SAM: Accelerating Strided Memory Accesses

Xin Xin
Electrical and Computer Engineering Department,
University of Pittsburgh
USA
xix59@pitt.edu

Yanan Guo
Electrical and Computer Engineering Department,
University of Pittsburgh
USA
yag45@pitt.edu

Youtao Zhang
Department of Computer Science, University of Pittsburgh
USA
zhangyt@cs.pitt.edu

Jun Yang
Electrical and Computer Engineering Department,
University of Pittsburgh
USA
juy9@pitt.edu

ABSTRACT

Strided memory accesses are an important type of operations for In-Memory Databases (IMDB) applications. Strided memory accesses often demand data at word granularity with fixed strides. Hence, they tend to produce sub-optimal performance on DRAM memory (the *de facto* standard memory in modern computer systems) that accesses data at cacheline granularity. Recently proposed optimizations either introduce significant reliability degradation or are limited to non-volatile crossbar memory structures.

In this paper, we propose a low-cost DRAM-based optimization scheme *SAM* for accelerating strided memory accesses. SAM consists of several designs. The primary design, termed *SAM-IO*, is to exploit under-utilized I/O resources in commodity DRAM chips to support high-performance strided memory accesses with near-zero hardware overhead. Based on SAM-IO, an enhanced design, termed *SAM-en*, is further proposed by combining several innovations to achieve overall efficiency on energy and area. Our evaluation of the proposed designs shows that SAM not only achieves high performance improvement (up to ~ 4.2×) but also maintains high-level reliability protection for server systems.

CCS CONCEPTS

• **Hardware** → **Dynamic memory**; • **Computer systems organization** → Special purpose systems.

KEYWORDS

DRAM, strided access, main memory

ACM Reference Format:
Xin Xin, Yanan Guo, Youtao Zhang, and Jun Yang. 2021. SAM: Accelerating Strided Memory Accesses. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '21), October 18–22, 2021, Virtual Event, Greece.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3466752.3480091

MICRO '21, October 18–22, 2021, Virtual Event, Greece

ACM ISBN 978-1-4503-8557-2/21/10…$15.00
https://doi.org/10.1145/3466752.3480091

1 INTRODUCTION

Strided memory accesses have become an important type of operations in IMDB applications, e.g., accessing one field of data from a list of multi-field records in a database. Strided memory accesses are often at word granularity with fixed strides. This exhibits low spatial locality and thus, high access overhead on DRAM memories, the *de facto* standard memory in modern computer systems. Figure 1 depicts an example of an array of records saved in an in-memory database: each memory row stores n records, each cacheline (64B) contains one record, and each record contains eight fields. To process a query that summarizes field $f1$ from all records, we generate consecutive memory accesses but each returned line provides only one field and there is no intra-cacheline data reuse.

A simple software based approach can speed up strided accesses by saving two data copies: one in row-wise while the other in column-wise. This leads to 100% storage overhead, as well as synchronization overhead between the two copies. Simple hardware based techniques, e.g., those reducing memory access granularities AGMS [52], DGMS [53], subchannel [7], and FGDRAM [35], divide the original rank (bank) into multiple sub-ranks (sub-banks) and let one memory access fetch a fraction of the data from one sub-rank (sub-bank), thereby supporting multiple accesses sharing the bandwidth simultaneously. While they speed up random accesses from different sub-ranks (sub-banks), they are ineffective for strided memory accesses whose data tend to reside in the same sub-rank (sub-bank).

Two recent studies proposed hardware support that can effectively improve the efficiency of strided accesses. Seshadri *et al.* [40] proposed GS-DRAM to distribute aligned data fields from different rows to different chips. They then enhanced the address bus such that one memory access drives different rows from different chips with each chip returning one requested field data. GS-DRAM can achieve significant memory performance improvement, e.g., the same number of memory requests return 8× useful data for a memory rank with eight chips. However, GS-DRAM faces one major limitation, i.e., it cannot effectively support error-checking and correcting (ECC) during memory accesses, which restricts its adoption in systems that demand reliability, e.g., the servers in data centers.

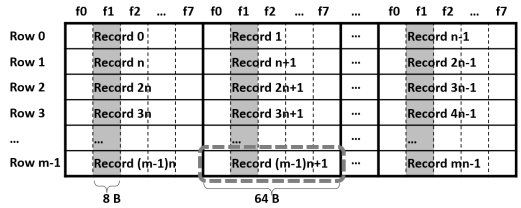


Figure 1: Strided memory accesses exhibit low spatial locality and low data reuse.

Wang *et al.* [48] proposed RC-NVM for supporting strided memory accesses on crossbar based non-volatile memories. RC-NVM exploits dual addressing to enable memory accesses in both row-wise and column-wise directions. Given RC-NVM leverages the symmetry of WLs (wordlines) and BLs (bitlines) in a crossbar, it is applicable only to crossbar structures. In addition, RC-NVM tends to introduce large hardware overhead. It duplicates peripheral circuit for the crossbar array, representing 33% area overhead. The complex layout in RC-NVM also demands two extra metal layers for the memory die, a large overhead for typical RRAM chips that have four metal layers [24].

In this paper, we propose a low-cost architectural scheme to accelerate strided memory (*SAM*) accesses for DRAM based memories. There are two design goals of SAM. The first one is to maintain high level data reliability protection for DRAM based computer systems, in particular, servers in data centers that adopt *chipkill* technology [2, 11, 16, 17] to prevent chip level failures. The second one is to minimize the hardware cost for DRAM chips, whose internals are highly optimized and thereby sensitive to hardware modifications. For these purposes, SAM mostly exploits the under-utilized resources in commodity DRAM chips to support strided memory accesses. SAM achieves chipkill compatibility by keeping the fine-grained accessed data consistent with chipkill codeword. In particular, *SAM* consists of three designs. *SAM-sub* utilizes the subarray organization to enable data aggregation in either column- or row- wise directions for supporting both strided accesses and traditional row accesses respectively. *SAM-IO* leverages the under-utilized I/O resources in commodity DRAM chips to support high-performance strided memory accesses. *SAM-en* combines SAM-sub and SAM-IO to achieve the overall efficiency on energy and area. Our evaluation of the proposed designs shows that SAM not only achieves high performance strided accesses but also maintains high reliability protection for server systems.

For the rest of the paper, we briefly discuss the background and motivation in Section 2 and Section 3, respectively. We elaborate the SAM design in Section 4 and its system support in Section 5. We present the evaluation results in Section 6. We conclude the paper in Section 7.

2 BACKGROUND

2.1 DRAM Basics

Figure 2 illustrates the DRAM basics, ranging from channel connection (Figure 2(a)) to cell array organization (Figure 2(d)). Each DRAM channel consists of multiple ranks while each rank consists of multiple chips. The choice of the number of chips depends on the computing environment, the channel width, and the chip I/O width. For servers in data centers, a typical channel configuration consists of 18 ×4 chips with 16 data chips and two parity chips (Figure 2(a)). Each chip has four pins such that a total of 64 bits from data chips can be communicated in one transfer, referred as one beat. Transferring a typical cacheline (64B) requires eight beats.

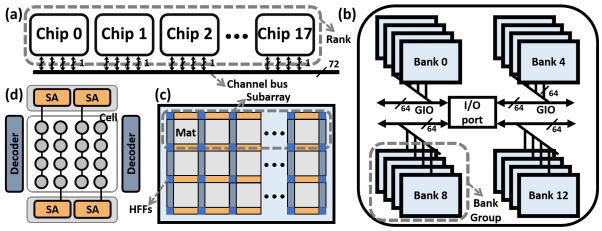


Figure 2: The DRAM (DDR4) hierarchical structure: (a) rank, (b) chip, (c) bank, and (d) mat.

Each DRAM chip consists of multiple banks. In modern DRAM, e.g., DDR4/DDR5 [19, 20], the banks are grouped as *bank groups* with each bank group typically having four banks and an independent GIO bus connecting to the I/O port (Figure 2(b)). Bank groups maintain semi-independent control logic such that the timing parameters for consecutive operations from different bank groups are shorter than those from the same bank group. Each bank contains multiple subarrays that are connected by global bitlines (BLs) and column select lines (CSLs) (Figure 2(c)). Each subarray consists of multiple mats with each mat being a 2D cell array (Figure 2(d)). Inside a mat, the cells are vertically connected by local BLs to local SAs, referred to as row buffers, and horizontally connected by local wordlines (WLs) [32, 54].

2.2 I/O Interface

I/O Buffer[1]**.** Recent technology advances achieve significant improvements on the I/O bus frequency, e.g., up to 1600 MHz in DDR3 and 3200 MHz in DDR4, but little reduction on the operation latency of DRAM arrays, i.e., banks and subarrays. To bridge the gap between I/O bus and inner arrays, modern DRAM chips augment the I/O buffer size to save all the data to be transferred in one burst (8 beats). For example, for the ×4 DRAM chip, a 32b I/O buffer is shared by four pins (DQs) so that the data gets transmitted to I/O bus in eight beats, as shown in Figure 3.

Common Die Design. Modern computer systems often employ 16, 8, and 4 data chips to construct a memory rank for servers, desktop, and embedded systems, respectively [45], so that each chip transmits 4, 8, and 16 bits per beat, referred to as ×4, ×8, ×16 configurations, respectively. A naive design thereby demands different I/O buffers for chips in different configurations, e.g., a ×4 configuration demands a 32-bit I/O buffer while a ×16 configuration demands a 128-bit buffer.

To mitigate the cost of design, DRAM manufacturers adopt the common die design for different I/O configurations [5, 13, 15, 27, 31, 41, 42], that is, the maximum of 128-bit I/O buffer (8-bit for each of the 16 pins) is integrated in all DRAM chips. The desired configuration is implemented by cutting electric fuses after passing tests for a specific I/O mode, as shown in Figure 3. For example, a

[1]Other articles may term it as FIFO, Read/Write Buffer, or In/Out Buffer

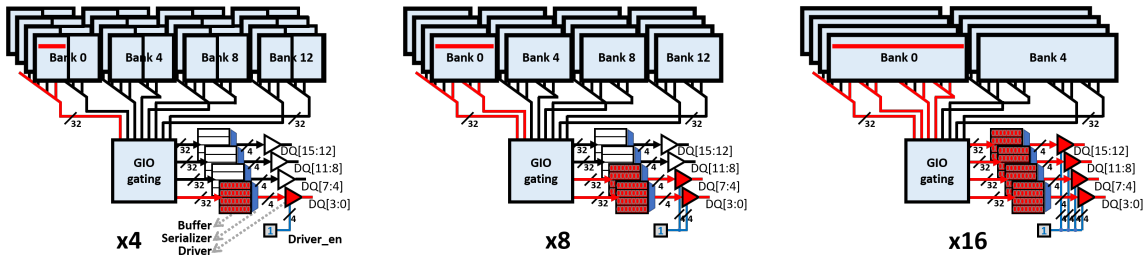


Figure 3: DRAM (DDR4) operation under different I/O mode (×4, ×8, ×16). Note that DRAM port contains a read path and a write path in separate. For simplicity, the figure shows an abstract diagram with one path.

×4 configuration activates one 32-bit I/O buffer to save 32-bit data fetched from DRAM arrays. A ×8 configuration activates two 32-bit I/O buffers (equal to one 64-bit buffer) to save 64-bit fetched data. The common die design not only mitigates the design, verification, and tooling costs, but also adapts better to market uncertainty. Of course, it leaves a subset of I/O buffers unused under ×4 and ×8 configurations.

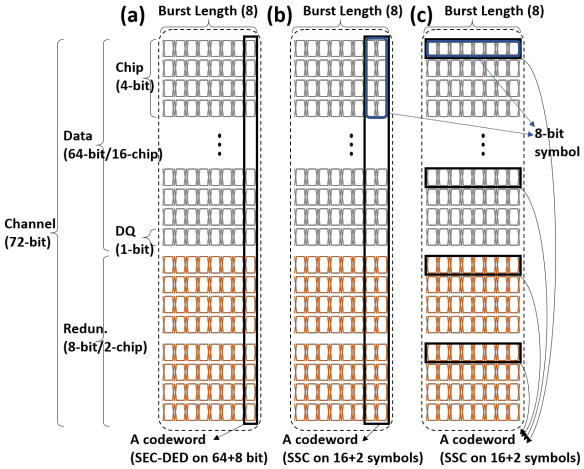


Figure 4: Three ECC schemes: (a) SEC-DED, (b) SSC, (c) SSC-variant

2.3 ECC for Servers

To improve the reliability of memory system, a DRAM rank usually integrates one or multiple ECC chips to detect and correct DRAM errors at runtime. Figure 4(a) illustrates the SEC-DED (single-bit error correction and double-bit error detection) scheme that is widely adopted for desktop computers. SEC-DED needs eight redundant bits for every 64 data bits to form a 72b codeword. Therefore, it integrates one extra parity chip for ×8 configuration, and two parity chips for ×4 configuration.

The server computers in data centers, to meet the high reliability demands from their applications, often adopt more advanced ECC schemes, i.e., chipkill ECC [44], to prevent system crashes under more errors. In particular, for ×4 configuration, an SSC (single symbol correction) chipkill ECC [16, 17] (Figure 4(b)) treats the eight data bits (in two beats) from each chip as a symbol and, with a total of 18 symbols (per every two beats), correct one symbol error, i.e., one DRAM chip is defective. Correspondingly, a codeword (144b) in SSC is the 18 symbols. To support multiple symbol detecting, e.g., Single Symbol Correcting-Double Symbol Detecting (SSC-DSD), a widely adopted strategy [1] is to double the channel width with 36 ×4 chips. Then, SSC-DSD treats the four bits (in one beat) from each chip as a symbol and, with a total of 36 symbols, correct (detect) one (two) chip fault. In this paper, we assume the baseline server computers adopt the SSC or SSC-DSD chipkill.

Different chipkill schemes have been proposed in the literature to achieve tradeoffs among performance, hardware overhead, and reliability factors [22, 23, 26]. Figure 4(c) shows a variant of SSC that treats the eight bits from each DQ as a data symbol. When being adopted for ×4 configuration, one burst (in the length of eight beats) returns a total of 4 SSC codewords, consisting of 72 symbols, which enables the detection and correction of up to four symbol errors, i.e., when all DQs from one chip fail.

Prior study [26] further extends this SSC variant and constructs a large codeword (512b) containing 72 8-bit symbols to provide even stronger protections, at the expense of decoding complexity and latency.

3 MOTIVATION

3.1 Strided Memory Accesses

In this paper, we elaborate our discussion by accelerating strided memory accesses from In-Memory DataBases (IMDBs). While IMDB is a typical application that requires a large number of strided memory accesses, our design is not bound to IMDB and is applicable to other application domains.

There are two types of workloads in IMDB: OLTP (on-line transactional processing) and OLAP (on-line analytical processing) [3, 4]. OLTP is a transactional system for querying specific records and generally characterized by the simple queries that insert, update, and delete information from the database. On the other hand, OLAP is a data retrieving and analysis system, which is characterized by complex queries that extract data from multiple transactions for analyzing.

The data in relational database are organized in a two-dimensional table. In the table with row-oriented layout, records (or tuples) are consecutively stored, which is generally preferred by OLTP. On the contrary, column-oriented layout, where fields are consecutively stored, generally works better on OLAP. However, neither row-oriented layout nor column-oriented layout can efficiently serve a mixed workload with both OLTP and OLAP, known as hybrid transaction-analytical processing (HTAP). Because modern memory is organized in a single dimension, the other dimensional accesses required by HTAP exhibit patterns of strided accesses.

3.2 Design Goal

In this paper, our goal is *to design a low-cost DRAM-based architectural scheme that achieves high performance for strided memory access while maintaining high-level ECC protection, i.e., chipkill ECC.* We face two design challenges.

- ECC compatibility. The memory reliability is critical when running IMDB on server computers as memory failure can crash the system, resulting in either data loss or severe performance degradation in data recovery. While it is beneficial to accelerate strided memory accesses, disabling chipkill ECC during the operation may not be desirable.
- Commodity DRAM readiness. Given DRAM remains the *de facto* memory standard for constructing main memory in modern computer systems, it is important to accelerate accesses for DRAM chips. Since DRAM internals are highly optimized, it is critical to minimize the hardware cost for the design to be applicable.

3.3 The State of the art

We next discuss the state-of-the-art architectural designs for accelerating strided memory accesses. We show how they work and why the preceding two design goals cannot be realized with existing designs.

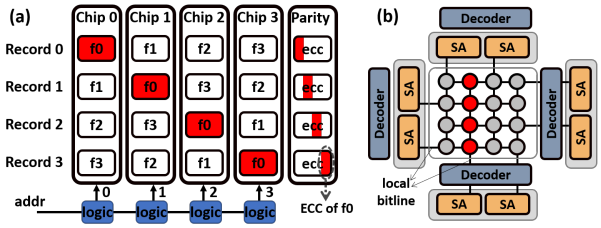


Figure 5: (a) Data allocation in Gather-Scatter approach, (b) Mat organization in Dual-Addressing approach

3.3.1 The GS-DRAM (Gather-Scatter) approach. Seshadri *et al.* proposed GS-DRAM to accelerate strided accesses at the chip level [40]. For the example shown in Figure 1, GS-DRAM first conducts intra-row shifts such that the data of the desired field from different records are distributed to different chips with aligned columns, e.g., the field f_0 data from four consecutive rows are distributed to four different chips in Figure 5(a). To enable accessing these data from different chips, GS-DRAM augments the address bus such that the same row address sent on the address bus is exploited to drive different rows in different chips. By returning field f_0 from different rows, one strided read can return all four f_0 field data and thus improve the overall memory performance. GS-DRAM adopts intra-row shift and thus is not suitable for accessing data with large strided patterns.

A major concern associated with GS-DRAM is its compatibility with ECC scheme. For example, one strided read can return all four f_0 fields (Figure 5(a)), but their ECC data, located at four addresses in one parity chip, cannot be returned simultaneously. Due to the same reason, neither chipkill nor SEC-DED ECC is compatible with GS-DRAM. To maintain the same reliability, a simple enhancement is to integrate two ECCs with one for normal accesses, and the other for strided accesses. Clearly, such an approach doubles the ECC space overhead and, more importantly, triggers multiple ECC updates when writing one cacheline. For example, since one memory transfer (72B) contains 4 SSC codewords, one write transfer can lead to updating five ECCs — four ECCs for accessing each codeword in its corresponding strided access, and one ECC for the normal access.

Note that GS-DRAM also varies the default data layout. For example, a 16B 'word', by default, is spread over all data chips, as shown in Figure 4(b), so that a critical word can be first accessed by changing burst order in DRAM, whereas the 16B word has to be concentrated on four ×4 data chips in GS-DRAM.

3.3.2 The RC-NVM (Dual-Addressing) Approach. Wang *et al.* proposed RC-NVM to accelerate strided memory accesses on NVM-memory based systems [48]. They exploited the symmetry of the crossbar structure of NVM so that they exchange WLs and BLs on demand to support row-wise and column-wise accesses, i.e., normal memory accesses and strided memory accesses, respectively, as shown in Figure 5(b).

RC-NVM tends to introduce large modifications in memory arrays. To support the symmetric access in two directions, it roughly duplicates all peripheral circuits (e.g., SAs and decoders) and connection wires (e.g., CSLs, local data lines (LDLs), global WLs, and global BLs). These lead to ~15% silicon overhead and two extra metal layers for a four-layer baseline design [24].

A major limitation of RC-NVM is that it binds to the crossbar structure and thus is not DRAM compatible. In addition, RC-NVM introduces long latency when leveraging the crossbar symmetry at bit-level. An optimization of RC-NVM reshapes the subarray structure from 1D to 2D (e.g., square shape of 4×4 mats). However, the reshaped structure increases the number of global BLs, which increases the area overhead up to ~33%. In this article, we term RC-NVM with and without reshaped subarray as RC-NVM-bit and RC-NVM-word respectively

To summarize, existing architectural optimizations lack the ability to achieve both of our design goals.

4 SAM DESIGN

In this section, we first introduce SAM-sub, an improvement of RC-NVM. Then we present our primaries, i.e., SAM-IO and SAM-en. Finally, we indicate an extension method to achieve finer-grained accesses in strided pattern.

4.1 SAM-sub

As discussed in Section 2.1, DRAM is well architected in a hierarchical structure, such as mat and subarray, each of which is organized in one dimension. For each access, multiple mats in one subarray are activated and a fraction of data are fetched from local row buffer to global row buffer with the help of HFFs (helper flip flops), which are in a latch-like structure for repeating signals [25]. With the same structure, we propose SAM-sub, which constructs a column-wise subarray by activating multiple mats in the same column, to implement strided access.

Figure 6 shows the logical structure of SAM-sub, where a set of row-oriented bitlines are added to connect multiple HFFs in one row. Therefore, the HFFs in the bank are symmetrically connected by the bitlines in two dimensions. Correspondingly, data from multiple mats aligned in the same column, termed column-wise subarray,

can be fetched to an additional global column buffer via the HFFs. Meanwhile, to enable the multiple mats in each column-wise subarray, one extra control line is added per column-wise subarray. Similar to RC-NVM, SAM-sub tends to cost the same power for accesses to row-wise subarray and column-wise subarray because of the symmetric data path.

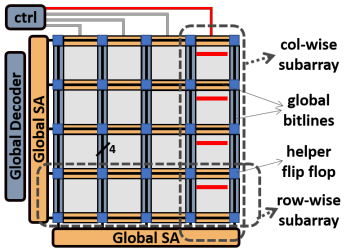


Figure 6: Bank structure of SAM-sub in logical layout[2]

SAM-sub outperforms RC-NVM in the following three aspects. First, SAM-sub avoids the limitation of RC-NVM that relies on specific crossbar-based memory substrate to implement strided access. SAM-sub can be applied to different memory technologies, as they are all built in a similar hierarchical architecture [28, 37, 51]. Second, SAM-sub does not require any modification in the mats, because each mat is still activated in row-wise in a column-wise subarray. Compared to RC-NVM, the slight hardware overhead of SAM-sub occurs only at the subarray level with a few extra wires (e.g., global bitlines), which does not need an extra metal layer for routing. Finally, it also avoids the challenge in RC-NVM where the crossbar symmetry is at bit-level, while the symmetry of SAM-sub is at word-level, as the HFFs of each mat have a width of 4 or 8 bits.

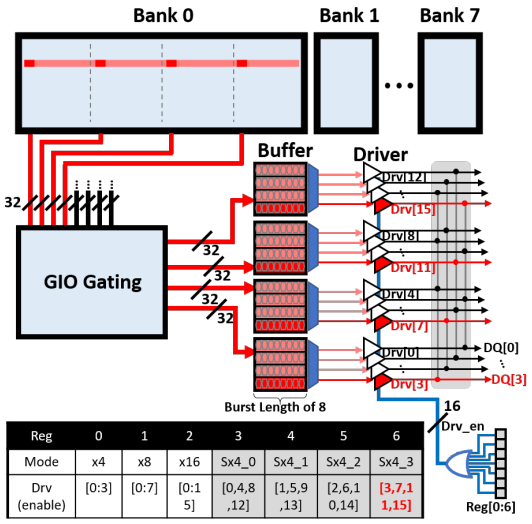
SAM-sub keeps the integrity of ECC code. For instance, in the example of SSC chipkill, each 8-bit symbol locates at the same address in each chip. Thereby, an 18-bit symbol codeword (Figure 4(b)) can always be transmitted by the 18 chips in tandem in two transfer beats, despite the access type, i.e. a regular or strided access. Although SAM-sub induces relatively smaller hardware overhead, it is still a challenge for memory manufacturers who are sensitive to area efficiency. We thereby propose our second design below, which keeps the integrity of DRAM and induces near-zero overhead.

4.2 SAM-IO

The key idea behind our proposed SAM-IO is to leverage underutilized resources in DRAM to implement strided access. As indicated in Section 2.2, × chips also contain the resources for ×8 and ×16 modes because of the common die technique. In this subsection, we leverage these redundant I/O resources to realize strided access.

4.2.1 Configurable I/O mode. Figure 7 shows the architecture of relevant components in the I/O path of a DRAM chip, including the GIO gating, I/O buffers, and drivers [18, 32]. There are 4 32-bit I/O buffers, each of which can be divided into 4 rows, termed as *lanes* in this paper. Each lane is connected to one DQ via a serializer.

In a ×4 chip, during each memory transfer (8-beat burst), 32 bits of data are fetched from slow memory arrays into one of the I/O buffers, e.g., the bottom of the four, via the GIO gating. Then, the buffered data are transmitted by the fast DDR interface. Hence, only one I/O buffer and its drivers (e.g., Drv[0:3]) are utilized, while all other I/O buffers are disabled. In a ×16 chip, the buffering size is extended by 4 times (128-bit). All 4 I/O buffers and 16 drivers are then utilized. The opportunity here is that if a ×4 chip can utilize all the unused resources, then we can perform wide buffering that stores them in all the I/O buffers, but only sends requested data to the CPU with desired strides.



Reg	0	1	2	3	4	5	6
Mode	x4	x8	x16	Sx4_0	Sx4_1	Sx4_2	Sx4_3
Drv (enable)	[0:3]	[0:7]	[0:15]	[0,4,8,12]	[1,5,9,13]	[2,6,10,14]	[3,7,11,15]

Figure 7: Structure of SAM-IO. Red color in bank/buffer represents useful data. Pink color in bank/buffer represents activated or fetched but unused data. The table below shows the I/O configuration under different modes.

Inspired from this observation, we propose SAM-IO, a strided access strategy leveraging configurable I/O modes. As shown in the table of Figure 7, we extend the regular I/O modes (×4, ×8, and ×16) with extra modes for strided access, termed stride mode Sx4_n, where n is the lane ID in the I/O buffer. For example, if only the bottom lane is requested by the CPU, then the I/O is configured as Sx4_3.

Figure 7 illustrates a detailed example. Four pieces of strided data (in red), allocated across one row in Bank0, are requested by the CPU. A regular ×4 mode would transfer them sequentially, but under an Sx4_3 mode, they are fetched into the 4 I/O buffers simultaneously, occupying the bottom lane of each buffer, and then transmitted via the 4 DQs, driven by the relevant 4 drivers (Drv[3, 7, 11, 15]), in one memory transfer. To enable this process, we redirect the control signals of drivers to an additional 7-bit I/O mode register, where each bit is responsible for one I/O configuration by enabling the corresponding drivers, as indicated by the table in Figure 7. There are also several additional interconnections among the drivers (as shown by the shaded region near DQs in Figure 7). Note that the interconnections can be implemented during post-manufacturing process, e.g., bonding the interconnected wires together during packaging, which does not induce any overhead to the silicon die. The only modification in SAM-IO is the extra mode register for driver control, which is negligible.

4.2.2 Discussion of SAM-IO. The main benefit of SAM-IO is that it induces negligible modifications to commodity DRAM. This is

[2]The figure shows a logic layout. In the physically layout, one local SA is shared by two adjacent mats. The column-wise subarray can be physically organized by every other mat.

important to memory technology whose manufacturing process has been well optimized, and non-trivial modifications could severely impact yield, reliability, and even timing parameters. Meanwhile, SAM-IO can use the SSC-variant scheme (Figure 4(c)), by storing an SSC symbol along the lane-wise of I/O buffer, to support chipkill ECC.

However, SAM-IO experiences increased power, as it internally activates and transfers many non-required data. During the strided access, 288B data (four cachelines plus ECC bits) are fetched to the I/O buffers (128 bits per chip) in the memory module, but only 72B data (one cacheline plus ECC) in strided pattern are sent to the channel. In addition, data layout now is transposed from the default Figure 4(b) to Figure 4(c). Correspondingly, one codeword transfer interval increases from 2 beats to 8 beats. This disables SAM-IO to transfer critical-word first, even though the impact on performance is moderate (<1% based on [53]). It also takes effort to transpose data in the CPU. To tackle these challenges, we propose SAM-en, an enhanced design that combines the advantage of both SAM-sub and SAM-IO.

4.3 SAM-en

SAM-en is based on SAM-IO, integrated with configurable I/O mode to implement strided access. Meanwhile, SAM-en offers two enhancement options. The first one employs an analogous technique to SAM-sub to reduce power consumption. The second is to enable the default data layout of Figure 4(b) by constructing a two-dimensional I/O buffer.

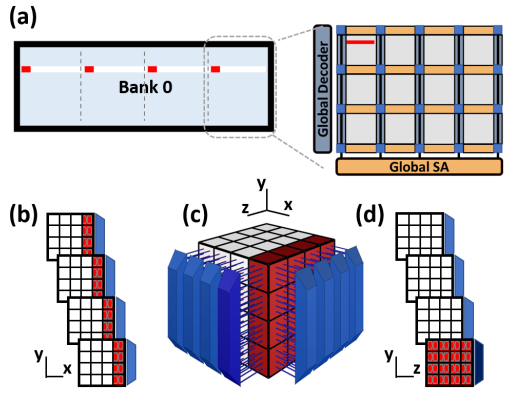


Figure 8: SAM-en Design. (a) SAM-en with fine-grained activation technique, (b)-(d) a two-dimensional I/O buffer from different views: xy-plane, 3D, and yz-plane

Option 1. Inspired by SAM-sub, the row activation in SAM-en can be conducted only on the mats that store the useful data. As shown in Figure 8(a), we leverage the fine-grained activation technique to selectively enable the mats storing the required data in strided pattern. As a result, only useful data are delivered to the I/O buffers (Figure 8(b)), which saves the power of activation and internal data transfer. This fine-grained activation technique has also been employed in several recent studies, such as FCRAM [10], SBA [46], HrDRAM [50], and half-dram [54].

Option 2. Instead of constructing the subarrays into two dimensions (row- or column-wise) in SAM-sub, we propose a lightweight two-dimensional I/O buffer access strategy for SAM-en. To better illustrate the design, we divide each I/O buffer into lanes (rows) and columns, creating 16 blocks with 2 bits per block, as shown in Figure 8(b). We can access the 4 I/O buffers from two directions with the help of two sets of serializers attached in the associated directions. Figure 8(c) shows a 3D view of the 4 buffers, where each small cube is equal to the 2-bit block. An extra set of serializers are located along the z-axis and the required data in a strided pattern (marked by red) are stored in the same level of cubes along the yz-plane. Correspondingly, the 4 buffers in Figure 8(b) can also be represented in the symmetric view (of the yz-plane) in Figure 8(d). The required data can thereby be accessed via the bottom buffer in the yz-plane. Different from SAM-IO where each word is stored in lane-wise, data in SAM-en now can be stored in column-wise, the default layout, in accordance with the critical-word-first principle. Since data paths through the two sets of serializers are symmetric, we thereby consider the accesses to the I/O buffer via the two dimensions have the same latency.

To summarize, these two options are independent, which provides flexibility to optional improved SAM design. In this article, SAM-en is implemented with both of them as default.

4.4 Support Finer Granularity

The SAM-IO and SAM-en designs select one 8-bit data in each I/O buffer and send four of them across four buffers in a strided pattern in each chip. Correspondingly, the strided granularity can be defined as 8 bits per chip, which is consistent with the 8-bit symbol, stored in one chip, in SSC. While for SSC-DSD, the symbol size is reduced to 4 bits. We thereby propose a finer strided granularity (4 bits per chip) to better support SSC-DSD and make full use of the bandwidth, even though SSC-DSD is not as widely adopted as SSC in modern server memories [26].

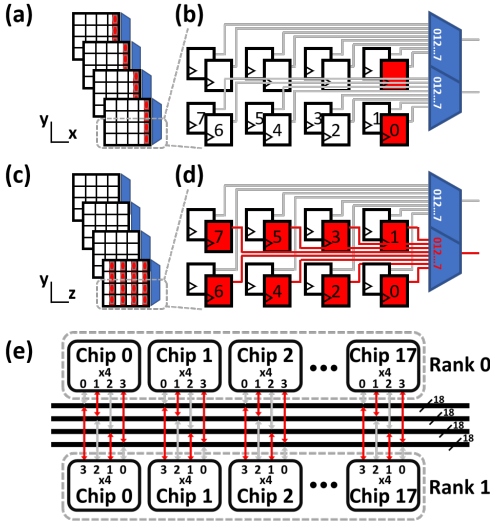


Figure 9: Support Finer Granularity. (a)-(d) I/O buffer and its details from different views. (e) DQ layout among two ranks

The strided granularity of SAM-sub is determined by the width of HFFs (4 or 8 bits) in a mat, which can be configured during manufacturing. In this subsection, we mainly indicate the implementation of finer strided granularity in SAM-IO and SAM-en. In practice, we can upgrade existing SAM-IO and SAM-en designs

without inducing additional hardware overhead. The upgrading can be based on a two-step procedure: first, data are aggregated to fewer drivers, i.e., using 2 DQs to send four 4-bit symbols in each chip, which leaves half channel pins unemployed; second, rank-level parallelism is leveraged to make full channel utilization.

For SAM-en, we can adjust the wire routing of the added serializers to aggregate data. Figure 9(a) and (c) show the layout of strided 4-bit symbols from the views of the xy-plane and yz-plane, respectively. Figure 9(b) shows a fraction of I/O buffer in detail, consisting of two lanes, where each lane contains 8 flip-flops (or latches). The 8 bits in one lane are selected by an 8:1 MUX. Correspondingly, 4 lanes attached to 4 MUXes, grouped into a 32:4 serializer. To support fine-grained strided access, the MUX in the added serializer is interleaved between two lanes. Specifically, it connects 4 bits from one lane and the other 4 bits from the other lane, as shown in Figure 8(d). Therefore, each of two 4-bit symbols can be connected to one driver via the interleaved MUX, and the 4 symbols can be transmitted via 2 DQs in one memory transfer (8 beats). Finally, we combine two ranks to make full use of channel bandwidth, as shown in Figure 9(e).

SAM-IO maintains a different layout from SAM-en, where each symbol is stored in lane-wise, and each lane is connected to one driver. As shown in Figure 7, symbols in strided pattern locate in different buffers with the same lane ID. Since each 4-bit symbol only occupies one half of a lane, we redirect two 4-bit symbols from two lanes, with the same ID, to one driver. Therefore, in each chip, four 4-bit symbols (in strided pattern) can be transmitted via 2 DQs in one memory transfer. This redirection can be implemented by the GIO gating (Figure 7) that already exists in DRAM. Similarly, we then leverage two ranks to fill up the channel.

5 DISCUSSION

We first discuss the system support for SAM. Although orthogonal to our research, which proposes an efficient memory substrate for strided access, a moderate adjustment in the system is necessary to support our design. Then we compare SAM to the state-of-the-art.

5.1 CPU End

5.1.1 Cache Hierarchy. In SAM, the accessed data under stride mode is in a strided pattern across multiple cachelines. To support the strided pattern in cache hierarchy, one choice is to employ a sector cache [30, 39] where the cacheline is divided into multiple sectors and each sector maintains its own valid and dirty bits. Since SAM is always compatible with chipkill ECC, each strided data is the 16B data in a chipkill codeword. Therefore, the cacheline can be divided into four 16B sectors, with moderate overhead, 6 bits per 64B, induced by the storage for valid and dirty bits.

Besides sector cache, previous study [14] also proposes multi-dimensional-access (MDA) cache architecture for strided access. However, it suffers from coherence issue induced by duplicated data. Additionally, compared to sector cache, MDA cache does not show a significant advantage in IMDB applications, where data reuse is not frequent in cache. Therefore, we choose a sector cache in this work to simplify performance evaluation and emphasize the performance improvement contributed by memory design.

5.1.2 ISA Extension. We add two instructions, termed `sload` and `sstore`, to enable IMDB applications to use the stride mode in SAM. The details of these two instructions are shown below:

```
sload  reg, addr
sstore  reg, addr
```

where `reg` is the destination register, `addr` is the data address. These instructions can inform memory controller to sets memory to the stride mode via the C/A (command/address) bus. While there exist other options, e.g. supplement new attribute to page table entry (PTE), for the system to implement a new function on memory, we choose ISA extension, a similar approach to prior work [40, 48], to make a fair comparison with the state-of-the-art.

5.2 OS support

By default, in order to maximize the possibility of row buffer hit, an OS page is generally mapped to one or two DRAM row segments [36, 47]. However, SAM reshapes the row organization in stride mode. Specifically, multiple rows in the column-wise subarray are activated simultaneously in SAM-sub. In SAM-IO (or SAM-en), the row size is extended by multiple times under stride mode. To ensure that an OS page can still be physically mapped to the reshaped DRAM rows, we can define a new scheme for virtual to physical address mapping under the stride mode.

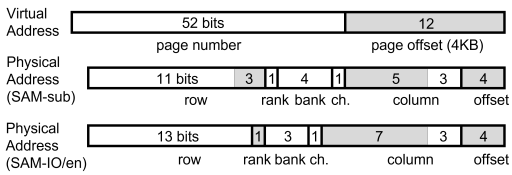


Figure 10: Virtual-Physical address mapping under stride mode

As shown in Figure 10, the shaded region in the virtual address represents the page offset (12 bits), which is directly mapped to the lower 12 bits in the physical address during regular memory access. The 4-bit offset in the physical addresses represents the 16B strided data. Under the stride mode, a small segment of the page offset is remapped. For example, in SAM-sub with 4-bit strided granularity (per chip), a 3-bit segment is swapped with the corresponding bits that are associated with subarrays. Similarly, in SAM-IO (or SAM-en), the 3-bit segment is swapped with the extended column bits and rank bit. For SAM designs with 8-bit strided granularity (per chip), the segment is only 2-bit. This extra address mapping scheme can be manually implemented by leveraging the huge-page technique. It can also be integrated into OS by adding a new kernel module. To summarize, with the knowledge of address mapping, an IMDB can explicitly control the data to be accessed and organize records with a specific layout, described in Section 5.4.1, in memory.

5.3 Interface to SAM

Different from prior work, which extends C/A width to implement the new command of strided access, SAM avoids any modification on the command interface by defining a new mode in DRAM. Commodity DRAM possesses a set of mode registers for internal control, such as refresh granularity and burst type [32]. To configure these mode registers, specific commands are sent to DRAM via C/A bus.

In SAM-IO and SAM-en, we extend the existing mode registers with the proposed extra mode register for configurable I/O, as shown in Figure 7. The different I/O modes, e.g. ×4, ×8, Sx4_n, etc., can thereby be configured via C/A bus, keeping the integrity of the command interface. Since SAM-sub does not require configurable I/O, it only needs to extend the existing mode registers with one extra bit to define the stride mode.

Given that initiating another I/O mode in SAM-IO or SAM-en could require switching DQ driver, which is similar to the process of rank switch [12], we thereby consider the switch delay among different I/O modes equals to rank-to-rank delay (tRTR). Note that in real applications, the mode switch does not happen frequently, incurring negligible performance overhead.

5.4 Compare to the State-of-the-art

5.4.1 Data Layout. Different data layouts can impact the efficiency of SAM, RC-NVM, and GS-DRAM. In this subsection, we first summarize how to allocate a chipkill codeword to multiple chips. Then, we discuss how to place an IMDB table in memory to facilitate strided access.

By default, to enable critical-word-first, each 144-bit SSC (or SSC-DSD) codeword is spread over 72 (or 144) pins, and one memory transfer can transmit 4 (or 8) codewords in burst length of 8. SAM-sub, SAM-en, and RC-NVM maintain the default layout, as shown in Figure 4(b). SAM-IO is consistent with Figure 4(c), where the codeword in each chip is in a transposed manner. GS-DRAM adopts a special layout, where each word is aggregated on a few chips, e.g. a 128-bit word is allocated to 4 data chips. This disables the chipkill function. Since maintaining a different layout from the default, SAM-IO and GS-DRAM cannot support critical-word-first.

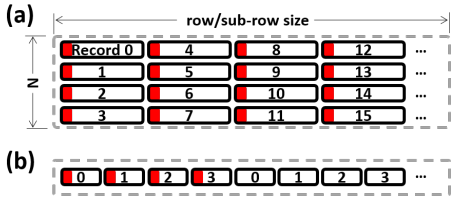


Figure 11: Two different record alignment strategies

Next, we discuss the layout of records for the above designs. In SAM, strided data are collected from multiple cachelines that are located in different rows (in SAM-sub) or sub-rows (in SAM-IO and SAM-en) with the same column offset. This requires the IMDB records to be aligned with the rows or sub-rows. As shown in Figure 11(a), the database is aligned by every N records in SAM, where N, determined by the strided granularity, equals 4 or 8. RC-NVM uses a similar alignment strategy, but with a much larger N (in the magnitude of KB). This requires the alignment to be implemented on a large physical space, which can lead to large fragments and degrade memory utilization.

Different from SAM and RC-NVM, where the database is aligned in the unit of record, in GS-DRAM, the database has to be aligned by each cacheline size (64B), as GS-DRAM cannot support large strided access. As shown in Figure 11(b), assuming the record size is 128B, the record has to be divided into two 64B segments and then aligned by each segment. This complicates the application of GS-DRAM for users.

5.4.2 Qualitatively Analysis. We qualitatively compare SAM and the state-of-the-art across three dimensions: system support, interface, and memory device, as summarized in Table 1. All designs require support from database alignment, ISA, and cache hierarchy. In the aspect of interface, GS-DRAM is the most complicated, which modifies the memory controller with a multi-level swapping logic and extends the width of command interface.

In the aspect of memory device, RC-NVM lags in performance and area efficiency. In contrast, GS-DRAM shows good performance and small area overhead, but cannot efficiently support ECC, which is an important consideration for both server memories and IMDB applications. SAM-sub and SAM-IO significantly relieve the challenges in RC-NVM and GS-DRAM. But there is still room for improvement from some perspectives, e.g., area overhead of SAM-sub, power of SAM-IO. Apparently, SAM-en achieves the best characters, which outperforms others from almost all perspectives. Again, Table 1 highlights the advantage of SAM as a promising memory substrate for strided access. The detailed quantitative evaluation is described later in Section 6.

Table 1: Comparison of Designs for Strided Access

		RC-NVM-bit	RC-NVM-wd	GS-DRAM	SAM-sub	SAM-IO	SAM-en
System	Database Alignment ISA Extension Sector Cache or MDA Cache						
Interface	Memory Controller	✓	✓	✗	✓	✓	✓
	Command Interface	✓	✓	✗	✓	✓	✓
	Critical-Word-First	✓	✓	✗	✓	✗	✓
Memory	Performance	✗	✗	✓	•	✓	✓
	Power Consumption	•	•	✓	✓	•	✓
	Area Overhead	✗	✗	✓	•	✓	✓
	Reliability	✓	✓	✗	✓	✓	✓
	Mode Switch Delay	•	•	✓	•	•	•

✓: good/unmodified •: fair/slightly_modified ✗: poor/modified

Note that GS-DRAM is better than SAM-en only from the perspective of mode switch delay. This is achieved by modifying the command interface. In other words, if GS-DRAM adopts the same mode switch strategy, it has the same switch delay as SAM-en.

6 EVALUATION

6.1 Experiment Setup

To quantitatively evaluate the benefits of SAM, we use a cycle-level memory simulator, NVMain [37], integrated with Gem5 [6] as our system simulator, on top of the x86 architecture. Table 2 lists the main parameters of the simulated system. All caches use 64B cachelines, corresponding to 16 ×4 memory chips with 32-bit I/O buffer. Multiple cores are applied to accelerate the computing process in workloads. The memory controller uses open-page and FR-FCFS scheduling policy.

In the evaluation, SAM is compared with the state-of-the-art. DRAM memory is provided for SAM and GS-DRAM, while RRAM is

the substrate of RC-NVM. The subarray in RRAM has been reshaped into a square structure (2K×2K) for RC-NVM. Each memory chip is in capacity of 8Gb, using DDR4 interface with ×4 I/O width. Timing parameters of DRAM and RRAM are obtained from industrial technical documentations [32] and academic studies [24, 37, 51], respectively. Specifically, we add mode switch parameter for SAM, equal to Rank-to-Rank delay (tRTR) that is 2 CK [21, 34, 49]. Given that memories with different capacities (core area) can maintain the same core frequency via deep pipelining [43], core frequencies in all the designs are not changed. Other latency parameters, such as tRCD, tAL, etc, are increased proportionally to the area overhead.

Table 2: Simulated System Parameters

Processor	4 cores, x86, 4.0GHz L1: 32KB, L2: 256KB, LLC: 8MB 64B cacheline, 8-way associative
Memory Controller	Write queue capacity: 32 Address mapping: rw:rk:bk:ch:cl:offset Page management: open-page, FR-FCFS
DRAM	DDR4-2400, ×4 I/O width 1 channel, 2 ranks, 16 banks 256 subarrays, 512 rows/subarray, 4Kb local row buffer CL-nRCD-nRP: 17-17-17 nRTR(or mode switch)-nCCDS-nCCDL: 2-4-6
RRAM	DDR4-2400, ×4 I/O width 1 channel, 2 ranks, 16 banks 128 subarrays, 2K rows/subarray, 2Kb local row buffer CL-nRCD-nRP: 17-35-1 nRTR-nCCDS-nCCDL: 2-4-6

Workloads. As pointed out by prior work [4, 14, 29, 48], there still lacks a standard benchmark for the mixed OLTP and OLAP processing, we follow prior work to construct the workloads.

Table 3 shows the benchmarks used in our evaluation. We first leverage the benchmark (Q1 to Q12) obtained from [29, 48]. In the benchmark, Q2 has f10 > x mostly false. Other queries have f10 > x in a constant possibility (selectivity) of 25%. However, we find all of them, which only read or write a subset of fields in a table, prefer column store, though the benchmark is a mix of OLTP- and OLAP-type queries (note that some OLTP-type queries could prefer row store to column store). We thereby supplement the benchmark with several queries (Qs1 to Qs6) that read or write the whole fields in a few records and prefer row store. Similar to Q queries, these Qs queries have f10 > x in a possibility of 25%.

We also conduct a comprehensive evaluation on a scenario that prefers a hybrid row and column store, using a similar method to [3, 4] by analyzing an arithmetic query, as well as an aggregate query, and adjusting the projectivity and selectivity, which determine the number of fields projected and the number of records that satisfy the predicate (e.g., f0 > x), respectively.

The query benchmark has two tables, a wide table (Ta) and a narrow table (Tb). Ta and Tb have 128 fields and 16 fields, each 8 bytes in size. The size of the records in Ta and Tb is 1KB and 128B, respectively. In all of our experiments, we load 10M records into each table in the database. We assume all the records are well aligned in each memory design.

Table 3: Benchmark Queries

No.	SQL Statement from [48] (prefer column store)
Q1	SELECT f3, f4 FROM Ta WHERE f10 > x
Q2	SELECT * FROM Tb WHERE f10 > x
Q3	SELECT SUM(f9) FROM Ta WHERE f10 > x
Q4	SELECT SUM(f9) FROM Tb WHERE f10 > x
Q5	SELECT AVG(f1) FROM Ta WHERE f10 > x
Q6	SELECT AVG(f1) FROM Tb WHERE f10 > x
Q7	SELECT Ta.f3, Tb.f4 FROM Ta, Tb WHERE Ta.f1 > Tb.f1 AND Ta.f9 = Tb.f9
Q8	SELECT Ta.f3, Tb.f4 FROM Ta, Tb WHERE Ta.f9 = Tb.f9
Q9	SELECT f3, f4 FROM Ta WHERE f1 > x AND f9 < y
Q10	SELECT f3, f4 FROM Ta WHERE f1 > x AND f2 < y
Q11	UPDATE Tb SET f3 = x, f4 = y WHERE f10 = z
Q12	UPDATE Tb SET f9 = x WHERE f10 = y
No.	**SQL Statement supplement (prefer row store)**
Qs1	SELECT * FROM Ta LIMIT 1024
Qs2	SELECT * FROM Tb LIMIT 1024
Qs3	SELECT * FROM Ta WHERE f10 > x
Qs4	SELECT * FROM Tb WHERE f10 > x
Qs5	INSERT INTO Ta VALUES (f0, f1, ..., fp)
Qs6	INSERT INTO Tb VALUES (f0, f1, ..., fp)
No.	**Arithmetic and Aggregate SQL (prefer row or col store)**
Arith.	SELECT fi + fj + ... + fk FROM Ta WHERE f0 < x
Aggr.	SELECT AVG(fi), ..., AVG(fj) FROM Ta WHERE f0 < x

Power. We estimate DRAM power consumption using Micron's power calculator [33], which is based on current (IDD) values measured on actual devices. The power parameters are excerpted from Micron's data sheet [32]. Specifically, for SAM-sub, power is calculated based on a ×4 chip with 2% increased background power from extra decoding and SA logic. For SAM-IO, power is calculated based on different modes. For example, a ×16 (×4) chip is employed to evaluate a stride (regular) mode. SAM-en adopts the same strategy as SAM-IO, but with optimized power parameters contributed by fine-grained activation. RRAM is modeled similar to Lee's work [28]. The parameters are obtained from [24, 37].

Area. There are two sources of area overhead. The first one is the extra wires routing in DRAM array, e.g. row-wise global BLs in SAM-sub, which can be evaluated by counting the number of wiring tracks in metal layers as indicated in [7, 35]. The array structure, e.g. bank and subarray, is based on the model from Rambus [38]. For SAM-sub, the 4 extra global BLs are routed in the same metal layer (M2) as global WLs and LDLs in horizontal (row-wise) direction. Given each subarray with 512 rows has 128 M2 routing tracks for global WLs and 12 M2 routing tracks for 4 differential LDLs and 4 local WLsels (wordline select lines), the extra global BLs, requiring 8 M2 routing tracks, lead to 5.7% area overhead. Correspondingly, the additional control lines for column-wise subarray are routed in M3, increasing the area by 0.7%. For SAM-IO, it does not induce any wire routing overhead. For SAM-en, it only requires the same additional control lines as SAM-sub, leading to 0.7% area overhead.

The second source of area overhead originates from the extra logic, e.g. additional global SA in SAM-sub, in the peripheral circuit, which can be derived from CACTI-3DD [8] (based on 32nm node). For SAM-sub, there are two types of extra logic. First, the area

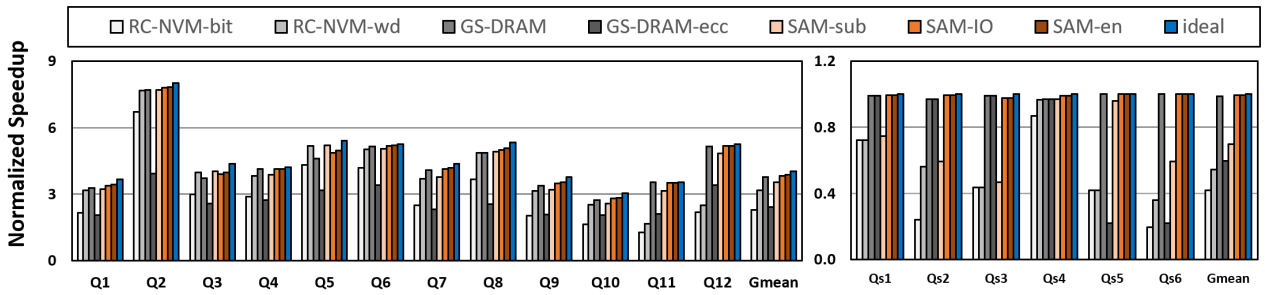


Figure 12: Speedup (normalized to row-store) of different designs on Q and Qs queries

of extra global SAs is 0.14 mm^2, corresponding to 0.8% overhead. Second, the extra control logic for the column-wise subarray, which is a simplified column decoder, occupies 0.002 mm^2 (less than 0.01% overhead). For SAM-IO, the only overhead is the 7-bit mode register (less than 0.01%), which is negligible. For SAM-en, it induces extra control logic, similar to SAM-sub, and another set of serializers, which is also negligible (less than 0.01% overhead).

Hence, the overall area overhead, including wire routing and peripheral logic, of SAM-sub, SAM-IO, and SAM-en are ~7.2%, <0.01%, and ~0.7%, respectively.

6.2 Results

Figure 12 shows the performance results of SAM, RC-NVM, and GS-DRAM. The memory module uses SSC-DSD by default, entailing 4-bit strided granularity of SAM. The baseline is a commodity DRAM with row-store layout. The ideal case represents either a row-store or column-store that is preferred by the queries. Specifically, column-store is applied to Q-type queries, and row-store is for Qs-type queries. As indicated in Section 3.3.2, two designs, RC-NVM-bit and RC-NVM-wd, are employed to represent RC-NVM. Given that GS-DRAM faces the challenge of reliability, it is not fair for other designs to compare with GS-DRAM from the perspective of performance, power, and area, but regardless of the reliability. An optimal solution to relieve this reliability challenge is to use embedded ECC, as indicated in [55], which stores ECC bits along with their associated data bits in the same page. We thereby enhance GS-DRAM with embedded ECC, termed GS-DRAM-ecc, and add it to the comparison.

All designs can outperform baseline (row-store) in Q queries. For Qs queries, the maximum speedup is one, as the baseline is the ideal case. Obviously, among all the designs, the performance of SAM-IO and SAM-en is closer to the ideal. On average, the naive SAM-sub design achieves 3.8× performance improvement on Q queries, but with 30% performance degradation on Qs queries. This already outperforms prior studies, where GS-DRAM-ecc, RC-NVM-bit, and RC-NVM-word achieve 2.7×, 2.6×, and 3.4× improvement on Q queries, with 41%, 58%, and 46% degradation on Qs queries. More importantly, our primary designs, SAM-IO, and SAM-en, can achieve 4.1× and 4.2× performance improvement on Q queries, meanwhile, without performance degradation (< 1%) on Qs queries.

SAM outperforms RC-NVM in all queries. This can be attributed to three reasons: first, the timing parameters of DRAM are generally better than NVM. Especially for the write requests, e.g., Q11, Q12, Qs5, and Qs6, the performance of RC-NVM is significantly degraded. Second, RC-NVM suffers from a high latency of field switch. When accessing a new field, RC-NVM has to conduct a column-to-column switch in the same bank, entailing bank conflict. This is more challenging for RC-NVM-bit, which has to collect multiple sub-fields to form one field because of the bit-level crossbar symmetry. Third, the consecutive records in RC-NVM are aligned across multiple rows in the same bank, which also increases the possibility of bank conflict when running Qs queries. While in SAM-IO and SAM-en, accesses to consecutive records or different fields have more chance of row buffer hit. SAM-sub also faces the second and third challenges similar to RC-NVM, thereby, its performance falls behind SAM-IO and SAM-en.

The performance of GS-DRAM is close to SAM. However, to achieve such performance, GS-DRAM requires a complex record alignment strategy, which divides a record into multiple 64B segments (Figure 11(b)). More importantly, it sacrifices reliability, which is unacceptable for IMDB applications. On the other hand, when enhanced with ECC, the performance of GS-DRAM-ecc declines distinctly. This is because the additional ECC data in GS-DRAM require extra data transfer and reduce the throughput. The results show that the performance of GS-DRAM-ecc is only 64~68% of SAM in Q-type queries.

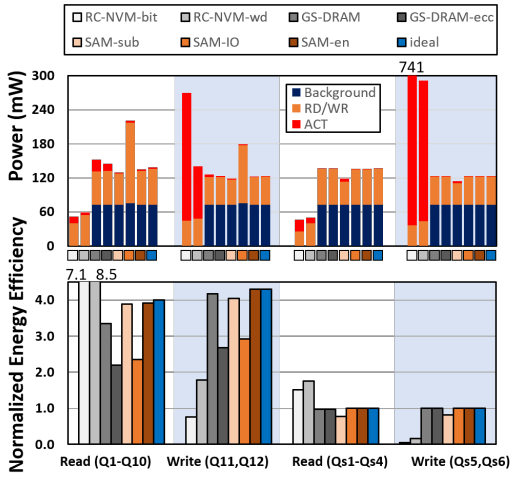


Figure 13: Power and energy efficiency (normalized to row-store) of different designs on Q and Qs queries

Figure 13 shows the power and energy efficiency, categorized by the read/write type of Q/Qs queries. We draw several conclusions from the results. First, compared to baseline (row-store), although the power of SAM-IO is increased by 1.8× and 1.5× when running the read-type and write-type Q queries, the energy efficiency of

SAM-IO is improved by 2.4× and 2.9×, respectively, as it avoids the transfer of unused data on channel. Second, the power of SAM-IO is higher than other DRAM-based designs, which present roughly the same power, because SAM-IO cannot avoid the internal transfer of unused data. Third, the NVM-based designs show better power and energy efficiency on read, but worse on write. This can be attributed to the character of RRAM, which consumes near-zero background power, but with significant write power. Fourth, for Qs queries, all DRAM-based designs show roughly the same power and energy efficiency as the baseline, because there is no strided access in Qs queries and they all work under a regular mode.

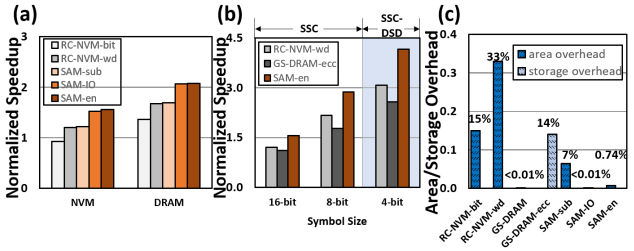


Figure 14: (a) Performance of RC-NVM and SAM using different memory technologies, (b) performance of RC-NVM-wd, GS-DRAM, and SAM-en with different granularities, (c) area overhead of different designs

To clarify the impact of memory technology, we construct a DRAM-based RC-NVM and an NVM-based SAM by configuring their timing parameters according to the related technology. Note that the RC-NVM based approach using a DRAM substrate can induce significant area overhead (larger than 200%) [9, 48]. As shown in Figure 14(a), RC-NVM-wd and SAM-sub show nearly the same performance on the same substrate. But RC-NVM always falls behind SAM-IO and SAM-en, regardless of what substrate they use. We also analyze the impact of different strided granularity. A finer granularity here corresponds to a smaller symbol size of chipkill ECC. As shown in Figure 14(b), finer granularity, which improves bandwidth utilization, helps achieve better performance. Obviously, SAM-en outperforms RC-NVM-word and GS-DRAM-ecc under all scenarios. Figure 14(c) shows the area or storage overhead of different designs. Again, our proposed SAM has an obvious advantage over others. Note that Figure 14(c) does not show the extra metal layers in the NVM-based designs.

Next, we analyze an arithmetic query and an aggregate query, as shown in table 3, with varied parameters, i.e., selectivity, productivity, and record size. The ideal case is either a row-store or column-store that is preferred when the query is configured with specific parameters. To simplify the simulation, we choose SAM-en, GS-DRAM-ecc, and RC-NVM-wd as representatives.

Figure 15(a), (b), and (c) show the speedup results of processing the arithmetic query under different selectivity settings when there are 8, 64, and 128 fields projected in a random manner. Across all these settings, the performance of SAM-en is either better than or comparable to the other designs, which highlights the benefits of SAM-en. Specifically, in Figure 15(a), when the selectivity is increased, the efficiency of strided accesses can be improved with more row buffer hit, the speedup thereby tends to rise. However, the tendency is diminished when there are more fields projected, which becomes more suitable for the baseline (row store), as shown in Figure 15(b) and (c).

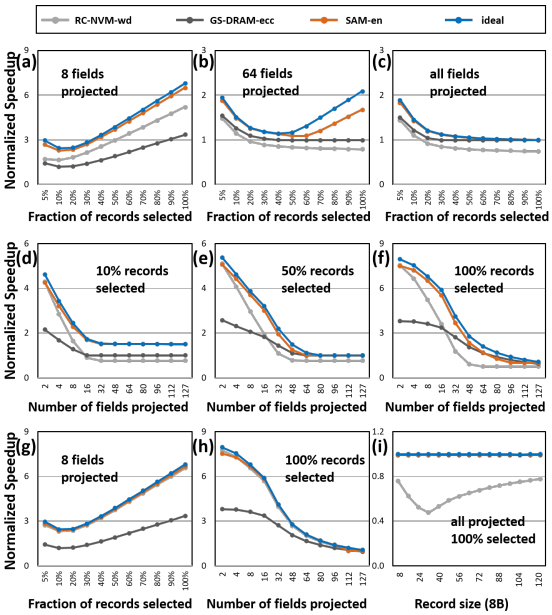


Figure 15: Speedup (normalized to row-store) of RC-NVM-wd, GS-DRAM-ecc, and SAM-en with different parameters, e.g., selectivity, projectivity, and record size

Figure 15(d), (e), and (f) show the results of experiments where the selectivity is fixed and projectivity is varied (in a random manner). From these results, we draw similar conclusions to Figure 15(a), (b), and (c). When the projectivity is increased, the efficiency of baseline is improved, the speedup thereby declines. On the other hand, the speedup rises with the increment of selectivity.

Figure 15(g) and (h) show the speedup results of processing an aggregate query with the same settings as Figure 15(a) and (f). The only difference is that the performance of RC-NVM-wd is improved, nearly the same as SAM-en. This is because the aggregate query can process each field independently, which relieves the challenge of field switch in RC-NVM. Figure 15(i) indicates the influence of record size on a query with 100% projectivity and selectivity. Only RC-NVM-wd shows degraded performance. This can be attributed to the inefficient record layout in RC-NVM, which allocates consecutive records across a large number of rows, increasing the possibility of bank conflict. In summary, SAM-en outperforms RC-NVM-wd and GS-DRAM-ecc and keeps close to the ideal case (row- or column-store) in almost all conditions.

7 CONCLUSION

In conclusion, we present SAM, a series of solutions for server memories (×4 DRAM chips) to achieve better efficiency of strided accesses in IMDB applications, meanwhile being compatible with chipkill ECC.

ACKNOWLEDGMENTS

This work is supported in part by US National Science Foundation #1617071, #1718080, #1725657, #1910413, and #2011146. The authors thank the anonymous reviewers for their constructive comments.

REFERENCES

[1] Inc. Advanced Micro Devices (AMD). 2007. kernel developer's guide for AMD NPT family 0Fh processors.

[2] Inc. Advanced Micro Devices (AMD). 2013. BIOS and Kernel Developer's Guide (BKDG) for AMD Family 15h Models 10h-1Fh Processors.

[3] Ioannis Alagiannis, Stratos Idreos, and Anastasia Ailamaki. 2014. H2O: a hands-free adaptive store. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. 1103–1114.

[4] Joy Arulraj, Andrew Pavlo, and Prashanth Menon. 2016. Bridging the archipelago between row-stores and column-stores for hybrid workloads. In *Proceedings of the 2016 International Conference on Management of Data*. 583–598.

[5] Kuljit S Bains and John Halbert. 2011. Common memory device for variable device width and scalable pre-fetch and page size. US Patent 7,957,216.

[6] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al. 2011. The gem5 simulator. *ACM SIGARCH computer architecture news* 39, 2 (2011), 1–7.

[7] Niladrish Chatterjee, Mike O'Connor, Donghyuk Lee, Daniel R Johnson, Stephen W Keckler, Minsoo Rhu, and William J Dally. 2017. Architecting an energy-efficient DRAM system for gpus. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 73–84.

[8] Ke Chen, Sheng Li, Naveen Muralimanohar, Jung Ho Ahn, Jay B Brockman, and Norman P Jouppi. 2012. CACTI-3DD: Architecture-level modeling for 3D die-stacked DRAM main memory. In *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 33–38.

[9] Yen-Hao Chen and Yi-Yu Liu. 2013. Dual-addressing memory architecture for two-dimensional memory access patterns. In *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 71–76.

[10] Elliott Cooper-Balis and Bruce Jacob. 2010. Fine-grained activation for power reduction in DRAM. *IEEE Micro* 30, 3 (2010), 34–47.

[11] Intel Corp. 2011. Intel Xeon Processor E7 Family: Reliability, Availability, and Serviceability".

[12] Leonardo Ecco, Adam Kostrzewa, and Rolf Ernst. 2016. Minimizing DRAM rank switching overhead for improved timing bounds and performance. In *2016 28th Euromicro Conference on Real-Time Systems (ECRTS)*. IEEE, 3–13.

[13] Gerd Frankowsky and Barbara Vasquez. 2004. Fuse programmable I/O organization. US Patent 6,707,746.

[14] Sumitha George, Minli Julie Liao, Huaipan Jiang, Jagadish B Kotra, Mahmut T Kandemir, Jack Sampson, and Vijaykrishnan Narayanan. 2018. MDACache: Caching for multi-dimensional-access memories. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 841–854.

[15] Product Guide. 2018. https://www.samsung.com/semiconductor/global.semi/file/resource/2018/06/DDR4_Product_guide_May.18.pdf

[16] Hewlett-Packard. 2011. HP Advanced Memory Error Detection Technology.

[17] International Business Machines Corp. (IBM). [n. d.]. Chipkill Memory. http://ps-2.kev009.com/pccbbs/pc_servers/chipkilf.pdf

[18] Bruce Jacob, David Wang, and Spencer Ng. 2010. *Memory systems: cache, DRAM, disk*. Morgan Kaufmann.

[19] JEDEC. 2012. JESD79-4: JEDEC Standard DDR4 SDRAM.

[20] JEDEC. 2020. JESD79-5: JEDEC Standard DDR5 SDRAM.

[21] Min Kyu Jeong, Doe Hyun Yoon, Dam Sunwoo, Mike Sullivan, Ikhwan Lee, and Mattan Erez. 2012. Balancing DRAM locality and parallelism in shared memory CMP systems. In *IEEE International Symposium on High-Performance Comp Architecture*. IEEE, 1–12.

[22] Xun Jian, Henry Duwe, John Sartori, Vilas Sridharan, and Rakesh Kumar. 2013. Low-power, low-storage-overhead chipkill correct via multi-line error correction. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. 1–12.

[23] Xun Jian and Rakesh Kumar. 2013. Adaptive reliability chipkill correct (arcc). In *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 270–281.

[24] Akifumi Kawahara, Ryotaro Azuma, Yuuichirou Ikeda, Ken Kawai, Yoshikazu Katoh, Yukio Hayakawa, Kiyotaka Tsuji, Shinichi Yoneda, Atsushi Himeno, Kazuhiko Shimakawa, et al. 2012. An 8 Mb multi-layered cross-point ReRAM macro with 443 MB/s write throughput. *IEEE Journal of Solid-State Circuits* 48, 1 (2012), 178–185.

[25] Brent Keeth, R Jacob Baker, Brian Johnson, and Feng Lin. 2007. *DRAM circuit design: fundamental and high-speed topics*. Vol. 13. John Wiley & Sons.

[26] Jungrae Kim, Michael Sullivan, and Mattan Erez. 2015. Bamboo ECC: Strong, safe, and flexible codes for reliable computer memory. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 101–112.

[27] Kie-Bong Ku and Tae-yun Kim. 2000. Circuit for setting width of input/output data in semiconductor memory device. US Patent 6,141,273.

[28] Benjamin C Lee, Engin Ipek, Onur Mutlu, and Doug Burger. 2009. Architecting phase change memory as a scalable dram alternative. In *Proceedings of the 36th annual international symposium on Computer architecture*. 2–13.

[29] Minli Julie Liao and Jack Sampson. 2020. D-SOAP: Dynamic Spatial Orientation Affinity Prediction for Caching in Multi-Orientation Memory Systems. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 581–595.

[30] JS Liptay. 2000. Structural aspects of the System/360 Model 85, PartII: the cache. In *Readings in computer architecture*. 373–379.

[31] Sangkug Lym, Heonjae Ha, Yongkee Kwon, Chun-kai Chang, Jungrae Kim, and Mattan Erez. 2018. ERUCA: Efficient DRAM Resource Utilization and Resource Conflict Avoidance for Memory System Parallelism. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 670–682.

[32] Micron. [n. d.]. Micron DDR4 Data Sheet. https://www.micron.com/-/media/client/global/documents/products/data-sheet/dram/ddr4/8gb_ddr4_sdram.pdf

[33] Micron. 2019. DDR4 Power Calculator 4.0. http://www.micron.com/products/dram

[34] Janani Mukundan, Hillery Hunter, Kyu-hyoun Kim, Jeffrey Stuecheli, and José F Martínez. 2013. Understanding and mitigating refresh overheads in high-density DDR4 DRAM systems. *ACM SIGARCH Computer Architecture News* 41, 3 (2013), 48–59.

[35] Mike O'Connor, Niladrish Chatterjee, Donghyuk Lee, John Wilson, Aditya Agrawal, Stephen W Keckler, and William J Dally. 2017. Fine-grained DRAM: Energy-efficient DRAM for extreme bandwidth systems. In *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 41–54.

[36] Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz, and Stefan Mangard. 2016. DRAMA: Exploiting DRAM addressing for cross-cpu attacks. In *25th {USENIX} security symposium ({USENIX} security 16)*. 565–581.

[37] Matthew Poremba, Tao Zhang, and Yuan Xie. 2015. Nvmain 2.0: A user-friendly memory simulator to model (non-) volatile memory systems. *IEEE Computer Architecture Letters* 14, 2 (2015), 140–143.

[38] Rambus. 2010. DRAM Power Model. https://www.rambus.com/energy/

[39] Jeffrey B Rothman and Alan Jay Smith. 2000. Sector cache design and performance. In *Proceedings 8th international symposium on modeling, analysis and simulation of computer and telecommunication systems (cat. no. pr00728)*. IEEE, 124–133.

[40] Vivek Seshadri, Thomas Mullins, Amirali Boroumand, Onur Mutlu, Phillip B Gibbons, Michael A Kozuch, and Todd C Mowry. 2015. Gather-scatter DRAM: In-DRAM address translation to improve the spatial locality of non-unit strided accesses. In *Proceedings of the 48th International Symposium on Microarchitecture*. 267–280.

[41] Mark E Shaw, Christian Petersen, and Lidia Mihaela Warnes. 2012. Memory module having a memory device configurable to different data pin configurations. US Patent 8,116,144.

[42] Kyomin Sohn, Taesik Na, Indal Song, Yong Shim, Wonil Bae, Sanghee Kang, Dongsu Lee, Hangyun Jung, Seokhun Hyun, Hanki Jeoung, et al. 2012. A 1.2 V 30 nm 3.2 Gb/s/pin 4 Gb DDR4 SDRAM with dual-error detection and PVT-tolerant data-fetch scheme. *IEEE journal of solid-state circuits* 48, 1 (2012), 168–177.

[43] Young Hoon Son, O Seongil, Yuhwan Ro, Jae W Lee, and Jung Ho Ahn. 2013. Reducing memory access latency with asymmetric DRAM bank organizations. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*. 380–391.

[44] Vilas Sridharan and Dean Liberty. 2012. A study of DRAM failures in the field. In *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–11.

[45] Synopsys. 2019. Synopsys White Paper. https://www.synopsys.com/dw/doc.php/wp/which_DDR_SDRAM_memory_to_use_and_when_LI.pdf

[46] Aniruddha N Udipi, Naveen Muralimanohar, Niladrish Chatterjee, Rajeev Balasubramonian, Al Davis, and Norman P Jouppi. 2010. Rethinking DRAM design and organization for energy-constrained multi-cores. In *Proceedings of the 37th annual international symposium on Computer architecture*. 175–186.

[47] Minghua Wang, Zhi Zhang, Yueqiang Cheng, and Surya Nepal. 2020. DRAMDig: a knowledge-assisted tool to uncover DRAM address mapping. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[48] Peng Wang, Shuo Li, Guangyu Sun, Xiaoyang Wang, Yiran Chen, Hai Li, Jason Cong, Nong Xiao, and Tao Zhang. 2018. Rc-nvm: Enabling symmetric row and column memory accesses for in-memory databases. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 518–530.

[49] Zheng Pei Wu, Rodolfo Pellizzoni, and Danlu Guo. 2016. A composable worst case latency analysis for multi-rank dram devices under open row policy. *Real-Time Systems* 52, 6 (2016), 761–807.

[50] Xin Xin, Youtao Zhang, and Jun Yang. 2020. Reducing DRAM access latency via helper rows. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[51] Cong Xu, Dimin Niu, Naveen Muralimanohar, Rajeev Balasubramonian, Tao Zhang, Shimeng Yu, and Yuan Xie. 2015. Overcoming the challenges of crossbar resistive memory architectures. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 476–488.

[52] Doe Hyun Yoon, Min Kyu Jeong, and Mattan Erez. 2011. Adaptive granularity memory systems: A tradeoff between storage efficiency and throughput. In *Proceedings of the 38th annual international symposium on Computer architecture*. 295–306.

[53] Doe Hyun Yoon, Min Kyu Jeong, Michael Sullivan, and Mattan Erez. 2012. The dynamic granularity memory system. In *2012 39th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 548–560.

[54] Tao Zhang, Ke Chen, Cong Xu, Guangyu Sun, Tao Wang, and Yuan Xie. 2014. Half-DRAM: A High-bandwidth and Low-power DRAM Architecture from the Rethinking of Fine-grained Activation. In *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*. IEEE, 349–360.

[55] Hongzhong Zheng, Jiang Lin, Zhao Zhang, Eugene Gorbatov, Howard David, and Zhichun Zhu. 2008. Mini-rank: Adaptive DRAM architecture for improving memory power efficiency. In *2008 41st IEEE/ACM International Symposium on Microarchitecture*. IEEE, 210–221.