3D-stacked DRAM energy consumption (operating at 200 and 266MHz) [Chandrasekar et al. 2013], to estimate 3D-stacked DRAM energy consumption across a wide range of IO frequencies (200–1600MHz). To do so, we extract two current sources: (1) current coupled with the clock (standby energy consumption in both active and standby modes) and (2) current decoupled from the clock (power-down current, and constant current consumed by operations such as activation, read, and precharge). Table I lists our estimates of 3D-stacked DRAM energy consumption for various operations, at four IO frequencies. As we expect, a large fraction of energy consumed in standby mode is coupled with the IO frequency. We use these estimates to evaluate overall DRAM energy consumption in Section 8.

7. METHODOLOGY

We use a cycle-accurate DRAM simulator, Ramulator [Kim et al. 2015], which models 3D-stacked memory. The front end of our simulator is based on Pin [Luk et al. 2005; Patil et al. 2004] and models processor cores, caches, and memory controllers.

We use an existing 3D-stacked DRAM, Wide I/O [JEDEC 2011], as our baseline. More recent 3D-stacked DRAM proposals (Wide I/O 2 [JEDEC 2014], HMC [Hybrid Memory Cube Consortium 2013, 2014], HBM [JEDEC 2013a; Lee et al. 2014]) can enable more bandwidth at the expense of greater area and power consumption (as shown in Section 2.4). As we discussed, the key advantage to our mechanisms (Dedicated-IO and Cascaded-IO) is our ability to deliver competitive bandwidth to these high-bandwidth proposals at a much lower cost than these recent proposals (due to the smaller number of global bitlines and sense amplifiers). It is also important to note that simply performing bandwidth scaling (as Wide I/O 2, HMC, and HBM have done) will eventually be constrained by the area and power overhead of the extra global sense amplifiers required. We show that our mechanisms overcome this challenge with a large reduction in global sense amplifier count. SMLA can still be used on top of these more recently proposed DRAM architectures to further increase off-chip bandwidth.

We evaluate our mechanisms for 3D-stacked DRAM with two/four/eight layers, with a corresponding increase in IO frequency of 2/4/8 times, respectively, over conventional 3D-stacked DRAM. We evaluate two rank organizations (Single-Layer Ranks and Multi-Layer Ranks). Table II summarizes the parameters for the baseline 3D-stacked memory system and our proposed memory systems, which, unless otherwise

Table I. Estimated Energy Consumption at Different Frequencies

Data Frequency (MHz)	200	400	800	1,600
Power-Down Current (mA, `IDD2P`)	0.24	0.24	0.24	0.24
Precharge Standby Current (mA, `IDD2N`)	4.24	5.39	6.54	8.84
Active Standby Current (mA, `IDD3N`)	7.33	8.50	9.67	12.0
Active Precharge Energy w/o Standby (nJ, extracted from `IDD1`)	1.36	1.37	1.38	1.41
Read Energy w/o Standby (nJ, extracted from `IDD4R`)	1.93	1.93	1.93	1.93
Write Energy w/o Standby (nJ, extracted from `IDD4W`)	1.33	1.33	1.33	1.33

Table II. 3D-Stacked DRAM Configurations Evaluated

IO Interface	Baseline	Dedicated-IO		Cascaded-IO	
Rank Organization	SLR	MLR	SLR	MLR	SLR
Number of Ranks	4	1	4	1	4
Clock Frequency (MHz)	200	800	800	800	800
Bandwidth (GBps)	3.2	12.8	12.8	12.8	12.8
Data Transfer Time (ns)	20.0	5.0	20.0	5.0	18.1†
No. of Simultaneously Accessible Layers	1	4	4	4	4

Global parameters: 4 layers, 2 banks/rank, 128 IOs/channel, 64 bytes/request.
†Average data transfer time of layers from bottom to top: 16.25ns/17.5ns/18.75ns/20ns.

noted, consist of four stacked layers and operate at 4 times the baseline system's IO frequency for our evaluation.

Table III shows the evaluated system configuration. We use a single-channel memory system for our single-core evaluations, and a four-channel system for multicore evaluations (identical to the configuration used in Wide I/O [JEDEC 2011]). Virtual memory pages are mapped to physical pages randomly across the layers.

We use 31 applications from the SPEC CPU2006 [SPEC 2006], TPC [TPC 2015], and STREAM [McCalpin 2007] application suites. For single-core studies, we report results that are averaged across all applications. For our multicore evaluation, we generate 16 multiprogrammed workloads for each case (four to 16 cores/two to eight stacked layers) by randomly selecting from our workload pool.

We execute all applications for 100 million instructions, similar to many prior works [Stuecheli et al. 2010; Mutlu and Moscibroda 2007; Chou et al. 2004; Kim et al. 2012; Mutlu and Moscibroda 2008; Kim et al. 2010a, 2010b; Lee et al. 2010; Muralidhara et al. 2011] that also studied memory systems. For multicore evaluations, we ensure that even the slowest core executes 100 million instructions, while other cores continue to exert pressure on the memory subsystem. To measure performance, we use instruction throughput (IPC) for single-core systems and *weighted speedup* [Snavely and Tullsen 2000; Eyerman and Eeckhout 2008] for multicore systems.

8. EVALUATION

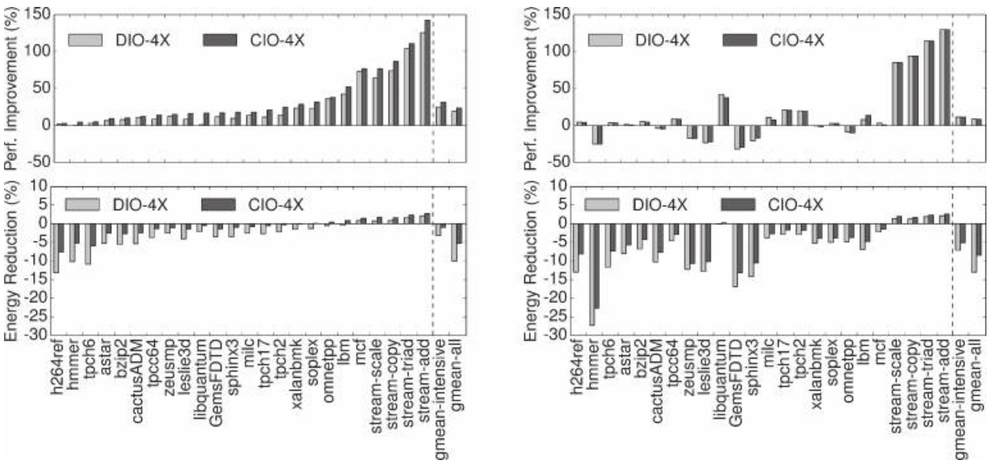
In our experiments for both single-core and multicore systems, we compare three different 3D-stacked DRAM designs: (1) the baseline 3D-stacked DRAM (Baseline SLR in Table II), (2) *SMLA using Dedicated-IO* (DIO), and (3) *SMLA using Cascaded-IO* (CIO), in terms of both performance and DRAM energy consumption.

8.1. Single-Core Results

Figure 14 compares the SMLA-based designs (Dedicated-IO and Cascaded-IO) to the baseline 3D-stacked DRAM. We present only 24 applications whose MPKIs (last-level cache Misses-Per-Kilo-Instructions) are larger than 1. The other benchmarks ($MPKI < 1$) show performance changes of less than 1%. We present both the geometric mean of these 24 memory-intensive workloads (*gmean-intensive*) and the geometric mean of *all 31 workloads* (*gmean-all*).

Table III. Configuration of Simulated System

Processor	1–16 cores, 3.2GHz, 3-wide issue, 8 MSHRs, 128-entry instruction window
Last-Level Cache	64B cache line, 16-way associative, 512KB private cache slice per core
Memory Controller	64/64-entry read/write queues/controller, FR-FCFS scheduler [Rixner et al. 2000]
Memory System	2–8 layer 3D-stacked DRAM, 1–4 channels



(a) Single-Layer Rank (4 ranks per channel) (b) Multi-Layer Rank (1 rank per channel)

Fig. 14. Single-core system performance (top) and DRAM energy reduction (bottom) of SMLA, normalized to the baseline 3D-stacked DRAM system (higher is better).

All three 3D-stacked DRAM designs have four stack layers (see Table II). By accessing those four layers simultaneously, SMLA-based mechanisms provide four times more bandwidth compared to the baseline. Each mechanism was evaluated with two rank organizations from Section 5: Single-Layer Rank (Figure 14(a)), and Multi-Layer Rank (Figure 14(b)). For each application, the figures show two metrics: (1) the performance improvement of SMLA-based mechanisms compared to the baseline and (2) the DRAM energy reduction of SMLA-based mechanisms relative to the baseline. We draw three major conclusions from Figure 14.

System Performance with Rank Organizations. First, *SMLA-based mechanisms provide significant average performance improvement in both rank organizations over the baseline. However, individual applications show different performance benefits depending on the rank organization.*

In SLR (Figure 14(a)), both of our mechanisms improve performance for all memory-intensive applications. The average improvement for memory-intensive applications is 25.2% and 31.6% for Dedicated-IO and Cascaded-IO, respectively (19.2%/23.9% averaged over *all 31 workloads*). Intuitively, performance benefits are higher for applications with higher memory intensity (the MPKI of the seven leftmost applications is less than eight, while the MPKI of the seven rightmost applications is more than 35).

In MLR (Figure 14(b)), while SMLA improves performance for most applications compared to the baseline (by 8.8% on average for both Dedicated-IO and Cascaded-IO, across all 31 workloads), there are a few applications that experience performance degradation. Note that even though both Dedicated-IO and Cascaded-IO enable much higher bandwidth, these mechanisms with MLR have only one rank per channel. This reduces the number of opportunities for rank-level parallelism. As a result, several applications (e.g., `hmmer`, `zeusmp`, `leslie3d`, `GemsFDTD`, `sphinx3`, `omnetpp`) that benefit more from having higher rank-level parallelism (in the baseline) than from having higher bandwidth (in MLR) experience performance loss.

For most applications, SMLA with SLR provides better performance compared to MLR. There are, however, a few noticeable exceptions (e.g., `libquantum`, `h264ref`) where Dedicated-IO and Cascaded-IO show better performance improvement with MLR. These applications are more latency sensitive, and hence they benefit more from the low latency in MLR than from rank-level parallelism in SLR.

System Performance with SMLA Designs. Second, *Cascaded-IO usually provides better performance improvements than Dedicated-IO in SLR*. This is because Cascaded-IO has a lower average latency than Dedicated-IO in SLR. However, in MLR, both mechanisms have the same latency (as shown in Table II). Furthermore, the upper layers in Cascaded-IO operate at a lower frequency, leading to a reduction in the command and address bandwidth for the upper layers, which in turn degrades the latency of some accesses that map to the upper layers. As a result, in MLR, Dedicated-IO shows slightly better performance than Cascaded-IO.

DRAM Energy Consumption. Third, *Cascaded-IO provides lower DRAM energy consumption than Dedicated-IO due to the reduction in the frequency of the third and fourth layers.* However, due to the increase in the average frequency of the layers (and especially for the lower layers in Cascaded-IO), both Dedicated-IO and Cascaded-IO consume more DRAM energy compared to the baseline (10.1%/5.2% more with SLR and 13.1%/8.5% more with MLR, respectively), as seen in Figure 14.

8.2. Multicore Results

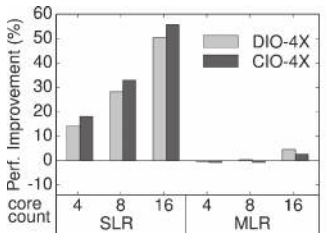
System Performance. Figure 15(a) shows the system performance improvement for our mechanisms in multiprogrammed workloads. Similar to the single-core results with SLR, both Dedicated-IO and Cascaded-IO provide significant performance improvements over the baseline in all system configurations. On average, Dedicated-IO improves performance by 14.4%/28.3%/50.4% for four-/eight-/16-core systems, respectively, while Cascaded-IO improves performance by 18.2%/32.9%/55.8%. As memory bandwidth becomes a bigger bottleneck with more cores, the performance benefits of our mechanisms increase. Due to the reduction in rank-level parallelism, the performance benefits of our mechanisms using MLR are lower than with SLR.

DRAM Energy Consumption. In the multicore case, our mechanisms reduce the DRAM energy consumption significantly using SLR, with respect to the baseline. Despite the additional *power* consumption, SMLA greatly reduces the execution time of our applications, resulting in much lower DRAM *energy* consumption.

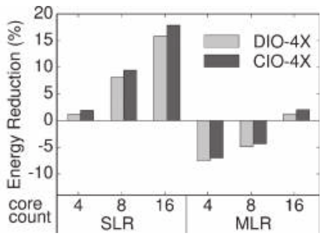
We see that unlike in the single-core case, our mechanisms with SLR actually deliver significant DRAM energy reductions as the number of cores (and therefore memory contention) increases. Figure 15(b) shows that Cascaded-IO with SLR reduces average DRAM energy consumption by 1.9%/9.4%/17.9% for four/eight/16 cores, respectively. This reduction mainly comes from the reduced contention in the memory system, leading to lower overall execution times for our workloads. Unlike SLR, MLR is geared toward latency and not parallelism, and as such is unable to ease the contention pressure as well, resulting in increased DRAM energy consumption. We conclude that SMLA with SLR is very effective at both improving system performance and reducing DRAM energy consumption.

8.3. Sensitivity to the Number of Stack Layers

The maximum bandwidth improvement in our mechanisms depends on the number of stack layers whose global bitline interfaces can operate simultaneously. For example, in



(a) System performance improvement



(b) DRAM energy reduction

Fig. 15. Multicore evaluation (four/eight/16 cores, four layers), normalized to the baseline.

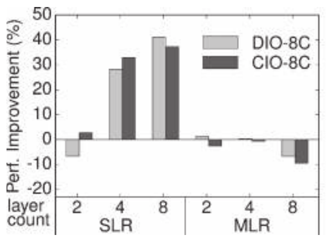
a two-layer stacked memory system, our mechanism can enable twice the bandwidth of the baseline 3D-stacked DRAM (4/8 times the bandwidth for four-/eight-layer DRAM, respectively). Figure 16 shows the system performance improvement and DRAM energy reduction of our mechanisms in two- to eight-layer 3D-stacked DRAM. We use eight-core multiprogrammed workloads, and our evaluated memory system has four channels.

Two observations are in order. First, as expected, the performance improvement and DRAM energy reduction of our mechanisms with SLR grow with the number of stack layers in DRAM. At the same time, we observe fewer performance benefits in eight-layer stacked DRAM with MLR, mainly due to reduction in the number of ranks (one rank in the MLR-based system vs. eight ranks in the baseline system). Second, a 3D-stacked DRAM with more layers (e.g., an eight-layer stacked DRAM) has better performance with Dedicated-IO than with Cascaded-IO, due to the reduced frequency of the higher layers in Cascaded-IO, as this leads to a reduction in the command bandwidth of upper layers. We conclude that the benefits of our mechanisms scale with the number of layers in 3D-stacked DRAM.

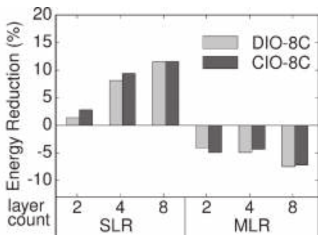
8.4. DRAM Energy Consumption with Varying Memory Intensity

Figure 17 shows the DRAM energy consumption of SMLA as we vary the memory intensity (MPKI) of a microbenchmark, which is similar to GUPS [Univ. of Tennessee 2015], in two different ways. First, we plot the absolute DRAM energy consumed when executing 100 million instructions (Figure 17(a)). As expected, the DRAM energy consumption of all three mechanisms grows with memory intensity (21 times more DRAM energy consumed at 51.2 MPKI than at 0.1 MPKI). Second, we analyze the relative DRAM energy increase when executing the same microbenchmarks (Figure 17(b)). We observe that the amount of DRAM energy increase for SMLA (relative to the baseline) significantly reduces at higher MPKIs. This is because, at high memory intensities, DRAM consumes more energy to serve more memory requests, and the amount of energy consumed by these requests is much less dependent on IO frequency than in low memory intensities. In other words, as much more DRAM energy is consumed by the baseline at higher memory intensities (Figure 17(a)), SMLA's relative energy overhead is much smaller at higher memory intensities (Figure 17(b)). We conclude that SMLA's energy overhead is low for memory-intensive workloads, where memory energy is a more important problem.

Cascaded-IO always consumes less energy than Dedicated-IO for our microbenchmark (about 30% lower), due to the lower energy consumed by the slower upper layers in Cascaded-IO. Based on this analysis, we conclude that (1) SMLA can increase DRAM energy consumption, but this overhead is relatively small compared to the overall DRAM energy when running memory-intensive applications, and is also small

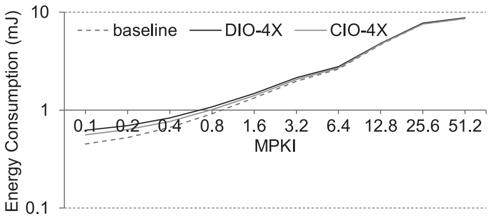


(a) System performance improvement

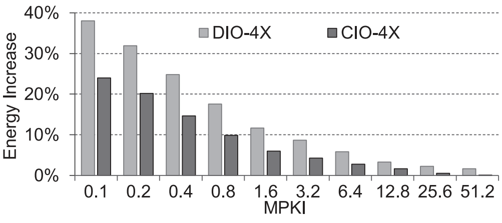


(b) DRAM energy reduction

Fig. 16. System performance and memory energy sensitivity to layer count (eight cores, two/four/eight layers), normalized to the baseline.



(a) DRAM energy consumption vs. MPKI



(b) DRAM energy increase vs. MPKI

Fig. 17. Memory intensity versus DRAM energy consumption with SMLA (over baseline).

compared to the overall DRAM energy when applications are not memory intensive, and (2) Cascaded-IO provides better DRAM energy efficiency than Dedicated-IO.

8.5. Thermal Analysis

A DRAM cell leaks charge over time and loses more charge at higher temperatures [Restle et al. 1992; Yaney et al. 1987; Mori et al. 2005; Khan et al. 2014; Liu et al. 2013]. Therefore, DRAM needs to refresh cells more frequently at higher temperatures, which can potentially lead to some performance loss [Liu et al. 2012, 2013; Chang et al. 2014; Qureshi et al. 2015]. To understand the thermal characteristics of our proposed mechanisms, we generate four thermal models of 3D-stacked DRAM using HotSpot [Huang et al. 2004]: (1) a baseline Wide I/O (Wide-I/O-200, operating at 200MHz), (2) a high-frequency Wide I/O that contains four times as many global sense amplifiers (Wide-I/O-1600, operating at 1600MHz), (3) Cascaded-IO (1600MHz), and (4) Dedicated-IO (1600MHz). Figure 18 compares the operating temperatures of these four models at each DRAM layer, with a workload that exerts full load on the TSV channel by issuing read requests with minimum delay. Compared to the baseline Wide-I/O-200, implementing Wide I/O at higher frequency (Wide-I/O-1600) increases the operating temperature by 13°C, mainly due to the activation of more global sense amplifiers and the increased clock frequency. Dedicated-IO operates at even higher temperatures than Wide-I/O-1600 (by 5.4°C), even though both Wide-I/O-1600 and Dedicated-IO have the same bandwidth and clock frequency. This is because Dedicated-IO simultaneously activates more layers (all four layers) than Wide-I/O-1600 (only one layer), leading to greater standby power consumption. Cascaded-IO operates at a higher operating temperature than Wide-I/O-200, but at a lower temperature than Wide-I/O-1600 and Dedicated-IO, due to the reduced upper layer clock frequencies.

Despite these increased temperatures over Wide I/O, it is important to note that contemporary DRAM specifications are designed to allow normal DRAM operation and

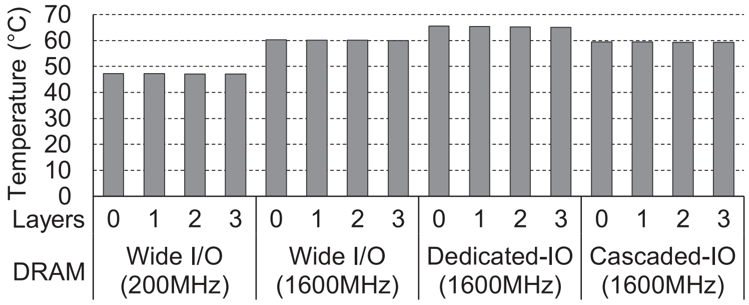


Fig. 18. Temperature analysis (ambient temperature: 45 °C).

refresh interval at temperatures as high as 85°C [Micron 2014; JEDEC 2012a], which is much higher than the temperatures observed for any of these models. Based on this, we conclude that Cascaded-IO achieves both higher bandwidth (than Wide-I/O-200) and lower area cost (than Wide-I/O-1600) at reasonable operating temperatures.

9. RELATED WORK

To our knowledge, this is the first work that (1) enables higher bandwidth at low cost by leveraging the existing global bitline interfaces in multiple layers of 3D-stacked DRAM and (2) provides higher bandwidth with low energy consumption by optimizing IO frequency individually for each layer in 3D-stacked DRAM. In this section, we discuss the prior works that aim to improve memory system bandwidth.

Bank Groups. LPDDR3 [JEDEC 2013b] and DDR4 [JEDEC 2012b] categorize banks into multiple groups (bank groups), where each group has its own set of global sense amplifiers. LPDDR3 and DDR4 increase the internal DRAM bandwidth by simultaneously accessing each bank group. In contrast, our design provides higher bandwidth in 3D-stacked DRAM by aggregating the DRAM internal bandwidth of multiple layers. These two approaches are orthogonal to each other and can be applied together to further increase the bandwidth in 3D-stacked DRAM.

High Bandwidth Memory. HBM [JEDEC 2013a; Lee et al. 2014] enables high bandwidth (128GBps) 3D-stacked DRAMs by (1) adding extra global bitlines per chip (2 times compared to Wide I/O), (2) allowing simultaneous accesses to different bank groups that have their own set of global sense amplifiers (same as LPDDR3 and DDR4 [JEDEC 2012b, 2013b]), and (3) aggregating the bandwidth of each layer by assigning exclusive IO channels to each layer. In contrast, SMLA provides higher bandwidth without adding any extra bitlines, by aggregating data available at different layers and transferring them simultaneously at a higher frequency. HBM can achieve performance and energy efficiency similar to Dedicated-IO, but lower than Cascaded-IO, while having higher cost than our proposals due to its doubling of the global sense amplifiers (as shown in Section 2.4).

Hybrid Memory Cube. The Hybrid Memory Cube (HMC) [Hybrid Memory Cube Consortium 2013, 2014] is a 3D-stacked DRAM that integrates a local memory controller within the stacked memory, a major difference from Wide I/O. So far, we have explained our mechanisms on top of the Wide I/O implementation, but our mechanism can also be integrated easily on top of HMC. HMC consists of a bottom layer that has a memory controller, and many stacked memory layers that contain the DRAM cells. Basically, these upper memory layers are conceptually the same as a Wide I/O chip. As such, our mechanisms can be implemented on top of the existing memory layer TSVs within HMC in the same way that we described in Section 4.

One major benefit of HMC is that it decouples the memory organization from the processor and enables many lower-level optimizations at the local memory controller. Our low-cost high-bandwidth mechanisms, when integrated within HMC, can also benefit from such controller optimizations. For example, data for latency-critical applications can be allocated at lower layers in Cascaded-IO with SLR, as the lower layers have a smaller latency than the upper layers. We believe that future work on such optimizations can yield further performance improvements, building upon the new SMLA substrate we propose (see Section 10).

3D-Stacked DRAM Studies. Prior works on 3D-stacked DRAM focused on utilizing its higher capacity as cache, and as main memory [Black et al. 2006; Loh 2008, 2009; Woo et al. 2010]. None of these works focus on solving the limited internal bandwidth of DRAM in the presence of a wider IO interface.

Multiplexing TSVs. Like our Cascaded-IO design, Tezzaron's DiRAM [Chapman 2013] also uses multiplexers to share TSVs. However, there are key architectural differences. Unlike the more common and widespread 3D-stacked DRAM architectures (e.g., Wide I/O, HBM, HMC), DiRAM removes most of its wordline and bitline decoding logic from the DRAM cell layers, and instead places this logic in the lowermost layer of the chip. To reduce the number of TSVs required for such an architecture, DiRAM performs TSV multiplexing across several bitlines *within a single layer*, in effect performing part of the decoding. In contrast, Cascaded-IO time-multiplexes data *across different 3D layers*, after the peripheral logic inside each layer has performed decoding. As the goal of TSV multiplexing in DiRAM is to reduce TSV count, there is no significant performance benefit from the multiplexing. The multiplexing in Cascaded-IO, in contrast, enables multiple layers in the widespread 3D DRAM architectures to operate simultaneously, leading to higher bandwidth utilization.

Reconfiguring DIMM Organization. Mini-ranks [Zheng et al. 2008] enable independent accesses to subdivided ranks, similar to our mechanisms. However, our mechanisms enable more bandwidth by mitigating the internal bandwidth bottleneck of 3D-stacked DRAM, which is not possible with mini-ranks. By integrating a buffer on DIMM, Decoupled-DIMM [Zheng et al. 2009] enables a higher-frequency memory channel by decoupling the memory channel from DRAM chips, similar to our mechanisms. However, our mechanisms can be implemented with small changes to the 3D-stacked DRAM, while Decoupled-DIMM requires an expensive high-performance buffer. Other recent proposals to change DIMM organization [Malladi et al. 2012; Seshadri et al. 2015] are complementary to our proposals.

Multi-Layer Rank Organization. Loh [2008] introduces a 3D-stacked DRAM that forms a rank over multiple layers. That work assumes that all upper layers only have DRAM cells, while the bottom layer contains control logic, *including both peripheral logic and sense amplifiers*, to access all DRAM cells in the upper layers. In contrast, our mechanisms enable MLR while retaining the architecture of existing 3D-stacked DRAMs (where all layers contain their own peripheral logic and sense amplifiers), by enabling simultaneous multi-layer access through minimal, uniform additions to the control logic.

High-Performance DRAM Architecture. Many previous works introduce new DRAM architectures for either achieving lower DRAM latency [Lee et al. 2013, 2015a; Seshadri et al. 2013; Malladi et al. 2012] or higher parallelism [Kim et al. 2012]. Our mechanisms are orthogonal to these works and can be applied together with them to further increase memory system performance.

10. FUTURE DIRECTIONS

In this work, we propose efficient mechanisms to increase the 3D-stacked DRAM bandwidth by accessing multiple layers simultaneously and by adapting the frequency of layers to the bandwidth requirements. Our mechanisms provide a substrate that can be leveraged in several ways for further performance improvement. We briefly introduce two potential research directions that build upon the SMLA substrate, which we believe are promising for future works to study.

The first direction is *SMLA-aware memory placement and allocation mechanisms*. As we explained, Cascaded-IO (with SLR) provides different access latencies for stacked layers. Hardware (i.e., the memory controller) and software (i.e., the operating system) are currently oblivious to this latency heterogeneity. Either can be modified to allocate data intelligently based on (1) the DRAM layout and (2) the latency criticality of the application/data, leading to improved system performance.

The second direction is *SMLA-aware dynamic energy management*. The upper layers in Cascaded-IO consume less power than the lower layers, resulting in heterogeneity in energy consumption. When systems execute less memory-intensive workloads, or when the memory systems are mostly idle, activating only the upper layer and powering down the other layers can lead to significant energy reduction. In such systems, only the TSV control logic in each layer must still be powered on to allow data to propagate from the upper layer to the processor.

11. CONCLUSION

In this work, we introduce Simultaneous Multi-Layer Access (SMLA), a new IO organization for 3D-stacked DRAM that enables greater off-chip memory bandwidth at low cost over existing 3D-stacked DRAM. We identify that the major bandwidth bottleneck of 3D-stacked DRAM is the costly global bitline interface that is internal to the DRAM chips. We offer a cheaper alternative to adding more global bitlines and sense amplifiers, by instead exploiting the otherwise idle internal global bitline interfaces of multiple DRAM layers in the 3D stack to deliver much greater DRAM bandwidth.

We propose two implementations of SMLA. The first, Dedicated-IO, assigns a subset of the TSVs exclusively to each layer and drives the TSVs at higher frequency, so that all of the DRAM layers can transmit data in parallel. The second, Cascaded-IO, overcomes the shortcomings of Dedicated-IO by time-multiplexing the IO bus across all of the DRAM layers, which reduces the DRAM design effort. Cascaded-IO also adapts the frequency of each layer to its bandwidth requirements, thereby reducing energy consumption at higher layers that transfer less data. We evaluate SMLA over a wide array of applications and show that our proposed mechanisms significantly improve performance and reduce DRAM energy consumption (on average 55%/18%, respectively, for 16-core multiprogrammed workloads) over a baseline 3D-stacked DRAM. We conclude that SMLA provides a high-performance and energy-efficient IO interface for building modern (and future) 3D-stacked memory systems at low cost.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their valuable suggestions. We thank the SAFARI group members for the feedback and the stimulating research environment they provide. An earlier, nonrefereed version of this work can be found at arXiv.org [Lee et al. 2015b].

REFERENCES

Bryan Black, Murali Annavaram, Ned Brekelbaum, John DeVale, Lei Jiang, Gabriel H. Loh, Don McCaule, Pat Morrow, Donald W. Nelson, Daniel Pantuso, Paul Reed, Jeff Rupley, Sadasivan Shankar, John Shen, and Clair Webb. 2006. Die stacking (3D) microarchitecture. In *MICRO*.

Shekhar Borkar. 2007. Thousand core chips: A technology perspective. In *DAC*.

Doug Burger, James R. Goodman, and Alain Kägi. 1996. Memory bandwidth limitations of future microprocessors. In *ISCA*.

Karthik Chandrasekar, Christian Weis, Benny Akesson, Norbert Wehn, and Kees Goossens. 2013. System and circuit level power modeling of energy-efficient 3D-stacked Wide I/O DRAMs. In *DATE*.

Kevin Kai-Wei Chang, Donghyuk Lee, Zeshan Chishti, Alaa R. Alameldeen, Chris Wilkerson, Yoongu Kim, and Onur Mutlu. 2014. Improving DRAM performance by parallelizing refreshes with accesses. In *HPCA*.

David Chapman. 2013. DiRAM architecture overview. In *MemCon*.

Yuan Chou, Brian Fahs, and Santosh Abraham. 2004. Microarchitecture optimizations for exploiting memory-level parallelism. In *ISCA*.

Howard David, Chris Fallin, Eugene Gorbatov, Ulf R. Hanebutte, and Onur Mutlu. 2011. Memory power management via dynamic voltage/frequency scaling. In *ICAC*.

Jeffrey Dean and Luiz André Barroso. 2013. The tail at scale. *Commun. ACM* 56, 2 (2013).

Qingyuan Deng, David Meisner, Luiz Ramos, Thomas F. Wenisch, and Ricardo Bianchini. 2011. MemScale: Active low-power modes for main memory. In *ASPLOS*.

Stijn Eyerman and Lieven Eeckhout. 2008. System-level performance metrics for multiprogram workloads. *IEEE Micro* 28, 3 (2008).

Qawi Harvard and R. Jacob Baker. 2011. A scalable I/O architecture for Wide I/O DRAM. In *MWSCAS*.

Wei Huang, Mircea R. Stan, Kevin Skadron, Karthik Sankaranarayanan, Shougata Ghosh, and Sivakumar Velusam. 2004. Compact thermal modeling for temperature-aware design. In *DAC*.

Cedric Huyghebaert, Jan Van Olmen, Okoro Chukwudi, Jens Coenen, Anne Jourdain, Marc Van Cauwenberghe, Rahul Agarwahl, Alain Phommahaxay, Michele Stucchi, and Philippe Soussan. 2010. Enabling 10μm pitch hybrid Cu-Cu IC stacking with through silicon vias. In *ECTC*.

Hybrid Memory Cube Consortium. 2013. HMC specification 1.1. (2013).

Hybrid Memory Cube Consortium. 2014. HMC specification 2.0. (2014).

ITRS. 2007. *International Technology Roadmap for Semiconductors*.

JEDEC. 2011. Wide I/O Single Data Rate (Wide I/O SDR). Standard No. JESD229. (2011).

JEDEC. 2012a. DDR3 SDRAM. Standard No. JESD79-3F. (2012).

JEDEC. 2012b. DDR4 SDRAM. Standard No. JESD79-4. (2012).

JEDEC. 2013a. High Bandwidth Memory (HBM) DRAM. Standard No. JESD235. (2013).

JEDEC. 2013b. Low Power Double Data Rate 3 (LPDDR3). Standard No. JESD209-3B. (2013).

JEDEC. 2014. Wide I/O 2 (WideIO2). Standard No. JESD229-2. (2014).

Uksong Kang, Hoe-Ju Chung, Seongmoo Heo, Soon-Hong Ahn, Hoon Lee, Soo-Ho Cha, Jaesung Ahn, DukMin Kwon, Jin-Ho Kim, Jae-Wook Lee, Han-Sung Joo, Woo-Seop Kim, Hyun-Kyung Kim, Eun-Mi Lee, So-Ra Kim, Keum-Hee Ma, Dong-Hyun Jang, Nam-Seog Kim, Man-Sik Choi, Sae-Jang Oh, Jung-Bae Lee, Tae-Kyung Jung, Jei-Hwan Yoo, and Changhyun Kim. 2009. 8Gb 3D DDR3 DRAM using through-silicon-via technology. In *ISSCC*.

Brent Keeth, R. Jacob Baker, Brian Johnson, and Feng Lin. 2007. *DRAM Circuit Design. Fundamental and High-Speed Topics*. Wiley-IEEE Press.

Samira Khan, Donghyuk Lee, Yoongu Kim, Alaa R. Alameldeen, Chris Wilkerson, and Onur Mutlu. 2014. The efficacy of error mitigation techniques for DRAM retention failures: A comparative experimental study. In *SIGMETRICS*.

Jung-Sik Kim, Chi Sung Oh, Hocheol Lee, Donghyuk Lee, Hyong-Ryol Hwang, Sooman Hwang, Byongwook Na, Joungwook Moon, Jin-Guk Kim, Hanna Park, Jang-Woo Ryu, Kiwon Park, Sang-Kyu Kang, So-Young Kim, Hoyoung Kim, Jong-Min Bang, Hyunyoon Cho, Minsoo Jang, Cheolmin Han, Jung-Bae Lee, Kyehyun Kyung, Joo-Sun Choi, and Young-Hyun Jun. 2011. A 1.2V 12.8GB/s 2Gb mobile Wide-I/O DRAM with 4x128 I/Os using TSV-based stacking. In *ISSCC*.

Yoongu Kim, Dongsu Han, Onur Mutlu, and Mor Harchol-Balter. 2010a. ATLAS: A scalable and high-performance scheduling algorithm for multiple memory controllers. In *HPCA*.

Yoongu Kim, Michael Papamichael, Onur Mutlu, and Mor Harchol-Balter. 2010b. Thread cluster memory scheduling: Exploiting differences in memory access behavior. In *MICRO*.

Yoongu Kim, Vivek Seshadri, Donghyuk Lee, Jamie Liu, and Onur Mutlu. 2012. A case for exploiting subarray-level parallelism (SALP) in DRAM. In *ISCA*.

Yoongu Kim, Weikun Yang, and Onur Mutlu. 2015. Ramulator: A fast and extensible dram simulator. *IEEE CAL* (2015).

Patrik Larsson. 1996. High-speed architecture for a programmable frequency divider and a dual-modulus prescaler. *IEEE J. Solid-State Circuits* 31, 5 (1996).

Chang Joo Lee, Veynu Narasiman, Eiman Ebrahimi, Onur Mutlu, and Yale N. Patt. 2010. *DRAM-Aware Last-Level Cache Writeback: Reducing Write-Caused Interference in Memory Systems*. Technical Report TR-HPS-2010-007. UT Austin HPS Group.

Donghyuk Lee, Yoongu Kim, Gennady Pekhimenko, Samira Khan, Vivek Seshadri, Kevin Chang, and Onur Mutlu. 2015a. Adaptive-latency DRAM: Optimizing dram timing for the common-case. In *HPCA*.

Donghyuk Lee, Yoongu Kim, Vivek Seshadri, Jamie Liu, Lavanya Subramanian, and Onur Mutlu. 2013. Tiered-latency DRAM: A low latency and low cost dram architecture. In *HPCA*.

Donghyuk Lee, Gennady Pekhimenko, Samira Khan, Saugata Ghose, and Onur Mutlu. 2015b. Simultaneous multi layer access: A high bandwidth and low cost 3D-stacked memory interface. *CoRR* abs/1506.03160 (2015b).

Dong Uk Lee, Kyung Whan Kim, Kwan Weon Kim, Hongjung Kim, Ju Young Kim, Young Jun Park, Jae Hwan Kim, Dae Suk Kim, Heat Bit Park, Jin Wook Shin, Jang Hwan Cho, Ki Hun Kwon, Min Jeong Kim, Jaejin Lee, Kun Woo Park, Byongtae Chung, and Sungjoo Hong. 2014. 25.2 A 1.2V 8Gb 8-channel 128GB/s high-bandwidth memory (HBM) stacked DRAM with effective microbump I/O test methods using 29nm process and TSV. In *ISSCC*.

Jamie Liu, Ben Jaiyen, Yoongu Kim, Chris Wilkerson, and Onur Mutlu. 2013. An experimental study of data retention behavior in modern DRAM devices: Implications for retention time profiling mechanisms. In *ISCA*.

Jamie Liu, Ben Jaiyen, Richard Veras, and Onur Mutlu. 2012. RAIDR: Retention-aware intelligent DRAM refresh. In *ISCA*.

Gabriel H. Loh. 2008. 3D-stacked memory architectures for multi-core processors. In *ISCA*.

Gabriel H. Loh. 2009. Extending the effectiveness of 3D-stacked DRAM caches with an adaptive multi-queue policy. In *MICRO*.

Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood. 2005. Pin: Building customized program analysis tools with dynamic instrumentation. In *PLDI*.

Krishna T. Malladi, Frank A. Nothaft, Karthika Periyathambi, Benjamin C. Lee, Christos Kozyrakis, and Mark Horowitz. 2012. Towards energy-proportional datacenter memory with mobile DRAM. In *ISCA*.

John D. McCalpin. 2007. The STREAM benchmark. Retrieved from http://www.streambench.org.

Micron. 2010. DDR3 SDRAM system-power calculator. Retrieved from http://www.micron.com/support/dram/power_calc/.

Micron. 2014. 2Gb: x4, x8, x16 DDR3 SDRAM. Retrieved from http://www.micron.com∼/media/documents/products/data-sheet/dram/ddr3/2gb_ddr3_sdram.pdf.

Yuki Mori, Kiyonori Ohyu, Kensuke Okonogi, and Ren-Ichi Yamada. 2005. The origin of variable retention time in DRAM. In *IEDM*.

Sai Prashanth Muralidhara, Lavanya Subramanian, Onur Mutlu, Mahmut Kandemir, and Thomas Moscibroda. 2011. Reducing memory interference in multicore systems via application-aware memory channel partitioning. In *MICRO*.

Onur Mutlu. 2013. Memory scaling: A systems architecture perspective. In *IMW*.

Onur Mutlu and Thomas Moscibroda. 2007. Stall-time fair memory access scheduling for chip multiprocessors. In *MICRO*.

Onur Mutlu and Thomas Moscibroda. 2008. Parallelism-aware batch scheduling: Enhancing both performance and fairness of shared DRAM systems. In *ISCA*.

Onur Mutlu and Lavanya Subramanian. 2014. Research problems and opportunities in memory systems. *SUPERFRI* 1, 3 (2014).

Ilya I. Novof, John Austin, Ram Kelkar, Don Strayer, and Steve Wyatt. 1995. Fully integrated CMOS phase-locked loop with 15 to 240 MHz locking range and 50ps jitter. *IEEE J. Solid-State Circuits* 30, 11 (1995).

Reum Oh, Byunghyun Lee, Sang-Woong Shin, Wonil Bae, Hundai Choi, Indal Song, Yun-Sang Lee, Jung-Hwan Choi, Chi-Wook Kim, Seong-Jin Jang, and Joo Sun Choi. 2014. Design technologies for a 1.2V 2.4Gb/s/pin high capacity DDR4 SDRAM with TSVs. In *VLSIC*.

Harish Patil, Robert Cohn, Mark Charney, Rajiv Kapoor, Andrew Sun, and Anand Karunanidhi. 2004. Pinpointing representative portions of large Intel itanium programs with dynamic instrumentation. In *MICRO*.

Moinuddin K. Qureshi, Dae-Hyun Kim, Samira Khan, Prashant J. Nair, and Onur Mutlu. 2015. AVATAR: A variable-retention-time (VRT) aware refresh for DRAM systems. In *DSN*.

Rambus. 2010. DRAM power model. Retrieved from http://www.rambus.com/energy.

Behzad Razavi. 2000. *Design of Analog CMOS Integrated Circuits*. McGraw-Hill.

Phillip J. Restle, J. W. Park, and Brian F. Lloyd. 1992. DRAM variable retention time. In *IEDM*.

Scott Rixner, William J. Dally, Ujval J. Kapasi, Peter Mattson, and John D. Owens. 2000. Memory access scheduling. In *ISCA*.

Brian M. Rogers, Anil Krishna, Gordon B. Bell, Ken Vu, Xiaowei Jiang, and Yan Solihin. 2009. Scaling the bandwidth wall: Challenges in and avenues for CMP scaling. In *ISCA*.

Vivek Seshadri, Yoongu Kim, Chris Fallin, Donghyuk Lee, Rachata Ausavarungnirun, Gennady Pekhimenko, Yixin Luo, Onur Mutlu, Phillip B. Gibbons, Michael A. Kozuch, and Todd C. Mowry. 2013. RowClone: Fast and energy-efficient in-DRAM bulk data copy and initialization. In *MICRO*.

Vivek Seshadri, Thomas Mullins, Amirali Boroumand, Onur Mutlu, Phillip B. Gibbons, Michael A. Kozuch, and Todd C. Mowry. 2015. Gather-scatter DRAM: In-DRAM address translation to improve the spatial locality of non-unit strided accesses. In *MICRO*.

Manjunath Shevgoor, Jung-Sik Kim, Niladrish Chatterjee, Rajeev Balasubramonian, Al Davis, and Aniruddha N. Udipi. 2013. Quantifying the relationship between the power delivery network and architectural policies in a 3D-stacked memory device. In *MICRO*.

Kenneth C. Smith, Alice Wang, and Laura C. Fujino. 2012. Through the looking glass: Trend tracking for ISSCC 2012. In *IEEE Solid-State Circuits Mag.* 4 (2012).

Allan Snavely and Dean M. Tullsen. 2000. Symbiotic jobscheduling for a simultaneous multithreaded processor. In *ASPLOS*.

SPEC. 2006. SPEC CPU2006 benchmark suite. Retrieved from http://www.spec.org/spec2006.

Jeffrey Stuecheli, Dimitris Kaseridis, David Daly, Hillery C. Hunter, and Lizy K. John. 2010. The virtual write queue: Coordinating DRAM and last-level cache policies. In *ISCA*.

TPC. 2015. TPC benchmarks. Retrieved from http://www.tpc.org.

Univ. of Tennessee. 2015. HPC challenge: GUPS. Retrieved from http://icl.cs.utk.edu/projectsfiles/hpcc/RandomAccess.

Thomas Vogelsang. 2010. Understanding the energy consumption of dynamic random access memories. In *MICRO*.

Jeff West, Youn Sung Choi, and Catherine Vartuli. 2012. Practical implications of via-middle Cu TSV-induced stress in a 28nm CMOS technology for Wide-IO logic-memory interconnect. In *VLSIT*.

Dong Hyuk Woo, Nak Hee Seong, Dean L. Lewis, and Hsien-Hsin S. Lee. 2010. An optimized 3D-stacked memory architecture by exploiting excessive, high-density TSV bandwidth. In *HPCA*.

David S. Yaney, Chih-Yuan Lu, Ross A. Kohler, Michael J. Kelly, and James T. Nelson. 1987. A meta-stable leakage phenomenon in DRAM charge storage - variable hold time. In *IEDM*.

Hongzhong Zheng, Jiang Lin, Zhao Zhang, Eugene Gorbatov, Howard David, and Zhichun Zhu. 2008. Mini-rank: Adaptive DRAM architecture for improving memory power efficiency. In *MICRO*.

Hongzhong Zheng, Jiang Lin, Zhao Zhang, and Zhichun Zhu. 2009. Decoupled DIMM: Building high-bandwidth memory system using low-speed DRAM devices. In *ISCA*.

Received June 2015; revised September 2015; accepted September 2015