ETH Library

Sectored DRAM: An Energy-Efficient High-Throughput and Practical Fine-Grained DRAM Architecture

Working Paper

Author(s):
Olgun, Ataberk; Bostanci, F. Nisa; Oliveira, Geraldo F.; Tuğrul, Yahya Can; Bera, Rahul; Yağlikçi, A. Giray; Hassan, Hasan; Ergin, Oğuz; Mutlu, Onur

Publication date:
2022-08-01

Permanent link:
https://doi.org/10.3929/ethz-b-000595585

Rights / license:
Creative Commons Attribution 4.0 International

Originally published in:
arXiv, https://doi.org/10.48550/ARXIV.2207.13795

This page was generated automatically upon download from the ETH Zurich Research Collection.
For more information, please consult the Terms of use.

Sectored DRAM: An Energy-Efficient High-Throughput and Practical Fine-Grained DRAM Architecture

Ataberk Olgun[§] F. Nisa Bostancı[§†] Geraldo F. Oliveira[§] Yahya Can Tuğrul[§†] Rahul Bera[§]
A. Giray Yağlıkcı[§] Hasan Hassan[§] Oğuz Ergin[†] Onur Mutlu[§]
[§]*ETH Zürich* [†]*TOBB University of Economics and Technology*

arXiv:2207.13795v2 [cs.AR] 1 Aug 2022

There are two major sources of inefficiency in computing systems that use modern DRAM devices as main memory. First, due to *coarse-grained* data transfers (size of a cache block, usually 64 B) between the DRAM and the memory controller, systems waste energy on transferring data that is not used in many workloads where a large fraction of words in a cache block is *not* used. Second, due to *coarse-grained* DRAM row activation, systems waste energy by activating DRAM cells that are unused in many workloads where spatial locality is lower than the large row size (usually 8–16 kB).

We propose Sectored DRAM, a new, low-overhead DRAM substrate that alleviates the two inefficiencies, by enabling *fine-grained* DRAM access and activation. To efficiently retrieve only the useful data from DRAM, Sectored DRAM exploits the observation that many cache blocks are not fully utilized in many workloads due to poor spatial locality. Sectored DRAM predicts the words in a cache block that will likely be accessed during the cache block's cache residency and: (i) transfers *only* the predicted words on the memory channel, as opposed to transferring the entire cache block, by dynamically tailoring the DRAM data transfer size for the workload and (ii) activates a smaller set of cells that contain the predicted words, as opposed to activating the entire DRAM row, by carefully operating physically isolated portions of DRAM rows (MATs). Activating a smaller set of cells on each access relaxes DRAM power delivery constraints and allows the memory controller to schedule DRAM accesses faster. Thereby, Sectored DRAM improves memory latency and system performance for many workloads that frequently and irregularly access memory.

Compared to prior work in fine-grained DRAM, Sectored DRAM greatly reduces DRAM energy consumption, does *not* reduce DRAM throughput, and can be implemented with low hardware cost. We evaluate Sectored DRAM using 41 workloads from widely-used benchmark suites. Compared to a system with coarse-grained DRAM, Sectored DRAM reduces the DRAM energy consumption of highly-memory-intensive workloads by up to (on average) 33% (20%) while improving their performance by up to (on average) 36% (17%). Sectored DRAM's DRAM energy savings, combined with its system performance improvement, allows system-wide energy savings of up to 23%.

1. Introduction

DRAM is hierarchically organized to improve scaling in density and performance. At the highest level of the hierarchy, to enhance parallelism, DRAM arrays are partitioned into banks that can be accessed simultaneously [1–4]. At the lowest level, a collection of DRAM rows (DRAM cells that are activated together) are typically divided into multiple *DRAM MATs* that can operate individually [5–8]. This allows for easier routing of the wires that connect the DRAM cells to command and data buses and improves access latency by reducing capacitive loads on long wires [5–8]. DRAM MAT organization provides a useful basis for low-cost DRAM substrates that improve DRAM energy efficiency.

Even though current DRAM chips are hierarchically organized, standard DRAM interfaces (e.g., DDRx [9–11]) do *not* expose DRAM MATs to the memory controller. To access even a single DRAM cell, the memory controller first needs to activate a large number of DRAM cells (e.g., 65'536 DRAM cells in a DRAM row in DDR4 [12]) and then transfer many bits (e.g., a cache block, typically 512 bits [13]) over the memory channel [12]. Thus, in current systems, both DRAM access and activation are *coarse-grained.* Coarse-grained access and activation cause significant energy inefficiency in systems that use DRAM as main memory for two major reasons.

First, coarse-grained DRAM access (at cache block granularity) causes unnecessary data movement between the processor and DRAM. Standard DRAM interfaces transfer data at cache block granularity over fixed-size data transfer bursts (e.g., 8 cycle bursts in DDR4 [10]), but a large fraction of data (§3) in a cache block is not used (i.e., referenced by CPU load/store instructions) during the cache block's residency in the cache hierarchy [14–19]. Thus, transferring unused portions of a cache block over the power-hungry memory channel wastes energy [20–30].

Second, coarse-grained DRAM activation causes an unnecessarily large amount of DRAM cells to be activated with each DRAM access. Although subsequent accesses to activated cells can be served faster, many modern memory-intensive workloads with irregular access patterns cannot benefit from fast accesses as the spatial locality in these workloads is lower than the DRAM row size [20, 31–38]. Thus, the energy cost of activating all cells in a DRAM row is not amortized over many accesses to the same row, leading to energy waste from activating a disproportionately large amount of cells.

Prior works develop DRAM substrates that enable *fine-grained* DRAM row activation, allowing a small number of DRAM cells to be activated with each DRAM access, and eliminate the energy wasted by activating unnecessarily large amount of DRAM cells [20–27, 39, 40]. However, fine-grained activation alone cannot eliminate the energy inefficiency caused by coarse-grained DRAM access. Moreover, these works have at least one of the three shortcomings: they (i) propose intrusive modifications to DRAM array circuit and organization [21–25], (ii) greatly reduce the throughput of DRAM data transfers [27, 40], (iii) introduce considerable DRAM chip area overhead [26, 39]. One prior DRAM substrate [20] enables *fine-grained DRAM activation* and a limited form of *fine-grained DRAM access*, allowing the processor to write to DRAM at word granularity (e.g., 8 bytes) instead of cache block granularity. Because this substrate does *not* allow the processor to read from DRAM at word granularity, it does *not* fully exploit fine-grained DRAM access (§3.1). We show that fully exploiting fine-grained DRAM access is critical to eliminate energy waste and thus justify the costs of enabling fine-grained DRAM access (§7.4).[1]

Our **goal** is to develop a new, low-cost, and high-throughput DRAM substrate that can mitigate the excessive energy consumption from both (i) transmitting unused data on the memory channel and (ii) activating a disproportionately large amount of DRAM cells. To this end, we develop Sectored DRAM. Sectored DRAM implements (i) Variable Burst Length (VBL) to enable fine-grained DRAM access, and Sectored Activation (SA) to enable fine-grained DRAM activation.

Sectored DRAM eliminates the need to transfer unused data over the memory channel by effectively enabling control over DRAM access granularity using VBL. VBL allows the memory controller to send and receive *dynamically-sized* portions of a cache block over the memory channel. VBL leverages the key observation that DRAM data transfers occur over multiple cycles in a burst. In each cycle, a portion of the cache block is transferred. VBL dynamically adjusts the number of cycles in a burst to transfer different portions of the cache block with each cycle, thus enabling fine-granularity DRAM access. To do so at low cost, VBL builds on existing DRAM I/O circuitry that already implements a mechanism to select one portion of a cache block to transfer in one cycle of a burst. Deciding which portions of a cache block to retrieve from DRAM is critical for maintaining high system performance. Portions that are not fetched but later referenced by load/store instructions cause additional high-latency memory accesses, potentially degrading system performance.

We develop two techniques, (i) Load/Store Queue (LSQ) Lookahead and (ii) Sector Predictor (SP) to effectively integrate VBL into a system. First, LSQ Lookahead accumulates the cache block offsets accessed by younger load/store instructions in older load/store instructions' memory requests. Thus, the execution of a load/store instruction prefetches the portions of cache blocks that will be accessed by the in-flight (i.e., not yet executed) load/store instructions. Second, SP predicts which portions of a cache block will be accessed by a load/store instruction based on that instruction's past cache block usage patterns. This allows SP to accurately predict the portions of a cache block that will be used by the processor during the cache block's residency in the cache hierarchy. The two techniques together minimize the number of additional high-latency memory accesses that occur when a portion of a cache block is missing in the cache hierarchy, allowing Sectored DRAM to provide significant DRAM access energy savings.

SA allows Sectored DRAM to activate a smaller set of DRAM cells on a DRAM row that fully store the portions of a cache block requested by a memory access. To enable SA with low hardware cost, Sectored DRAM builds on the structures that already exist in the lowest level of the DRAM organization hierarchy. We make the key observation that DRAM rows are already partitioned into independent MATs that can be individually activated with small modifications to the DRAM chip. We refer to parts of a DRAM row that can be individually activated as a *sector.* [2] SA (i) implements *sector transistors* that are each turned on to activate one of the independent MATs, and (ii) *sector latches* that control the sector transistors. SA exposes the sector latches to the memory controller by making use of an existing DRAM command (§4.1), thereby keeping the overhead of enabling fine-granularity activation to a minimum. As the power required to activate a MAT in a DRAM row is only a fraction of the power required to activate the whole row, Sectored DRAM also relaxes the power delivery constraints in DRAM chips [20, 39]. This allows for the activation of DRAM rows at a higher rate, increasing memory-level parallelism for memory-intensive workloads that have irregular access patterns.

We evaluate the performance and energy of Sectored DRAM using 41 workloads from SPEC2006/2017 [41, 42] and DAMOV [43] benchmarks. Sectored DRAM significantly reduces system energy consumption and improves system performance for memory-intensive workloads with irregular access patterns. Compared to a system with coarse-grained DRAM, Sectored DRAM reduces DRAM energy consumption by 20%, improves system performance by 17% on average, and reduces system energy consumption by 14% on average. We estimate the DRAM area overheads of Sectored DRAM using CACTI [44] and find that it can be implemented with low hardware complexity. Sectored DRAM incurs 0.39 mm^2 DRAM area overhead (1.7% of a DRAM chip) and does *not* require modifications to the physical DRAM interface.

We make the following contributions:

- We introduce Sectored DRAM and its two key mechanisms Variable Burst Length and Sectored Activation. Sectored DRAM improves system performance and alleviates the energy consumed on the memory channel by enabling fine-grained DRAM access and activation.
- We develop two techniques (LSQ Lookahead and SP) to effectively integrate Sectored DRAM into a typical computing system. Our techniques reduce the number of additional high-latency memory accesses by 82% as they accurately identify the portions of a cache block will be used by the

[1]No prior work develops a DRAM substrate that enables only fine-grained DRAM access.

[2]We use the word *sector* to distinguish between what exists today in DRAM chips (MATs) and what is proposed by Sectored DRAM (sectors).

processor.

- We rigorously evaluate Sectored DRAM's system performance and energy savings using workloads from SPEC2006/2017 [41, 42] and DAMOV [43] benchmark suites. Compared to a system with coarse-grained DRAM, Sectored DRAM reduces DRAM energy consumption by 20%, improves system performance by 17%, and reduces system energy by 14% on average across memory-intensive workloads. We find that implementing the proposed Sectored DRAM design takes up only 1.7% of DRAM chip area.

2. DRAM Background

We provide relevant background on DRAM organization. For more detailed background on DRAM, we refer the reader to prior works [8, 45–65].

2.1. DRAM Organization

Fig. 1 illustrates the hierarchical DRAM organization. The *memory controller* in the processor is connected to multiple DRAM *modules* over multiple *memory channels*. One or multiple *ranks* of DRAM *chips* constitute a DRAM module, where all ranks share a single memory channel. All DRAM chips in a rank operate in lockstep. A DRAM chip has multiple DRAM *banks* that can be accessed in parallel. All banks in a DRAM chip share the same set of inputs and outputs.

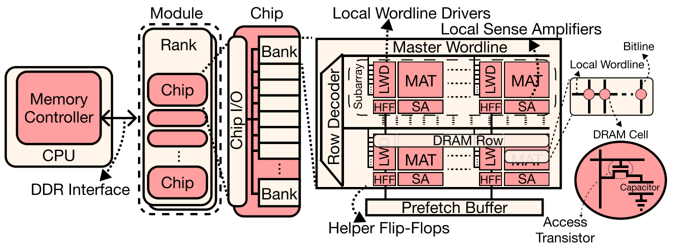

Figure 1: DRAM organization.

A DRAM bank consists of a set of *global sense amplifiers* (alternatively, the *prefetch buffer*), one *row decoder*, and multiple *subarrays*. Each subarray contains a set of *local wordline drivers*, *sense amplifiers*, *helper flip-flops (HFFs)*, and *MATs*. Inside every MAT, *DRAM cells* are placed onto a two-dimensional array of *local wordlines* and *bitlines*. A DRAM cell constitutes of an *access transistor* and a *capacitor*. DRAM cells that lie on the same local wordline across different MATs form a *DRAM row*.

2.2. Accessing DRAM

The memory controller accesses DRAM in two steps [9–11, 66, 67]. First, the memory controller sends an $ACTIVATE$ (ACT) command with a *row address* to enable (i.e., make accessible) a DRAM row. The row decoder drives the master wordline corresponding to the higher-order bits of the row address. The master wordline enables a single local wordline in every MAT in a subarray depending on the lower-order bits of the row address. Consequently, all cells on the same DRAM row start sharing the charge in their capacitors with their bitlines as the access transistors are enabled. Then, the local sense amplifiers amplify the voltage difference on the bitline and read out the value encoded in each cell.

Second, the memory controller sends a $READ$ command with a *column address* to retrieve multiple bytes of data (e.g., 8 B for an x8 DDR4 chip [10]) from the enabled DRAM row. The $READ$ command copies the data in the local sense amplifiers, over the HFFs, to the prefetch buffer. Thus, the *throughput* of internal DRAM data transfers (i.e., between the global and local sense amplifiers) is constrained by the number of HFFs per MAT. Once the data is in the prefetch buffer, the DRAM chip sends the data to the memory controller over the memory channel in a *burst*.

Row Buffer. Once a row is enabled, subsequent $READ$ and $WRITE$ commands targeting that row can be served at a fast rate (i.e., row is buffered). An access that targets an enabled row is referred to as a *row buffer hit*. An access that targets a row other than the enabled row in a bank causes a *row buffer conflict*. When no row in a bank is enabled and the bank is targeted by an access, the access causes a *row buffer miss*.

Accessing Another Row. When a bank already has an enabled row, and the memory controller wants to access a disabled row, the memory controller first issues a $PRECHARGE$ (PRE) command to disable the already-enabled DRAM row.

2.3. Data Transfer Bursts

DRAM modules transfer data on the memory channel over multiple *interface cycles*. For example, a $READ$ command transfers a cache block (512 bits) over eight interface cycles in a DDR4 module [12]. Each such transfer is referred to as a burst, and the *burst length* defines the number of double-data-rate interface cycles it takes to transfer the data.

Accesses to DRAM-based main memory have a granularity of a cache block. A cache block is partitioned across chips in fixed size blocks (e.g., if there are eight chips, each chip receives $\frac{1}{8}$ of the cache block). These fixed size blocks are further split into multiple MATs inside a bank in the chip. We depict how a cache block is scattered across multiple chips and MATs for a DDR4 module in Fig. 2 (top). Fig. 2 (bottom) shows the timing diagram of the command (cmd) and data buses during a $WRITE$ transfer, which happens in three steps. First, the memory controller drives 64 *DQ* signals (data) to transfer a 64-bit portion of the cache block in each *beat* (i.e., cycle) of the burst. Second, each chip receives 8 bits in a beat as each set of 8 DQ signals is routed to one of the chips. A chip accumulates 64 bits of data during the burst and places this data in its prefetch buffer. Third, these 64 bits are copied into the MATs inside the chip. Only 8 bits of data is transferred in a burst to (from) a MAT with a $WRITE$ ($READ$) command. Thus, the *maximum DRAM throughput* can *only* be obtained if every MAT contributes 8 bits to the data transfer burst.

2.4. The t_{FAW} Timing Parameter

DDRx specifications define the t_{FAW} timing parameter, which specifies the time window where no more than four ACT commands are allowed (i.e., the memory controller can only schedule four ACT commands in any t_{FAW}-wide time window) [9–11]. t_{FAW} allows correctly delivering the power that is required to activate large DRAM rows in a DRAM chip.

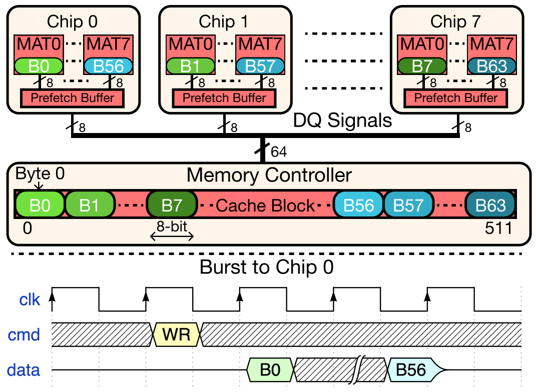

Figure 2: DRAM example cache block placement.

However, it limits row activation frequency and diminishes the memory-level parallelism, degrading the performance of memory intensive workloads [68].

3. Motivation

A modern DRAM system exploits applications' spatial locality on two levels. First, the system exploits *word-level spatial locality* by transferring an entire cache block, with multiple words[3], from DRAM to the memory controller over a single memory burst (*coarse-grained DRAM access*). Second, the system exploits *cache block-level spatial locality* by activating an entire DRAM row, with multiple cache blocks, during a read-/write operation (*coarse-grained DRAM activation*). Coarse-grained DRAM access and activation can increase DRAM throughput and reduce main memory access latency for applications with enough spatial locality. However, it can also increase energy consumption for applications with inherent low spatial locality [15, 18, 19, 39, 69].

We study the impact of coarse-grained DRAM access (*Coarse-DRAM-Access*) and activation (*Coarse-DRAM-Act*) for a wide range of applications to understand their impact on system and DRAM energy consumption. To do so, we execute 41 single-core workloads from a wide range of workload domains from the DAMOV [43], SPEC2006 [41], and SPEC2017 [42] benchmark suites (see §6 for our evaluation methodology). We compare their energy consumption to a system that performs (i) fine-grained DRAM access (*Fine-DRAM-Access*) in word granularity (i.e., the memory controller can transfer individual words from/to DRAM), and (ii) fine-grained DRAM activation (*Fine-DRAM-Act*) in MAT granularity (i.e., DRAM MATs can be activated individually).

Fig. 3 (left) shows the DRAM access energy across workloads from our three benchmark suites for the *Coarse-DRAM-Access* system, normalized to the *Fine-DRAM-Access* system. We observe that the DRAM access energy of the *Coarse-DRAM-Access* system is 1.27× that of the *Fine-DRAM-Access*. The large increase in energy consumption in the *Coarse-DRAM-Access* system is caused by retrieving words in a cache block that the processor does *not* entirely use. This leads to a 45% increase in the data movement between DRAM and the CPU in the *Coarse-DRAM-Access* system, on average. We conclude that current systems that perform coarse-grained DRAM accesses significantly waste energy by transferring a fixed number of words from DRAM to the processor, which are not entirely used by the processor.

[3]We refer to each non-overlapping 64-bit portion of data in a cache block as a word.

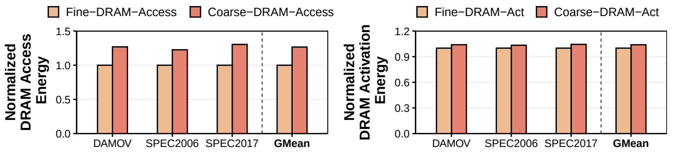

Figure 3: Normalized DRAM access (left) and DRAM activation (right) energy consumption.

Fig. 3 (right) shows the DRAM activation energy across workloads from our three benchmark suites for the *Coarse-DRAM-Act* system, normalized to the *Fine-DRAM-Act* system. We observe that the DRAM activation energy of the *Coarse-DRAM-Act* system is 1.04× that of the *Fine-DRAM-Act* system. Like the system that performs coarse-grained DRAM accesses, the increase in energy consumption in the coarse-grained DRAM activation system is caused by activating a large, fixed-size DRAM row that the processor does *not* entirely use. As prior works show [20, 31–38], such an increase in energy consumption when executing coarse-grained DRAM activation is because modern memory-intensive workloads with irregular access patterns suffer from low spatial locality, which reduces the benefit of a large DRAM row buffer. We conclude that systems that employ coarse-grained DRAM activation suffer from energy inefficiency.

3.1. Enabling Fine-Grained DRAM: Challenges and Limitations

Efficiently enabling fine-grained DRAM access and activation can significantly improve system energy. However, three main challenges must be overcome in doing so.

(1) Maintaining High DRAM Throughput. Current DRAM systems leverage coarse-grained accesses to maximize DRAM's throughput since throughput directly impacts applications' performance. Thus, it is critical to sustain high DRAM throughput when enabling fine-grained DRAM. However, naively enabling fine-grained DRAM, particularly Fine-DRAM-Act, reduces DRAM throughput as one MAT contributes only a fraction of the total DRAM internal throughput (§2.3). This issue can be alleviated by increasing the number of the HFFs. However, this approach is not practical since it severely complicates DRAM array routing and leads to significant DRAM area overheads [20, 39].

(2) Incurring Low DRAM Area Overhead. DRAM manufacturing is highly optimized for density and cost [70]. While enabling fine-grained DRAM, one must avoid applying intrusive modifications to the DRAM array since such modifications are difficult to integrate into real designs.

(3) Fully Exploiting Fine-Grained DRAM. The energy waste of current coarse-grained DRAM systems stems from rigid DRAM access and activation granularities. Thus, a fine-grained DRAM system must enable flexible DRAM access *and* activation granularities for *both* read and write operations to eliminate such energy waste. However, integrating such a fine-grained DRAM into current systems is challenging as systems are typically designed to access DRAM in cache block granularity.

Several prior works [20–27, 39, 40] propose different mechanisms to enable fine-grained DRAM substrates, aiming to alleviate the energy waste caused by coarse-grained DRAM. Such works can be divided into two broader groups: (1) works that propose *intrusive* modifications to the DRAM array circuit and organization (e.g., new DRAM interconnects, considerably more HFFs) [21–25] and (2) works that aim to enable coarse-grained DRAM *without intrusive* modifications to DRAM [20, 26, 27, 39, 40]. The intrusive DRAM modifications proposed by the first group lead to significant DRAM area overheads, which makes it difficult to integrate this group of works into real DRAM designs.

Table 1 qualitatively compares how prior works from the second group address the three challenges of enabling fine-grained DRAM. We observe that no prior work can *simultaneously* provide (i) high DRAM throughput (FGA [40] and SBA [27] change the cache block mapping such that DRAM transfers can be served from only one MAT, but reduce the throughput of data transfers in doing so); (ii) low area overhead (HalfDRAM [39] and HalfPage [26] require changes to the number and organization of DRAM's HFFs, leading to non-negligible area overheads); and (iii) mechanisms that fully exploit fine-grained DRAM (PRA [20] only enables fine-grained DRAM access and activation for write operations; HalfDRAM, HalfPages, FGA, and SBA still impose a rigid DRAM access granularity). We conclude that no prior work efficiently enables fine-grained DRAM access and activation.

Table 1: Sectored DRAM vs. prior works.

	High Throughput	Low Area Overhead	Fully Exploit Fine-Grained DRAM
FGA [40]	✗	✓	✗
SBA [27]	✗	✓	✗
HalfDRAM [39]	✓	✗	✗
HalfPage [26]	✓	✗	✗
PRA [20]	✓	✓	✗
This Work	✔	✔	✔

Our **goal** is to address prior works' limitations while efficiently mitigating the excessive energy consumed by transferring unused data on the memory channel and activating unused DRAM cells. We aim to maintain high DRAM throughput, incur low area overhead, and fully exploit fine-grained DRAM access and activation. To this end, we develop Sectored DRAM, a new, practical and high-performance fine-grained DRAM substrate.

4. Sectored DRAM

We introduce Sectored DRAM, the first fine-grained DRAM substrate that mitigates the energy inefficiencies in DRAM operation while improving system performance. To do so, we leverage two key observations regarding DRAM chip design. First, we observe that the hierarchical structure within a DRAM subarray naturally splits DRAM rows into fixed-size portions. Second, the DRAM I/O circuitry already implements a mechanism to select one portion of a cache block to transfer it in one beat of a burst.

We leverage these two observations to implement Sectored DRAM, a fine-grained DRAM substrate that enables energy savings and performance improvements via fine-grained DRAM activation and access at a low area overhead. Sectored DRAM consists of two new mechanisms implemented in a DRAM chip: Sectored Activation (SA) and Variable Burst Length (VBL). SA enables fine-grained control over the activation of sectors in DRAM by making minimal modifications to how local wordlines are driven. VBL enables fine-grained control over data transfer bursts by transferring only the portions of a cache block that correspond to the activated sectors in the DRAM chip.

We expose the two mechanisms to the memory controller such that the system can benefit from Sectored DRAM, with *no* changes to the physical DRAM interface and only small changes to DRAM interface specification.

4.1. Sectored Activation

Fig. 4-A depicts the architecture of a DRAM array with 8 MATs in one subarray [5, 26, 71]. We make a **key observation** from this figure. The hierarchical structure within the subarray naturally splits DRAM rows into fixed-size portions (i.e., MATs). SA augments these portions by allowing them to be activated individually. We refer to these augmented portions as *sectors*.

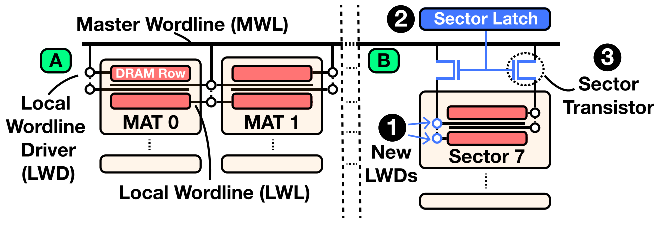

Figure 4: Wordline organization in a subarray.

SA Design. To implement SA, we propose minor modifications to the existing architecture. Fig. 4-B depicts our modifications to the DRAM subarray with 8 MATs (the modifications are highlighted in blue color). First, we insert new (❶) local wordline drivers (LWDs) such that each LWD drives only one local wordline (LWL). Thus, by enabling a single LWD, we make sure that only the cells that belong in a single MAT get activated, as opposed to cells from multiple MATs getting activated in the existing architecture (e.g., LWDs between MAT 0 and MAT 1 in Fig. 4-A drive two LWLs that extend onto both of the MATs). Second, we place one *sector latch* (❷) for every sector in the horizontal direction to control the activation of individual sectors in every bank. Third, we isolate the master wordline (MWL) from the LWDs using *sector transistors* (❸). With the proposed modifications in place, sectors that have their corresponding latches set will be activated when the MWL is driven (with an ACT command), because two sector transistors will connect the MWL to the LWDs.

Exposing SA. To make use of SA, the memory controller needs to be able to set and reset sector latches. We propose exposing the control of sector latches to the memory controller via the unused signals in DDR4 command encoding [10]. Doing so allows SA to be implemented with minor modifications to the memory controller and the DRAM chip's I/O control logic, and no modifications to the DRAM interface signals (i.e., the physical interface remains the same). We use the unused bits in the PRE command's encoding to store the *sector bits*. Each sector bit sets (when logic-1) or resets (when logic-0) a sector latch. When the memory controller sends a PRE to disable the activated row in a bank, it also encodes within the PRE command the sector bits that will activate the required sectors for the next ACT command to the same bank. When the bank is closed (i.e., no enabled rows in the bank), the memory controller schedules a PRE command before the first ACT command to convey the sector bits. The slack between successive PRE and ACT commands targeting the same bank (t_{RP}, $\sim 13\ ns$ in DDR4 [10, 12]) ensures that the sector bits can be propagated from the DRAM chip's inputs to the sector latches.

Because activating one sector requires considerably less power (§7.1) than activating all sectors, the t_{FAW} timing constraint can be relaxed for successive ACT commands that target different subsets of sectors according to the amount of sectors activated with each command [20, 24, 39].

In our Sectored Activation (SA) implementation, we consider DRAM subarrays with 8 sectors. We discuss how Sectored DRAM can implement a finer-granularity activation mechanism (i.e., with a larger number of sectors) in §8.2.

4.2. Variable Burst Length

The goal of VBL is to allow the DRAM chip to transmit (i.e., $READ$) and receive (i.e., $WRITE$) data in variable lengths of bursts such that each beat of the burst transfers only the data corresponding to one of the enabled sectors.

VBL Design. Fig. 5 depicts the I/O read/write circuitry of modern DRAM chips [12]. In such a chip, data is first moved from the DRAM array to the *Read FIFO* (❶), in case of a $READ$ transfer. The *Read FIFO* comprises eight entries, and each entry stores the data that will be transmitted over the DQ pins in one beat (i.e., DDR interface cycle) of the data transfer burst. The *Read MUX* (❷) selects one entry in the *Read FIFO* based on the value of the *burst counter* (not shown in the figure), which counts the number of beats in the transfer (e.g., from 0 to 7).

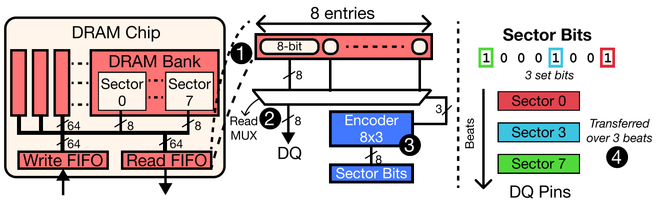

Figure 5: Variable Burst Length, example over $READ$ transfers.

By studying the I/O read/write circuitry of modern DRAM chips, we observe that current DRAM chips select (using the burst counter) individual entries in the *Read FIFO* to drive the DQ pins within a beat [12]. Based on that, VBL's **key idea** is to modify the selection criteria slightly. To do so, we replace the burst counter with an $8{\times}3$ encoder (❸) that takes sector bits as input, and outputs only the indices for the *Read FIFO* entries that contain data from one of the enabled sectors. In this way, the *Read MUX* skips the entries in the *Read FIFO* that correspond to the disabled sectors, driving the DQ pins *only* with the data that comes from the enabled sectors (❹ in Fig. 5, right). Since the *Write FIFO* is organized in the same way as the *Read FIFO*, we reuse the same encoder for $WRITE$ transfers to correctly fill the entries in the *Write FIFO* upon the receipt of a data transfer burst.

Exposing VBL. To make use of VBL, the memory controller and the DRAM chip need to agree on the length of the burst *before* the data transfer starts. This is important for both parties to calibrate their I/O drivers correctly and capture the signals on the high-frequency DRAM interface [9–11]. Since VBL enables dynamically sized data transfer bursts, the memory controller and the DRAM chip could need to agree on a differently burst length per data transfer. A straightforward approach to communicate the burst length from the memory controller to the DRAM chip is to implement a new DRAM command that conveys the burst length. However, using this command would result in the DRAM command bus to be underutilized, and thus, the data transfer throughput to be reduced, as the memory controller would need to schedule the new command prior to every $READ$ and $WRITE$ access. Instead, we do *not* implement a new command. We take a different approach where we use the sector bits that are already communicated to the DRAM chip by fine-grained DRAM activation operations to determine the burst length (§4.1).

We make two modifications to enable the described approach. First, we implement a low overhead 8-bit *popcount* circuit [72, 73] in both the DRAM chip and the memory controller to count the number of set sector bits in the DRAM bank targeted by a $READ$ or a $WRITE$ access. The popcount circuitry requires *only* 34 logic gates to be implemented [73], introducing negligible area overhead. Second, we extend the *bank state table* of the memory controller with sector bits. The bank state table already contains metadata, such as the address of the enabled row, for every bank. The additional storage requirement for sector bits in the bank state table is small, only 128 bits (8 bits for each bank [10, 12]).

5. System Integration

We describe the challenges in integrating Sectored DRAM into a typical system and propose solutions. We assume that the system uses a DDR4 module with *eight chips* as main memory and that each chip has *eight sectors* to explain the challenges and our solutions clearly.[4] Since there are eight sectors in every chip, one sector from each DRAM chip collectively stores *one word* (64 bits) of the cache block.

5.1. Integration Challenges

We identify two challenges in integrating the Sectored DRAM to a system. First, VBL allows transferring data at

[4]In §8, we discuss how Sectored DRAM can be integrated into systems with different parameters (e.g., more sectors per chip).

word granularity (instead of cache block granularity). Therefore, a cache block may have both *valid* (up to date) and *invalid* (stale or evicted) words present in the processor. However, the processor keeps track of the valid on-chip data at cache block granularity. This granularity is too coarse to keep track of valid words in a cache block. Second, because some words in a cache block can be invalid, references to these words (e.g., made by load/store instructions) would result in a cache miss. This can increase the average memory latency and induce performance overheads. Thus, the system must carefully retrieve all words in a cache block that will be referenced by load/store instructions during the cache block's residency in the caches (i.e., from the moment the cache block is brought to on-chip caches until the block is evicted).

We propose the following minor modifications to overcome the challenges in integrating Sectored DRAM. First, to track which words in a cache block are valid, we extend cache blocks with additional bits, each indicating if a 64-bit word in the cache block is valid. Second, to accurately retrieve all words within a cache block that will be used during the cache block's residency, we propose (i) a microarchitectural optimization in the processor's load/store queue, the LSQ Lookahead, and (ii) a useful word predictor, the Sector Predictor.

5.2. Tracking Valid Words

Since Sectored DRAM can retrieve individual words of a cache block from DRAM using VBL, on-chip caches must store data at a granularity that is finer than the typical 512-bit granularity. One straightforward approach in allowing word granularity storage in caches is to reduce the cache block size from 512 bits to 64 bits. However, this reduces the capacity of the cache by $8\times$. Reducing cache block size while maintaining the same cache capacity requires implementing $8\times$ as much storage for *cache block tags*, which introduces significant area overhead. Instead, we extend cache blocks with just eight additional bits that each indicate whether a word in the cache block is valid or invalid, using sectored caches [74–77].

Fig. 6 depicts the modifications required (in blue color) by our implementation. We extend cache blocks with sector bits that each indicate if a word in the cache block at that level in the cache hierarchy is valid (Fig. 6-a). We store sector bits in a CAM array (similar to how cache block tags are stored). This allows the system to find out if a *sector is missing* (i.e., a word is invalid) in the cache block in parallel with the tag lookup. We divide a cache block into eight sectors, each corresponding to a word. Thus, we only add eight sector bits to the cache block, keeping the additional storage requirements to a minimum (§7.5). We describe how the processor makes use of the sector bits field in the memory request in §5.3.

Fig. 6-a depicts how a cache block is accessed with a memory request. First, the processor sends a memory request with an address and sector bits (❶). The sector bits indicate the words in the cache block that the request will access. Second, the cache block index is extracted from the address to locate the cache block in the cache (❷). Third, the cache block tag extracted from the address together with the sector bits (❸) are

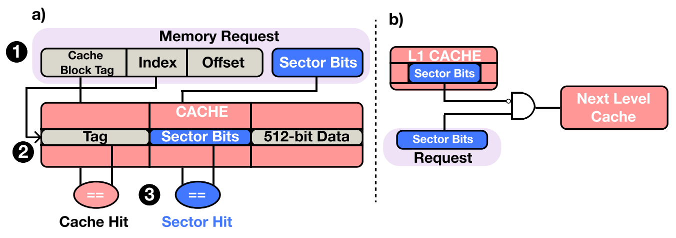

Figure 6: Cache with sector bits.

used to find out if (i) the cache block is missing (i.e., *cache miss*) or (ii) any of the sectors are missing (i.e., *sector miss*). If a cache miss occurs (i.e., the tag is not present), the system forwards the memory request to a lower-level cache or to DRAM (i.e., next component in the memory hierarchy). Upon receipt of the missing cache block, the system sets the sector bits in the cache block according to the sector bits in the memory request. If a cache miss does not occur but a sector miss occurs (i.e., the tag is present but some sectors are missing), the system will request the missing sectors from the next component in the memory hierarchy (Fig. 6-b). When the system receives the missing sectors, it sets the sector bits in the cache block accordingly.

We discuss various aspects (e.g., cache block management, coherency) of Sectored DRAM's implementation of sectored caches in detail:

Cache Block Evictions. When a cache block with a dirty sector is evicted from a higher level cache, the dirty sector in that cache block overwrites the copy of the same cache block in the lower level cache and updates the sector bits of the copy of the cache block in the lower level cache. A cache block without any dirty sectors is simply invalidated as nothing needs to be written back to a lower level component (e.g., L2 cache) in the memory hierarchy.

Cache Inclusion Policy. We implement Sectored DRAM using an inclusive caching policy for cache blocks. If a cache block is valid in a lower level cache, it is also valid in the higher level caches. A cache block might be invalid in a lower level cache, but it can be valid in a higher level cache. If a sector in a clean cache block is valid in a higher-level cache (e.g., L1), it is also valid in a lower level cache (e.g., L2). However, the CPU may update an invalid sector of a valid cache block in the higher level cache by executing store instructions (making the sector dirty in doing so). This dirty sector can be valid in the higher level cache and invalid in the lower level cache.

Cache Coherence. Sectored DRAM requires no modifications to existing cache coherence protocols that operate at the granularity of a cache block since cache coherence in Sectored DRAM is maintained at the granularity of a cache block. A core can only modify a sector in a cache block if the core owns the cache block (e.g., the cache block is in the M state in a MESI protocol). A cache block shared across multiple cores may have different valid sectors among its copies in different private caches. However, this does not violate cache coherence protocols.

Compatibility with SRAM ECC. Sector caches are compatible with existing ECC schemes. The only difference between a sector cache's cache block and a baseline cache's cache block is that some portions (sectors/words) of the data in the sector cache's cache block might be invalid.

We describe one way to support ECC in sectored caches. First, the cache controller fills the invalid sectors in a cache block with an all zeros or an all ones data pattern. Second, the cache controller performs an ECC encoding operation on the cache block's data. Third, the cache controller updates the cache's data array with the cache block's data and parity bits (i.e., the codeword). To update an invalid sector in the cache block (e.g., when a LOAD request causes a sector miss and the missing sector is brought from a lower level cache), the cache controller i) reads the cache block from the data array, ii) updates the missing sector in the cache block, and iii) performs ECC encoding on the updated cache block.

Other Cache Architectures. There are numerous other multi-granularity cache architectures [15–17,78–80] that could be used instead of sectored caches in Sectored DRAM to improve cache utilization (e.g., by reducing the number of invalid words stored in a cache block) at the cost of increased storage for tags and hardware complexity [15]. We use sectored caches to minimize the storage and hardware complexity overheads in Sectored DRAM and leave the exploration of other cache architectures in Sectored DRAM to future work.

5.3. Accurate Word Retrieval

With support for tracking valid words in cache blocks, the processor can access words individually (instead of retrieving cache blocks as a whole) by executing load/store instructions. We describe the simplest way that the system can make use of Sectored DRAM. To do so, the processor uses the sector bits field in a memory request and sets only the bit (among all eight bits) that corresponds to the word a load/store instruction needs to access. When the memory request is forwarded to the memory controller (by missing in all caches), the memory controller only retrieves the word indicated by the sector bits, benefiting from the performance improvements provided by SA and energy savings provided by both SA and VBL. However, even though cache blocks are not fully utilized by the processor (§3), workloads often access multiple words in a cache block during the cache block's cache residency. Therefore, retrieving only a single word with each memory access can result in a high number of sector misses per cache block (due to load/store memory requests to the missing words in a cache block), causing multiple high latency DRAM accesses to retrieve the missing words. Retrieving cache blocks word-by-word from DRAM can greatly reduce system performance compared to bringing cache blocks as a whole.

To minimize the performance overheads induced by additional DRAM accesses and to better benefit from the energy savings provided by Sectored DRAM, we propose two mechanisms that greatly reduce the number of sector misses.

5.3.1. Load/Store Queue (LSQ) Lookahead. LSQ Lookahead accumulates the words that are referenced by the load/store instructions in the processor's load/store queues. Using the accumulated word offsets, the mechanism *prefetches* the sectors that will be accessed by future load/store instructions. The key idea behind LSQ Lookahead is to exploit the spatial locality in subsequent load/store instructions that target the same cache block.

Fig. 7 depicts how LSQ Lookahead is implemented over an example. We extend each load address queue (LAQ) entry with sector bits that accumulate future references to different words in the same cache block.

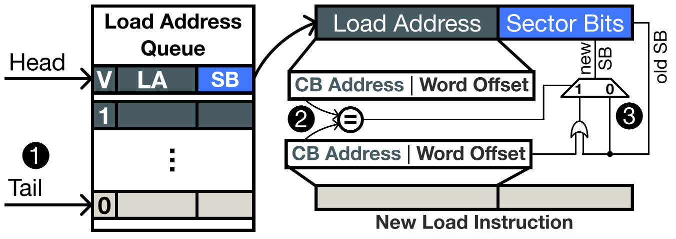

Figure 7: LSQ Lookahead mechanism.

LSQ Lookahead works in two steps. First, when a new entry is allocated at LAQ's tail (❶), the LSU compares the new entry's cache block address (CB address) with each of the existing entries' cache block addresses (❷). Second, when it finds a matching cache block address, it updates the existing entry's sector bits by setting the bit that corresponds to the word referenced by the new entry (❸). Thus, when a load request misses in the caches, the memory controller uses the sector bits to fetch multiple words that are likely to be referenced in the future by the load requests in the LAQ.

5.3.2. Sector Predictor. Although LSQ Lookahead prevents some of the sector misses by prefetching the words that will be used by the processor in the near future, it alone cannot significantly reduce the number of sector misses. This is because not all references to a word in a cache block is made by the immediately subsequent load/store instructions to the load/store instruction that brings the cache block to the cache. Therefore, we require a more powerful mechanism to complement LSQ Lookahead and minimize the number of sector misses.

We develop the Sector Predictor (SP) to further alleviate sector misses and reduce the amount of high-latency memory accesses in Sectored DRAM. The key idea of SP is to associate the words that are used in a cache block during the cache block's residency in the L1 cache to the signature (instruction address and memory address) of the memory instruction that brings the cache block to the L1 cache. The SP records the used words in the cache block in a table. The intuition behind SP is that when the same instruction is executed in the future and misses in the L1 cache, further references to the words in this missing cache block will be the same as the used words that are stored in the table. SP builds on a class of predictors referred to as spatial pattern predictors [14, 81]. We tailor SP for predicting the useful words in a cache block, similar to what is done by [19].

Fig. 8 depicts the organization of the SP. The Sector History Table (SHT) stores the *previously used sectors* that are used to predict the words in a missing cache block (❶). The table is accessed with a *table index* that is computed by XOR-ing parts of the instruction address with the *word offset* in the data address upon an L1 cache miss (❷). We extend the L1 cache to store the table index and *currently used sectors* (❸). The currently used sectors in the cache track which sectors are used during a cache block's residency in the L1 cache. The table index is used to update the previously used sectors in an entry in the SHT with the currently used sectors stored in the cache block (❹). Next, we briefly describe how the system manages the SP.

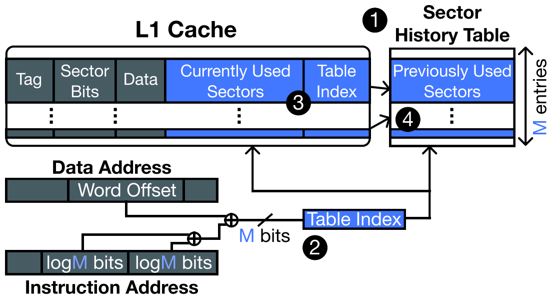

Figure 8: Sector predictor table and indexing.

When a memory request causes an L1 miss or a sector miss, the SHT is queried with the table index to retrieve the *used sectors*. These bits are added to the sector bits of the missing memory request and forwarded to the next level in the memory hierarchy. The table index of the newly allocated cache block (to store the missing cache block in) is updated with the table index used to access the SHT and the cache block's currently used sectors are set to 0 (i.e., none of the sectors are referenced by an instruction yet). Once the cache block is placed in the L1 cache, the cache block's currently used sectors will track the words that are accessed by load/store instructions. When the same block is evicted from the L1 cache, the SHT entry corresponding to the cache block's table index will be updated with the currently used sectors.

6. Evaluation Methodology

We describe the workloads (§6.1), power model (§6.2), and our simulation infrastructure used to evaluate Sectored DRAM's performance and energy (§6.3). Table 2 shows our system configuration.

Table 2: System configuration.

Processor	1-16 cores, 3.6 GHz clock frequency, 4-wide issue 8 MSHRs per core, 128-entry issue window 32 KiB L1, 256 KiB L2, 8 MiB L3 caches Dynamic Power: 101.7 W [82], Static Power: 32.0 W [82]
Mem. Ctrl.	64 entry read/write request queue, FR-FCFS-Cap [34] scheduling policy, Open-page row buffer policy Row-Bank-Rank-Column-Channel address mapping [58]
DRAM	DDR4 [10], 1600 MHz bus frequency, 1 channel 4 ranks, 16 banks/rank, 32K rows/bank 64 subarrays/bank, 8 sectors/subarray $tRCD/tRAS/tRC/tFAW$ 13.75/35.00/48.75/25 ns
Sectored DRAM	128-entry LSQ Lookahead (default) 512-entry Sector Predictor (default)

6.1. Workloads

We use 41 workloads from the SPEC2006 [41] (23 workloads), SPEC2017 [42] (12 workloads), and DAMOV [43] (6 workloads) to evaluate Sectored DRAM. For every workload, we generate memory traces corresponding to 100 million instructions from representative regions in each workload using SimPoint [83]. We classify the workloads into three memory intensity categories, which Table 3 describes, using their observed last-level-cache (LLC) misses-per-kilo-instruction (MPKI). We use LLC MPKI threshold values prior works report [43, 84] in our classification.

Multi-core workloads. We create 2-, 4-, 8-, and 16-core workloads by replicating the same single-core workload over multiple cores. We create 16 eight-core workload mixes for each memory intensity category by randomly picking eight single-core workloads from every category.

Table 3: Evaluated workloads. -2006/-2017 indicates SPEC workloads.

LLC MPKI	Workloads
≥ 10 (High)	ligraPageRank, mcf-2006, libquantum-2006, gobmk-2006, ligraMIS, GemsFDTD-2006, bwaves-2006, lbm-2006, lbm -2017, hashjoinPR
1..10 (Medium)	omnetpp-2006, gcc-2017, mcf-2017, cactusADM-2006, zeusmp-2006, xalancbmk-2006, ligraKCore, astar-2006, cactus-2017, parest-2017, ligraComponents
≤ 1 (Low)	splash2Ocean, tonto-2006, xz-2017, wrf-2006, bzip2-2006, xalancbmk-2017, h264ref-2006, hmmer-2006, namd-2017, blender-2017, sjeng-2006, perlbench-2006, x264-2017, deepsjeng-2017, gromacs-2006, gcc-2006, imagick-2017, leela-2017, povray-2006, calculix-2006

6.2. Power Model

DRAM Power Model. We use the Rambus Power Model [7] to model a DDR4 device (Table 2) that supports Sectored DRAM. We augment the Rambus Power Model to simulate Sectored Activation and Variable Burst Length for varying number of activated sectors and varying burst lengths. We modify the model to (i) enable a smaller number of local wordlines (i.e., activate a smaller number of sectors) and (ii) reduce the burst size of data transfers for partially activated DRAM rows. Our model considers the power overheads introduced by the sector transistors and the sector latches.

Processor Power Model. We use an IPC-based model [19, 85] to estimate the power consumed by the processor core. The total power our 8-core processor consumes, according to the IPC-based power model, is equal to: $\frac{IPC}{4} \times Dynamic\ Power + Static\ Power$. Our model includes the dynamic and static power consumed by the sector prediction and the additional cache storage Sectored DRAM requires, which we model using CACTI [44].

6.3. Performance and Energy

We evaluate Sectored DRAM's performance and energy using a modified version of Ramulator [58], a cycle-accurate, trace-based DRAM simulator, and a modified version of DRAMPower [86], a DRAM power and energy estimation tool. We extend Ramulator by implementing Sectored DRAM's LSQ Lookahead and Sector Predictor. We modify DRAMPower by integrating the Sectored DRAM's DRAM power we obtain from the Rambus Power Model.

Performance Metrics. We measure single-workload performance using *parallel speedup* (i.e., the baseline single-core execution time divided by the multi-core baseline/Sectored DRAM execution time), which allows us to evaluate Sectored DRAM's scalability for a single workload. We measure workload mix performance using *weighted speedup* [87], which allow us to evaluate Sectored DRAM's system throughput [88] in a heterogeneous computing environment.

7. Evaluation Results

We evaluate Sectored DRAM's impact on DRAM power, sector misses, performance, energy, and DRAM area.

7.1. Impact on DRAM Power

Fig. 9 shows Sectored DRAM's impact on DRAM power consumption. We analyze the DRAM array power and DRAM periphery circuitry power required by Sectored DRAM to perform ACT, $READ$, and $WRITE$ DRAM operations for 8, 4, 2, and 1 sectors. Values are normalized to the power required to perform the same operation using the baseline DDR4 module. We make three observations. First, activating only one sector greatly reduces the power consumed by the DRAM array compared to activating all eight sectors. Because SA enables activating a small set of DRAM sense amplifiers in a DRAM row ($\frac{1}{8}$ of all sense amplifiers in the row), activating a single sector consumes 66.5% less DRAM array power compared to activating eight sectors. However, we find that activating one sector reduces the overall power consumption of an ACT operation by only 12.7% compared to the baseline DDR4 module. The effect of reducing activation power on overall power consumption is small since the power consumed by the periphery circuitry makes up a large proportion of the activation power and is not affected by the number of sectors activated.

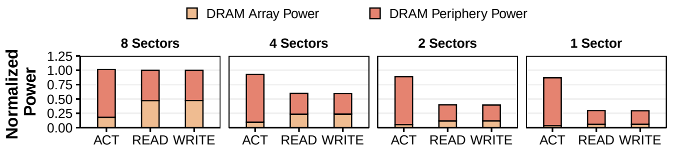

Figure 9: $ACT/READ/WRITE$ power for varying number of sectors.

Second, SA together with VBL significantly improve $READ$ and $WRITE$ power consumption. We find that the power consumed by the DRAM module while reading from and writing to a sector is 70.0% and 70.6% smaller than reading from and writing to all sectors, respectively. This improvement is due to the (i) reduced sense amplifier activity in the DRAM array, (ii) reduced switching on the DRAM periphery circuitry that transfers data between the DRAM array and the DRAM I/O, and (iii) smaller number of bursts to transfer data between the DRAM module and the memory controller. Third, the additional circuitry required to implement SA incurs little activation power overhead. Compared to the baseline DDR4 module, SA increases activation power by only 0.26% due to additional switching activity in master wordline drivers from the sector transistors used to control the activation of each sector (§4.1).

We conclude that Sectored DRAM's fine-grained DRAM access and activation provides a significant reduction in DRAM's power consumption.

7.2. Sector Misses

To quantify the number of sector misses (recorded when a cache block is present, but some sectors are missing, §5.2), we look at the LLC MPKI of workloads when they are executed with different Sectored DRAM configurations. Fig. 10 plots the LLC MPKI for different LSQ Lookahead (LA<*number*> where *number* is the number of entries looked ahead in the LSQ) and Sector Predictor (SP<*number*> where *number* is the number of entries in the SHT) configurations along with the *Basic* Sectored DRAM configuration which does not use LSQ Lookahead nor SP. Each bar shows the number of LLC MPKI for a Sectored DRAM configuration, clustered based on the three benchmarks suites we execute. We plot the average number of LLC MPKI for every single-core workload in all benchmarks.

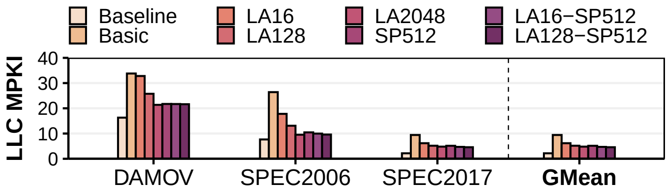

Figure 10: LLC MPKI for different Sectored DRAM configurations.

We make three major observations from Fig. 10. First, the basic configuration greatly increases the LLC MPKI of a workload, on average by 3.08×, compared to the baseline, due to sector misses. Second, LSQ Lookahead can reduce the number of additional LLC misses generated by the basic configuration by 39%, 65%, and 83% by looking ahead 16, 128, and a currently impractical 2048 younger entries in the LSQ, respectively. This is because LSQ Lookahead can identify the words that will be used in a cache block and retrieve these from DRAM with a single request when a cache miss occurs. Third, LSQ Lookahead together with the Sector Predictor (LA128-SP512) can reduce the number of additional LLC misses generated by the basic configuration by 82%, performing almost as good as the currently impractical implementation of LSQ Lookahead of 2048 entries. SP greatly reduces the number of additional LLC misses as it can pick up intra-cache-block access patterns from previously executed memory requests and correctly predict the words that will be used based on these patterns.

We conclude that LSQ Lookahead with a 128 lookahead size together with SP minimizes the additional LLC misses caused by sector misses. We use the LA128-SP512 configuration in the remainder of our evaluation.

7.3. Single-Workload Performance and Energy

We evaluate Sectored DRAM's performance and energy using (i) single-core workloads, and (ii) 2-, 4-, 8-, and 16-core multi-programmed workloads that are made up of identical single-core workloads. We compare Sectored DRAM's performance and system energy to a baseline coarse-grained DRAM system.

Performance. The two lines in Fig. 11 show the normalized parallel speedup (on the primary/left y-axis) of three representative high MPKI (top row), medium MPKI (middle row), and low MPKI (bottom row) workloads for the baseline system (solid lines) and Sectored DRAM (dashed lines). Fig. 12 (top row) shows the distribution of normalized parallel speedups of all high, medium, and low MPKI workloads. We make two observations from the two figures. First, Sectored DRAM provides higher parallel speedup over the baseline for high MPKI workloads when the number of cores is larger than two. For example, Sectored DRAM provides 26% higher parallel speedup than the baseline for all 16-core high MPKI workloads on average. As the average row buffer hit rate for 16-core high MPKI workloads is only 18%, the memory controller needs to issue many ACT (activate) commands to serve the memory requests. Sectored DRAM's t_{FAW} reduction (§4.1) allows the memory controller to issue the large number of ACT commands required by these workloads (i.e., 82% of all main memory requests) at a higher rate, reducing the average memory access latency for these workloads (by 25% on average for 16-core high MPKI workloads). The parallel speedup of two displayed applications, *ligraPageRank* and *hashjoinPR*, follow the common pattern we describe for the high MPKI workloads. However, *libquantum* behaves differently where Sectored DRAM cannot provide substantially higher parallel speedups than the baseline as the number of cores increase. This workload has a high row buffer hit rate (62% for the 16-core *libquantum*). Thus, the memory controller does *not* need to issue many ACT commands to serve *libquntum*'s memory requests, leaving a small window of opportunity for the memory controller to schedule ACT commands at a higher rate provided by the t_{FAW} reduction. Second, Sectored DRAM, on average, provides smaller parallel speedup compared to the baseline for low and medium MPKI workloads. Although Sectored DRAM's t_{FAW} reduction reduces the proportion of processor cycles where the memory controller has to stall to satisfy the t_{FAW} timing parameter from 14.4% in the baseline to 6.5% in Sectored DRAM for 16-core low and medium MPKI workloads, the average memory latency for these workloads increase by 0.5% in Sectored DRAM compared to the baseline. Moreover, for these workloads, sector misses increase the number of memory requests on average by 69%. Because a larger number of memory requests experience higher latencies in Sectored DRAM compared to the baseline, Sectored DRAM provides smaller parallel speedup for these workloads.

System Energy Consumption. The bars in Fig. 11 show the system energy consumption (on the secondary/right y-axis) of Sectored DRAM normalized to the system energy consumption of the baseline. Fig. 12 (bottom) shows the distribution of normalized system energy consumption for workloads from three categories (low, medium, and high MPKI). We make two observations. First, Sectored DRAM reduces system energy consumption for high MPKI workloads when the number of cores is larger than two. On average at 16-cores, high MPKI workloads' system energy consumption reduce by 20%. Sectored DRAM achieves this by a combination of (i) reduced DRAM energy consumption due to SA and VBL operation (see §7.4), and (ii) reduced background power consumption by the processor as workloads execute faster. Second, for medium

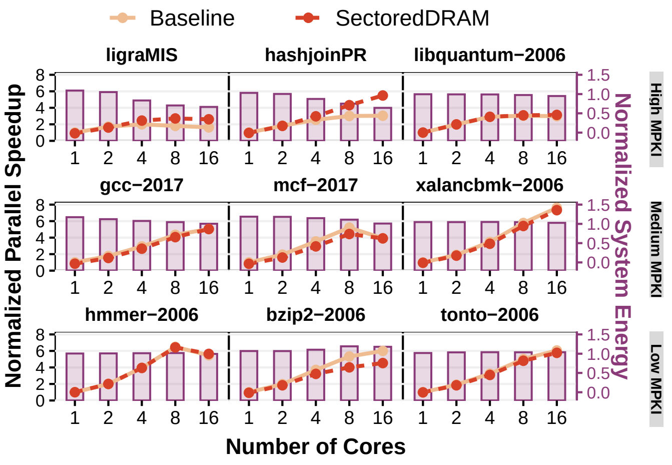

Figure 11: Normalized parallel speedup (primary/left y-axis) and system energy (secondary/right x-axis) of representative high, medium, and low LLC MPKI workloads for varying number of cores.

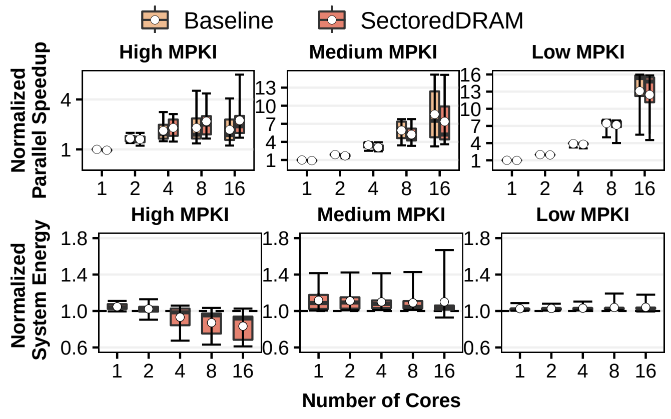

Figure 12: Normalized parallel speedup (top) and system energy (bottom) distribution of all high, medium, and low LLC MPKI workloads for varying number of cores.[5]

and low MPKI workloads, Sectored DRAM increases system energy consumption. We observe that Sectored DRAM increases the average DRAM energy consumption by 12% for 16-core medium and low MPKI workloads on average. The increase in DRAM energy consumption together with the increase in background power consumption by the processor (as workloads execute slower, Fig. 11) increases the system energy consumption for these workloads.

We conclude that Sectored DRAM improves system performance and reduces system energy consumption in high MPKI workloads where: (i) a high number of $ACTs$ targets different DRAM banks (i.e., the workload is bound by t_{FAW}), and (ii) the sector predictor can accurately predict the used words.

7.4. Workload Mix Performance and Energy

We evaluate Sectored DRAM's performance and energy using multi-programmed workload mixes. To stress DRAM and

[5]Each box is lower-bounded by the first quartile and upper-bounded by the third quartile. The median falls within the box. The inter-quartile range (IQR) is the distance between the first and third quartiles (i.e., box size). Whiskers extend to the minimum and maximum data point values on either sides of the box, while a bubble depicts average values.

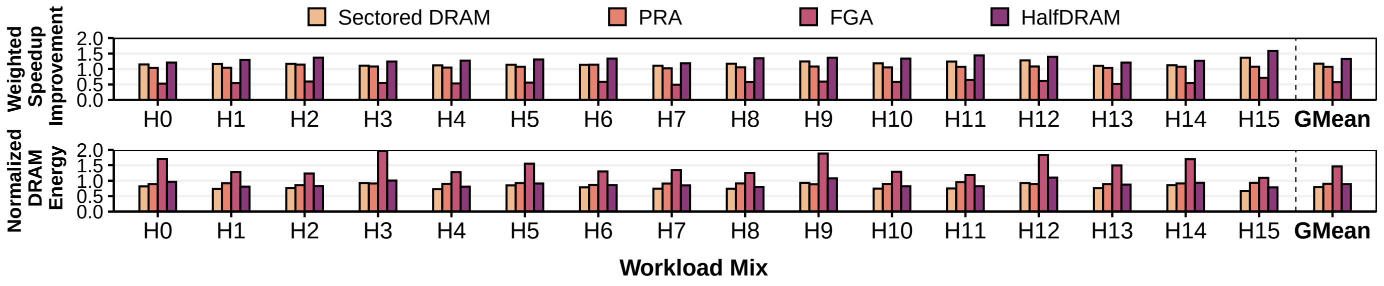

Figure 13: Weighted speedup improvement over the baseline (higher is better) on top. DRAM energy normalized to the baseline (lower is better) on the bottom.

cache hierarchy, we use high MPKI workload mixes. We compare Sectored DRAM's performance and main memory access energy to a baseline coarse-grained DRAM system and three state-of-the-art fine-grained DRAM mechanisms: (i) Fine-Grained Activation (FGA) [27, 40], (ii) Partial Row Activation (PRA) [20], and (iii) HalfDRAM [26, 39].

Performance. Fig. 13 (top) shows the weighted speedup [88–90] of 16 workload mixes for Sectored DRAM and the three state-of-the-art fine-grained DRAM mechanisms, normalized to the baseline system. We make four observations. First, Sectored DRAM's weighted speedup is 1.17× that of the baseline, on average, across all workload mixes. This is due to Sectored DRAM's reduction in t_{FAW}, which allows Sectored DRAM's memory controller to schedule ACT commands at a higher rate than in the baseline. In this way, Sectored DRAM can serve $READ$ requests faster (Sectored DRAM's average DRAM read latency is 25% smaller compared to the baseline), thus improving the performance of the memory-intensive workloads. Second, Sectored DRAM greatly outperforms naive fine-grained DRAM mechanisms (i.e., FGA [27, 40]). We observe that Sectored DRAM's weighted speedup is 2.05× that of FGA, on average across all workloads. Naive fine-grained DRAM mechanisms greatly reduce the throughput of DRAM data transfers as they are limited to fetching a cache block from a single MAT (§3.1). Compared to the baseline, FGA incurs a 43% *reduction* in weighted speedup, on average.

Third, Sectored DRAM's weighted speedup is 1.10× that of PRA, on average across all workloads. Sectored DRAM outperforms PRA by enabling fine-grained DRAM access and activation for *both* $READ$ and $WRITE$ operations, while PRA is limited to $WRITE$ operations.

Fourth, Sectored DRAM's weighted speedup is 0.89× that of HalfDRAM, on average across all workloads. Sectored DRAM cannot improve performance as much as HalfDRAM since, in Sectored DRAM, the memory controller needs to service additional memory requests caused by sector misses. However, as we show next, HalfDRAM's higher performance benefits come at the cost of *higher* area overheads (§7.5) and *lower* energy savings than Sectored DRAM.

DRAM Energy Consumption. Fig. 13 (bottom) shows the DRAM energy consumption of each workload mix for Sectored DRAM and the state-of-the-art mechanisms. Values are normalized to the DRAM energy in the baseline system. We draw two observations. First, Sectored DRAM *significantly* reduces DRAM energy consumption compared to the baseline, leading to up to (average) 33% (20%) lower DRAM energy consumption. Second, Sectored DRAM enables larger DRAM energy savings compared to prior works. On average, across all workload mixes, Sectored DRAM reduces DRAM energy consumption by 84%, 13%, and 12% compared to FPA, PRA, and HalfDRAM. Note that, while HalfDRAM can provide better performance than Sectored DRAM, our mechanism provides higher energy savings than HalfDRAM since it enables accessing DRAM at a finer granularity.

We analyze the impact of Sectored DRAM on the energy consumed by DRAM operations to better understand our DRAM access energy results. Fig. 14 (left) shows the DRAM energy broken down into ACT, background, and RD/WR consumption, normalized to the baseline system DRAM energy consumption, averaged across all workload mixes. We make two observations.

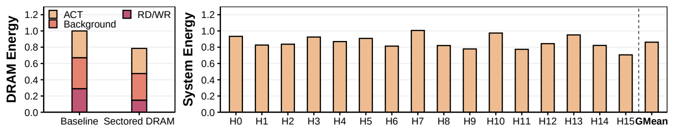

Figure 14: Components that contribute to DRAM energy normalized to total DRAM energy consumed by the baseline (left). System energy normalized to the baseline (right) for different workload mixes.

First, Variable Burst Length (VBL) greatly reduces the RD/WR energy by 51%, on average. Using VBL, the system retrieves only the required (and predicted to be required) words of a cache block from the DRAM module. On average, the number of bytes transferred between the memory controller and the DRAM module is reduced by 55% (not shown) with Sectored DRAM compared to the baseline. In this way, the system uses the power-hungry memory channel more energy-efficiently, eliminating unnecessary data movement. Second, Sectored Activation (SA) can reduce the energy spent on activating DRAM rows by 6% on average. The reduction in ACT energy is relatively small. This is because the memory controller issues more ACT commands compared to the baseline in Sectored DRAM. The new ACT commands (i) respond to the additional memory requests caused by sector misses, and (ii) resolve the row conflicts that occur due to interference created by the sector misses.

System Energy Consumption. Fig. 14 (right) shows the energy consumed by the Sectored DRAM system (processor *and*

DRAM) normalized to the energy consumed by the baseline system for all workloads. We observe that Sectored DRAM reduces system energy consumption, on average, by 14%. Sectored DRAM does so by (i) reducing DRAM energy consumption, and (ii) reducing background power consumption by the processor as workloads execute faster.

We conclude that Sectored DRAM improves performance and provides energy savings compared to current coarse-grained DRAM systems and prior state-of-the-art fine-grained DRAM systems.

7.5. Area Overhead

DRAM. We use CACTI [44] to model the area of a DRAM chip (Table 2) at $22nm$ technology node. Table 4 gives a detailed breakdown of the modeled DRAM bank's area.

Table 4: DRAM bank area breakdown.

DRAM Component	Area (mm^2)
DRAM cells	8.3
Wordline drivers	3.2
Sense amplifiers	4.6
Row decoder	0.1
Column decoder	<0.1
Data & address bus	0.4

Sectored DRAM. We model the overhead of (i) 8 additional LWD stripes, (ii) sector transistors, (iii) sector latches, and (iv) wires that propagate sector bits from sector latches to sector transistors to implement Sectored Activation (§4.1). Sectored DRAM introduces 2.26% area overhead over the baseline DRAM bank. Overall, Sectored DRAM increases the area of the chip (16 banks and I/O circuitry) by *only* 1.72%.

FGA [27,40] and PRA [20]. We estimate the area overhead of these architectures to be the same as Sectored DRAM because they require the same set of modifications to the DRAM array to enable Fine-DRAM-Act.

HalfDRAM [39] and HalfPage [26]. We estimate the chip area overheads of HalfDRAM and HalfPage as 2.6% and 5.2%, respectively. Both HalfDRAM and HalfPage require 8 additional LWD stripes like Sectored DRAM does. HalfDRAM further requires implementing double the number of CSL signals [39] to enable mirrored connection, and HalfPage requires doubling the number of HFFs per MAT [26].

Processor. We use CACTI to model the storage overhead of sector bits in caches (one byte per cache block) and the sector predictor (1088 bytes per core). The sector bits and the predictor storage increase the area of the 8-core processor (Table 2) by 1.22%.

We conclude that Sectored DRAM can be implemented at low area overhead.

7.6. Cache Access Latency

Sectored caches [74–77] (§5.2) require minor modifications to existing cache design to enable word-granularity access. Our CACTI simulations show that these modifications increase the L1 cache's access latency from $0.78\,\mathrm{ns}$ to $0.79\,\mathrm{ns}$. This small increase in latency is unlikely to increase the number of processor cycles it takes to access the L1 cache. However, finer-granularity sectored cache support may require more sector bits and increase the additional L1 cache access latency by a greater amount, thereby causing the number of processor cycles it takes to access the L1 cache to increase. To understand the overheads of the increase in processor cycles, we simulate a Sectored DRAM design (called SlowCache) with L1, L2, and L3 cache access latency values that are one processor cycle higher than the default Sectored DRAM design. SlowCache performs as well as the default Sectored DRAM design, providing 17.0% weighted speedup improvement (compared to the baseline) for the high MPKI workload mixes on average, whereas the default Sectored DRAM design provides 17.2% weighted speedup improvement. We conclude that Sectored DRAM's impact on cache access latency has a negligible effect on Sectored DRAM's system performance improvement.

8. Discussion

8.1. Non-Memory-Intensive Workloads

Sectored DRAM's current system integration can lead to varying performance benefits depending on the workload's memory intensity, for two reasons. First, non-memory-intensive workloads (i.e., workloads with low and medium LLC MPKI; see §6.1) do *not* issue enough concurrent memory requests to main memory so they can benefit from Sectored DRAM's $tFAW$ reduction. Second, the additional memory accesses caused by sector misses can increase the average memory access latency, causing performance degradation. We propose two approaches to overcome the performance degradation in non-memory-intensive workloads.

Dynamically Turning Sectored DRAM Off. Sectored DRAM can be turned off while the system executes non-memory-intensive workloads. To do so, we leverage the performance counters already present in modern processors [91–93] to periodically compute the average occupancy of the memory controller's read request queue and turn Sectored DRAM on/off when the computed value exceeds an empirically determined threshold (set by the system designer). We implement the described mechanism where we (i) periodically (i.e., every 1000 cycles) compute the average occupancy of the memory controller's read request queue, and (ii) turn Sectored DRAM on for the next 1000 cycles if it exceeds 30 or turn it off otherwise.

Fig. 15 shows the weighted speedup for 16 workload mixes from each memory intensity category (H: high, M: medium, L: low LLC MPKI), normalized to the coarse-grained DRAM baseline. We show results for two Sectored DRAM configurations: *Always ON* never turns Sectored DRAM off and *Dynamic* turns Sectored DRAM on and off as described above. We observe that the *Dynamic* configuration allows Sectored DRAM to perform as well as the baseline for non-memory-intensity workloads, while maintaining better performance than the baseline for memory intensive workloads. *Dynamic* provides higher speedups than *Always ON*, on average across all workload mixes and classes, even though *Dynamic* provides slightly lower speedups than *Always ON* for high memory intensive workloads.

Better Sector Prediction. Improving the Sector Predictor's (SP) accuracy would reduce the number of additional LLC

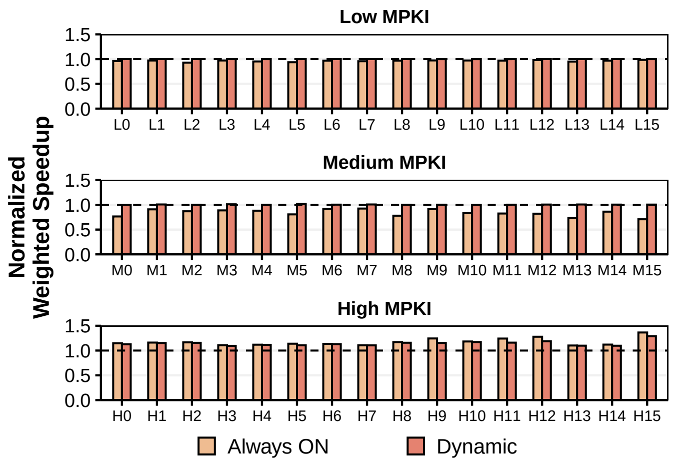

Figure 15: Weighted speedup for *Always ON* and *Dynamic*.

misses. A more sophisticated SP could be developed by tracking the intra-cache-block access patterns of instructions with deeper history, or other techniques (e.g., reinforcement learning [94, 95], perceptron-based prediction [96–101]) could be applied to predict the useful words.

Implementing SP in the lower level caches (e.g., L2) might improve sector prediction accuracy, potentially at the cost of designing a more complex SP. This is because the access patterns of requests that arrive to the lower level caches could be different than those that arrive to the L1 cache. We implement the SP only in the L1 cache because the L1 cache is typically tightly integrated with the core. This allows us to transfer the memory instruction addresses from the core to the L1 cache at low cost.

8.2. Finer-Granularity Sector Support

We design and evaluate Sectored DRAM with 8 sectors. Extending Sectored DRAM to support more sectors could enable higher energy and performance benefits but would require (i) transferring additional sector bits to DRAM and (ii) more DRAM circuit area to place additional sector latches.

Our implementation allows us to transfer up to 14 sector bits with each PRE command to DRAM (§4.1). To transfer more than 14 sector bits, (i) DRAM command encoding could be extended with new signals to carry the sector bits (e.g., another signal/pin for one additional sector bit), (ii) a new DRAM command with enough space allocated in its encoding for sector bits could be implemented, or (iii) parts of sector bits could be distributed over multiple PRE commands.

The area required by additional sector latches are negligible. Implementing 8 more sector latches brings Sectored DRAM's DRAM chip area overhead to 1.78% from 1.72%.

8.3. DRAM Error Correcting Codes (ECC)

Some computing systems are required to operate reliably while executing highly-intensive workloads over long periods of time (e.g., cloud systems) [102]. These systems employ error correcting codes (ECC) to detect and/or prevent errors from manifesting in DRAM cells. Single-error-correcting double-error-detecting code (SECDED-ECC) [103] used in today's systems is naturally compatible with Sectored DRAM.

SECDED-ECC encodes a beat of the DRAM data transfer burst into a codeword by adding parity bits. The DRAM module is extended with additional chips to store the parity bits, and the memory channel is extended with additional DQ pins to transfer the parity bits. Since with Sectored DRAM, each DRAM access transfers at least one full codeword (64 bits data + 8 bits parity), ECC encoding or decoding operations can be correctly performed after each access.

Other systems make use of more specialized, Chipkill-like, ECC (e.g., Single Device Data Correction [104, 105]). Sectored DRAM can easily support the single symbol error correction (SSC [105–107]) scheme. In the SSC scheme, the total ECC codeword consists of 32 4-bit data symbols and 4 4-bit ECC symbols with a total size of 144 bits. 128 bits of data is encoded to form ECC symbols. The DRAM module consists of 16 x4 chips to store data symbols and 2 x4 chips to store ECC symbols. The module transmits 72 bits with each beat of the data transfer burst and an ECC codeword is transmitted over two beats of a data transfer burst. To support the SSC scheme, Sectored DRAM can use burst lengths that are multiples of two (i.e., 2, 4, 6, or 8), which allows the DRAM module to transmit whole ECC codewords.

8.4. Burst Chop in DDRx

Recent DDRx standards [9–11] support burst chop operation that allows the DRAM chip to cut the data transfer burst's length in half. Burst chop can be used to enable Fine-DRAM-Access in a system *without* any modifications to DRAM chips (although at an access granularity of only half of the cache block size).

We evaluate Sectored DRAM with only the burst chop operation (i.e., no SA and no VBL) to see how system performance and DRAM energy are affected using high MPKI workload mixes. We observe that (i) Sectored DRAM reduces DRAM energy on average compared to the baseline system by 18%, and (ii) Sectored DRAM reduces the average weighted speedup for the workload mixes by 5%. Since Sectored DRAM with only the burst chop operation does not implement SA, these workloads cannot benefit from the reduction in $tFAW$. Moreover, sector misses result in extra memory accesses, increasing the average memory latency observed by these workloads. We conclude that SA and VBL are critical for Sectored DRAM to enable high performance and low energy consumption.

9. Related Work

Sectored DRAM is the first low-cost and high-performance DRAM substrate that can mitigate the excessive energy consumed by enabling (i) Fine-DRAM-Access and (ii) Fine-DRAM-Act. We extensively compared Sectored DRAM against the relevant, low-cost, state-of-the-art fine-grained DRAM architectures [20, 26, 27, 39, 40] in §3.1 and §7.4. In this section, we discuss other related works.

Other Fine-Grained Activation Mechanisms. Prior works [21–25] propose other fine-grained DRAM architectures at the cost of reorganizing the DRAM array and/or modifying the DRAM on-chip interconnect. A subset of these works [23–25] targets higher-bandwidth DRAM standards and offer significant performance and activation energy improvements. [21] develops a new interconnect that serializes the data fetched from multiple partially activated banks. [22] divides DRAM MATs into subMATs by adding more helper flip-flops to mitigate the throughput loss of fine-grained activation. These works do *not* reduce the energy wasted on the memory channel by transferring unused words, which Sectored DRAM does at low chip area cost.

DRAM Module-Level Fine-DRAM-Access. A class of prior work [18, 19, 108–110] proposes new DRAM module designs (e.g., subranked DIMMs [18, 19, 108]) that allow independently operating each DRAM chip in a DRAM module (§2.1) in order to implement Fine-DRAM-Access. From a system standpoint, Sectored DRAM is a more practical mechanism to implement compared to module-level mechanisms because Sectored DRAM requires no modifications to the physical DRAM interface. Similar to Sectored DRAM, module-level mechanisms (e.g., DGMS [19]) require modifications to DRAM (i.e., modifications to the DRAM module in DGMS and to the DRAM chip in Sectored DRAM) and the processor. However, on top of these modifications, module-level mechanisms also require modifications to the physical DRAM interface (e.g., three additional pins to select one of the eight chips in a rank [108]), thus making module-level mechanisms incompatible with industry standards (i.e., JEDEC specifications [10]). In contrast, Sectored DRAM does *not* require modifications to the physical DRAM interface, and thus, Sectored DRAM chips comply with industry standards and specifications.

We quantitatively evaluate a subranked DIMM design (DGMS [19]) that can be implemented with minimal modifications to the physical DRAM interface. This design can operate subranks independently and each subrank can receive one DRAM command per DRAM command bus cycle (i.e., 1x ABUS scheme [18]). We find that this subranked DIMM design *reduces* system performance for the high MPKI workload mixes, causing 23% reduction in weighted speedup on average. Even though the subranked DIMM allows requests to be served from different subranks in parallel, the DRAM command bus bandwidth is insufficient to allow timely scheduling of these requests to different subranks [19] (i.e., command bus becomes the bottleneck). The DRAM command bus bandwidth can be increased to enable higher-performance subranked DIMM designs. However, this comes at additional hardware cost and modifications to the physical DRAM interface [19]. In contrast, Sectored DRAM improves the weighted speedup for the same set of workloads by 17% on average and requires no modifications to the physical DRAM interface.

Sectored DRAM can be used along with a module-level mechanism to further improve energy efficiency by enabling finer-granularity DRAM activation and access within DRAM chips. In systems that employ a single DRAM chip as main memory (typically using LPDDRx [66, 67]), Sectored DRAM could improve energy efficiency where module-level power saving mechanisms are inapplicable.

10. Conclusion

We develop a new, high-throughput, energy-efficient, and practical fine-grained DRAM architecture. Compared to prior fine-grained DRAM architectures, our design (Sectored DRAM) significantly improves both system energy and performance. It does so by eliminating (i) the energy waste caused by transferring unused words between the processor and DRAM, (ii) the energy spent on activating DRAM cells that are not accessed by memory requests. Activating a smaller number of cells allows the memory controller to serve memory requests with lower memory access latency. While effective at improving both system energy and performance, Sectored DRAM is also practical and can be implemented at low hardware cost.

Acknowledgements

We thank the SAFARI Research Group members for valuable feedback and the stimulating intellectual environment they provide. We acknowledge the generous gifts provided by our industrial partners, including Google, Huawei, Intel, Microsoft, and VMware, and support from the Microsoft Swiss Joint Research Center.

References

[1] O. Mutlu and T. Moscibroda, "Parallelism-Aware Batch Scheduling: Enhancing Both Performance and Fairness of Shared DRAM Systems," in *ISCA*, 2008.
[2] Y. Kim *et al.*, "Thread Cluster Memory Scheduling: Exploiting Differences in Memory Access Behavior," in *MICRO*, 2010.
[3] C. J. Lee *et al.*, "Improving Memory Bank-Level Parallelism in the Presence of Prefetching," in *MICRO*, 2009.
[4] X. Tang *et al.*, "Improving Bank-Level Parallelism for Irregular Applications," in *MICRO*, 2016.
[5] B. Keeth *et al.*, *DRAM Circuit Design: Fundamental and High-Speed Topics*, 2008.
[6] K. Itoh, *VLSI Memory Chip Design.* Springer Science & Business Media, 2013.
[7] T. Vogelsang, "Understanding the Energy Consumption of Dynamic Random Access Memories," in *ISCA*, 2010.
[8] Y. Kim *et al.*, "A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM," in *ISCA*, 2012.
[9] JEDEC, "DDR3 SDRAM Standard," *JESD79-3*, 2007.
[10] JEDEC, *JESD79-4C: DDR4 SDRAM Standard*, 2020.
[11] JEDEC, *JESD79-5: DDR5 SDRAM Standard*, 2020.
[12] Micron Technology, "SDRAM, 4Gb: x4, x8, x16 DDR4 SDRAM Features," 2014.
[13] P. Hammarlund *et al.*, "Haswell: The Fourth-Generation Intel Core Processor," *IEEE Micro*, 2014.
[14] S. Kumar and C. Wilkerson, "Exploiting Spatial Locality in Data Caches Using Spatial Footprints," in *ISCA*, 1998.
[15] S. Kumar *et al.*, "Amoeba-Cache: Adaptive Blocks for Eliminating Waste in the Memory Hierarchy," in *MICRO*, 2012.
[16] P. Pujara and A. Aggarwal, "Increasing the Cache Efficiency by Eliminating Noise," in *HPCA*, 2006.
[17] M. K. Qureshi *et al.*, "Line Distillation: Increasing Cache Capacity by Filtering Unused Words in Cache Lines," in *HPCA*, 2007.
[18] D. H. Yoon *et al.*, "Adaptive Granularity Memory Systems: A Tradeoff Between Storage Efficiency and Throughput," in *ISCA*, 2011.
[19] D. H. Yoon *et al.*, "The Dynamic Granularity Memory System," in *ISCA*, 2012.
[20] Y. Lee *et al.*, "Partial Row Activation for Low-Power DRAM System," in *HPCA*, 2017.
[21] C. Zhang and X. Guo, "Enabling Efficient Fine-Grained DRAM Activations with Interleaved I/O," in *ISLPED*, 2017.
[22] T. Alawneh *et al.*, "Dynamic Row Activation Mechanism for Multi-Core Systems," in *CF*, 2021.
[23] Y. H. Son *et al.*, "Microbank: Architecting Through-Silicon Interposer-Based Main Memory Systems," in *SC*, 2014.
[24] M. O'Connor *et al.*, "Fine-Grained DRAM: Energy-Efficient DRAM for Extreme Bandwidth Systems," in *MICRO*, 2017.
[25] N. Chatterjee *et al.*, "Architecting an Energy-Efficient DRAM System for GPUs," in *HPCA*, 2017.
[26] H. Ha *et al.*, "Improving Energy Efficiency of DRAM by Exploiting Half Page Row Access," in *MICRO*, 2016.
[27] A. N. Udipi *et al.*, "Rethinking DRAM Design and Organization for Energy-Constrained Multi-Cores," in *ISCA*, 2010.
[28] I. Paul *et al.*, "Harmonia: Balancing Compute and Memory Power in High-Performance GPUs," in *ISCA*, 2015.
[29] M. Ware *et al.*, "Architecting for Power Management: The IBM® POWER7™ Approach," in *HPCA*, 2010.

[30] C. Lefurgy *et al.*, “Energy Management for Commercial Servers,” *Computer*, 2003.
[31] S. Ghose *et al.*, “Demystifying Complex Workload-DRAM Interactions: An Experimental Study,” in *SIGMETRICS*, 2019.
[32] O. Mutlu, “Memory Scaling: A Systems Architecture Perspective,” in *IMW*, 2013.
[33] T. M. O. Mutlu, “Memory Performance Attacks: Denial of Memory Service in Multi-Core Systems,” in *USENIX Security*, 2007.
[34] O. Mutlu and T. Moscibroda, “Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors,” in *MICRO*, 2007.
[35] K. Sudan *et al.*, “Micro-Pages: Increasing DRAM Efficiency with Locality-Aware Data Placement,” in *ISCA*, 2010.
[36] G. L. Yuan *et al.*, “Complexity Effective Memory Access Scheduling for Many-Core Accelerator Architectures,” in *MICRO*, 2009.
[37] K. J. Nesbit *et al.*, “Fair Queuing Memory Systems,” in *MICRO*, 2006.
[38] L. Subramanian *et al.*, “BLISS: Balancing Performance, Fairness and Complexity in Memory Access Scheduling,” *TPDS*, 2016.
[39] T. Zhang *et al.*, “Half-DRAM: A High-Bandwidth and Low-Power DRAM Architecture from the Rethinking of Fine-Grained Activation,” in *ISCA*, 2014.
[40] E. Cooper-Balis and B. Jacob, “Fine-Grained Activation for Power Reduction in DRAM,” *IEEE Micro*, 2010.
[41] Standard Performance Evaluation Corp., “SPEC CPU® 2006,” http://www.spec.org/cpu2006, 2006.
[42] Standard Performance Evaluation Corp., “SPEC CPU® 2017,” http://www.spec.org/cpu2017, 2017.
[43] G. F. Oliveira *et al.*, “DAMOV: A New Methodology and Benchmark Suite for Evaluating Data Movement Bottlenecks,” *IEEE Access*, 2021.
[44] N. Muralimanohar *et al.*, “CACTI 6.0: A Tool to Model Large Caches,” HP Laboratories, Tech. Rep. HPL-2009-85, 2009.
[45] I. Bhati *et al.*, “Flexible Auto-Refresh: Enabling Scalable and Energy-Efficient DRAM Refresh Reductions,” in *ISCA*, 2015.
[46] K. K. Chang *et al.*, “Understanding Latency Variation in Modern DRAM Chips: Experimental Characterization, Analysis, and Optimization,” in *SIGMETRICS*, 2016.
[47] K. K.-W. Chang *et al.*, “Improving DRAM Performance by Parallelizing Refreshes with Accesses,” in *HPCA*, 2014.
[48] K. K. Chang *et al.*, “Low-Cost Inter-Linked Subarrays (LISA): Enabling Fast Inter-Subarray Data Movement in DRAM,” in *HPCA*, 2016.
[49] K. K. Chang *et al.*, “Understanding Reduced-Voltage Operation in Modern DRAM Devices: Experimental Characterization, Analysis, and Mechanisms,” in *SIGMETRICS*, 2017.
[50] L. Cojocar *et al.*, “Are We Susceptible to Rowhammer? An End-to-End Methodology for Cloud Providers,” in *SP*, 2020.
[51] P. Frigo *et al.*, “TRRespass: Exploiting the Many Sides of Target Row Refresh,” in *SP*, 2020.
[52] H. Hassan *et al.*, “CROW: A Low-Cost Substrate for Improving DRAM Performance, Energy Efficiency, and Reliability,” in *ISCA*, 2019.
[53] H. Hassan *et al.*, “ChargeCache: Reducing DRAM Latency by Exploiting Row Access Locality,” in *HPCA*, 2016.
[54] J. S. Kim *et al.*, “The DRAM Latency PUF: Quickly Evaluating Physical Unclonable Functions by Exploiting the Latency-Reliability Tradeoff in Modern Commodity DRAM Devices,” in *HPCA*, 2018.
[55] J. S. Kim *et al.*, “D-RaNGe: Using Commodity DRAM Devices to Generate True Random Numbers With Low Latency and High Throughput,” in *HPCA*, 2019.
[56] J. S. Kim *et al.*, “Revisiting RowHammer: An Experimental Analysis of Modern DRAM Devices and Mitigation Techniques,” in *ISCA*, 2020.
[57] Y. Kim *et al.*, “Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors,” in *ISCA*, 2014.
[58] Y. Kim *et al.*, “Ramulator: A Fast and Extensible DRAM Simulator,” *CAL*, 2016.
[59] D. Lee *et al.*, “Adaptive-Latency DRAM: Optimizing DRAM Timing for the Common-Case,” in *HPCA*, 2015.
[60] D. Lee *et al.*, “Simultaneous Multi-Layer Access: Improving 3D-Stacked Memory Bandwidth at Low Cost,” *TACO*, 2016.
[61] D. Lee *et al.*, “Design-Induced Latency Variation in Modern DRAM Chips: Characterization, Analysis, and Latency Reduction Mechanisms,” in *SIGMETRICS*, 2017.
[62] D. Lee *et al.*, “Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture,” in *HPCA*, 2013.
[63] D. Lee *et al.*, “Decoupled Direct Memory Access: Isolating CPU and IO Traffic by Leveraging a Dual-Data-Port DRAM,” in *PACT*, 2015.
[64] A. Olgun *et al.*, “QUAC-TRNG: High-Throughput True Random Number Generation Using Quadruple Row Activation in Commodity DRAMs,” in *ISCA*, 2021.
[65] N. Hajinazar *et al.*, “SIMDRAM: A Framework for Bit-Serial SIMD Processing Using DRAM,” in *ASPLOS*, 2021.
[66] JEDEC, *JESD209-5A: LPDDR5 SDRAM Standard*, 2020.
[67] JEDEC, *JESD209-4B: Low Power Double Data Rate 4 (LPDDR4) Standard*, 2017.
[68] D. Kaseridis *et al.*, “Minimalist Open-Page: A DRAM Page-Mode Scheduling Policy for the Many-Core Era,” in *MICRO*, 2011.
[69] M. Rhu *et al.*, “A Locality-Aware Memory Hierarchy for Energy-Efficient GPU Architectures,” in *MICRO*, 2013.
[70] S. Li *et al.*, “DRISA: A DRAM-Based Reconfigurable In-Situ Accelerator,” in *MICRO*, 2017.
[71] K. Koo *et al.*, “A 1.2V 38nm 2.4Gb/s/pin 2Gb DDR4 SDRAM with Bank Group and ×4 Half-Page Architecture,” in *ISSCC*, 2012.
[72] L. Frontini *et al.*, “A Very Compact Population Count Circuit for Associative Memories,” in *MOCAST*, 2018.
[73] A. Dalalah *et al.*, “New Hardware Architecture for Bit-Counting,” in *WSEAS*, 2006.
[74] J. S. Liptay, “Structural Aspects of the System/360 Model 85, II: The Cache,” *IBM Systems Journal*, 1968.
[75] A. J. Smith, “Line (Block) Size Choice for CPU Cache Memories,” *TC*, 1987.
[76] D. Alpert and M. Flynn, “Performance Trade-Offs for Microprocessor Cache Memories,” *IEEE Micro*, 1988.
[77] M. D. Hill and A. J. Smith, “Experimental Evaluation of On-Chip Microprocessor Cache Memories,” in *ISCA*, 1984.
[78] J. B. Rothman and A. J. Smith, “The Pool of Subsectors Cache Design,” in *ICS*, 1999.
[79] A. Seznec, “Decoupled Sectored Caches: Conciliating Low Tag Implementation Cost,” in *ISCA*, 1994.
[80] K. Inoue *et al.*, “Dynamically Variable Line-Size Cache Exploiting High On-chip Memory Bandwidth of Merged DRAM/Logic LSIs,” in *HPCA*, 1999.
[81] C. Chen *et al.*, “Accurate and Complexity-Effective Spatial Pattern Prediction,” in *HPCA*, 2004.
[82] S. Li *et al.*, “The McPAT Framework for Multicore and Manycore Architectures Simultaneously Modeling Power, Area, and Timing,” *TACO*, 2013.
[83] G. Hamerly *et al.*, “Simpoint 3.0: Faster and More Flexible Program Phase Analysis,” *Journal of Instruction Level Parallelism*, 2005.
[84] M. Hashemi *et al.*, “Accelerating Dependent Cache Misses with an Enhanced Memory Controller,” in *ISCA*, 2016.
[85] J. H. Ahn *et al.*, “Future Scaling of Processor-Memory Interfaces,” in *SC*, 2009.
[86] K. Chandrasekar *et al.*, “DRAMPower: Open-Source DRAM Power & Energy Estimation Tool,” http://www.drampower.info, 2012.
[87] A. Snavely and D. M. Tullsen, “Symbiotic Jobscheduling for a Simultaneous Multithreaded Processor,” in *ASPLOS*, 2000.
[88] S. Eyerman and L. Eeckhout, “System-Level Performance Metrics for Multiprogram Workloads,” *IEEE Micro*, 2008.
[89] R. Das *et al.*, “Application-Aware Prioritization Mechanisms for On-Chip Networks,” in *MICRO*, 2009.
[90] Y. Kim *et al.*, “ATLAS: A Scalable and High-Performance Scheduling Algorithm for Multiple Memory Controllers,” in *HPCA*, 2010.
[91] Intel, “Intel Alder Lake Events,” https://perfmon-events.intel.com/, 2022.
[92] AMD, “uProf User Guide,” https://bit.ly/3L0xwl3, 2022.
[93] Intel, “Intel® Performance Counter Monitor - A Better Way to Measure CPU Utilization,” https://intel.ly/3xLo80Y, 2022.
[94] R. Bera *et al.*, “Pythia: A Customizable Hardware Prefetching Framework Using Online Reinforcement Learning,” in *MICRO*, 2021.
[95] E. Ipek *et al.*, “Self-Optimizing Memory Controllers: A Reinforcement Learning Approach,” in *ISCA*, 2008.
[96] D. A. Jiménez and C. Lin, “Dynamic Branch Prediction with Perceptrons,” in *HPCA*, 2001.
[97] D. A. Jiménez and C. Lin, “Neural Methods for Dynamic Branch Prediction,” *TOCS*, 2002.
[98] D. A. Jiménez, “Fast Path-Based Neural Branch Prediction,” in *MICRO*, 2003.
[99] E. Teran *et al.*, “Perceptron Learning for Reuse Prediction,” in *MICRO*, 2016.
[100] D. A. Jiménez and E. Teran, “Multiperspective Reuse Prediction,” in *MICRO*, 2017.
[101] E. Garza *et al.*, “Bit-level Perceptron Prediction for Indirect Branches,” in *ISCA*, 2019.
[102] C. Slayman, “Cache and Memory Error Detection, Correction, and Reduction Techniques for Terrestrial Servers and Workstations,” *IEEE Transactions on Device and Materials Reliability*, 2005.
[103] S. Mukherjee, *Architecture Design for Soft Errors*. Morgan Kaufmann Publishers Inc., 2008.
[104] Intel, “Intel® E7500 Chipset MCH x4 Single Device Data Correction (x4 SDDC) Implementation and Validation,” Application Note, 2002.
[105] AMD, “BIOS and Kernel Developer’s Guide (BKDG) for AMD Family 15h Models 00h-0Fh Processors,” Developer’s Guide, 2013.
[106] R. Yeleswarapu and A. K. Somani, “Addressing Multiple Bit/Symbol Errors in DRAM Subsystem,” arXiv:1908.01806, 2020.
[107] C. Chen, “Symbol error correcting codes for memory applications,” in *Proceedings of Annual Symposium on Fault Tolerant Computing*, 1996, pp. 200–207.
[108] J. H. Ahn *et al.*, “Multicore DIMM: An Energy Efficient Memory Module with Independently Controlled DRAMs,” *CAL*, 2009.
[109] H. Zheng *et al.*, “Mini-Rank: Adaptive DRAM Architecture for Improving Memory Power Efficiency,” in *MICRO*, 2008.
[110] T. M. Brewer, “Instruction Set Innovations for the Convey HC-1 Computer,” *IEEE Micro*, 2010.