Low-Cost Inter-Linked Subarrays (LISA): Enabling Fast Inter-Subarray Data Movement in DRAM

Kevin K. Chang†, Prashant J. Nair⋆, Donghyuk Lee†, Saugata Ghose†, Moinuddin K. Qureshi⋆, and Onur Mutlu†

†*Carnegie Mellon University* ⋆*Georgia Institute of Technology*

ABSTRACT

This paper introduces a new DRAM design that enables fast and energy-efficient bulk data movement across subarrays in a DRAM chip. While bulk data movement is a key operation in many applications and operating systems, contemporary systems perform this movement inefficiently, by transferring data from DRAM to the processor, and then back to DRAM, across a narrow off-chip channel. The use of this narrow channel for bulk data movement results in high latency and energy consumption. Prior work proposed to avoid these high costs by exploiting the existing *wide internal DRAM bandwidth for bulk data movement, but the limited connectivity of wires within DRAM allows fast data movement within only a single DRAM subarray. Each subarray is only a few megabytes in size, greatly restricting the range over which fast bulk data movement can happen within DRAM.*

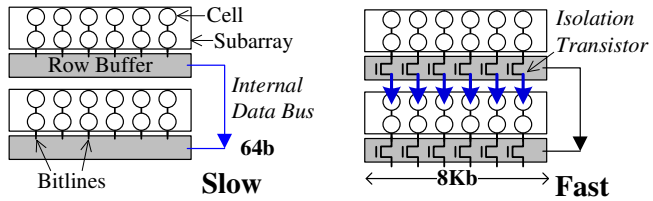
We propose a new DRAM substrate, Low-Cost Inter-Linked Subarrays (LISA), whose goal is to enable fast and efficient data movement across a large range of memory at low cost. LISA adds low-cost connections between *adjacent subarrays. By using these connections to interconnect the existing internal wires (*bitlines*) of adjacent subarrays, LISA enables wide-bandwidth data transfer across multiple subarrays with little (only 0.8%) DRAM area overhead. As a DRAM substrate, LISA is versatile, enabling an array of new applications. We describe and evaluate three such applications in detail: (1) fast inter-subarray bulk data copy, (2) in-DRAM caching using a DRAM architecture whose rows have heterogeneous access latencies, and (3) accelerated bitline precharging by linking multiple precharge units together. Our extensive evaluations show that each of LISA's three applications significantly improves performance and memory energy efficiency, and their combined benefit is higher than the benefit of each alone, on a variety of workloads and system configurations.*

1. Introduction

Bulk data movement, the movement of thousands or millions of bytes between two memory locations, is a common operation performed by an increasing number of real-world applications (e.g., [28, 40, 56, 63, 67, 69, 71, 74, 83]). Therefore, it has been the target of several architectural optimizations (e.g., [1, 26, 69, 80, 83]). In fact, bulk data movement is important enough that modern commercial processors are adding specialized support to improve its performance, such as the ERMSB instruction recently added to the x86 ISA [17].

In today's systems, to perform a bulk data movement between two locations in memory, the data needs to go through the processor *even though both the source and destination are within memory*. To perform the movement, the data is first read out one cache line at a time from the source location in memory into the processor caches, over a pin-limited off-chip channel (typically 64 bits wide). Then, the data is written back to memory, again one cache line at a time over the pin-limited channel, into the destination location. By going through the processor, this data movement incurs a significant penalty in terms of latency and energy consumption.

To address the inefficiencies of traversing the pin-limited channel, a number of mechanisms have been proposed to accelerate bulk data movement (e.g., [26, 45, 69, 83]). The state-of-the-art mechanism, RowClone [69], performs data movement *completely within a DRAM chip*, avoiding costly data transfers over the pin-limited memory channel. However, its effectiveness is limited because RowClone can enable *fast* data movement *only* when the source and destination are within the same DRAM *subarray*. A DRAM chip is divided into multiple *banks* (typically 8), each of which is further split into many *subarrays* (16 to 64) [36], shown in Figure 1a, to ensure reasonable read and write latencies at high density [4, 22, 24, 36, 77]. Each subarray is a two-dimensional array with hundreds of rows of DRAM cells, and contains only a few megabytes of data (e.g., 4MB in a rank of eight 1Gb DDR3 DRAM chips with 32 subarrays per bank). While two DRAM rows in the *same* subarray are connected via a wide (e.g., 8K bits) bitline interface, rows in *different* subarrays are connected via only a *narrow 64-bit data bus* within the DRAM chip (Figure 1a). Therefore, even for previously-proposed in-DRAM data movement mechanisms such as RowClone [69], *inter-subarray* bulk data movement incurs long latency and high memory energy consumption even though data does not move out of the DRAM chip.



(a) RowClone [69] (b) LISA

Figure 1: Transferring data between subarrays using the internal data bus takes a long time in state-of-the-art DRAM design, RowClone [69] (a). Our work, LISA, enables fast inter-subarray data movement with a low-cost substrate (b).

While it is clear that fast *inter-subarray* data movement can have several applications that improve system performance and memory energy efficiency [28, 56, 63, 67, 69, 83], there is currently no mechanism that performs such data movement quickly and efficiently. This is because *no wide datapath exists today between subarrays* within the same bank (i.e., the connectivity of subarrays is low in modern DRAM). **Our goal** is to design a low-cost DRAM substrate that enables fast and energy-efficient data movement *across subarrays*.

978-1-4673-9211-2/16/$31.00 ©2016 IEEE

We make two key observations that allow us to improve the connectivity of subarrays within each bank in modern DRAM. First, accessing data in DRAM causes the transfer of an entire row of DRAM cells to a buffer (i.e., the *row buffer*, where the row data temporarily resides while it is read or written) via the subarray's *bitlines*. Each bitline connects a column of cells to the row buffer, interconnecting every row within the same subarray (Figure 1a). Therefore, the bitlines essentially serve as a very wide bus that transfers a *row's worth of data* (e.g., 8K bits) at once. Second, subarrays within the same bank are placed in close proximity to each other. Thus, the bitlines of a subarray are very close to (but are not currently connected to) the bitlines of neighboring subarrays (as shown in Figure 1a).

Key Idea. Based on these two observations, we introduce a new DRAM substrate, called *Low-cost Inter-linked SubArrays* (*LISA*). LISA enables *low-latency, high-bandwidth inter-subarray connectivity* by linking neighboring subarrays' bitlines together with *isolation transistors*, as illustrated in Figure 1b. We use the new inter-subarray connection in LISA to develop a new DRAM operation, *row buffer movement (RBM)*, which moves data that is latched in an activated row buffer in one subarray into an inactive row buffer in another subarray, without having to send data through the narrow internal data bus in DRAM. RBM exploits the fact that the activated row buffer has enough drive strength to induce charge perturbation within the idle (i.e., *precharged*) bitlines of neighboring subarrays, allowing the destination row buffer to sense and latch this data when the isolation transistors are enabled.

By using a rigorous DRAM circuit model that conforms to the JEDEC standards [22] and ITRS specifications [18, 19], we show that RBM performs inter-subarray data movement at 26x the bandwidth of a modern 64-bit DDR4-2400 memory channel (500 GB/s vs. 19.2 GB/s; see §3.3), even after we conservatively add a large (60%) timing margin to account for process and temperature variation.

Applications of LISA. We exploit LISA's *fast inter-subarray movement* to enable many applications that can improve system performance and energy efficiency. We implement and evaluate the following three applications of LISA:

- **Bulk data copying.** Fast inter-subarray data movement can eliminate long data movement latencies for copies between two locations in the same DRAM chip. Prior work showed that such copy operations are widely used in today's operating systems [56, 63] and datacenters [28]. We propose Rapid Inter-Subarray Copy (*RISC*), a new bulk data copying mechanism based on LISA's RBM operation, to reduce the latency and DRAM energy of an inter-subarray copy by 9.2x and 48.1x, respectively, over the best previous mechanism, RowClone [69] (§4).
- **Enabling access latency heterogeneity within DRAM.** Prior works [40, 71] introduced non-uniform access latencies within DRAM, and harnessed this heterogeneity to provide a data caching mechanism within DRAM for hot (i.e., frequently-accessed) pages. However, these works do not achieve either one of the following goals: (1) low area overhead, and (2) fast data movement from the slow portion of DRAM to the fast portion. By exploiting the LISA substrate, we propose a new DRAM design, VarIabLe LAtency (*VILLA*) DRAM, with asymmetric subarrays that reduce the access latency to hot rows by up to 63%, delivering high system performance and achieving both goals of low overhead and fast data movement (§5).
- **Reducing precharge latency.** Precharge is the process of preparing the subarray for the next memory access [22, 36, 39, 40]. It incurs latency that is on the critical path of a bank-conflict memory access. The precharge latency of a subarray is limited by the drive strength of the precharge unit attached to its row buffer. We demonstrate that LISA enables a new mechanism, LInked Precharge (*LIP*), which connects a subarray's precharge unit with the idle precharge units in the neighboring subarrays, thereby accelerating precharge and reducing its latency by 2.6x (§6).

These three mechanisms are complementary to each other, and we show that when combined, they provide additive system performance and energy efficiency improvements (§9.4). LISA is a versatile DRAM substrate, capable of supporting several other applications beyond these three, such as performing efficient data remapping to avoid conflicts in systems that support subarray-level parallelism [36], and improving the efficiency of bulk bitwise operations in DRAM [67] (see §10).

This paper makes the following major contributions:

- We propose a new DRAM substrate, *Low-cost Inter-linked SubArrays* (*LISA*), which provides high-bandwidth connectivity between subarrays within the same bank to support bulk data movement at low latency, energy, and cost (§3).
- We propose and evaluate three new applications that take advantage of LISA: (1) Rapid Inter-Subarray Copy (*RISC*), which copies data across subarrays at low latency and low DRAM energy; (2) Variable Latency (*VILLA*) DRAM, which reduces the access latency of hot data by caching it in fast subarrays; and (3) Linked Precharge (*LIP*), which reduces the precharge latency for a subarray by linking its precharge units with neighboring idle precharge units.
- We extensively evaluate LISA's applications individually and combined together. Our evaluation shows that (1) RISC improves average system performance by 66% over workloads that perform intensive bulk data movement and (2) VILLA/LIP improve performance by 5%/8% over a wide variety of workloads. Combining all three applications improves system performance by 94% and reduces memory energy by 49% on a 4-core system running workloads with intensive bulk data movement (§9.4).

2. Background: DRAM Organization

A modern DRAM system consists of a hierarchy of components: channels, ranks, banks, and subarrays. A memory channel drives DRAM commands, addresses, and data between a memory controller and a group of DRAM ranks. Within a rank, there are multiple banks that can serve memory requests (i.e., reads or writes) concurrently, independent of one another.[1]

[1]Physically, a rank consists of multiple DRAM chips. Every chip in a rank operates in lockstep to serve fragments of data for the same request. Many prior works provide further details on DRAM rank organization [77, 82].

2.1. DRAM Subarrays

In this work, we focus on operations across subarrays within the *same* bank. Typically, a bank is subdivided into multiple *subarrays* [4, 36, 69, 79], as shown in Figure 2. Each subarray consists of a 2D-array of DRAM cells that are connected to sense amplifiers through *bitlines*. Because the size of a sense amplifier is more than 100x the size of a cell [40], modern DRAM designs fit in only enough sense amplifiers in a row to sense *half a row of cells*. To sense the *entire* row of cells, each subarray has bitlines that connect to *two rows* of sense amplifiers — one above and one below the cell array (① and ② in Figure 2, for Subarray 1). This DRAM design is known as the *open bitline architecture*, and is commonly used to achieve high-density DRAM [42, 75]. For the rest of the paper, we refer to a single row of sense amplifiers, which holds the data from *half* a row of activated cells, as a *row buffer*.

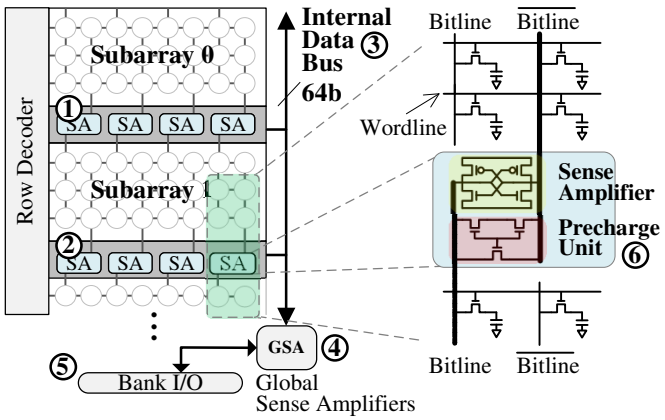


Figure 2: Bank and subarray organization in a DRAM chip.

2.2. DRAM Subarray Operation

Accessing data in a subarray requires two steps. The DRAM row (typically 8KB across a rank of eight x8 chips) must first be *activated*. Only after activation completes, a column command (i.e., a `READ`/`WRITE`) can operate on a piece of data (typically 64B across a rank; the size of a single cache line) from that row.

When an `ACTIVATE` command with a row address is issued, the data stored within a row in a subarray is read by *two* row buffers (i.e., the row buffer at the top of the subarray ① and the one at the bottom ②). First, a wordline corresponding to the row address is selected by the subarray's row decoder. Then, the top row buffer and the bottom row buffer each sense the charge stored in half of the row's cells through the bitlines, and amplify the charge to full digital logic values (0 or 1) to latch in the cells' data.

After an `ACTIVATE` finishes latching a row of cells into the row buffers, a `READ` or a `WRITE` can be issued. Because a typical read/write memory request is made at the granularity of a single cache line, only a subset of bits are selected from a subarray's row buffer by the column decoder. On a `READ`, the selected column bits are sent to the global sense amplifiers through the *internal data bus* (also known as the global data lines) ③, which has a narrow width of 64B across a rank of eight chips. The global sense amplifiers ④ then drive the data to the bank I/O logic ⑤, which sends the data out of the DRAM chip to the memory controller.

While the row is activated, a consecutive column command to the same row can access the data from the row buffer without performing an additional `ACTIVATE`. This is called a *row buffer hit*. In order to access a different row, a `PRECHARGE` command is required to reinitialize the bitlines' values for another `ACTIVATE`. This re-initialization process is completed by a set of *precharge units* ⑥ in the row buffer. For more detail on DRAM commands and internal DRAM operation, we refer the reader to prior works [36, 39, 40, 44, 69, 71].

3. Low-Cost Inter-Linked Subarrays (LISA)

We propose a new DRAM substrate, LISA, which enables fast and energy-efficient data movement across subarrays within a DRAM chip. First, we discuss the low-cost design changes to DRAM to enable high-bandwidth connectivity across neighboring subarrays (Section 3.1). We then introduce a new DRAM command that uses this new connectivity to perform bulk data movement (Section 3.2). Finally, we conduct circuit-level studies to determine the latency of this command (Sections 3.3 and 3.4).

3.1. LISA Design

LISA is built upon two key characteristics of DRAM. First, large data bandwidth *within* a subarray is already available in today's DRAM chips. A row activation transfers an entire DRAM row (e.g., 8KB across all chips in a rank) into the row buffer via the bitlines of the subarray. These bitlines essentially serve as a wide bus that transfers an entire row of data in parallel to the respective subarray's row buffer. Second, every subarray has its own set of bitlines, and subarrays within the same bank are placed in close proximity to each other. Therefore, a subarray's bitlines are very close to its neighboring subarrays' bitlines, although these bitlines are *not* directly connected together.[2]

By leveraging these two characteristics, we propose to *build a wide connection path between subarrays* within the same bank at low cost, to overcome the problem of a narrow connection path between subarrays in commodity DRAM chips (i.e., the internal data bus ③ in Figure 2). Figure 3 shows the subarray structures in LISA. To form a new, low-cost inter-subarray datapath with the same wide bandwidth that already exists inside a subarray, we join neighboring subarrays' bitlines together using *isolation transistors*. We call each of these isolation transistors a *link*. A link connects the bitlines for the same column of two adjacent subarrays.

When the isolation transistor is turned on (i.e., the link is enabled), the bitlines of two adjacent subarrays are connected. Thus, the sense amplifier of a subarray that has already driven its bitlines (due to an `ACTIVATE`) can also drive its neighboring subarray's precharged bitlines through the enabled link. This causes the neighboring sense amplifiers to sense the charge difference, and simultaneously help drive both sets of bitlines. When the isolation transistor is turned off (i.e., the link is

[2]Note that matching the bitline pitch across subarrays is important for a high-yield DRAM process [42, 75].

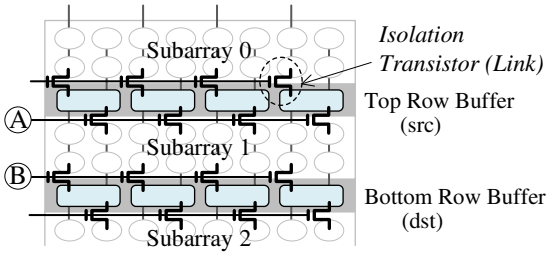


Figure 3: Inter-linked subarrays in LISA.

disabled), the neighboring subarrays are disconnected from each other and thus operate as in conventional DRAM.

3.2. Row Buffer Movement (RBM) Through LISA

Now that we have inserted physical links to provide high-bandwidth connections across subarrays, we must provide a way for the memory controller to make use of these new connections. Therefore, we introduce a new DRAM command, RBM, which triggers an operation to move data from one row buffer (half a row of data) to another row buffer within the same bank through these links. This operation serves as the building block for our architectural optimizations.

To help explain the RBM process between two row buffers, we assume that the *top row buffer* and the *bottom row buffer* in Figure 3 are the source (`src`) and destination (`dst`) of an example RBM operation, respectively, and that `src` is activated with the content of a row from Subarray 0. To perform this RBM, the memory controller enables the links (Ⓐ and Ⓑ) between `src` and `dst`, thereby connecting the two row buffers' bitlines together (*bitline* of `src` to *bitline* of `dst`, and $\overline{bitline}$ of `src` to $\overline{bitline}$ of `dst`).

Figure 4 illustrates how RBM drives the data from `src` to `dst`. For clarity, we show only one column from each row buffer. State ① shows the initial values of the bitlines (BL and $\overline{BL}$) attached to the row buffers — `src` is activated and has fully driven its bitlines (indicated by thick bitlines), and `dst` is in the precharged state (thin bitlines indicating a voltage state of $V_{DD}/2$). In state ②, the links between `src` and `dst` are turned *on*. The charge of the `src` bitline (BL) flows to the connected bitline (BL) of `dst`, raising the voltage level of `dst`'s BL to $V_{DD}/2 + \Delta$. The other bitlines ($\overline{BL}$) have the opposite charge flow direction, where the charge flows from the $\overline{BL}$ of `dst` to the $\overline{BL}$ of `src`. This phase of charge flowing between the bitlines is known as *charge sharing*. It triggers `dst`'s row buffer to sense the increase of differential voltage between BL and $\overline{BL}$, and amplify the voltage difference further. As a result, *both* `src` and `dst` start driving the bitlines with the same values. This *double sense amplification* process pushes both sets of bitlines to reach the final *fully sensed* state (③), thus completing the RBM from `src` to `dst`.

Extending this process, RBM can move data between two row buffers that are *not* adjacent to each other as well. For example, RBM can move data from the `src` row buffer (in Figure 3) to a row buffer, `dst2`, that is *two* subarrays away (i.e., the bottom row buffer of Subarray 2, not shown in Figure 3). This operation is similar to the movement shown in Figure 4, except that the RBM command turns on *two extra links* (Ł ②

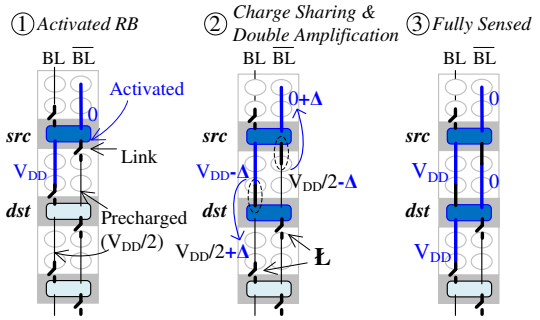


Figure 4: Row buffer movement process using LISA.

in Figure 4), which connect the bitlines of `dst` to the bitlines of `dst2`, in state ②. By enabling RBM to perform row buffer movement across non-adjacent subarrays via a single command, instead of requiring multiple commands, the movement latency and command bandwidth are reduced.

3.3. Row Buffer Movement (RBM) Latency

To validate the RBM process over LISA links and evaluate its latency, we build a model of LISA using the Spectre Circuit Simulator [2], with the NCSU FreePDK 45nm library [54]. We configure the DRAM using the JEDEC DDR3-1600 timings [22], and attach each bitline to 512 DRAM cells [40, 71]. We conservatively perform our evaluations using *worst-case cells*, with the resistance and capacitance parameters specified in the ITRS reports [18, 19] for the metal lanes. Furthermore, we conservatively model the worst RC drop (and hence latency) by evaluating cells located at the edges of subarrays.

We now analyze the process of using one RBM operation to move data between *two non-adjacent row buffers* that are two subarrays apart. To help the explanation, we use an example that performs RBM from `RB0` to `RB2`, as shown on the left side of Figure 5. The right side of the figure shows the voltage of a single bitline BL from each subarray during the RBM process over time. The voltage of the $\overline{BL}$ bitlines show the same behavior, but have inverted values. We now explain this RBM process step by step.

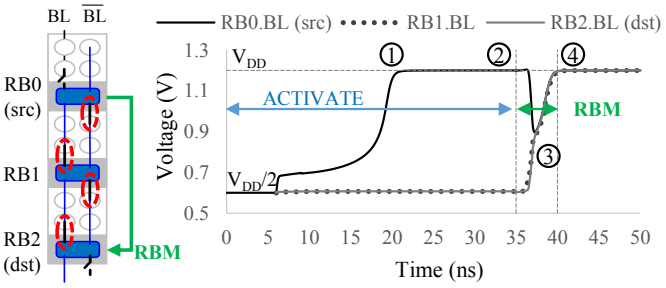


Figure 5: SPICE simulation results for transferring data across two subarrays with LISA.

First, before the RBM command is issued, an `ACTIVATE` command is sent to `RB0` at time 0. After roughly 21ns (①), the bitline reaches V_{DD}, which indicates the cells have been fully restored (tRAS). Note that, in our simulation, restoration happens more quickly than the standard-specified tRAS value of 35ns, as the standard includes a guardband on top of the typical cell restoration time to account for process and temperature

variation [3, 39]. This amount of margin is on par with values experimentally observed in commodity DRAMs at 55°C [39].

Second, at 35ns (②), the memory controller sends the RBM command to move data from RB0 to RB2. RBM simultaneously turns on the *four* links (circled on the left in Figure 5) that connect the subarrays' bitlines.

Third, after a small amount of time (③), the voltage of RB0's bitline drops to about 0.9V, as the fully-driven bitlines of RB0 are now charge sharing with the precharged bitlines attached to RB1 and RB2. This causes both RB1 and RB2 to sense the charge difference and start amplifying the bitline values. Finally, after amplifying the bitlines for a few nanoseconds (④ at 40ns), all three bitlines become fully driven with the value that is originally stored in RB0.

We thus demonstrate that RBM moves data from one row buffer to a row buffer two subarrays away at very low latency. Our SPICE simulation shows that the RBM latency across two LISA links is approximately 5ns (② → ④). To be conservative, we do *not* allow data movement across more than two subarrays with a single RBM command.[3]

3.4. Handling Process and Temperature Variation

On top of using worst-case cells in our SPICE model, we add in a latency guardband to the RBM latency to account for process and temperature variation, as DRAM manufacturers commonly do [3, 39]. For instance, the ACTIVATE timing (tRCD) has been observed to have margins of 13.3% [3] and 17.3% [39] for different types of commodity DRAMs. To conservatively account for process and temperature variation in LISA, we add a large timing margin, of *60%*, to the RBM latency. Even then, RBM latency is 8ns and RBM provides a 500 GB/s data transfer bandwidth across two subarrays that are one subarray apart from each other, which is 26x the bandwidth of a DDR4-2400 DRAM channel (19.2 GB/s) [24].

4. Application 1: Rapid Inter-Subarray Bulk Data Copying (LISA-RISC)

Due to the narrow memory channel width, bulk copy operations used by applications and operating systems are performance limiters in today's systems [26, 28, 69, 83]. These operations are commonly performed due to the memcpy and memmov. Recent work reported that these two operations consume 4-5% of *all of* Google's datacenter cycles, making them an important target for lightweight hardware acceleration [28]. As we show in Section 4.1, the state-of-the-art solution, RowClone [69], has poor performance for such operations when they are performed *across* subarrays in the same bank.

Our goal is to provide an architectural mechanism to accelerate these inter-subarray copy operations in DRAM. We propose LISA-RISC, which uses the RBM operation in LISA to perform rapid data copying. We describe the high-level operation of LISA-RISC (Section 4.2), and then provide a detailed look at the memory controller command sequence required to implement LISA-RISC (Section 4.3).

[3]In other words, RBM has two variants, one that moves data between immediately adjacent subarrays (Figure 4) and one that moves data between subarrays that are one subarray apart from each other (Figure 5).

4.1. Shortcomings of the State-of-the-Art

Previously, we have described the state-of-the-art work, RowClone [69], which addresses the problem of costly data movement over memory channels by coping data completely in DRAM. However, RowClone does *not* provide fast data copy between subarrays. The main latency benefit of RowClone comes from intra-subarray copy (*RC-IntraSA* for short) as it copies data at the row granularity. In contrast, inter-subarray RowClone (*RC-InterSA*) requires transferring data at the cache line granularity (64B) through the internal data bus in DRAM. Consequently, RC-InterSA incurs *16x longer latency* than RC-IntraSA. Furthermore, RC-InterSA is a long *blocking operation* that prevents reading from or writing to the other banks in the same rank, reducing bank-level parallelism [38, 53].

To demonstrate the ineffectiveness of RC-InterSA, we compare it to today's currently-used copy mechanism, memcpy, which moves data via the memory channel. In contrast to RC-InterSA, which copies data in DRAM, memcpy copies data by sequentially reading out source data from the memory and then writing it to the destination data in the on-chip caches. Figure 6 compares the average system performance and queuing latency of RC-InterSA and memcpy, on a quad-core system across 50 workloads that contain bulk (8KB) data copies (see Section 8 for our methodology). RC-InterSA actually *degrades* system performance by 24% relative to memcpy, mainly because RC-InterSA increases the overall memory queuing latency by 2.88x, as it blocks other memory requests from being serviced by the memory controller performing the RC-InterSA copy. In contrast, memcpy is *not* a long or blocking DRAM command, but rather a long sequence of memory requests that can be interrupted by other critical memory requests, as the memory scheduler can issue memory requests out of order [34, 35, 52, 53, 62, 73, 78, 84].

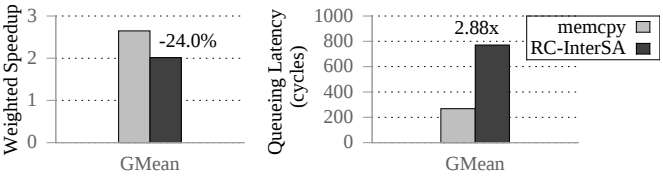


Figure 6: Comparison of RowClone to memcpy over the memory channel, on workloads that perform bulk data copy across subarrays on a 4-core system.

On the other hand, RC-InterSA offers energy savings of 5.1% on average over memcpy by *not* transferring the data over the memory channel. Overall, these results show that neither of the existing mechanisms (memcpy or RowClone) offers *fast and energy-efficient* bulk data copy across subarrays.

4.2. In-DRAM Rapid Inter-Subarray Copy (RISC)

Our goal is to design a new mechanism that enables *low-latency* and *energy-efficient* memory copy between rows *in different subarrays* within the same bank. To this end, we propose a new in-DRAM copy mechanism that uses LISA to exploit the high-bandwidth links between subarrays. The key idea, step by step, is to: (1) activate a source row in a subarray; (2) rapidly transfer the data in the activated source row buffers to the destination subarray's row buffers, through LISA's wide

inter-subarray links, without using the narrow internal data bus; and (3) activate the destination row, which enables the contents of the destination row buffers to be latched into the destination row. We call this inter-subarray row-to-row copy mechanism *LISA-Rapid Inter-Subarray Copy* (LISA-RISC).

As LISA-RISC uses the full row bandwidth provided by LISA, it reduces the copy latency by 9.2x compared to RC-InterSA (see Section 4.5). An additional benefit of using LISA-RISC is that its inter-subarray copy operations are performed *completely inside a bank*. As the internal DRAM data bus is untouched, *other* banks can concurrently serve memory requests, exploiting bank-level parallelism. This new mechanism is complementary to RowClone, which performs fast *intra-subarray* copies. Together, our mechanism and RowClone can enable a complete set of fast in-DRAM copy techniques in future systems. We now explain the step-by-step operation of how LISA-RISC copies data across subarrays.

4.3. Detailed Operation of LISA-RISC

Figure 7 shows the command service timelines for both LISA-RISC and RC-InterSA, for copying a single row of data across two subarrays, as we show on the left. Data is copied from subarray SA0 to SA2. We illustrate four row buffers (RB0–RB3): recall from Section 2.1 that in order to activate one row, a subarray must use *two* row buffers (at the top and bottom), as each row buffer contains only half a row of data. As a result, LISA-RISC must copy half a row at a time, first moving the contents of RB1 into RB3, and then the contents of RB0 into RB2, using two RBM commands.

First, the LISA-RISC memory controller activates the source row ($\texttt{ACT}_{\texttt{SA0}}$) to latch its data into two row buffers (RB0 and RB1). Second, LISA-RISC invokes the first RBM operation ($\texttt{RBM}_{1\rightarrow3}$) to move data from the bottom source row buffer (RB1) to the respective destination row buffer (RB3), thereby linking RB1 to both RB2 and RB3, which activates both RB2 and RB3. After this step, LISA-RISC *cannot* immediately invoke another RBM to transfer the remaining half of the source row in RB0 into RB2, as a row buffer (RB2) needs to be in the *precharged state* in order to receive data from an activated row buffer (RB0). Therefore, LISA-RISC completes copying the first half of the source data into the destination row before invoking the second RBM, by writing the row buffer (RB3) into the cells through an activation ($\texttt{ACT}_{\texttt{SA2}}$). This activation enables the contents of the sense amplifiers (RB3) to be driven into the destination row. To address the issue that modern DRAM chips do not allow a second ACTIVATE to an already-activated bank, we use the *back-to-back* ACTIVATE command that is used to support RowClone [69].

Third, to move data from RB0 to RB2 to complete the copy transaction, we need to precharge both RB1 and RB2. The challenge here is to precharge all row buffers *except* RB0. This cannot be accomplished in today's DRAM because a precharge is applied at the bank level to *all* row buffers. Therefore, we propose to add a new *precharge-exception* command, which prevents a row buffer from being precharged and keeps it activated. This bank-wide exception signal is supplied to all row buffers, and when raised for a particular row buffer, the selected row buffer retains its state while the other row buffers are precharged. After the precharge-exception ($\texttt{PRE}_{\texttt{E}}$) is complete, we then invoke the second RBM ($\texttt{RBM}_{0\rightarrow2}$) to copy RB0 to RB2, which is followed by an activation ($\texttt{ACT}_{\texttt{SA2}}'$) to write RB2 into SA2. Finally, LISA-RISC finishes the copy by issuing a PRECHARGE command (PRE in Figure 7) to the bank.

In comparison, the command service timeline of RC-InterSA is much longer, as RowClone can copy only *one cache line* of data at a time (as opposed to half a row buffer). This requires 128 *serial cache line transfers* to read the data from RB0 and RB1 into a temporary row in another bank, followed by another 128 serial cache line transfers to write the data into RB2 and RB3. LISA-RISC, by moving half a row using a single RBM command, achieves 9.2x lower latency than RC-InterSA.

4.4. Data Coherence

When a copy is performed in DRAM, one potential disadvantage is that the data stored in the DRAM may not be the most recent version, as the processor may have dirty cache lines that belong to the section of memory being copied. Prior works on in-DRAM migration have proposed techniques to accommodate data coherence [67, 69]. Alternatively, we can accelerate coherence operations by using structures like the Dirty-Block Index [66].

4.5. Comparison of Copy Techniques

Figure 8 shows the DRAM latency and DRAM energy consumption of different *copy commands* for copying a row of data (8KB). The exact latency and energy numbers are listed in Table 1.[4] We derive the copy latency of each command sequence using equations based on the DDR3-1600 timings [22] (available in our technical report [5]), and the DRAM energy using the Micron power calculator [49]. For LISA-RISC, we define a *hop* as the number of subarrays that LISA-RISC needs to copy data *across* to move the data from the source subarray to the destination subarray. For example, if the source and destination subarrays are adjacent to each other, the number of

[4]Our reported numbers differ from prior work [69] because: (1) we use faster DRAM timing parameters (1600-11-11-11 vs 1066-8-8-8), and (2) we use the 8KB row size of most commercial DRAM instead of 4KB [69].

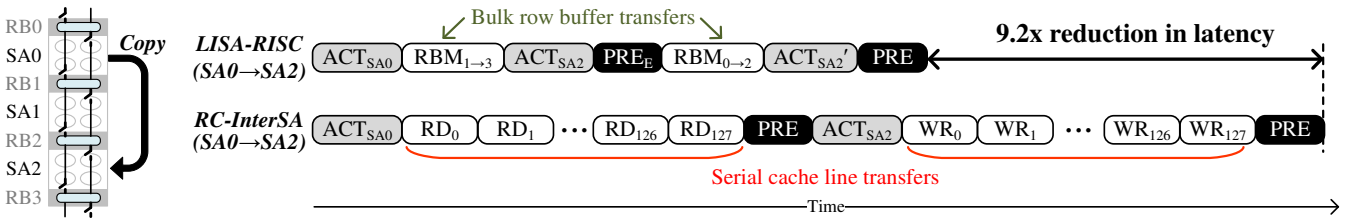


Figure 7: Command service timelines of a row copy for LISA-RISC and RC-InterSA (command latencies not drawn to scale).

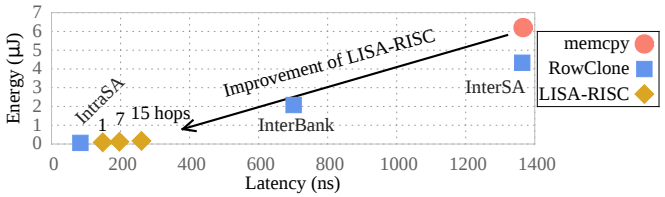


Figure 8: Latency and DRAM energy of 8KB copy.

Copy Commands (8KB)	**Latency** (ns)	**Energy** (μJ)
`memcpy` (via mem. channel)	1366.25	6.2
RC-InterSA / Bank / IntraSA	1363.75 / 701.25 / 83.75	4.33 / 2.08 / 0.06
LISA-RISC (1 / 7 / 15 hops)	148.5 / 196.5 / 260.5	0.09 / 0.12 / 0.17

Table 1: Copy latency and DRAM energy.

hops is 1. The DRAM chips that we evaluate have 16 subarrays per bank, so the maximum number of hops is 15.

We make two observations from these numbers. First, although RC-InterSA incurs similar latencies as `memcpy`, it consumes 29.6% less energy, as it does *not* transfer data over the channel and DRAM I/O for each copy operation. However, as we showed in Section 4.1, RC-InterSA incurs a higher system performance penalty because it is a long-latency *blocking* memory command. Second, copying between subarrays using LISA achieves significantly lower latency and energy compared to RowClone, even though the total latency of LISA-RISC grows linearly with the hop count.

By exploiting the LISA substrate, we thus provide a more complete set of in-DRAM copy mechanisms. Our workload evaluation results show that LISA-RISC outperforms RC-InterSA and `memcpy`: its average performance improvement and energy reduction over the best performing inter-subarray copy mechanism (i.e., `memcpy`) are 66.2% and 55.4%, respectively, on a quad-core system, across 50 workloads that perform bulk copies (see Section 9.1).

5. Application 2: In-DRAM Caching Using Heterogeneous Subarrays (LISA-VILLA)

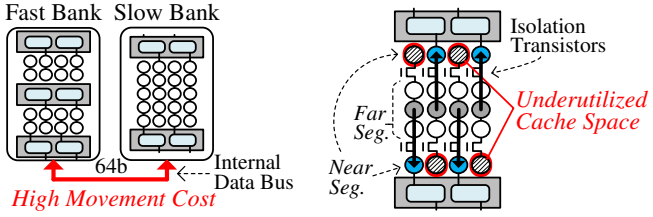
Our second application aims to reduce the DRAM access latency for frequently-accessed (hot) data. Prior work introduces heterogeneity into DRAM, where one region has a fast access latency but small capacity (fewer DRAM rows), while the other has a slow access latency but high capacity (many more rows) [40, 71]. To yield the highest performance benefits, the fast region is used as a dynamic cache that stores the hot rows. There are two design constraints that must be considered: (1) *ease of implementation*, as the fast caching structure needs to be low-cost and non-intrusive; and (2) *data movement cost*, as the caching mechanism should adapt to dynamic program phase changes, which can lead to changes in the set of hot DRAM rows. As we show in Section 5.1, prior work has not balanced the trade-off between these two constraints.

Our goal is to design a heterogeneous DRAM that offers fast data movement with a low-cost and easy-to-implement design. To this end, we propose *LISA-VILLA (VarIabLe LAtency)*, a mechanism that uses LISA to provide fast row movement into the cache when the set of hot DRAM rows changes. LISA-VILLA is also easy to implement, as discussed in Section 5.2. We describe our hot row caching policy in Section 5.3.

5.1. Shortcomings of the State-of-the-Art

We observe that two state-of-the-art techniques for heterogeneity within a DRAM chip are not effective at providing *both* ease of implementation and low movement cost.

CHARM [71] introduces heterogeneity *within a rank* by designing a few fast banks with (1) shorter bitlines for faster data sensing, and (2) closer placement to the chip I/O for faster data transfers. To exploit these low-latency banks, CHARM uses an OS-managed mechanism to statically allocate hot data to them based on program profile information. Unfortunately, this approach cannot adapt to program phase changes, limiting its performance gains. If it were to adopt dynamic hot data management, CHARM would incur high movement cost over the narrow 64-bit internal data bus in DRAM, as illustrated in Figure 9a, since it does not provide high-bandwidth connectivity between banks.



(a) CHARM [71] (b) TL-DRAM [40]

Figure 9: Drawbacks of existing heterogeneous DRAMs.

TL-DRAM [40] provides heterogeneity *within a subarray* by dividing it into fast (near) and slow (far) segments that have short and long bitlines, respectively, using isolation transistors. To manage the fast segment as an OS-transparent hardware cache, TL-DRAM proposes a *fast intra-subarray movement* scheme similar to RowClone [69]. The main disadvantage is that TL-DRAM needs to cache each hot row in *two near segments*, as shown in Figure 9b, as each subarray uses two row buffers on *opposite ends* to sense data in the open-bitline architecture. This prevents TL-DRAM from using the *full* near segment capacity. TL-DRAM's area overhead is also sizable (3.15%) in an open-bitline architecture. As we can see, neither CHARM nor TL-DRAM strike a good trade-off between the two design constraints.

5.2. Variable Latency (VILLA) DRAM

We propose to introduce heterogeneity *within a bank* by designing *heterogeneous-latency subarrays*. We call this heterogeneous DRAM design *VarIabLe LAtency DRAM* (VILLA-DRAM). To design a low-cost fast subarray, we take an approach similar to prior work, attaching fewer cells to each bitline to reduce the parasitic capacitance and resistance. This reduces the sensing (tRCD), restoration (tRAS), and precharge (tRP) time of the fast subarrays [40, 51, 71]. In this work, we focus on managing the fast subarrays in hardware, as it offers better adaptivity to dynamic changes in the hot data set.

In order to take advantage of VILLA-DRAM, we rely on LISA-RISC to rapidly copy rows across subarrays, which significantly reduces the caching latency. We call this synergistic

design, which builds VILLA-DRAM using the LISA substrate, *LISA-VILLA*. Nonetheless, the cost of transferring data to a fast subarray is still non-negligible, especially if the fast subarray is far from the subarray where the data to be cached resides. Therefore, an intelligent cost-aware mechanism is required to make astute decisions on which data to cache and when.

5.3. Caching Policy for LISA-VILLA

We design a simple epoch-based caching policy to evaluate the benefits of caching a row in LISA-VILLA. Every epoch, we track the number of accesses to rows by using a set of 1024 saturating counters for each bank.[5] The counter values are halved every epoch to prevent staleness. At the end of an epoch, we mark the 16 most frequently-accessed rows as *hot*, and cache them when they are accessed the next time. For our cache replacement policy, we use the *benefit-based caching* policy proposed by Lee et al. [40]. Specifically, it uses a benefit counter for each row cached in the fast subarray: whenever a cached row is accessed, its counter is incremented. The row with the least benefit is replaced when a new row needs to be inserted. Note that a large body of work proposed various caching policies (e.g., [11, 13, 15, 25, 30, 48, 61, 68, 81]), each of which can potentially be used with LISA-VILLA.

Our evaluation shows that LISA-VILLA improves system performance by 5.1% on average, and up to 16.1%, for a range of 4-core workloads (see Section 9.2).

6. Application 3: Fast Precharge Using Linked Precharge Units (LISA-LIP)

Our third application aims to accelerate the process of precharge. The precharge time for a subarray is determined by the drive strength of the precharge unit. We observe that in modern DRAM, while a subarray is being precharged, the precharge units (PUs) of *other* subarrays remain idle.

We propose to exploit these idle PUs to accelerate a precharge operation by connecting them to the subarray that is being precharged. Our mechanism, *LISA-LInked Precharge* (LISA-LIP), precharges a subarray using *two* sets of PUs: one from the row buffer that is being precharged, and a second set from a neighboring subarray's row buffer (which is already in the precharged state), by enabling the links between the two subarrays.

Figure 10 shows the process of *linked precharging* using LISA. Initially, only one subarray (top) is fully activated (state ①) while the neighboring (bottom) subarray is in the precharged state. The neighboring subarray is in the precharged state, as only one subarray in a bank can be activated at a time, while the other subarrays remain precharged. In state ②, we begin the precharge operation by disabling the sense amplifier in the top row buffer and enabling its PU. After we enable the links between the top and bottom subarrays, the bitlines start sharing charge with each other, and both PUs *simultaneously* reinitialize the bitlines, eventually fully pulling the bitlines to $V_{DD}/2$ (state ③). Note that we are using *two* PUs to pull down *only one* set of activated bitlines, which is why the precharge process is shorter.

[5] The hardware cost of these counters is low, requiring only 6KB of storage in the memory controller (see Section 7.1).

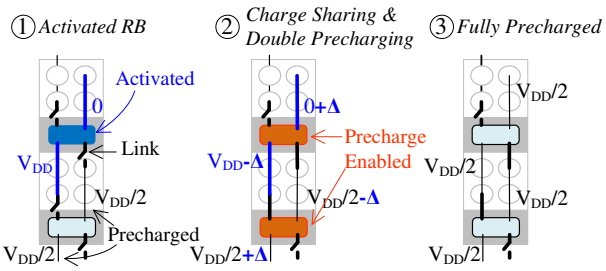


Figure 10: Linked precharging through LISA.

To evaluate the accelerated precharge process, we use the same methodology described in Section 3.3 and simulate the linked precharge operation in SPICE. Figure 11 shows the resulting timing diagram. During the first 2ns, the wordline is lowered to disconnect the cells from the bitlines ①. Then, we enable the links to begin precharging the bitlines ②. The result shows that the precharge latency reduces significantly due to having two PUs to perform the precharge. LISA enables a shorter precharge latency of approximately 3.5ns ③ versus the baseline precharge latency of 13.1ns ④.

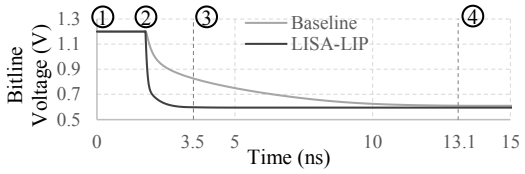


Figure 11: SPICE simulation of precharge operation.

To account for **process and temperature variation**, we add a guardband to the SPICE-reported latency, increasing it to 5ns (i.e., by *42.9%*), which still achieves 2.6x lower precharge latency than the baseline. Our evaluation shows that LISA-LIP improves performance by 10.3% on average, across 50 four-core workloads (see Section 9.3).

7. Hardware Cost

7.1. Die Area Overhead

To evaluate the area overhead of adding isolation transistors, we use area values from prior work, which adds isolation transistors to disconnect bitlines from sense amplifiers [55]. That work shows that adding an isolation transistor to every bitline incurs a total of 0.8% die area overhead in a 28nm DRAM process technology. Similar to prior work that adds isolation transistors to DRAM [40, 55], our LISA substrate also requires additional control logic outside the DRAM banks to control the isolation transistors, which incurs a small amount of area and is non-intrusive to the cell arrays. For LISA-VILLA, we use 1024 six-bit saturating counters to track the access frequency of rows in every bank; this requires an additional 6KB storage within a memory controller connected to one rank.

7.2. Handling Repaired Rows

To improve yield, DRAM manufacturers often employ post-manufacturing repair techniques that can remap faulty rows to spare rows provisioned in every subarray [31]. Therefore, consecutive row addresses as observed by the memory controller may physically reside in *different* subarrays. To handle

this issue for techniques that require the controller to know the subarray a row resides in (e.g., RowClone [69], LISA-RISC), a simple approach can be used to expose the repaired row information to the memory controller. Since DRAM already stores faulty rows' remapping information inside the chip, this information can be exposed to the controller through the serial presence detect (SPD) [23], which is an EEPROM that stores DRAM information such as timing parameters. The memory controller can read this stored information at system boot time so that it can correctly determine a repaired row's location in DRAM. Note that similar techniques may be necessary for other mechanisms that require information about physical location of rows in DRAM (e.g., [4, 29, 33, 36, 40, 43]).

8. Methodology

We evaluate our system using a variant of Ramulator [32], an open-source cycle-accurate DRAM simulator, driven by traces generated from Pin [46]. We will make our simulator publicly available [6]. We use a row buffer policy that closes a row only when there are no more outstanding requests in the memory controller to the same row [62]. Unless stated otherwise, our simulator uses the parameters listed in Table 2.

Processor	1–4 OoO cores, 4GHz, 3-wide issue
Cache	L1: 64KB, L2: 512KB per core, L3: 4MB, 64B lines
Mem. Controller	64/64-entry read/write queue, FR-FCFS [62, 84]
DRAM	DDR3-1600 [50], 1–2 channels, 1 rank/channel, 8 banks/rank, 16 subarrays/bank

Table 2: Evaluated system configuration.

Benchmarks and Workloads. We primarily use benchmarks from TPC(-C/-H) [76], DynoGraph (BFS, PageRank) [58], SPEC CPU2006 [72], and STREAM [47], along with a random-access microbenchmark similar to HPCC RandomAccess [14]. Because these benchmarks predominantly stress the CPU and memory while rarely invoking `memcpy`, we use the following benchmarks to evaluate different copy mechanisms: (1) `bootup`, (2) `forkbench`, and (3) `Unix shell`. These were shared by the authors of RowClone [69]. The `bootup` benchmark consists of a trace collected while a Debian operating system was booting up. The `forkbench` kernel forks a child process that copies 1K pages from the parent process by randomly accessing them from a 64MB array. The `Unix shell` is a script that runs `find` in a directory along with `ls` on each subdirectory. More information on these is in [69].

To evaluate the benefits of different data copy mechanisms in isolation, we use a copy-aware page mapping policy that allocates destination pages to certain DRAM structures (i.e., subarrays, banks) such that only the specified copy mechanism is used for copy operations. For example, when evaluating RC-IntraSA, the page mapper allocates the destination page only within the same subarray as the source page [69].

To construct multi-core workloads for evaluating the benefits of data copy mechanisms, we randomly assemble 50 workloads, each comprising 50% copy-intensive benchmarks and 50% non-copy-intensive benchmarks. To evaluate the benefits of in-DRAM caching and reduced precharge time, we restrict our workloads to randomly-selected memory-intensive (≥ 5 misses per thousand instructions) non-copy-intensive benchmarks. Due to the large number of workloads, we present detailed results for only five workload mixes (Table 3), along with the average results across all 50 workloads.

Mix 1	`tpcc64, forkbench, libquantum, bootup`
Mix 2	`bootup, xalancbmk, pagerank, forkbench`
Mix 3	`libquantum, pagerank, forkbench, bootup`
Mix 4	`mcf, forkbench, random, forkbench`
Mix 5	`bfs, bootup, tpch2, bootup`

Table 3: A subset of copy workloads with detailed results.

Performance Metrics. We measure single-core and multi-core performance using IPC and Weighted Speedup (WS) [70], respectively. Prior work showed that WS is a measure of system throughput [8]. To report DRAM energy consumption, we use the Micron power calculator [49]. We run all workloads for 100 million instructions, as done in many recent works [35, 36, 40, 41, 52].

VILLA-DRAM Configuration. For our simulated VILLA-DRAM, each fast subarray consists of 32 rows to achieve low latency on sensing, precharge, and restoration (a typical subarray has 512 rows). Our SPICE simulation reports the following new timing parameters for a 32-row subarray: tRCD=7.5ns, tRP=8.5ns, and tRAS=13ns, which are reduced from the original timings by respectively, 45.5%, 38.2%, and 62.9%. For each bank, we allocate 4 fast subarrays in addition to the 16 512-row subarrays, incurring a 1.6% area overhead. We set the epoch length for our caching policy to 10,000 cycles.

9. Evaluation

We quantitatively evaluate our proposed applications of LISA: (1) rapid bulk copying (LISA-RISC), (2) in-DRAM caching with heterogeneous subarrays (LISA-VILLA), and (3) reduced precharge time (LISA-LIP).

9.1. Bulk Memory Copy

9.1.1. Single-Core Workloads. Figure 12 shows the performance of three copy benchmarks on a single-core system with one memory channel and 1MB of last-level cache (LLC). We evaluate the following bulk copy mechanisms: (1) `memcpy`, which copies data over the memory channel; (2) RowClone [69]; and (3) LISA-RISC. We use two different hop counts between the source and destination subarray for LISA-RISC: 15 (longest) and 1 (shortest). They are labeled as LISA-RISC-15 and LISA-RISC-1, respectively, in the figure. We make four major observations.

First, LISA-RISC achieves significant improvement over RC-InterSA for all three benchmarks in terms of both IPC and memory energy consumption, shown in Figure 12a and Figure 12b, respectively. This shows that the LISA substrate is effective at performing fast inter-subarray copies.

Second, both LISA-RISC-1 and LISA-RISC-15 significantly reduce the memory energy consumption over `memcpy`. This is due to (1) reduced memory traffic over the channel by keeping the data within DRAM, and (2) higher performance.

Third, LISA-RISC-1/-15 provides 12.6%/10.6%, 4.9x/4.3x, and 1.8%/0.7% speedup for `bootup`, `forkbench`, and `shell`, respectively, over `memcpy`. The performance gains are smaller

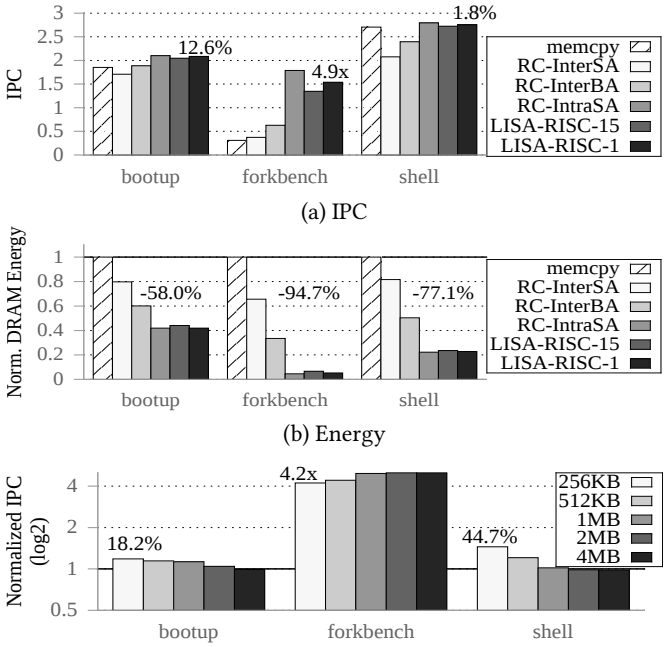


Figure 12: Comparison of copy mechanisms in a single-core system. Value (%) on top indicates the improvement of LISA-RISC-1 over memcpy.

for `bootup` and `shell`. Both of these benchmarks invoke fewer copy operations (i.e., 2171 and 2682, respectively) than `forkbench`, which invokes a large number (40952) of copies. As a result, `forkbench` is more sensitive to the memory latency of copy commands. Furthermore, the large LLC capacity (1MB) helps absorb the majority of memory writes resulting from `memcpy` for `bootup` and `shell`, thereby reducing the effective latency of `memcpy`.

Fourth, RC-InterSA performs *worse* than `memcpy` for `bootup` and `shell` due to its long *blocking* copy operations. Although, it attains a 19.4% improvement on `forkbench` because `memcpy` causes severe *cache pollution* by installing a large amount of copied data into the LLC. Compared to the 20% cache hit rate for `memcpy`, RC-InterSA has a much higher hit rate of 67.2% for `forkbench`. The copy performance of `memcpy` is strongly correlated with the LLC management policy and size.

To understand performance sensitivity to LLC size, Figure 12c shows the speedup of LISA-RISC-1 over `memcpy` for different LLC capacities. We make two observations, which are also similar for LISA-RISC-15 (not shown). First, for `bootup` and `shell`, the speedup of LISA over `memcpy` reduces as the LLC size increases because the destination locations of `memcpy` operations are more likely to hit in the larger cache.

Second, for `forkbench`, LISA-RISC's performance gain over `memcpy` decreases as cache size reduces from 1MB to 256KB. This is because the LLC hit rate reduces much more significantly for LISA-RISC, from 67% (1MB) to 10% (256KB), than for `memcpy` (from 20% at 1MB, to 19% at 256KB). When `forkbench` uses LISA-RISC for copying data, its working set mainly consists of non-copy data, which has good locality. As the LLC size reduces by 4x, the working set no longer fits in the smaller cache, thus causing a significant hit rate reduction. On the other hand, when `memcpy` is used as the copy mechanism, the working set of `forkbench` is mainly from bulk copy data, and is less susceptible to cache size reduction. Nonetheless, LISA-RISC still provides an improvement of 4.2x even with a 256KB cache.

We conclude that LISA-RISC significantly improves performance and memory energy efficiency in single-core workloads that invoke bulk copies.

9.1.2. Multi-Core Workloads. Figure 13 shows the system performance and energy efficiency (i.e., memory energy per instruction) of different copy mechanisms across 50 workloads, on a quad-core system with two channels and 4MB of LLC. The error bars in this figure (and other figures) indicate the 25th and 75th percentile values across all 50 workloads. Similar to the performance trends seen in the single-core system, LISA-RISC consistently outperforms other mechanisms at copying data between subarrays. LISA-RISC-1 attains a high average system performance improvement of 66.2% and 2.2x over `memcpy` and RC-InterSA, respectively. Although Mix 5 has the smallest number of copy operations out of the five presented workload mixes, LISA-RISC still improves its performance by 6.7% over `memcpy`. By moving copied data only within DRAM, LISA-RISC significantly reduces memory energy consumption (55.4% on average) over `memcpy`. In summary, LISA-RISC provides both high performance and high memory energy efficiency for bulk data copying for a wide variety of single- and multi-core workloads.

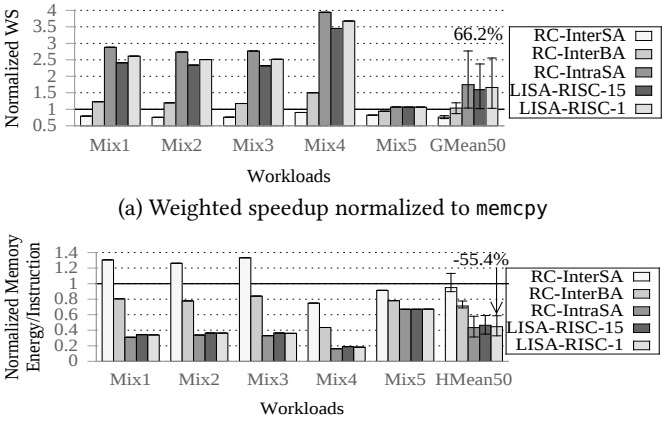


Figure 13: Four-core system evaluation: (a) weighted speedup and (b) memory energy per instruction.

9.2. In-DRAM Caching with LISA-VILLA

Figure 14 shows the system performance improvement of LISA-VILLA over a baseline without any fast subarrays in a four-core system. It also shows the hit rate in VILLA-DRAM, i.e., the fraction of accesses that hit in the fast subarrays. We make two main observations. First, by exploiting LISA-RISC to quickly cache data in VILLA-DRAM, LISA-VILLA improves system performance for a wide variety of workloads — by up to 16.1%, with a geometric mean of 5.1%. This is mainly due to reduced DRAM latency of accesses that hit in the fast subarrays (which comprise 16MB of total storage across two memory channels). The performance improvement heavily

correlates with the VILLA cache hit rate. Our work does not focus on optimizing the caching scheme, but the hit rate may be increased by an enhanced caching policy (e.g., [61, 68]), which can further improve system performance.

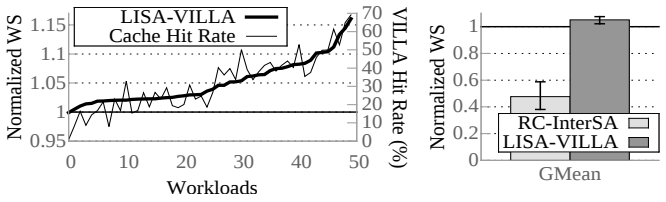


Figure 14: Performance improvement and hit rate with LISA-VILLA, and performance comparison to using RC-InterSA with VILLA-DRAM.

Second, the VILLA-DRAM design, which consists of heterogeneous subarrays, is not practical without LISA. Figure 14 shows that using RC-InterSA to move data into the cache *reduces* performance by 52.3% due to slow data movement, which overshadows the benefits of caching. The results indicate that LISA is an important substrate to enable not only fast bulk data copy, but also a fast in-DRAM caching scheme.

9.3. Accelerated Precharge with LISA-LIP

Figure 15 shows the system performance improvement of LISA-LIP over a baseline that uses the standard DRAM precharge latency, as well as LISA-LIP's row-buffer hit rate, on a four-core system across 50 workloads. LISA-LIP attains a maximum gain of 13.2%, with a mean improvement of 8.1%. The performance gain becomes higher as the row-buffer hit rate decreases, which leads to more precharge commands. These results show that LISA is a versatile substrate that effectively reduces precharge latency in addition to accelerating data movement.

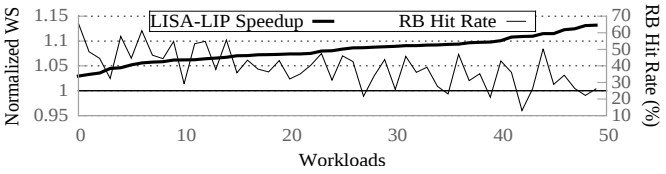


Figure 15: Speedup and row buffer (RB) hit rate of LISA-LIP.

We also evaluate the effectiveness of combining LISA-VILLA and LISA-LIP (not shown, but available in our technical report [5]). The combined mechanism, which is transparent to software, improves system performance by 12.2% on average and up to 23.8% across the same set of 50 workloads without bulk copies. Thus, LISA is an effective substrate that can enable mechanisms to fundamentally reduce memory latency.

9.4. Putting Everything Together

As all of the three proposed applications are complementary to each other, we evaluate the effect of putting them together on a four-core system. Figure 16 shows the system performance improvement of adding LISA-VILLA to LISA-RISC (15 hops), as well as combining all three optimizations, compared to our baseline using `memcpy` and standard DDR3-1600 memory. We draw several key conclusions. First, the performance benefits from each scheme are additive. On average, adding LISA-VILLA improves performance by 16.5% over LISA-RISC alone, and adding LISA-LIP further provides an 8.8% gain over LISA-(RISC+VILLA). Second, although LISA-RISC alone provides a majority of the performance improvement over the baseline (59.6% on average), the use of both LISA-VILLA and LISA-LIP further improves performance, resulting in an average performance gain of 94.8% and memory energy reduction (not plotted) of 49.0%. Taken together, these results indicate that LISA is an effective substrate that enables a wide range of high-performance and energy-efficient applications in the DRAM system.

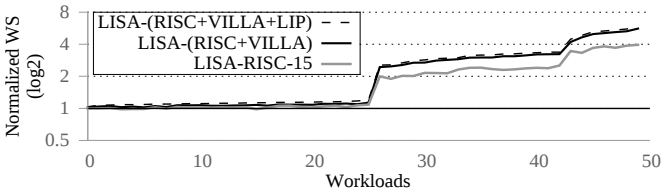


Figure 16: Combined WS improvement of LISA applications.

9.5. Sensitivity to System Configuration

Figure 17 shows the weighted speedup for `memcpy` and LISA-All (i.e., all three applications) on a 4-core system using varying memory channel counts and LLC sizes. The results show that performance improvement increases with fewer memory channels, as memory contention increases. On the other hand, adding more memory channels increases memory-level parallelism, allowing more of the copy latency to be hidden. Similar trends are observed with the LLC capacity. As LLC size decreases, the working set becomes less likely to fit with `memcpy`, worsening its performance. LISA-All provides significant performance benefits for all configurations.

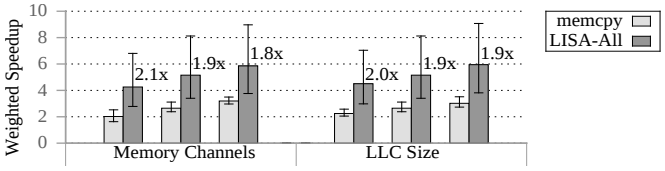


Figure 17: Performance sensitivity to channels and LLC size.

9.6. Effect of Copy Distance on LISA-RISC

Table 4 shows that the performance gain and memory energy savings of LISA-RISC over `memcpy` increases as the copy distance reduces. This is because with fewer subarrays between the source and destination subarrays, the number of RBM commands invoked by LISA-RISC reduces accordingly, which decreases the latency and memory energy consumption of bulk data copy.

Copy Distance (hops)	1	3	7	15	31	63
RISC Copy Latency (ns)	148.5	164.5	196.5	260.5	388.5	644.5
WS Improvement (%)	66.2	65.3	63.3	59.6	53.0	42.4
DRAM Energy Savings (%)	55.4	55.2	54.6	53.6	51.9	48.9

Table 4: Effect of copy distance on LISA-RISC.

10. Other Applications Enabled by LISA

We describe two additional applications that can potentially benefit from LISA. We describe them at a high level, and defer evaluations to future work.

Reducing Subarray Conflicts via Remapping. When two memory requests access two different rows in the same bank, they have to be served serially, even if they are to different subarrays. To mitigate such *bank conflicts*, Kim et al. [36] propose *subarray-level parallelism (SALP)*, which enables multiple subarrays to remain activated at the same time. However, if two accesses are to the same subarray, they still have to be served serially. This problem is exacerbated when frequently-accessed rows reside in the same subarray. To help alleviate such *subarray conflicts*, LISA can enable a simple mechanism that efficiently remaps or moves the conflicting rows to different subarrays by exploiting fast RBM operations.

Extending the Range of In-DRAM Bulk Operations. To accelerate bitwise operations, Seshadri et al. [67] propose a new mechanism that performs bulk bitwise AND and OR operations in DRAM. Their mechanism is restricted to applying bitwise operations only on rows within the *same subarray* as it requires the copying of source rows before performing the bitwise operation. The high cost of *inter-subarray* copies makes the benefit of this mechanism inapplicable to data residing in rows in different subarrays. LISA can enable efficient inter-subarray bitwise operations by using LISA-RISC to copy rows to the same subarray at low latency and low energy.

11. Related Work

To our knowledge, this is the first work to propose a DRAM substrate that supports fast data movement between subarrays in the same bank, which enables a wide variety of applications for DRAM systems. We already provided extensive comparisons to RowClone [69], CHARM [71], and TL-DRAM [40]. We now discuss prior works that focus on each of the optimizations that LISA enables.

Bulk Data Transfer Mechanisms. Prior works [9, 10, 27] have added scratchpad memories to reduce CPU pressure during bulk data transfers, which can also enable sophisticated data movement (e.g., scatter-gather), but they still require data to first be moved on-chip. A patent proposes a DRAM that can copy a page across memory blocks [65], but lacks concrete analysis and evaluation of the underlying copy operations. Intel I/O Acceleration Technology [16] allows for memory-to-memory DMA transfers *across a network*, but cannot transfer data within main memory.

Zhao et al. [83] propose to add a bulk data movement engine inside the memory controller to speed up bulk-copy operations. Jiang et al. [26] design a different copy engine, placed within the cache controller, to alleviate pipeline and cache stalls that occur when these transfers take place. However, these works do not directly address the problem of data movement across the narrow memory channel.

A concurrent work by Lu et al. [45] proposes a heterogeneous DRAM design similar to VILLA-DRAM, called DAS-DRAM, but with a very different data movement mechanism from LISA. It introduces a row of *migration cells* into each subarray to move rows across subarrays. Unfortunately, the latency of DAS-DRAM is not scalable with movement distance, because it requires writing the migrating row into each intermediate subarray's migration cells before the row reaches its destination, which prolongs data transfer latency. In contrast, LISA provides a *direct path* to transfer data *between row buffers* without requiring intermediate data writes into the subarray.

Caching. Several prior works (e.g., [11, 13, 15, 30]) have proposed to add a small SRAM cache to DRAM chips, which obtains lower access latency at the cost of high area overhead. Many other works (e.g., [7, 20, 21, 57, 59, 60, 61, 68]) have proposed various cache management policies. Such caching policies can potentially be integrated with LISA-VILLA to achieve a better hit rate in the fast subarrays.

Reducing DRAM Latency. Several types of DRAM provide low latency, such as Micron's RLDRAM [51] or Fujitsu's FCDRAM [64]. These have fewer cells connected to each bitline, which comes with a large overhead [36, 40].

O et al. [55] propose to add isolation transistors that separate bitlines from their sense amplifiers so that the bitlines can be precharged without deactivating the row buffer. This helps to hide the latency impact of precharge. These isolation transistors are orthogonal to isolation transistors in LISA, which interconnect bitlines of *neighboring subarrays*, and LISA can be combined with [55] to further improve performance. LISA can also be combined with many other proposals that reduce DRAM latency (e.g., [4, 12, 36, 37, 39, 40]).

12. Conclusion

We present a new DRAM substrate, *low-cost inter-linked subarrays (LISA)*, that expedites bulk data movement across subarrays in DRAM. LISA achieves this by creating a new high-bandwidth datapath at low cost between subarrays, via the insertion of a small number of isolation transistors. We describe and evaluate three applications that are enabled by LISA. First, LISA significantly reduces the latency and memory energy consumption of bulk copy operations between subarrays over two state-of-the-art mechanisms [69]. Second, LISA enables an effective in-DRAM caching scheme on a new heterogeneous DRAM organization, which uses fast subarrays for caching hot data in every bank. Third, we reduce precharge latency by connecting two precharge units of adjacent subarrays together using LISA. We experimentally show that the three applications of LISA greatly improve system performance and memory energy efficiency when used individually or together, across a variety of workloads and system configurations.

We conclude that LISA is an effective substrate that enables several effective applications. We believe that this substrate, which enables low-cost interconnections between DRAM subarrays, can pave the way for other applications that can further improve system performance and energy efficiency through fast data movement in DRAM.

Acknowledgments

We thank the anonymous reviewers and SAFARI group members for their helpful feedback. We acknowledge the support of Google, Intel, Nvidia, Samsung, and VMware. This research was supported in part by the ISTC-CC, SRC, CFAR, and NSF (grants 1212962, 1319587, and 1320531). Kevin Chang is supported in part by the SRCEA/Intel Fellowship.

References

[1] S. Blagodurov *et al.*, "A Case for NUMA-Aware Contention Management on Multicore Systems," in *USENIX ATC*, 2011.
[2] Cadence Design Systems, Inc., "Spectre Circuit Simulator," http://www.cadence.com/products/rf/spectre_circuit/pages/default.aspx.
[3] K. Chandrasekar *et al.*, "Exploiting Expendable Process-Margins in DRAMs for Run-Time Performance Optimization," in *DATE*, 2014.
[4] K. K. Chang *et al.*, "Improving DRAM Performance by Parallelizing Refreshes with Accesses," in *HPCA*, 2014.
[5] K. K. Chang *et al.*, "Low-Cost Inter-Linked Subarrays (LISA): A New DRAM Substrate with Higher Connectivity," Carnegie Mellon Univ., SAFARI Research Group, Tech. Rep., 2016.
[6] CMU SAFARI Research Group, https://github.com/CMU-SAFARI.
[7] J. D. Collins and D. M. Tullsen, "Hardware Identification of Cache Conflict Misses," in *MICRO*, 1999.
[8] S. Eyerman and L. Eeckhout, "System-Level Performance Metrics for Multiprogram Workloads," *IEEE Micro*, 2008.
[9] M. Gschwind, "Chip Multiprocessing and the Cell Broadband Engine," in *CF*, 2006.
[10] J. Gummaraju *et al.*, "Architectural Support for the Stream Execution Model on General-Purpose Processors," in *PACT*, 2007.
[11] C. A. Hart, "CDRAM in a Unified Memory Architecture," in *Intl. Computer Conference*, 1994.
[12] H. Hassan *et al.*, "ChargeCache: Reducing DRAM Latency by Exploiting Row Access Locality," in *HPCA*, 2016.
[13] H. Hidaka *et al.*, "The Cache DRAM Architecture," *IEEE Micro*, 1990.
[14] HPC Challenge, "RandomAccess," http://icl.cs.utk.edu/hpcc.
[15] W.-C. Hsu and J. E. Smith, "Performance of Cached DRAM Organizations in Vector Supercomputers," in *ISCA*, 1993.
[16] Intel Corp., "Intel®I/O Acceleration Technology," http://www.intel.com/content/www/us/en/wireless-network/accel-technology.html.
[17] Intel Corp., "Intel 64 and IA-32 Architectures Optimization Reference Manual," 2012.
[18] ITRS, http://www.itrs.net/ITRS1999-2014Mtgs,Presentations&Links/2013ITRS/2013Tables/FEP_2013Tables.xlsx, 2013.
[19] ITRS, http://www.itrs.net/ITRS1999-2014Mtgs,Presentations&Links/2013ITRS/2013Tables/Interconnect_2013Tables.xlsx, 2013.
[20] A. Jaleel *et al.*, "Adaptive Insertion Policies for Managing Shared Caches," in *PACT*, 2008.
[21] A. Jaleel *et al.*, "High Performance Cache Replacement Using Re-reference Interval Prediction (RRIP)," in *ISCA*, 2010.
[22] JEDEC, "DDR3 SDRAM Standard," 2010.
[23] JEDEC, "Standard No. 21-C. Annex K: Serial Presence Detect (SPD) for DDR3 SDRAM Modules," 2011.
[24] JEDEC, "DDR4 SDRAM Standard," 2012.
[25] X. Jiang *et al.*, "CHOP: Adaptive Filter-Based DRAM Caching for CMP Server Platforms," in *HPCA*, 2010.
[26] X. Jiang *et al.*, "Architecture Support for Improving Bulk Memory Copying and Initialization Performance," in *PACT*, 2009.
[27] J. A. Kahle *et al.*, "Introduction to the Cell Multiprocessor," *IBM JRD*, 2005.
[28] S. Kanev *et al.*, "Profiling a Warehouse-Scale Computer," in *ISCA*, 2015.
[29] U. Kang *et al.*, "Co-Architecting Controllers and DRAM to Enhance DRAM Process Scaling," in *The Memory Forum*, 2014.
[30] G. Kedem and R. P. Koganti, "WCDRAM: A Fully Associative Integrated Cached-DRAM with Wide Cache Lines," *CS-1997-03, Duke*, 1997.
[31] B. Keeth and R. J. Baker, *DRAM Circuit Design: A Tutorial*. Wiley, 2000.
[32] Y. Kim *et al.*, "Ramulator: A Fast and Extensible DRAM Simulator," *CAL*, 2015.
[33] Y. Kim *et al.*, "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors," in *ISCA*, 2014.
[34] Y. Kim *et al.*, "ATLAS: A Scalable and High-Performance Scheduling Algorithm for Multiple Memory Controllers," in *HPCA*, 2010.
[35] Y. Kim *et al.*, "Thread Cluster Memory Scheduling: Exploiting Differences in Memory Access Behavior," in *MICRO*, 2010.
[36] Y. Kim *et al.*, "A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM," in *ISCA*, 2012.
[37] C. J. Lee *et al.*, "DRAM-Aware Last-Level Cache Writeback: Reducing Write-Caused Interference in Memory Systems," Tech. Rep., 2010.
[38] C. J. Lee *et al.*, "Improving Memory Bank-Level Parallelism in the Presence of Prefetching," in *MICRO*, 2009.
[39] D. Lee *et al.*, "Adaptive-Latency DRAM: Optimizing DRAM Timing for the Common-Case," in *HPCA*, 2015.
[40] D. Lee *et al.*, "Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture," in *HPCA*, 2013.
[41] D. Lee *et al.*, "Simultaneous Multi Layer Access: A High Bandwidth and Low Cost 3D-Stacked Memory Interface," *TACO*, 2016.
[42] K.-N. Lim *et al.*, "A 1.2V 23nm 6F2 4Gb DDR3 SDRAM with Local-Bitline Sense Amplifier, Hybrid LIO Sense Amplifier and Dummy-Less Array Architecture," in *ISSCC*, 2012.
[43] J. Liu *et al.*, "An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms," in *ISCA*, 2013.
[44] J. Liu *et al.*, "RAIDR: Retention-Aware Intelligent DRAM Refresh," in *ISCA*, 2012.
[45] S.-L. Lu *et al.*, "Improving DRAM Latency with Dynamic Asymmetric Subarray," in *MICRO*, 2015.
[46] C.-K. Luk *et al.*, "Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation," in *PLDI*, 2005.
[47] J. D. McCalpin, "STREAM Benchmark."
[48] J. Meza *et al.*, "Enabling Efficient and Scalable Hybrid Memories Using Fine-Granularity DRAM Cache Management," *CAL*, 2012.
[49] Micron Technology, "Calculating Memory System Power for DDR3," 2007.
[50] Micron Technology, Inc., "4Gb: x4, x8, x16 DDR3 SDRAM," 2011.
[51] Micron Technology, Inc., "576Mb: x18, x36 RLDRAM3," 2011.
[52] O. Mutlu and T. Moscibroda, "Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors," in *MICRO*, 2007.
[53] O. Mutlu and T. Moscibroda, "Parallelism-Aware Batch Scheduling: Enhancing Both Performance and Fairness of Shared DRAM Systems," in *ISCA*, 2008.
[54] North Carolina State Univ., "FreePDK45," http://www.eda.ncsu.edu/wiki/FreePDK.
[55] S. O *et al.*, "Row-Buffer Decoupling: A Case for Low-Latency DRAM Microarchitecture," in *ISCA*, 2014.
[56] J. K. Ousterhout, "Why Aren't Operating Systems Getting Faster as Fast as Hardware?" in *USENIX Summer Conf.*, 1990.
[57] T. Piquet *et al.*, "Exploiting Single-Usage for Effective Memory Management," in *ACSAC*, 2007.
[58] J. Poovey *et al.*, "DynoGraph," https://github.com/sirpoovey/DynoGraph.
[59] M. Qureshi *et al.*, "A Case for MLP-Aware Cache Replacement," in *ISCA*, 2006.
[60] M. Qureshi and Y. Patt, "Utility-Based Cache Partitioning: A Low-Overhead, High-Performance, Runtime Mechanism to Partition Shared Caches," in *MICRO*, 2006.
[61] M. K. Qureshi *et al.*, "Adaptive Insertion Policies for High-Performance Caching," in *ISCA*, 2007.
[62] S. Rixner *et al.*, "Memory Access Scheduling," in *ISCA*, 2000.
[63] M. Rosenblum *et al.*, "The Impact of Architectural Trends on Operating System Performance," in *SOSP*, 1995.
[64] Y. Sato *et al.*, "Fast cycle RAM (FCRAM): A 20-ns Random Row Access, Pipe-Lined Operating DRAM," in *VLSIC*, 1998.
[65] S.-Y. Seo, "Methods of Copying a Page in a Memory Device and Methods of Managing Pages in a Memory System," U.S. Patent Application 20140185395, 2014.
[66] V. Seshadri *et al.*, "The Dirty-Block Index," in *ISCA*, 2014.
[67] V. Seshadri *et al.*, "Fast Bulk Bitwise AND and OR in DRAM," *CAL*, 2015.
[68] V. Seshadri *et al.*, "The Evicted-Address Filter: A Unified Mechanism to Address Both Cache Pollution and Thrashing," in *PACT*, 2012.
[69] V. Seshadri *et al.*, "RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization," in *MICRO*, 2013.
[70] A. Snavely and D. Tullsen, "Symbiotic Jobscheduling for a Simultaneous Multithreading Processor," in *ASPLOS*, 2000.
[71] Y. H. Son *et al.*, "Reducing Memory Access Latency with Asymmetric DRAM Bank Organizations," in *ISCA*, 2013.
[72] Standard Performance Evaluation Corp., "SPEC CPU2006 Benchmarks," http://www.spec.org/cpu2006.
[73] L. Subramanian *et al.*, "The Blacklisting Memory Scheduler: Achieving High Performance and Fairness at Low Cost," in *ICCD*, 2014.
[74] K. Sudan *et al.*, "Micro-Pages: Increasing DRAM Efficiency with Locality-Aware Data Placement," in *ASPLOS*, 2010.
[75] T. Takahashi *et al.*, "A Multigigabit DRAM Technology with 6F2 Open-Bitline Cell, Distributed Overdriven Sensing, and Stacked-Flash Fuse," *JSSC*, 2001.
[76] Transaction Performance Processing Council, "TPC Benchmarks," http://www.tpc.org/.
[77] A. N. Udipi *et al.*, "Rethinking DRAM Design and Organization for Energy-Constrained Multi-Cores," in *ISCA*, 2010.
[78] H. Usui *et al.*, "DASH: Deadline-Aware High-Performance Memory Scheduler for Heterogeneous Systems with Hardware Accelerators," *TACO*, 2016.
[79] T. Vogelsang, "Understanding the Energy Consumption of Dynamic Random Access Memories," in *MICRO*, 2010.
[80] S. Wong *et al.*, "A Hardware Cache memcpy Accelerator," in *FPT*, 2006.
[81] H. Yoon *et al.*, "Row Buffer Locality Aware Caching Policies for Hybrid Memories," in *ICCD*, 2012.
[82] T. Zhang *et al.*, "Half-DRAM: A High-Bandwidth and Low-Power DRAM Architecture from the Rethinking Of Fine-grained Activation," in *ISCA*, 2014.
[83] L. Zhao *et al.*, "Hardware Support for Bulk Data Movement in Server Platforms," in *ICCD*, 2005.
[84] W. Zuravleff and T. Robinson, "Controller for a Synchronous DRAM That Maximizes Throughput by Allowing Memory Requests and Commands to Be Issued Out of Order," U.S. Patent 5630096, 1997.