A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM

Yoongu Kim Vivek Seshadri Donghyuk Lee Jamie Liu Onur Mutlu
Carnegie Mellon University

Abstract

Modern DRAMs have multiple banks to serve multiple memory requests in parallel. However, when two requests go to the same bank, they have to be served serially, exacerbating the high latency of off-chip memory. Adding more banks to the system to mitigate this problem incurs high system cost. Our goal in this work is to achieve the benefits of increasing the number of banks with a low cost approach. To this end, we propose three new mechanisms that overlap the latencies of different requests that go to the same bank. The key observation exploited by our mechanisms is that a modern DRAM bank is implemented as a collection of subarrays that operate largely independently while sharing few global peripheral structures.

Our proposed mechanisms (SALP-1, SALP-2, and MASA) mitigate the negative impact of bank serialization by overlapping different components of the bank access latencies of multiple requests that go to different subarrays within the same bank. SALP-1 requires no changes to the existing DRAM structure and only needs reinterpretation of some DRAM timing parameters. SALP-2 and MASA require only modest changes (< 0.15% area overhead) to the DRAM peripheral structures, which are much less design constrained than the DRAM core. Evaluations show that all our schemes significantly improve performance for both single-core systems and multi-core systems. Our schemes also interact positively with application-aware memory request scheduling in multi-core systems.

1. Introduction

To improve overall performance, modern systems expose multiple independent memory requests to the memory controller using two orthogonal approaches. First, a single processing core employs techniques to tolerate memory latency which generate multiple concurrent cache misses that can potentially be served in parallel (e.g., out-of-order execution [60], non-blocking caches [27], fine-grained multithreading [52, 58], runahead execution [5, 39] and other memory-level parallelism mechanisms [4, 45]). Second, multiple cores concurrently run multiple applications or threads, each of which generates independent memory requests. The effectiveness of these approaches in improving system performance critically depends on whether the concurrent memory requests are actually served in parallel. To be able to serve multiple requests in parallel, modern DRAM systems employ multiple banks that can be accessed independently. Unfortunately, if two memory requests go to the same bank, they have to be served one after another. This is what is referred to as a *bank conflict*.

Bank conflicts have two negative consequences. First, they serialize requests that can potentially be served in parallel. This, coupled with the long latency of bank access, significantly reduces memory bandwidth utilization and can cause cores to stall, leading to lower system performance. Furthermore, a request scheduled after a write request to the same bank incurs an additional delay called the *write-recovery latency*. This aggravates the impact of bank conflicts by increasing the latency of subsequent requests.

Second, a bank conflict can lead to thrashing in the bank's row-buffer. A *row-buffer*, present in each bank, effectively acts as a single-entry direct-mapped "cache" for the rows in the bank. Memory requests that hit in the row-buffer incur much lower latency than those that miss in the row-buffer. In a multi-core system, requests from different applications are interleaved with each other. When such interleaved requests lead to bank conflicts, they can "evict" the row that is present in the row-buffer. As a result, requests of an application that could have otherwise hit in the row-buffer will miss in the row-buffer, significantly degrading the performance of the application (and potentially the overall system) [37, 40, 56, 67].

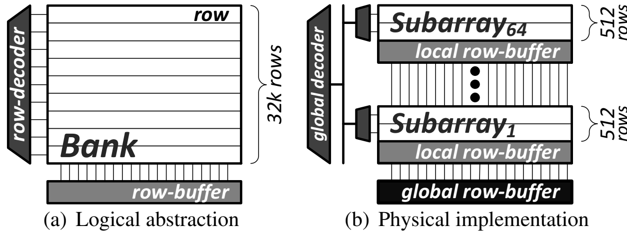
A solution to the bank conflict problem is to increase the number of DRAM banks in the system. While current memory subsystems theoretically allow for three ways of doing so, they all come at a significantly high cost. First, one can increase the number of banks in the DRAM chip itself. However, for a constant storage capacity, increasing the number of banks-per-chip significantly increases the DRAM die area (and thus chip cost) due to replicated decoding logic, routing, and drivers at each bank [66]. Second, one can increase the number of banks in a channel by multiplexing the channel with many memory modules, each of which is a collection of banks. Unfortunately, this increases the electrical load on the channel, causing it to run at a significantly reduced frequency [8, 9]. Third, one can add more memory channels to increase the overall bank count. Unfortunately, this increases the pin-count in the processor package, which is an expensive resource.[1] Considering both the low growth rate of pin-count and the prohibitive cost of pins in general, it is clear that increasing the number of channels is not a scalable solution.

Our goal in this paper is to mitigate the detrimental effects of bank conflicts with a low-cost approach. We make two key observations that lead to our proposed mechanisms.

Observation 1. A modern DRAM bank is *not* implemented as a monolithic component with a single row-buffer. Implementing a DRAM bank as a monolithic structure requires very long wires (called bitlines), to connect the row-buffer to all the rows in the bank, which can significantly increase the access latency (Section 2.3). Instead, a bank consists of multiple *subarrays*, each with its own local row-buffer, as shown in Figure 1. Subarrays within a bank share *i)* a global row-address decoder and *ii)* a set of global bitlines which connect their local row-buffers to a global row-buffer.

Observation 2. The latency of bank access consists of three major components: *i)* opening a row containing the required data (referred to as *activation*), *ii)* accessing the data (*read* or *write*), and *iii)* closing the row (*precharging*). In existing systems, all three operations must be completed for one memory request before serving another request to a different row within the same bank, even if the two rows reside in different subarrays. However, this need not be the case for two reasons. First, the activation and precharging operations are mostly local to each subarray, enabling the opportunity to overlap these operations to different subarrays within the same bank. Second, if we reduce the resource sharing among subarrays, we can enable activation operations to different subarrays to be performed in parallel and, in addition, also exploit the existence of multiple local row-buffers to cache more than one row in a single bank, enabling the opportunity to improve row-buffer hit rate.

[1] Intel Sandy Bridge dedicates 264 pins for two channels [14]. IBM POWER7 dedicates 640 pins for eight channels [51].



(a) Logical abstraction (b) Physical implementation

Figure 1. DRAM bank organization

Based on these observations, our thesis in this paper is that exposing the subarray-level internal organization of a DRAM bank to the memory controller would allow the controller to exploit the independence between subarrays within the same bank and reduce the negative impact of bank conflicts. To this end, we propose three different mechanisms for exploiting *subarray-level parallelism*. Our proposed mechanisms allow the memory controller to overlap or eliminate different latency components required to complete multiple requests going to different subarrays within the same bank.

First, *SALP-1* (Subarray-Level-Parallelism-1) overlaps the latency of closing a row of one subarray with that of opening a row in a different subarray within the same bank by pipelining the two operations one after the other. SALP-1 requires *no* changes to the existing DRAM structure. Second, *SALP-2* (Subarray-Level-Parallelism-2) allows the memory controller to start opening a row in a subarray before closing the currently open row in a different subarray. This allows SALP-2 to overlap the latency of opening a row with the write-recovery period of another row in a different subarray, and further improve performance compared to SALP-1. SALP-2 requires the addition of small latches to each subarray's peripheral logic. Third, *MASA* (Multitude of Activated Subarrays) exploits the fact that each subarray has its own *local row-buffer* that can potentially "cache" the most recently accessed row in that subarray. MASA reduces hardware resource sharing between subarrays to allow the memory controller to *i)* activate multiple subarrays in parallel to reduce request serialization, *ii)* concurrently keep local row-buffers of multiple subarrays active to significantly improve row-buffer hit rate. In addition to the change needed by SALP-2, MASA requires only the addition of a single-bit latch to each subarray's peripheral logic as well as a new 1-bit global control signal.

Our paper makes the following **contributions**.

- We exploit the existence of subarrays within each DRAM bank to mitigate the effects of bank conflicts. We propose three mechanisms, SALP-1, SALP-2, and MASA, that overlap (to varying degrees) the latency of accesses to different subarrays. SALP-1 does not require any modifications to existing DRAM structure, while SALP-2 and MASA introduce small changes only to the subarrays' peripheral logic.

- We exploit the existence of local subarray row-buffers within DRAM banks to mitigate row-buffer thrashing. We propose MASA that allows multiple such subarray row-buffers to remain activated at any given point in time. We show that MASA can significantly increase row-buffer hit rate while incurring only modest implementation cost.

- We perform a thorough analysis of area and power overheads of our proposed mechanisms. MASA, the most aggressive of our proposed mechanisms, incurs a DRAM chip area overhead of 0.15% and a modest power cost of 0.56mW per each additionally activated subarray.

- We identify that *tWR* (*bank write-recovery*[2]) exacerbates the negative impact of bank conflicts by increasing the latency of critical read requests. We show that SALP-2 and MASA are effective at minimizing the negative effects of *tWR*.

- We evaluate our proposed mechanisms using a variety of system configurations and show that they significantly improve performance for single-core systems compared to conventional DRAM: 7%/13%/17% for SALP-1/SALP-2/MASA, respectively. Our schemes also interact positively with application-aware memory scheduling algorithms and further improve performance for multi-core systems.

2. Background: DRAM Organization

As shown in Figure 2, DRAM-based main memory systems are logically organized as a hierarchy of channels, ranks, and banks. In today's systems, *banks* are the smallest memory structures that can be accessed in parallel with respect to each other. This is referred to as *bank-level parallelism* [24, 41]. Next, a *rank* is a collection of banks across multiple DRAM chips that operate in lockstep.[3] Banks in different ranks are fully decoupled with respect to their device-level electrical operation and, consequently, offer better bank-level parallelism than banks in the same rank. Lastly, a *channel* is the collection of all banks that share a common physical link (command, address, data buses) to the processor. While banks from the same channel experience contention at the physical link, banks from different channels can be accessed completely independently of each other. Although the DRAM system offers varying degrees of parallelism at different levels in its organization, two memory requests that access the same bank must be served one after another. To understand why, let us examine the logical organization of a DRAM bank as seen by the memory controller.

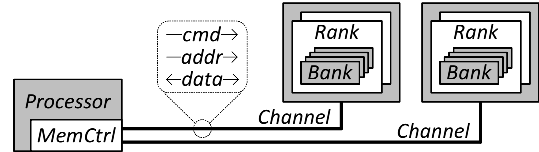


Figure 2. Logical hierarchy of main memory

2.1. Bank: Logical Organization & Operation

Figure 3 presents the logical organization of a DRAM bank. A DRAM bank is a two-dimensional array of capacitor-based DRAM cells. It is viewed as a collection of *rows*, each of which consists of multiple *columns*. Each bank contains a *row-buffer* which is an array of sense-amplifiers that act as latches. Spanning a bank in the column-wise direction are the *bitlines*, each of which can connect a sense-amplifier to any of the cells in the same column. A *wordline* (one for each row) determines whether or not the corresponding row of cells is connected to the bitlines.

To serve a memory request that accesses data at a particular row and column address, the memory controller issues three commands to a bank in the order listed below. Each command triggers a specific sequence of events within the bank.

1. `ACTIVATE`: read the entire row into the row-buffer
2. `READ`/`WRITE`: access the column from the row-buffer
3. `PRECHARGE`: de-activate the row-buffer

[2] Write-recovery (explained in Section 2.2) is different from the *bus-turnaround penalty* (read-to-write, write-to-read), which is addressed by several prior works [3, 29, 55].

[3] A DRAM rank typically consists of eight DRAM chips, each of which has eight banks. Since the chips operate in lockstep, the rank has only eight independent banks, each of which is the set of the ith bank across all chips.

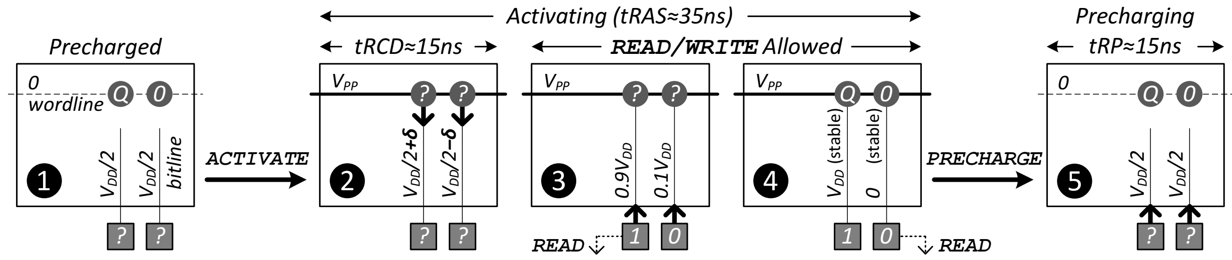


Figure 4. DRAM bank operation: Steps involved in serving a memory request [17] ($V_{PP} > V_{DD}$)

Category	RowCmd↔RowCmd			RowCmd↔ColCmd			ColCmd↔ColCmd			ColCmd→DATA	
Name	tRC	$tRAS$	tRP	$tRCD$	$tRTP$	tWR^*	$tCCD$	$tRTW^\dagger$	$tWTR^*$	CL	CWL
Commands	A→A	A→P	P→A	A→R/W	R→P	W*→P	R(W)→R(W)	R→W	W*→R	R→DATA	W→DATA
Scope	Bank	Bank	Bank	Bank	Bank	Bank	Channel	Rank	Rank	Bank	Bank
Value (ns)	**∼50**	∼35	13-15	13-15	∼7.5	**15**	5-7.5	11-15	∼7.5	13-15	10-15

A: ACTIVATE– P: PRECHARGE– R: READ– W: WRITE
∗ Goes into effect after the last write *data*, not from the WRITE command
† Not explicitly specified by the JEDEC DDR3 standard [18]. Defined as a function of other timing constraints.

Table 1. Summary of DDR3-SDRAM timing constraints (derived from Micron's 2Gb DDR3-SDRAM datasheet [33])

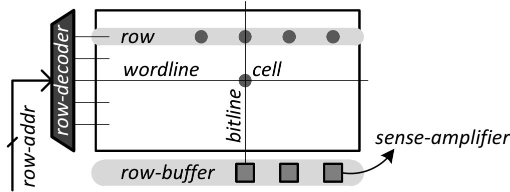


Figure 3. DRAM Bank: Logical organization

ACTIVATE **Row**. Before a DRAM row can be activated, the bank must be in the *precharged state* (State ❶, Figure 4). In this state, all the bitlines are maintained at a voltage-level of $\frac{1}{2}V_{DD}$. Upon receiving the ACTIVATE command along with a row-address, the wordline corresponding to the row is raised to a voltage of V_{PP}, connecting the row's cells to the bitlines (State ❶→❷). Subsequently, depending on whether a cell is charged (Q) or uncharged (0), the bitline voltage is slightly perturbed towards V_{DD} or 0 (State ❷). The row-buffer "senses" this perturbation and "amplifies" it in the same direction (State ❷→❸). During this period when the bitline voltages are still in transition, the cells are left in an undefined state. Finally, once the bitline voltages stabilize, cell charges are restored to their original values (State ❹). The time taken for this entire procedure is called *tRAS* ($\approx$ 35ns).

READ/WRITE **Column**. After an ACTIVATE, the memory controller issues a READ or a WRITE command, along with a column address. The timing constraint between an ACTIVATE and a subsequent column command (READ/WRITE) is called *tRCD* ($\approx$ 15ns). This reflects the time required for the data to be latched in the row-buffer (State ❸). If the next request to the bank also happens to access the same row, it can be served with only a column command, since the row has already been activated. As a result, this request is served more quickly than a request that requires a new row to be activated.

PRECHARGE **Bank**. To activate a new row, the memory controller must first take the bank back to the precharged state (State ❺). This happens in two steps. First, the wordline corresponding to the currently activated row is lowered to zero voltage, disconnecting the cells from the bitlines. Second, the bitlines are driven to a voltage of $\frac{1}{2}V_{DD}$. The time taken for this operation is called *tRP* ($\approx$ 15ns).

2.2. Timing Constraints

As described above, different DRAM commands have different latencies. Undefined behavior may arise if a command is issued before the previous command is fully processed. To prevent such occurrences, the memory controller must obey a set of timing constraints while issuing commands to a bank. These constraints define when a command becomes ready to be scheduled, depending on all other commands issued before it to the same channel, rank, or bank. Table 1 summarizes the most important timing constraints between ACTIVATE (A), PRECHARGE (P), READ (R), and WRITE (W) commands. Among these, two timing constraints (highlighted in bold) are the critical bottlenecks for bank conflicts: tRC and tWR.

tRC. Successive ACTIVATEs to the same bank are limited by *tRC* (row-cycle time), which is the sum of tRAS and tRP [17]. In the worst case, when N requests all access different rows within the same bank, the bank must activate a new row and precharge it for each request. Consequently, the last request experiences a DRAM latency of $N \cdot tRC$, which can be hundreds or thousands of nanoseconds.

tWR. After issuing a WRITE to a bank, the bank needs additional time, called *tWR* (write-recovery latency), while its row-buffer drives the bitlines to their new voltages. A bank cannot be precharged before then – otherwise, the new data may not have been safely stored in the cells. Essentially, after a WRITE, the bank takes longer to reach State ❹ (Figure 4), thereby delaying the next request to the same bank even longer than tRC.

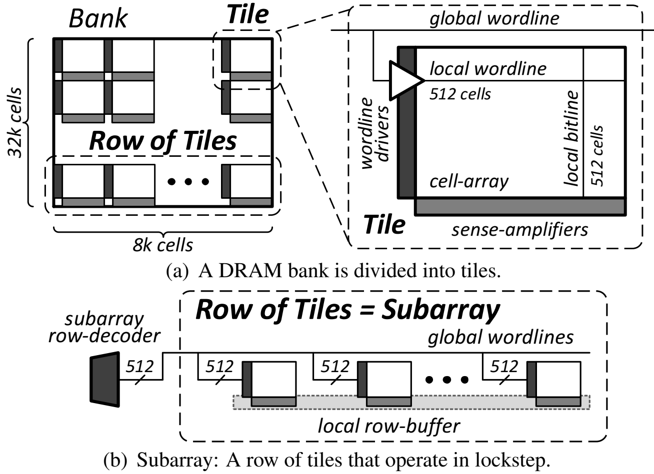
2.3. Subarrays: Physical Organization of Banks

Although we have described a DRAM bank as a monolithic array of rows equipped with a single row-buffer, implementing a large bank (e.g., 32k rows and 8k cells-per-row) in this manner requires long bitlines. Due to their large parasitic capacitance, long bitlines have two disadvantages. First, they make it difficult for a DRAM cell to cause the necessary perturbation required for reliable sensing [21]. Second, a sense-amplifier takes longer to drive a long bitline to a target voltage-level, thereby increasing the latency of activation and precharging.

To avoid the disadvantages of long bitlines, as well as long wordlines, a DRAM bank is divided into a two-dimensional array of tiles [17, 21, 63], as shown in Figure 5(a). A *tile* comprises *i)* a cell-array, whose typical dimensions are 512 cells×512 cells [63], *ii)* sense-amplifiers, and *iii)* wordline-drivers that strengthen the signals on the *global wordlines* before relaying them to the local wordlines.

All tiles in the horizontal direction – a "row of tiles" – share the same set of global wordlines, as shown in Figure 5(b). Therefore, these tiles are activated and precharged in lockstep.

We abstract such a "row of tiles" as a single entity that we refer to as a *subarray*.[4] More specifically, a subarray is a collection of cells that share a *local row-buffer* (all sense-amplifiers in the horizontal direction) and a *subarray row-decoder* [17].



(a) A DRAM bank is divided into tiles.

(b) Subarray: A row of tiles that operate in lockstep.

Figure 5. A DRAM bank consists of tiles and subarrays.

As shown in Figure 6, all subarray row-decoders in a bank are driven by the shared *global row-address latch* [17]. The latch holds a partially pre-decoded row-address (from the global row-decoder) that is routed by the *global address-bus* to all subarray row-decoders, where the remainder of the decoding is performed. A partially pre-decoded row-address allows subarray row-decoders to remain small and simple without incurring the large global routing overhead of a fully pre-decoded row-address [17]. All subarrays in a bank also share a *global row-buffer* [17, 22, 36] that can be connected to any one of the local row-buffers through a set of *global bitlines* [17]. The purpose of the global row-buffer is to sense the perturbations caused by the local row-buffer on the global bitlines and to amplify the perturbations before relaying them to the I/O drivers. Without a global row-buffer, the local row-buffers will take a long time to drive their values on the global bitlines, thereby significantly increasing the access latency.[5]

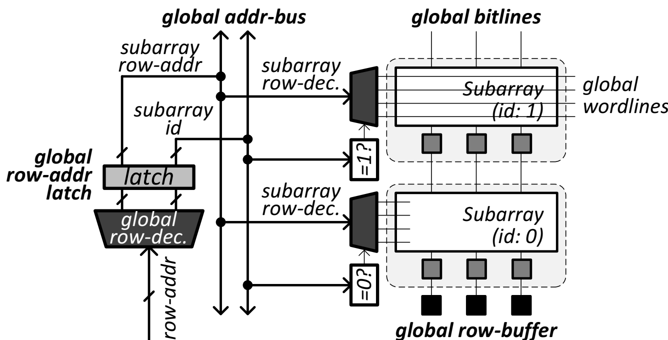


Figure 6. DRAM Bank: Physical organization

Although all subarrays within a bank share some global structures (e.g., the global row-address latch and the global bitlines), some DRAM operations are completely local to a subarray or use the global structures minimally. For example, precharging is completely local to a subarray and does not use any of the shared structures, whereas activation uses only the global row-address latch to drive the corresponding wordline.

Unfortunately, existing DRAMs cannot fully exploit the independence between different subarrays for two main reasons. First, *only one* row can be activated (i.e., only one wordline can be raised) within each bank at a time. This is because the global row-address latch, which determines which wordline within the bank is raised, is shared by all subarrays. Second, although each subarray has its own local row-buffer, *only one* subarray can be activated at a time. This is because all local row-buffers are connected to the global row-buffer by a *single* set of global bitlines. If multiple subarrays were allowed to be activated[6] at the same time when a column command is issued, all of their row-buffers would attempt to drive the global bitlines, leading to a short-circuit.

Our goal in this paper is to reduce the performance impact of bank conflicts by exploiting the existence of subarrays to enable their parallel access and to allow multiple activated local row-buffers within a bank, using low cost mechanisms.

3. Motivation

To understand the benefits of exploiting the subarray-organization of DRAM banks, let us consider the two examples shown in Figure 7. The first example (top) presents the timeline of four memory requests being served at the same bank in a subarray-oblivious baseline.[7] The first two requests are write requests to two rows in different subarrays. The next two requests are read requests to the same two rows, respectively. This example highlights three key problems in the operation of the baseline system. First, successive requests are completely serialized. This is in spite of the fact that they are to different subarrays and could potentially have been partially parallelized. Second, requests that immediately follow a `WRITE` incur the additional write-recovery latency (Section 2.2). Although this constraint is completely local to a subarray, it delays a subsequent request even to a different subarray. Third, both rows are activated twice, once for each of their two requests. After serving a request from a row, the memory controller is forced to de-activate the row since the subsequent request is to a different row within the same bank. This significantly increases the overall service time of the four requests.

The second example (bottom) in Figure 7 presents the timeline of serving the four requests when the two rows belong to different *banks*, instead of to different subarrays within the same bank. In this case, the overall service time is significantly reduced due to three reasons. First, rows in different banks can be activated in parallel, overlapping a large portion of their access latencies. Second, the write-recovery latency is local to a bank and hence, does not delay a subsequent request to another bank. In our example, since consecutive requests to the same bank access the same row, they are also not delayed by the write-recovery latency. Third, since the row-buffers of the two banks are completely independent, requests do not evict each other's rows from the row-buffers. This eliminates the need for extra `ACTIVATE`s for the last two requests, further reducing the overall service time. However, as we described in Section 1, increasing the number of banks in the system significantly increases the system cost.

[4] We use the term subarray to refer to a *single* "row of tiles" (alternatively, a block [28]). Others have used the term subarray to refer to *i)* an individual tile [62, 63], *ii)* a single "row of tiles" [66], or *iii)* multiple "rows of tiles" [36].

[5] Better known as *main* [17] or *I/O* [22, 36] *sense-amplifiers*, the global row-buffer lies between the local row-buffers and the I/O driver. It is narrower than a local row-buffer; column-selection logic (not shown in Figure 6) multiplexes the wide outputs of the local row-buffer onto the global row-buffer.

[6] We use the phrases "subarray is activated (precharged)" and "row-buffer is activated (precharged)" interchangeably as they denote the same phenomenon.

[7] This timeline (as well as other timelines we will show) is for illustration purposes and does not incorporate all DRAM timing constraints.

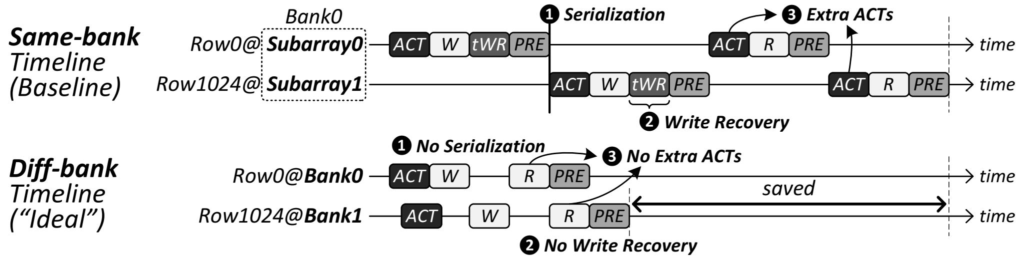


Figure 7. Service timeline of four requests to two different rows. The rows are in the same bank (top) or in different banks (bottom).

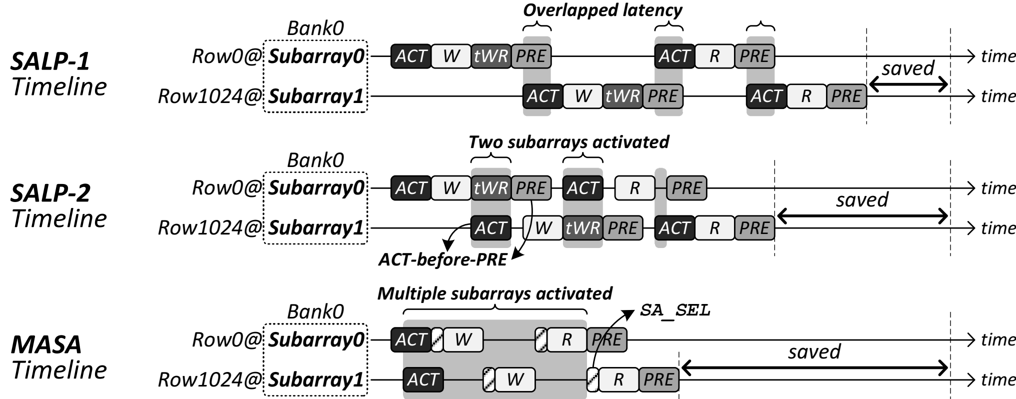


Figure 8. Service timeline of four requests to two different rows. The rows are in the same bank, but in different subarrays.

The **core thesis** of this paper is that most of the performance benefits of having multiple banks can be achieved at a significantly lower cost by exploiting the potential parallelism offered by subarrays within a bank. To this end, we propose three mechanisms that exploit the existence of subarrays with little or no change to the existing DRAM designs.

4. Overview of Proposed Mechanisms

We call our three proposed schemes *SALP-1*, *SALP-2* and *MASA*. As shown in Figure 8, each scheme is a successive refinement over the preceding scheme such that the performance benefits of the most sophisticated scheme, MASA, subsumes those of SALP-1 and SALP-2. We explain the key ideas of each scheme below.

4.1. SALP-1: Subarray-Level-Parallelism-1

The key observation behind SALP-1 is that precharging and activation are mostly local to a subarray. SALP-1 exploits this observation to overlap the precharging of one subarray with the activation of another subarray. In contrast, existing systems always serialize precharging and activation to the same bank, conservatively provisioning for when they are to the same subarray. SALP-1 requires *no modifications* to existing DRAM structure. It only requires reinterpretation of an existing timing constraint (tRP) and, potentially, the addition of a new timing constraint (explained in Section 5.1). Figure 8 (top) shows the performance benefit of SALP-1.

4.2. SALP-2: Subarray-Level-Parallelism-2

While SALP-1 pipelines the precharging and activation of different subarrays, the relative ordering between the two commands is still preserved. This is because existing DRAM banks do not allow two subarrays to be activated at the same time. As a result, the write-recovery latency (Section 2.2) of an activated subarray not only delays a `PRECHARGE` to itself, but also delays a subsequent `ACTIVATE` to another subarray. Based on the observation that the write-recovery latency is also local to a subarray, SALP-2 (our second mechanism) issues the `ACTIVATE` to another subarray *before* the `PRECHARGE` to the currently activated subarray. As a result, SALP-2 can overlap the write-recovery of the currently activated subarray with the activation of another subarray, further reducing the service time compared to SALP-1 (Figure 8, middle).

However, as highlighted in the figure, SALP-2 requires two subarrays to remain activated at the same time. This is not possible in existing DRAM banks as the global row-address latch, which determines the wordline in the bank that is raised, is shared by all the subarrays. In Section 5.2, we will show how to enable SALP-2 by eliminating this sharing.

4.3. MASA: Multitude of Activated Subarrays

Although SALP-2 allows two subarrays within a bank to be activated, it requires the controller to precharge one of them before issuing a column command (e.g., `READ`) to the bank. This is because when a bank receives a column command, all activated subarrays in the bank will connect their local row-buffers to the global bitlines. If more than one subarray is activated, this will result in a short circuit. As a result, SALP-2 cannot allow multiple subarrays to concurrently remain activated and serve column commands.

The key idea of MASA (our third mechanism) is to allow *multiple* subarrays to be activated at the same time, while allowing the memory controller to *designate* exactly one of the activated subarrays to drive the global bitlines during the next column command. MASA has two advantages over SALP-2. First, MASA overlaps the activation of different subarrays within a bank. Just before issuing a column command to any of the activated subarrays, the memory controller *designates* one particular subarray whose row-buffer should serve the column command. Second, MASA eliminates extra `ACTIVATE`s to the same row, thereby mitigating row-buffer thrashing. This is because the local row-buffers of multiple subarrays can remain activated at the same time without experiencing collisions on the global bitlines. As a result, MASA further improves performance compared to SALP-2 (Figure 8, bottom).

As indicated in the figure, to designate one of the multi-

ple activated subarrays, the controller needs a new command, SA_SEL (*subarray-select*). In addition to the changes required by SALP-2, MASA requires a single-bit latch per subarray to denote whether a subarray is *designated* or not (Section 5.3).

5. Implementation

Our three proposed mechanisms assume that the memory controller is aware of the existence of subarrays (to be described in Section 5.4) and can determine which subarray a particular request accesses. All three mechanisms require reinterpretation of existing DRAM timing constraints or addition of new ones. SALP-2 and MASA also require small, non-intrusive modifications to the DRAM chip. In this section, we describe the changes required by each mechanism in detail.

5.1. SALP-1: Relaxing tRP

As previously described, SALP-1 overlaps the precharging of one subarray with the subsequent activation of another subarray. However, by doing so, SALP-1 violates the timing constraint *tRP* (row-precharge time) imposed between consecutive PRECHARGE and ACTIVATE commands to the same bank. The reason why tRP exists is to ensure that a previously activated subarray (Subarray X in Figure 9) has fully reached the precharged state before it can again be activated. Existing DRAM banks provide that guarantee by conservatively delaying an ACTIVATE to any subarray, even to a subarray that is not the one being precharged. But, for a subarray that is *already* in the precharged state (Subarray Y in Figure 9), it is safe to activate it while another subarray is being precharged. So, as long as consecutive PRECHARGE and ACTIVATE commands are to different subarrays, the ACTIVATE can be issued before tRP has been satisfied.[8]

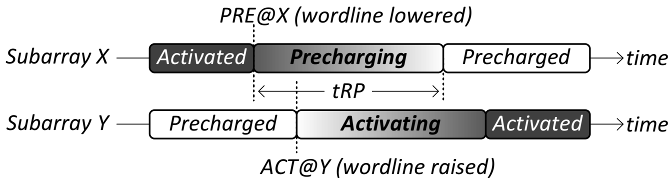


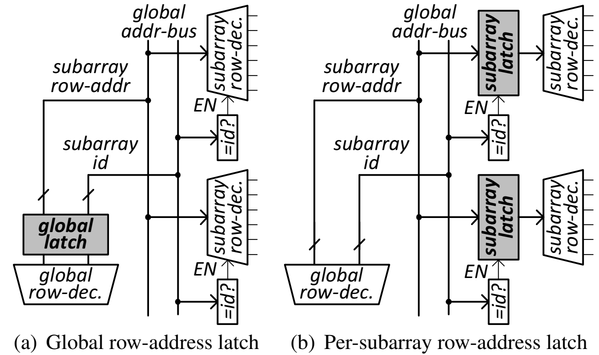
Figure 9. Relaxing tRP between two different subarrays.

Limitation of SALP-1. SALP-1 *cannot* overlap the write-recovery of one subarray with the activation of another subarray. This is because both write-recovery and activation require their corresponding wordline to remain raised for the entire duration of the corresponding operation. However, in existing DRAM banks, the global row-address latch determines the unique wordline within the bank that is raised (Section 2.3). Since this latch is shared across all subarrays, it is not possible to have two raised wordlines within a bank, even if they are in different subarrays. SALP-2 addresses this issue by adding row-address latches to each subarray.

5.2. SALP-2: Per-Subarray Row-Address Latches

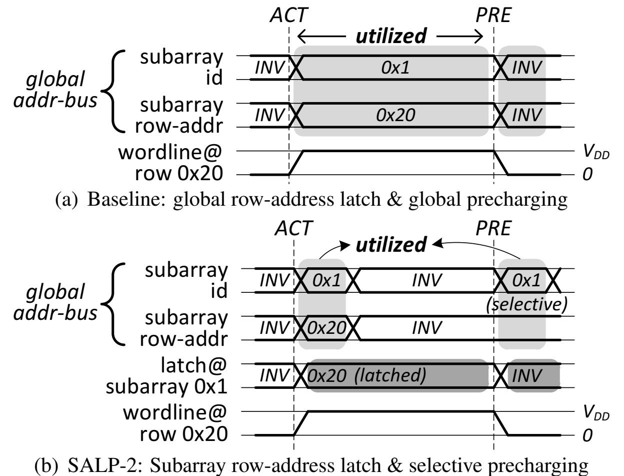
The goal of SALP-2 is to further improve performance compared to SALP-1 by overlapping the write-recovery latency of one subarray with the activation of another subarray. For this purpose, we propose two changes to the DRAM chip: *i)* latched subarray row-decoding and *ii)* selective precharging.

Latched Subarray Row-Decoding. The key idea of *latched subarray row-decoding* (LSRD) is to push the global row-address latch to individual subarrays such that each subarray has its own row-address latch, as shown in Figure 10. When an ACTIVATE is issued to a subarray, the subarray row-address is stored in the latch. This latch feeds the subarray row-decoder, which in turn drives the corresponding wordline within the subarray. Figure 11 shows the timeline of subarray activation with and without LSRD. Without LSRD, the global row-address bus is utilized by the subarray until it is precharged. This prevents the controller from activating another subarray. In contrast, with LSRD, the global address-bus is utilized only until the row-address is stored in the corresponding subarray's latch. From that point on, the latch drives the wordline, freeing the global address-bus to be used by another subarray.



(a) Global row-address latch (b) Per-subarray row-address latch

Figure 10. SALP-2: Latched Subarray Row-Decoding



(a) Baseline: global row-address latch & global precharging

(b) SALP-2: Subarray row-address latch & selective precharging

Figure 11. Activating/precharging wordline-0x20 of subarray-0x1.

Selective Precharging. Since existing DRAMs do not allow a bank to have more than one raised wordline, a PRECHARGE is designed to lower *all* wordlines within a bank to zero voltage. In fact, the memory controller does not even specify a row address when it issues a PRECHARGE. A bank lowers all wordlines by broadcasting an INV (invalid) value on the global row-address bus.[9] However, when there are two activated subarrays (each with a raised wordline) SALP-2 needs to be able to *selectively precharge* only one of the subarrays. To achieve this, we require that PRECHARGEs be issued with the corresponding subarray ID. When a bank receives a PRECHARGE to a subarray, it places the subarray ID and INV (for the subarray row-address) on the global row-address bus. This ensures that only that specific subarray is precharged. Selective precharging requires the memory controller to remember the ID of the subarray to be precharged. This requires mod-

[8] We assume that it is valid to issue the two commands in consecutive DRAM cycles. Depending on vendor-specific microarchitecture, an additional precharge-to-activate timing constraint *tPA* ($<$ tRP) may be required.

[9] When each subarray receives the INV values for both subarray ID and subarray row-address, it lowers all its wordlines and precharges all its bitlines.

est storage overhead at the memory controller – one subarray ID per bank.

Timing Constraints. Although SALP-2 allows two activated subarrays, no column command can be issued during that time. This is because a column command electrically connects the row-buffers of all activated subarrays to the global bitlines – leading to a short-circuit between the row-buffers. To avoid such hazards on the global bitlines, SALP-2 must wait for a column command to be processed before it can activate another subarray in the same bank. Hence, we introduce two new timing constraints for SALP-2: *tRA* (read-to-activate) and *tWA* (write-to-activate).

Limitation of SALP-2. As described above, SALP-2 requires a bank to have exactly one activated subarray when a column command is received. Therefore, SALP-2 cannot address the row-buffer thrashing problem.

5.3. MASA: Designating an Activated Subarray

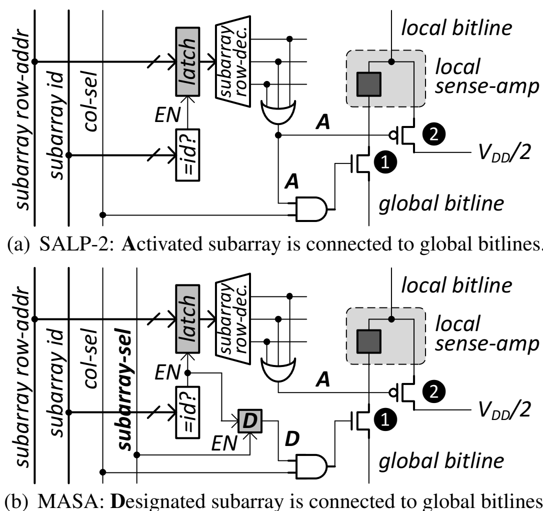
The key idea behind MASA is to allow multiple activated subarrays, but to ensure that only a single subarray's row-buffer is connected to the global bitlines on a column command. To achieve this, we propose the following changes to the DRAM microarchitecture in addition to those required by SALP-2: *i)* addition of a *designated-bit latch* to each subarray, *ii)* introduction of a new DRAM command, `SA_SEL` (subarray-select), and *iii)* routing of a new global wire (subarray-select).

Designated-Bit Latch. In SALP-2 (and existing DRAM), an activated subarray's local sense-amplifiers are connected to the global bitlines on a column command. The connection between each sense-amplifier and the corresponding global bitline is established when an access transistor, ❶ in Figure 12(a), is switched on. All such access transistors (one for each sense-amplifier) within a subarray are controlled by the same 1-bit signal, called *activated* (**A** in figure), that is raised only when the subarray has a raised wordline.[10] As a result, it is *not* possible for a subarray to be activated while at the same time be disconnected from the global bitlines on a column command.

To enable MASA, we propose to decouple the control of the access transistor from the wordlines, as shown in Figure 12(b). To this end, we propose a separate 1-bit signal, called *designated* (**D** in figure), to control the transistor independently of the wordlines. This signal is driven by a *designated-bit latch*, which must be set by the memory controller in order to enable a subarray's row-buffer to be connected to the global bitlines. To access data from one particular activated subarray, the memory controller sets the designated-bit latch of the subarray and clears the designated-bit latch of all other subarrays. As a result, MASA allows multiple subarrays to be activated within a bank while ensuring that one subarray (the *designated* one) can at the same time serve column commands. Note that MASA still requires the *activated* signal to control the precharge transistors ❷ that determine whether or not the row-buffer is in the precharged state (i.e., connecting the local bitlines to $\frac{1}{2}V_{DD}$).

Subarray-Select Command. To allow the memory controller to selectively set and clear the designated-bit of any subarray, MASA requires a new DRAM command, which we call `SA_SEL` (subarray-select). To set the designated-bit of a particular subarray, the controller issues a `SA_SEL` along with the row-address that corresponds to the raised wordline within the subarray. Upon receiving this command, the bank sets the designated-bit for only the subarray and clears the designated-bits of all other subarrays. After this operation, all subsequent

[10]The *activated* signal can be abstracted as a logical *OR* across all wordlines in the subarray, as shown in Figure 12(a). The exact implementation of the signal is microarchitecture-specific.



(a) SALP-2: **A**ctivated subarray is connected to global bitlines.

(b) MASA: **D**esignated subarray is connected to global bitlines.

Figure 12. MASA: Designated-bit latch and subarray-select signal

column commands are served by the designated subarray.

To update the designated-bit latch of each subarray, MASA requires a new global control signal that acts as a strobe for the latch. We call this signal *subarray-select*. When a bank receives the `SA_SEL` command, it places the corresponding subarray ID and subarray row-address on the global address-bus and briefly raises the subarray-select signal. At this point, the subarray whose ID matches the ID on the global address-bus will set its designated-bit, while all other subarrays will clear their designated-bit. Note that `ACTIVATE` also sets the designated-bit for the subarray it activates, as it expects the subarray to serve all subsequent column commands. In fact, from the memory controller's perspective, `SA_SEL` is the same as `ACTIVATE`, except that for `SA_SEL`, the supplied row-address corresponds to a wordline that is already raised.

Timing Constraints. Since designated-bits determine which activated subarray will serve a column command, they should not be updated (by `ACTIVATE`/`SA_SEL`) while a column command is in progress. For this purpose, we introduce two timing constraints called tRA (read-to-activate/select) and tWA (write-to-activate/select). These are the same timing constraints introduced by SALP-2.

Additional Storage at the Controller. To support MASA, the memory controller must track the status of all subarrays within each bank. A subarray's status represents *i)* whether the subarray is activated, *ii)* if so, which wordline within the subarray is raised, and *iii)* whether the subarray is designated to serve column commands. For the system configurations we evaluate (Section 8), maintaining this information incurs a storage overhead of less than 256 bytes at the memory controller.

While MASA overlaps multiple `ACTIVATE`s to the same bank, it must still obey timing constraints such as tFAW and tRRD that limit the rate at which `ACTIVATE`s are issued to the entire DRAM chip. We evaluate the power and area overhead of our three proposed mechanisms in Section 6.

5.4. Exposing Subarrays to the Memory Controller

For the memory controller to employ our proposed schemes, it requires the following three pieces of information: *i)* the number of subarrays per bank, *ii)* whether the DRAM supports SALP-1, SALP-2 and/or MASA, and *iii)* the values for the timing constraints tRA and tWA. Since these parameters are heavily dependent on vendor-specific microarchitecture and process technology, they may be difficult to standardize. Therefore, we describe an alternate way of exposing these parameters to the

memory controller.

Serial Presence Detect. Multiple DRAM chips are assembled together on a circuit board to form a DRAM module. On every DRAM module lies a separate 256-byte EEPROM, called the *serial presence detect* (SPD), which contains information about both the chips and the module, such as timing, capacity, organization, etc. [19]. At system boot time, the SPD is read by the BIOS, so that the memory controller can correctly issue commands to the DRAM module. In the SPD, more than a hundred extra bytes are set aside for use by the manufacturer and the end-user [19]. This storage is more than sufficient to store subarray-related parameters required by the controller.

Number of Subarrays per Bank. The number of subarrays within a bank is expected to increase for larger capacity DRAM chips that have more rows. However, certain manufacturing constraints may prevent all subarrays from being accessed in parallel. To increase DRAM yield, every subarray is provisioned with a few spare rows that can replace faulty rows [17, 21]. If a faulty row in one subarray is mapped to a spare row in another subarray, then the two subarrays can no longer be accessed in parallel. To strike a trade-off between high yield and the number of subarrays that can be accessed in parallel, spare rows in each subarray can be restricted to replace faulty rows only within a subset of the other subarrays. With this guarantee, the memory controller can still apply our mechanisms to different subarray groups. In our evaluations (Section 9.2), we show that just having 8 subarray groups can provide significant performance improvements. From now on, we refer to an independently accessible subarray group as a "subarray."

6. Power & Area Overhead

Of our three proposed schemes, SALP-1 does not incur any additional area or power overhead since it does not make any modifications to the DRAM structure. On the other hand, SALP-2 and MASA require subarray row-address latches that minimally increase area and power. MASA also consumes additional static power due to multiple activated subarrays and additional dynamic power due to extra `SA_SEL` commands. We analyze these overheads in this section.

6.1. Additional Latches

SALP-2 and MASA add a subarray row-address latch to each subarray. While MASA also requires an additional single-bit latch for the designated-bit, its area and power overheads are insignificant compared to the subarray row-address latches. In most of our evaluations, we assume 8 subarrays-per-bank and 8 banks-per-chip. As a result, a chip requires a total of 64 row-address latches, where each latch stores the 40-bit partially pre-decoded row-address.[11] Scaling the area from a previously proposed latch design [26] to $55nm$ process technology, each row-address latch occupies an area of $42.9\mu m^2$. Overall, this amounts to a 0.15% area overhead compared to a 2Gb DRAM chip fabricated using $55nm$ technology (die area = $73mm^2$ [46]). Similarly, normalizing the latch power consumption to $55nm$ technology and 1.5V operating voltage, a 40-bit latch consumes 72.2μW additional power for each `ACTIVATE`. This is negligible compared to the activation power, 51.2mW (calculated using DRAM models [31, 46, 63]).

6.2. Multiple Activated Subarrays

To estimate the additional static power consumption of multiple activated subarrays, we compute the difference in the maximum current between the cases when *all* banks are activated (I_{DD3N}, 35mA) and when *no* bank is activated (I_{DD2N}, 32mA) [33]. For a DDR3 chip which has 8 banks and operates at 1.5V, each activated local row-buffer consumes at most 0.56mW additional static power in the steady state. This is small compared to the baseline static power of 48mW per DRAM chip.

6.3. Additional `SA_SEL` Commands

To switch between multiple activated subarrays, MASA issues additional `SA_SEL` commands. Although `SA_SEL` is the same as `ACTIVATE` from the memory controller's perspective (Section 5.3), internally, `SA_SEL` does not involve the subarray core, i.e., a subarray's cells. Therefore, we estimate the dynamic power of `SA_SEL` by subtracting the subarray core's power from the dynamic power of `ACTIVATE`, where the subarray core's power is the sum of the wordline and row-buffer power during activation [46]. Based on our analysis using DRAM modeling tools [31, 46, 63], we estimate the power consumption of `SA_SEL` to be 49.6% of `ACTIVATE`. MASA also requires a global subarray-select wire in the DRAM chip. However, compared to the large amount of global routing that is already present within a bank (40 bits of partially pre-decoded row-address and 1024 bits of fully decoded column-address), the overhead of one additional wire is negligible.

6.4. Comparison to Expensive Alternatives

As a comparison, we present the overhead incurred by two alternative approaches that can mitigate bank conflicts: *i)* increasing the number of DRAM banks and *ii)* adding an SRAM cache inside the DRAM chip.

More Banks. To add more banks, per-bank circuit components such as the global decoders and I/O-sense amplifiers must be replicated [17]. This leads to significant increase in DRAM chip area. Using the DRAM area model from Rambus [46, 63], we estimate that increasing the number of banks from 8 to 16, 32, and 64, increases the chip area by 5.2%, 15.5%, and 36.3%, respectively. Larger chips also consume more static power.

Additional SRAM Cache. Adding a separate SRAM cache within the DRAM chip (such "Cached DRAM" proposals are discussed in Section 7), can achieve similar benefits as utilizing multiple row-buffers across subarrays. However, this increases the DRAM chip area and, consequently, its static power consumption. We calculate the chip area penalty for adding SRAM caches using CACTI-D [59]. An SRAM cache that has a size of 8 Kbits (same as a row-buffer), 64 Kbits, and 512 Kbits increases DRAM chip area by 0.6%, 5.0%, and 38.8%, respectively. These figures do not include the additional routing logic that is required between the I/O sense-amplifiers and the SRAM cache.

7. Related Work

In this work, we propose three schemes that exploit the existence of subarrays within DRAM banks to mitigate the negative effects of bank conflicts. Prior works proposed increasing the performance and energy-efficiency of DRAM through approaches such as DRAM module reorganization, changes to DRAM chip design, and memory controller optimizations.

DRAM Module Reorganization. Threaded Memory Module [64], Multicore DIMM [1], and Mini-Rank [70] are all techniques that partition a DRAM rank (and the DRAM data-bus) into multiple rank-subsets [2], each of which can be operated independently. Although partitioning a DRAM rank into smaller rank-subsets increases parallelism, it narrows the data-bus of each rank-subset, incurring longer latencies to transfer a 64 byte cache-line. Fore example, having 8 mini-ranks

[11] A 2Gb DRAM chip with 32k rows has a 15-bit row-address. We assume 3:8 pre-decoding, which yields a 40-bit partially pre-decoded row-address.

Processor	1-16 cores, 5.3GHz, 3-wide issue, 8 MSHRs, 128-entry instruction window
Last-level cache	64B cache-line, 16-way associative, 512kB private cache-slice per core
Memory controller	64/64-entry read/write request queues per controller, FR-FCFS scheduler [47, 71], writes are scheduled in batches [3, 29, 55]
Memory	Timing: DDR3-1066 (8-8-8) [33], tRA: 4tCK, tWA: 14tCK Organization (default in bold): **1**-8 channels, **1**-8 ranks-per-channel, **8**-64 banks-per-rank, 1-**8**-128 subarrays-per-bank

Table 2. Configuration of simulated system

increases the data-transfer latency by 8 times (to 60 ns, assuming DDR3-1066) for all memory accesses. In contrast, our schemes increase parallelism without increasing latency. Furthermore, having many rank-subsets requires a correspondingly large number of DRAM chips to compose a DRAM rank, an assumption that does not hold in mobile DRAM systems where a rank may consist of as few as two chips [32]. However, since the parallelism exposed by rank-subsetting is orthogonal to our schemes, rank-subsetting can be combined with our schemes to further improve performance.

Changes to DRAM Design. Cached DRAM organizations, which have been widely proposed [7, 10, 11, 13, 20, 42, 49, 65, 69] augment DRAM chips with an additional SRAM cache that can store recently accessed data. Although such organizations reduce memory access latency in a manner similar to MASA, they come at increased chip area and design complexity (as Section 6.4 showed). Furthermore, cached DRAM only provides parallelism when accesses hit in the SRAM cache, while serializing cache misses that access the same DRAM bank. In contrast, our schemes parallelize DRAM bank accesses while incurring significantly lower area and logic complexity.

Since a large portion of the DRAM latency is spent driving the local bitlines [35], Fujitsu's FCRAM and Micron's RLDRAM proposed to implement shorter local bitlines (i.e., fewer cells per bitline) that are quickly drivable due to their lower capacitances. However, this significantly increases the DRAM die size (30-40% for FCRAM [50], 40-80% for RLDRAM [21]) because the large area of sense-amplifiers is amortized over a smaller number of cells.

A patent by Qimonda [44] proposed the high-level notion of separately addressable sub-banks, but it lacks concrete mechanisms for exploiting the independence between sub-banks. In the context of embedded DRAM, Yamauchi et al. proposed the Hierarchical Multi-Bank (HMB) [66] that parallelizes accesses to different subarrays in a fine-grained manner. However, their scheme adds complex logic to all subarrays. For example, each subarray requires a timer that automatically precharges a subarray after an access. As a result, HMB cannot take advantage of multiple row-buffers.

Although only a small fraction of the row is needed to serve a memory request, a DRAM bank wastes power by always activating an entire row. To mitigate this "overfetch" problem and save power, Udipi et al. [62] proposed two techniques (SBA and SSA).[12] In SBA, global wordlines are segmented and controlled separately so that tiles in the horizontal direction are not activated in lockstep, but selectively. However, this increases DRAM chip area by 12-100% [62]. SSA combines SBA with chip-granularity rank-subsetting to achieve even higher energy savings. But, both SBA and SSA increase DRAM latency, more significantly so for SSA (due to rank-subsetting).

A DRAM chip experiences bubbles in the data-bus, called the bus-turnaround penalty (*tWTR* and *tRTW* in Table 1), when transitioning from serving a write request to a read request, and vice versa [3, 29, 55]. During the bus-turnaround penalty, Chatterjee et al. [3] proposed to internally "prefetch" data for subsequent read requests into extra registers that are added to the DRAM chip.

[12] Udipi et al. use the term subarray to refer to an individual tile.

An IBM patent [25] proposed latched row-decoding to activate multiple wordlines in a DRAM bank simultaneously, in order to expedite the testing of DRAM chips by checking for defects in multiple rows at the same time.

Memory Controller Optimizations. To reduce bank conflicts and increase row-buffer locality, Zhang et al. proposed to randomize the bank address of memory requests by XOR hashing [68]. Sudan et al. proposed to improve row-buffer locality by placing frequently referenced data together in the same row [56]. Both proposals can be combined with our schemes to further improve parallelism and row-buffer locality.

Prior works have also proposed memory scheduling algorithms (e.g., [6, 16, 23, 24, 38, 40, 41, 43]) that prioritize certain favorable requests in the memory controller to improve system performance and/or fairness. Subarrays expose more parallelism to the memory controller, increasing the controller's flexibility to schedule requests.

8. Evaluation Methodology

We developed a cycle-accurate DDR3-SDRAM simulator that we validated against Micron's Verilog behavioral model [34] and DRAMSim2 [48]. We use this memory simulator as part of a cycle-level in-house x86 multi-core simulator, whose front-end is based on Pin [30]. We calculate DRAM dynamic energy consumption by associating an energy cost with each DRAM command, derived using the tools [31, 46, 63] and the methodology as explained in Section 6.[13]

Unless otherwise specified, our default system configuration comprises a single-core processor with a memory subsystem that has 1 channel, 1 rank-per-channel (RPC), 8 banks-per-rank (BPR), and 8 subarrays-per-bank (SPB). We also perform detailed sensitivity studies where we vary the numbers of cores, channels, ranks, banks, and subarrays. More detail on the simulated system configuration is provided in Table 2.

We use *line-interleaving* to map the physical address space onto the DRAM hierarchy (channels, ranks, banks, etc.). In line-interleaving, small chunks of the physical address space (often the size of a cache-line) are striped across different banks, ranks, and channels. Line-interleaving is utilized to maximize the amount of memory-level parallelism and is employed in systems such as Intel Nehalem [15], Sandy Bridge [14], Sun OpenSPARC T1 [57], and IBM POWER7 [51]. We use the *closed-row policy* in which the memory controller precharges a bank when there are no more outstanding requests to the activated row of that bank. The closed-row policy is often used in conjunction with line-interleaving since row-buffer locality is expected to be low. Additionally, we also show results for *row-interleaving* and the *open-row policy* in Section 9.3.

We use 32 benchmarks from SPEC CPU2006, TPC [61], and STREAM [54], in addition to a random-access mi-

[13] We consider dynamic energy dissipated by only the DRAM chip itself and do not include dynamic energy dissipated at the channel (which differs on a motherboard-by-motherboard basis).

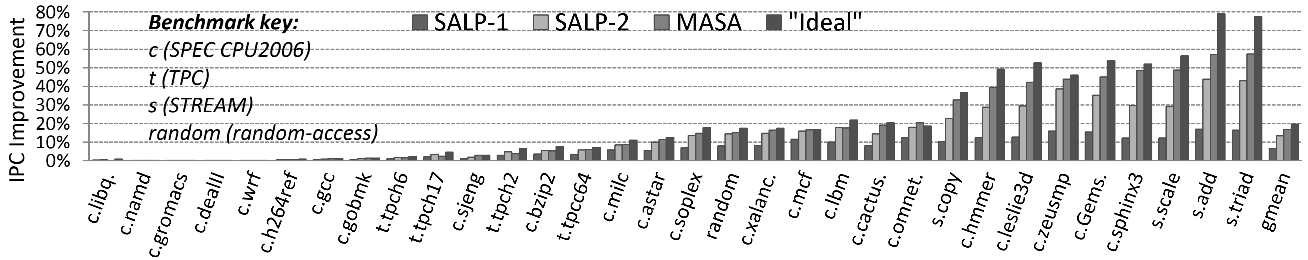


Figure 13. IPC improvement over the conventional subarray-oblivious baseline

crobenchmark similar in behavior to HPCC RandomAccess [12]. We form multi-core workloads by randomly choosing from only the benchmarks that access memory at least once every 1000 instructions. We simulate all benchmarks for 100 million instructions. For multi-core evaluations, we ensure that even the slowest core executes 100 million instructions, while other cores still exert pressure on the memory subsystem. To measure performance, we use instruction throughput for single-core systems and *weighted speedup* [53] for multi-core systems. We report results that are averaged across all 32 benchmarks for single-core evaluations and averaged across 16 different workloads for each multi-core system configuration.

9. Results

9.1. Individual Benchmarks (Single-Core)

Figure 13 shows the performance improvement of SALP-1, SALP-2, and MASA on a system with 8 subarrays-per-bank over a subarray-oblivious baseline. The figure also shows the performance improvement of an "Ideal" scheme which is the subarray-oblivious baseline with 8 times as many banks (this represents a system where all subarrays are fully independent). We draw two conclusions. First, SALP-1, SALP-2 and MASA consistently perform better than baseline for all benchmarks. On average, they improve performance by 6.6%, 13.4%, and 16.7%, respectively. Second, MASA captures most of the benefit of the "Ideal," which improves performance by 19.6% compared to baseline.

The difference in performance improvement across benchmarks can be explained by a combination of factors related to their individual memory access behavior. First, subarray-level parallelism in general is most beneficial for memory-intensive benchmarks that frequently access memory (benchmarks located towards the right of Figure 13). By increasing the memory throughput for such applications, subarray-level parallelism significantly alleviates their memory bottleneck. The average memory-intensity of the rightmost applications (i.e., those that gain >5% performance with SALP-1) is 18.4 MPKI (last-level cache misses per kilo-instruction), compared to 1.14 MPKI of the leftmost applications.

Second, the advantage of SALP-2 is large for applications that are write-intensive. For such applications, SALP-2 can overlap the long write-recovery latency with the activation of a subsequent access. In Figure 13, the three applications (that improve more than 38% with SALP-2) are among both the most memory-intensive (>25 MPKI) and the most write-intensive (>15 WMPKI).

Third, MASA is beneficial for applications that experience frequent bank conflicts. For such applications, MASA parallelizes accesses to different subarrays by concurrently activating multiple subarrays (`ACTIVATE`) and allowing the application to switch between the activated subarrays at low cost (`SA_SEL`). Therefore, the subarray-level parallelism offered by MASA can be gauged by the `SA_SEL`-to-`ACTIVATE` ratio. For the nine applications that benefit more than 30% from MASA, on average, one `SA_SEL` was issued for every two `ACTIVATEs`, compared to one-in-seventeen for all the other applications. For a few benchmarks, MASA performs slightly worse than SALP-2. The baseline scheduling algorithm used with MASA tries to overlap as many `ACTIVATEs` as possible and, in the process, inadvertently delays the column command of the most critical request which slightly degrades performance for these benchmarks.[14]

9.2. Sensitivity to Number of Subarrays

With more subarrays, there is greater opportunity to exploit subarray-level parallelism and, correspondingly, the improvements provided by our schemes also increase. As the number of subarrays-per-bank is swept from 1 to 128, Figure 14 plots the IPC improvement, average read latency,[15] and memory-level parallelism[16] of our three schemes (averaged across 32 benchmarks) compared to the subarray-oblivious baseline.

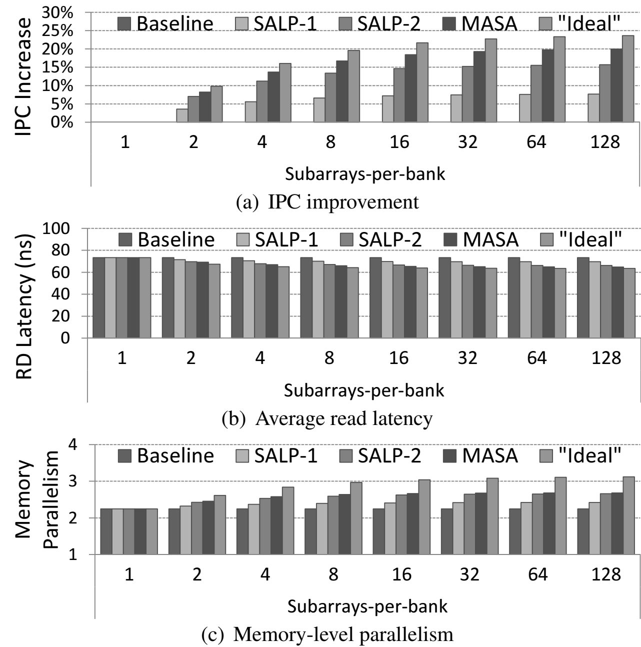


Figure 14. Sensitivity to number of subarrays-per-bank

Figure 14(a) shows that SALP-1, SALP-2, and MASA consistently improve IPC as the number of subarrays-per-bank increases. But, the gains are diminishing because most of the bank conflicts are parallelized for even a modest number of subarrays. Just 8 subarrays-per-bank captures more than

[14]For one benchmark, MASA performs slightly better than the "Ideal" due to interactions with the scheduler.

[15]Average memory latency for read requests, which includes: *i)* queuing delay at the controller, *ii)* bank access latency, and *iii)* data-transfer latency.

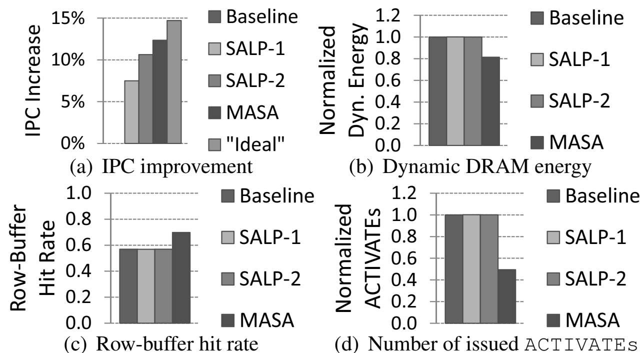
[16]The average number of requests that are *being served*, given that there is at least one such request. A request is defined as *being served* from when the first command is issued on its behalf until its data-transfer has completed.

80% of the IPC improvement provided by the same mechanism with 128 subarrays-per-bank. The performance improvements of SALP-1, SALP-2, and MASA are a direct result of reduced memory access latency and increased memory-level parallelism, as shown in Figures 14(b) and 14(c), respectively. These improvements are two-sides of the same coin: by increasing the parallelism across subarrays, our mechanisms are able to overlap the latencies of multiple memory requests to reduce the average memory access latency.

9.3. Sensitivity to System Configuration

Mapping and Row Policy. In *row-interleaving*, as opposed to line-interleaving, a contiguous chunk of the physical address space is mapped to each DRAM row. Row-interleaving is commonly used in conjunction with the *open-row policy* so that a row is never eagerly closed – a row is left open in the row-buffer until another row needs to be accessed. Figure 15 shows the results (averaged over 32 benchmarks) of employing our three schemes on a row-interleaved, open-row system.

As shown in Figure 15(a), the IPC improvements of SALP-1, SALP-2, and MASA are 7.5%, 10.6%, and 12.3%, where MASA performs nearly as well as the "Ideal" (14.7%). However, the gains are lower than compared to a line-interleaved, closed-row system. This is because the subarray-oblivious baseline performs better on a row-interleaved, open-row system (due to row-buffer locality), thereby leaving less headroom for our schemes to improve performance. MASA also improves DRAM energy-efficiency in a row-interleaved system. Figure 15(b) shows that MASA decreases DRAM dynamic energy consumption by 18.6%. Since MASA allows multiple row-buffers to remain activated, it increases the row-buffer hit rate by 12.8%, as shown in Figure 15(c). This is clear from Figure 15(d), which shows that 50.1% of the `ACTIVATEs` issued in the baseline are converted to `SA_SELs` in MASA.



(a) IPC improvement (b) Dynamic DRAM energy

(c) Row-buffer hit rate (d) Number of issued `ACTIVATEs`

Figure 15. Row-interleaving and open-row policy.

Number of Channels, Ranks, Banks. Even for highly provisioned systems with unrealistically large numbers of channels, ranks, and banks, exploiting subarray-level parallelism improves performance significantly, as shown in Figure 16. This is because even such systems cannot completely remove all bank conflicts due to the well-known birthday paradox: even if there were 365 banks (very difficult to implement), with just 23 concurrent memory requests, the probability of a bank conflict between any two requests is more than 50% (for 64 banks, only 10 requests are required). Therefore, exploiting subarray-level parallelism still provides performance benefits. For example, while an 8-channel baseline system provides more than enough memory bandwidth (<4% data-bus utilization), MASA reduces access latency by parallelizing bank conflicts, and improves performance by 8.6% over the baseline.

As more ranks/banks are added to the same channel, increased contention on the data-bus is likely to be the performance limiter. That is why adding more ranks/banks does not provide as large benefits as adding more channels (Figure 16).[17] Ideally, for the highest performance, one would increase the numbers of all three: channels/ranks/banks. However, as explained in Section 1, adding more channels is very expensive, whereas the number of ranks-/banks-per-channel is limited to a low number in modern high-frequency DRAM systems. Therefore, exploiting subarray-level parallelism is a cost-effective way of achieving the performance of many ranks/banks and, as a result, extracting the most performance from a given number of channels.

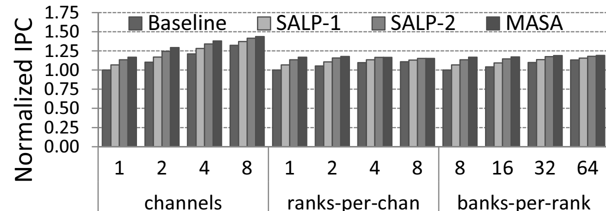


Figure 16. Memory configuration sweep (line-interleaved, closed-row). IPC normalized to: 1-channel, 1-RPC, 8-BPR, 8-SPB.

Number of Cores. As shown in Figure 17, our schemes improve performance of 8-core and 16-core systems with the FR-FCFS memory scheduler [47, 71]. However, previous studies have shown that destructive memory interference among applications due to FR-FCFS scheduling can severely degrade system performance [37, 40]. Therefore, to exploit the full potential of subarray-level parallelism, the scheduler should resolve bank conflicts in an application-aware manner. To study this effect, we evaluate our schemes with TCM [24], a state-of-the-art scheduler that mitigates inter-application interference. As shown in Figure 17, TCM outperforms FR-FCFS by 3.7%/12.3% on 8-core/16-core systems. When employed with the TCM scheduler, SALP-1/SALP-2/MASA further improve performance by 3.9%/5.9%/7.4% on the 8-core system and by 2.5%/3.9%/8.0% on the 16-core system. We also observe similar trends for systems using row-interleaving and the open-row policy (not shown due to space constraints). We believe that further performance improvements are possible by designing memory request scheduling algorithms that are both application-aware and subarray-aware.

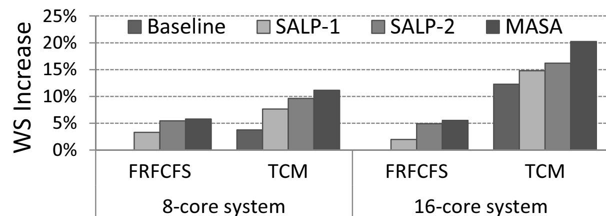


Figure 17. Multi-core weighted speedup improvement. Configuration: 2-channel, 2-RPC, line-interleaved, closed-row policy.

10. Conclusion

We introduced new techniques that exploit the existence of subarrays within a DRAM bank to mitigate the performance impact of bank conflicts. Our mechanisms are built on the key observation that subarrays within a DRAM bank operate largely independently and have their own row-buffers. Hence, the latencies of accesses to different subarrays within the same bank can potentially be overlapped to a large degree. We introduce three schemes that take advantage of this fact and pro-

[17]Having more ranks (as opposed to having just more banks) aggravates data-bus contention by introducing bubbles in the data-bus due to *tRTRS* (rank-to-rank switch penalty).

gressively increase the independence of operation of subarrays by making small modifications to the DRAM chip. Our most sophisticated scheme, MASA, enables *i)* multiple subarrays to be accessed in parallel, and *ii)* multiple row-buffers to remain activated at the same time in different subarrays, thereby improving both memory-level parallelism and row-buffer locality. We show that our schemes significantly improve system performance on both single-core and multi-core systems on a variety of workloads while incurring little (<0.15%) or no area overhead in the DRAM chip. Our techniques can also improve memory energy efficiency. We conclude that exploiting subarray-level parallelism in a DRAM bank can be a promising and cost-effective method for overcoming the negative effects of DRAM bank conflicts, without paying the large cost of increasing the number of banks in the DRAM system.

Acknowledgments

Many thanks to Uksong Kang, Hak-soo Yu, Churoo Park, Jung-Bae Lee, and Joo Sun Choi from Samsung for their helpful comments. We thank the anonymous reviewers for their feedback. We gratefully acknowledge members of the SAFARI group for feedback and for the stimulating intellectual environment they provide. We acknowledge the generous support of AMD, Intel, Oracle, and Samsung. This research was also partially supported by grants from NSF (CAREER Award CCF-0953246), GSRC, and Intel ARO Memory Hierarchy Program. Yoongu Kim is partially supported by a Ph.D. fellowship from the Korea Foundation for Advanced Studies.

References

[1] J. H. Ahn et al. Multicore DIMM: An energy efficient memory module with independently controlled DRAMs. *IEEE CAL*, Jan. 2009.
[2] J. H. Ahn et al. Improving system energy efficiency with memory rank subsetting. *ACM TACO*, Mar. 2012.
[3] N. Chatterjee et al. Staged reads: Mitigating the impact of DRAM writes on DRAM reads. In *HPCA*, 2012.
[4] Y. Chou et al. Microarchitecture optimizations for exploiting memory-level parallelism. In *ISCA*, 2004.
[5] J. Dundas and T. Mudge. Improving data cache performance by pre-executing instructions under a cache miss. In *ICS*, 1997.
[6] E. Ebrahimi et al. Parallel application memory scheduling. In *MICRO*, 2011.
[7] Enhanced Memory Systems. Enhanced SDRAM SM2604, 2002.
[8] H. Fredriksson and C. Svensson. Improvement potential and equalization example for multidrop DRAM memory buses. *IEEE Transactions on Advanced Packaging*, 2009.
[9] B. Ganesh et al. Fully-buffered DIMM memory architectures: Understanding mechanisms, overheads and scaling. In *HPCA*, 2007.
[10] C. A. Hart. CDRAM in a unified memory architecture. In *Compcon*, 1994.
[11] H. Hidaka et al. The cache DRAM architecture: A DRAM with an on-chip cache memory. *IEEE Micro*, Mar. 1990.
[12] HPCC. RandomAccess. `http://icl.cs.utk.edu/hpcc/`.
[13] W.-C. Hsu and J. E. Smith. Performance of cached DRAM organizations in vector supercomputers. In *ISCA*, 1993.
[14] Intel. 2nd Gen. Intel Core Processor Family Desktop Datasheet, 2011.
[15] Intel. Intel Core Desktop Processor Series Datasheet, 2011.
[16] E. Ipek et al. Self optimizing memory controllers: A reinforcement learning approach. In *ISCA*, 2008.
[17] K. Itoh. *VLSI Memory Chip Design*. Springer, 2001.
[18] JEDEC. Standard No. 79-3E. DDR3 SDRAM Specification, 2010.
[19] JEDEC. Standard No. 21-C. Annex K: Serial Presence Detect (SPD) for DDR3 SDRAM Modules, 2011.
[20] G. Kedem and R. P. Koganti. WCDRAM: A fully associative integrated cached-DRAM with wide cache lines. *CS-1997-03, Duke*, 1997.
[21] B. Keeth et al. *DRAM Circuit Design. Fundamental and High-Speed Topics*. Wiley-IEEE Press, 2007.
[22] R. Kho et al. 75nm 7Gb/s/pin 1Gb GDDR5 graphics memory device with bandwidth-improvement techniques. In *ISSCC*, 2009.
[23] Y. Kim et al. ATLAS: A scalable and high-performance scheduling algorithm for multiple memory controllers. In *HPCA*, 2010.
[24] Y. Kim et al. Thread cluster memory scheduling: Exploiting differences in memory access behavior. In *MICRO*, 2010.
[25] T. Kirihata. Latched row decoder for a random access memory. U.S. patent number 5615164, 1997.
[26] B.-S. Kong et al. Conditional-capture flip-flop for statistical power reduction. *IEEE JSSC*, 2001.
[27] D. Kroft. Lockup-free instruction fetch/prefetch cache organization. In *ISCA*, 1981.
[28] B. C. Lee et al. Architecting phase change memory as a scalable DRAM alternative. In *ISCA*, 2009.
[29] C. J. Lee et al. DRAM-aware last-level cache writeback: Reducing write-caused interference in memory systems. *TR-HPS-2010-002, UT Austin*, 2010.
[30] C.-K. Luk et al. Pin: Building customized program analysis tools with dynamic instrumentation. In *PLDI*, 2005.
[31] Micron. DDR3 SDRAM System-Power Calculator, 2010.
[32] Micron. 2Gb: x16, x32 Mobile LPDDR2 SDRAM, 2012.
[33] Micron. 2Gb: x4, x8, x16, DDR3 SDRAM, 2012.
[34] Micron. DDR3 SDRAM Verilog Model, 2012.
[35] M. J. Miller. Bandwidth engine serial memory chip breaks 2 billion accesses/sec. In *HotChips*, 2011.
[36] Y. Moon et al. 1.2V 1.6Gb/s 56nm 6F2 4Gb DDR3 SDRAM with hybrid-I/O sense amplifier and segmented sub-array architecture. In *ISSCC*, 2009.
[37] T. Moscibroda and O. Mutlu. Memory performance attacks: Denial of memory service in multi-core systems. In *USENIX SS*, 2007.
[38] S. P. Muralidhara et al. Reducing memory interference in multicore systems via application-aware memory channel partitioning. In *MICRO*, 2011.
[39] O. Mutlu et al. Runahead execution: An alternative to very large instruction windows for out-of-order processors. In *HPCA*, 2003.
[40] O. Mutlu and T. Moscibroda. Stall-time fair memory access scheduling for chip multiprocessors. In *MICRO*, 2007.
[41] O. Mutlu and T. Moscibroda. Parallelism-aware batch scheduling: Enhancing both performance and fairness of shared DRAM systems. In *ISCA*, 2008.
[42] NEC. Virtual Channel SDRAM uPD4565421, 1999.
[43] K. J. Nesbit et al. Fair queuing memory systems. In *MICRO*, 2006.
[44] J.-h. Oh. Semiconductor memory having a bank with sub-banks. U.S. patent number 7782703, 2010.
[45] M. K. Qureshi et al. A case for MLP-aware cache replacement. In *ISCA*, 2006.
[46] Rambus. DRAM Power Model, 2010.
[47] S. Rixner et al. Memory access scheduling. In *ISCA*, 2000.
[48] P. Rosenfeld et al. DRAMSim2: A cycle accurate memory system simulator. *IEEE CAL*, Jan. 2011.
[49] R. H. Sartore et al. Enhanced DRAM with embedded registers. U.S. patent number 5887272, 1999.
[50] Y. Sato et al. Fast Cycle RAM (FCRAM); a 20-ns random row access, pipe-lined operating DRAM. In *Symposium on VLSI Circuits*, 1998.
[51] B. Sinharoy et al. IBM POWER7 multicore server processor. *IBM Journal Res. Dev.*, May. 2011.
[52] B. J. Smith. A pipelined shared resource MIMD computer. In *ICPP*, 1978.
[53] A. Snavely and D. M. Tullsen. Symbiotic jobscheduling for a simultaneous multithreaded processor. In *ASPLOS*, 2000.
[54] STREAM Benchmark. `http://www.streambench.org/`.
[55] J. Stuecheli et al. The virtual write queue: Coordinating DRAM and last-level cache policies. In *ISCA*, 2010.
[56] K. Sudan et al. Micro-pages: Increasing DRAM efficiency with locality-aware data placement. In *ASPLOS*, 2010.
[57] Sun Microsystems. OpenSPARC T1 microarch. specification, 2006.
[58] J. E. Thornton. Parallel operation in the control data 6600. In *AFIPS*, 1965.
[59] S. Thoziyoor et al. A comprehensive memory modeling tool and its application to the design and analysis of future memory hierarchies. In *ISCA*, 2008.
[60] R. M. Tomasulo. An efficient algorithm for exploiting multiple arithmetic units. *IBM Journal Res. Dev.*, Jan. 1967.
[61] TPC. `http://www.tpc.org/`.
[62] A. N. Udipi et al. Rethinking DRAM design and organization for energy-constrained multi-cores. In *ISCA*, 2010.
[63] T. Vogelsang. Understanding the energy consumption of dynamic random access memories. In *MICRO*, 2010.
[64] F. Ware and C. Hampel. Improving power and data efficiency with threaded memory modules. In *ICCD*, 2006.
[65] W. A. Wong and J.-L. Baer. DRAM caching. *CSE-97-03-04, UW*, 1997.
[66] T. Yamauchi et al. The hierarchical multi-bank DRAM: A high-performance architecture for memory integrated with processors. In *Advanced Research in VLSI*, 1997.
[67] G. L. Yuan et al. Complexity effective memory access scheduling for many-core accelerator architectures. In *MICRO*, 2009.
[68] Z. Zhang et al. A permutation-based page interleaving scheme to reduce row-buffer conflicts and exploit data locality. In *MICRO*, 2000.
[69] Z. Zhang et al. Cached DRAM for ILP processor memory access latency reduction. *IEEE Micro*, Jul. 2001.
[70] H. Zheng et al. Mini-rank: Adaptive DRAM architecture for improving memory power efficiency. In *MICRO*, 2008.
[71] W. K. Zuravleff and T. Robinson. Controller for a synchronous DRAM that maximizes throughput by allowing memory requests and commands to be issued out of order. U.S. patent number 5630096, 1997.