2018 IEEE International Symposium on High Performance Computer Architecture

# ERUCA: Efficient DRAM Resource Utilization and Resource Conflict Avoidance for Memory System Parallelism

Sangkug Lym† Heonjae Ha‡ Yongkee Kwon† Chun-kai Chang† Jungrae Kim§ Mattan Erez†

†The University of Texas at Austin ‡Stanford University §Microsoft

†{*sklym, yongkee.kwon, chunkai, mattan.erez*}*@utexas.edu* ‡*hunjaeha@stanford.edu* §*jungrkim@microsoft.com*

***Abstract*—Memory system performance is measured by access latency and bandwidth, and DRAM access parallelism critically impacts for both. To improve DRAM parallelism, previous research focused on increasing the number of effective banks by sub-dividing one physical bank. We find that without avoiding conflicts on the shared resources among (sub)banks, the benefits are limited. We propose mechanisms for efficient DRAM resource utilization and resource-conflict avoidance (ERUCA). ERUCA reduces conflicts on shared (sub)bank resources utilizing row address locality between sub-banks and improving the DRAM chip-level data bus. Area overhead for ERUCA is kept near zero with a unique implementation that exploits under-utilized resources available in commercial DRAM chips. Overall ERUCA provides 15% speedup while incurring <0.3% DRAM die area overhead.**

***Keywords- Memory System; Memory System Performance; DRAM Access parallelism;***

## I. Introduction

Main memory remains an important bottleneck for many applications. Both the latency of memory access and overall memory bandwidth are of concern and both vary depending on application's memory access pattern and available parallel resources in the memory system. Memory system parallelism improves latency by overlapping memory accesses and mitigates bank conflicts as more parallel resources (banks) can better utilize page locality. A cost-effective way for increasing parallelism is to grow the number of banks in each DRAM chip, as has been the trend with recent DRAM generations (Tab. I). However, additional memory banks require extra peripheral logic [1], which leads to substantial chip area overheads. Researchers have therefore proposed to improve effective bank parallelism with low cost using various sub-banking schemes [2], [3], [4], [5]. However, as sub-banks share resources within one physical bank, conflicts on such shared resources hamper performance. Conflict rate depends on the memory access pattern, increasing system sensitivity and decreasing performance robustness.

Continued DRAM channel clock frequency scaling places another constraint on DRAM parallelism because of mechanisms needed to keep channel utilization high. Historically, DRAM channel clock frequency has doubled with every new DRAM generation (Tab. I). At the same time, DRAM core frequency remains roughly fixed [6]. Two techniques are used to bridge the gap between channel and DRAM core frequencies. The first is to coarsen the granularity of access such that a wide internal fetch (prefetch) is used and the data is then rapidly delivered over the channel (long burst length). Increasing the burst length (prefetch depth) is costly because it requires additional internal bandwidth out of the DRAM arrays and increases access granularity. The second technique to bridge frequency disparity relies on *bank grouping*, as in DDR4, GDDR5, and HBM [7], [8], [9], [10]. With bank grouping, the global chip-wide buses internal to a DRAM chip are partitioned such that accesses to different bank groups are handled in parallel; each access requires higher latency than the time to traverse the DRAM channel and internal bus conflicts are avoided by partitioning. Although bank grouping helps alleviate internal data bus conflicts, consecutive DRAM accesses to the same bank group still suffer from intra-group bus contention, increasing the performance impact of access patterns. *In summary, increasing the number of banks, while important, is not sufficient to fully utilize memory system parallelism.*

| | DDR | DDR2 | DDR3 | DDR4 |
|---|---|---|---|---|
| Bank count | 4 | 4∼8 | 8 | 16 |
| Channel clock(MHz) | 133∼200 | 266∼400 | 533∼800 | 1066∼1600 |
| DRAM core clock(MHz) | 133∼200 | 133∼200 | 133∼200 | 133∼200 |
| Internal prefetch | 2n | 4n | 8n | 8n |

Tab. I: Specifications of DRAM generations

In this paper, we introduce ERUCA, an efficient sub-banking and frequency-scalable DRAM architecture and memory controller. The main goal of ERUCA is to improve effective parallelism by avoiding or minimizing inter-(sub)bank resource conflicts with very low area overhead mechanisms. One of our important observations is that conflicts on the shared resources among sub-banks can be effectively avoided when memory access address locality is taken into account. While physical address hashing schemes already consider channel and bank conflicts, and row locality, no prior work discusses the interactions between sub-banks, and other fine-grained DRAM parallelism mechanisms, and address locality. We develop two techniques that take this locality into account, reduce inter-sub-bank resource conflicts, and improve performance and efficiency. *Row address permutation* (RAP) reduces conflicts on the shared row address latches between sub-banks by breaking locality on row addresses within a bank without any changes in DRAM chip. While reducing conflicts is

2378-203X/18/$31.00 ©2018 IEEE
DOI 10.1109/HPCA.2018.00063

desirable, the design of ERUCA exploits internal DRAM architecture to improve performance when multiple sub-banks require physically-close row indices. *Effective word-line range* (EWLR) enables sub-banks that activate nearby rows to share expensive word-select signals with a small number of additional row address latches. EWLR is also very area-efficient ($< 0.1\%$ DRAM die area).

To bridge the frequency gap between DRAM core and I/O channel, we propose the *dual data bus* (DDB). DDB consists of a pair of buses and small switches, and removes or mitigates the conflicts on chip-level global data buses to double internal bandwidth. We show how to do this without requiring sophisticated and non-scalable circuits like prior work proposed [6], and utilize already-existing routing resources when possible. As the performance penalty from global data bus conflicts increases with the growing frequency gap, and our evaluation demonstrates that the benefits from DDB increase as DRAM frequency grows.

Even when performance benefits are demonstrated, adding new features to commercial DRAM designs is difficult. To make our proposals more practical, we introduce a unique implementation mechanism to realize ERUCA that exploits modern DRAM architectures and requires near-zero area overhead. This implementation is based on the observation that current server and desktop DRAM chips utilize a *Combo DRAM* architecture where a DRAM chip can be configured post manufacturing to use either a x4 or a x8 memory interface. Combo DRAM is adopted by all DRAM manufacturers [11], [12], [13] because it offers major cost benefits, reducing design, verification, tooling, and demand-fluctuation costs. The structure of a Combo DRAM chip leaves some internal resources unused, and others not concurrently, when configured as a x4 device. We propose to use these resources to realize all the proposed schemes of ERUCA, improving performance without increasing cost (just $0.05\%$ and $0.06\%$ for EWLR and DDB respectively).

When EWLR and RAP are used together, performance with ERUCA improves by $13\%$ compared to a DDR4 baseline. DDB further improves the performance by $2\%$, and this additional performance gain scales to $5\%$ when DRAM I/O frequency is increased. By relying on the already existing structures in Combo DRAM, implementing all schemes in ERUCA requires less than $0.3\%$ additional DRAM chip area—five times less overhead than the lowest-cost prior work that discusses sub-banking [14], [4]. Implementing ERUCA for x4 Combo DRAM chips is important because such chips are preferred by servers for larger memory capacity and stronger error correction [15], [16], and more than half of all CPU DRAM sales are expected to be for servers [17]. Our design optimizes the x4 DRAM configuration and adds only negligible overhead when used as a x8 DRAM configuration and carries huge benefits. We also generalize ERUCA to non-Combo DRAM designs and describe and evaluate such implementations. We also note that our paper describes and evaluates the tradeoffs associated with sub-banks to much greater detail than prior work, which only hinted at similar possible optimizations [2], [3].

To summarize our contributions in ERUCA:

- We observe and discuss hardware resource conflicts among sub-banks. We explain how conflicts on shared resources hamper sub-banking effectiveness and quantify the impact on performance and energy. To mitigate the interference and further improve the performance, we introduce the RAP and EWLR mechanisms.
- We propose DDB frequency-scalable chip-level data bus scheme. DDB enables gap-less memory access to the DRAM (sub)banks, and its performance benefit grows with increasing channel and DRAM core frequency mismatch.
- We implement the proposed DRAM parallelism mechanisms of ERUCA with very low area overhead, exploiting the under-utilized resources of Combo DRAM chips. Combo DRAM is the current de facto standard DRAM architecture and ERUCA improves its performance nearly for free. We also discuss how ERUCA can be implemented in other DRAM architectures.
- We quantitatively compare our proposed mechanisms to prior work in terms of both performance and hardware cost. Our proposed schemes (EWLR, RAP, and DDB) are partially orthogonal to prior schemes and we demonstrate significant performance synergy when used together.
- Because DRAM performance in general is very access-pattern dependent (and the performance of RAP and EWLR is particularly sensitive), we evaluate our techniques using physical addresses captured on a full system under a range of physical memory fragmentation scenarios.

## II. Background

We summarize DRAM design aspects necessary to understand ERUCA in this paper. We rely on a fairly-detailed understanding of DRAM and encourage the reader to review this background.

| Acronym | Description | Acronym | Description |
|---|---|---|---|
| CSL | chip select | LWL | local wordline |
| SBL | sub-bitline | LWL_DRV | local wordline driver |
| GBL | global bitline | LWL_SEL | local wordline select |
| SA | sense amplifier | MWL | main wordline |

Tab. II: Acronyms used in this paper.

### A. DRAM Organization and Operation

The memory sub-system is logically organized as a hierarchy of channels, ranks, and banks. A channel is the most independent resource in the memory system hierarchy in that each channel has a dedicated transaction queue, a command scheduler, and data and command buses. All these resources are shared by ranks within each channel. A rank consists of multiple DRAM chips where each chip contributes a fraction of the bus width; each chip is typically 4, 8, or 16-bits wide (referred to as x4, x8, and x16 devices, respectively). All DRAM chips in a rank operate in parallel and share

a single command. Within a rank (and thus within each chip), a number of banks share the I/O pins but otherwise operate independently. Each bank holds an active row that can be accessed by column read and write commands. There is thus a hierarchy of resources with significant parallelism overall. The memory controller is responsible for scheduling memory operations to take advantage of this parallelism while adhering to all resource constraints. How much parallelism can be exploited depends on the address mapping function and memory access pattern—if accesses within a short time window target a small subset of the parallel resources, performance is limited.

Each DRAM bank consists of memory sub-arrays, row and column address decoders, write drivers, and an IO sense amplifier (IOSA) array (Fig. 1). The row address decoder selects the target row in one sub-array, the column address decoder chooses which bits to access within a row in that sub-array, and the IOSA and write drivers assist with performing read and write operations from the bank edge to the sub-arrays, respectively.

A sub-array is the finest-grained unit of control within a bank. It performs wordline activation which caches all data from the selected wordline to a sense amplifier (SA) array (also called a row buffer or page). To activate a wordline (row), the target local wordline is raised by a *local wordline driver* (LWL_DRV). The LWL_DRV is selected by a combination of the *main wordline* (MWL) and the *local wordline select* (LWL_SEL) signals ❶. Next, the data cells in the selected wordline are sensed to the SA array through local bit lines ❷. All wordlines in one sub-array share an SA array, and each SA array is shared by two vertically-adjacent sub-arrays. This sharing is done because an SA is wider than a column of cells and DRAM designers aggressively minimize cell pitch [18]. Therefore, one half of a row is fetched to the upper SA array while the other half is fetched to the lower SA array.

After activation is complete, the bank becomes ready for read and write operations to the active row. For a read

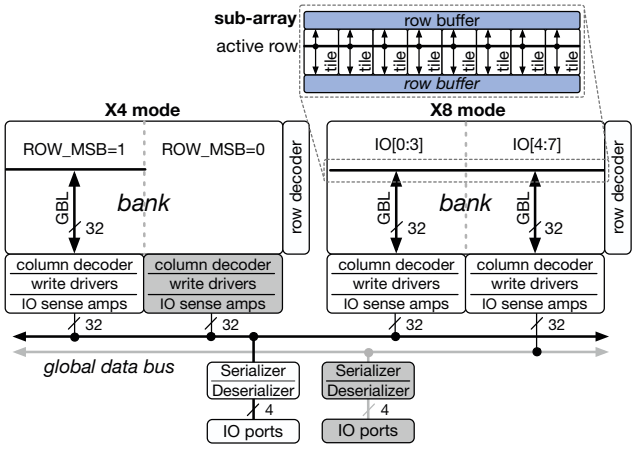
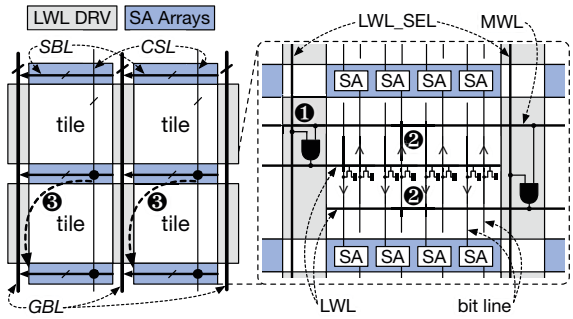
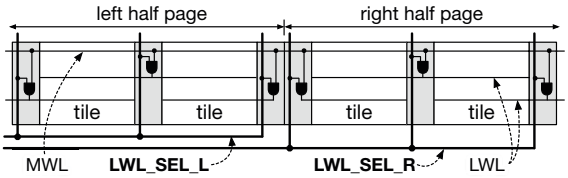



Fig. 1: DDR4 Combo DRAM bank organization: Grey blocks are underutilized or unused in x4 mode.




(a) DRAM sub-array composed of tiles, SA array, local wordline drivers, data paths, and control signals




(b) DDR4 half page structure: Two LWL_SEL signal organization and local wordline driver layout for half wordline control [19]

Fig. 2: DRAM sub-array structure.

operation, the target data point within the activated row in the SA array is selected by *column select* (CSL) signals and moved to *global bitlines* (GBLs) through an intermediate *sub-bitline* (SBL) level ❸. Because all selection wires and data path wires are compactly laid out between and over array tiles (2D arrays of cells that comprise sub-arrays), any additional wire inserted expands the DRAM die footprint and incurs significant overhead.

### B. DRAM I/O Configuration

DRAM chips with different I/O widths offer different tradeoffs. Narrower x4 DRAM allows larger memory capacity because more chips are used for a given rank width than when using x8 chips. In addition, stronger memory protection is possible with x4 chips [15], [16]. This combination of capacity and reliability makes x4 chips very common in server systems. However, x8 DRAM requires less dynamic and static power (and energy) and x8 DIMMs are cheaper because fewer chips are needed in each (non-ECC) rank. Thus, x8 DRAMs are often used in personal computing systems.

***Combo DRAM.*** All DRAM vendors design a single DRAM chip in a way to support multiple I/O width options [11], [12], [13]. The final chip width is programed post manufacturing, and the resources to support all chip configurations are always implemented within each chip. Fig. 1 shows how these resources are utilized in x4 and x8 Combo DRAM chips. When set to x8 mode, all chip-level bus interconnects are utilized, and one full bank is activated and accessed at any time. On the other hand, a chip in x4 mode uses only half of the available bus and only half a bank is activated and accessed at any time. Such under-utilized resources in x4 chips can be used for various purposes, including expanding

DRAM parallelism. One important benefit of this *half-page DRAM* design [7] is that the row buffer of each chip is half as big in x4 vs. x8 mode (Fig. 2b). This means that the rank-level DRAM page has the same width when implemented with either x4 or x8 DDR4 $ComboDRAM$ chips, making x4 DRAM more power and energy competitive.

## III. Related Work

### A. Sub-bank parallelism and restrictions

We briefly discuss previously proposed sub-bank parallelism schemes and compare their benefits and constraints to our proposed mechanisms. We can classify the sub-banking schemes into two groups: bitline- and wordline-direction sub-banking.

***Bitline direction sub-banking.*** Bitline direction sub-banking is proposed by Kim et al [2] and also discussed by Son et al. [3]. This scheme divides a physical bank into multiple horizontal sub-array groups and utilizes each sub-array group as a sub-bank for access interleaving and parallelism. By inserting a dedicated set of row address latches to each sub-bank, each sub-bank can hold an active wordline and we can interleave sub-banks to hide memory access latency.

Unfortunately, bitline direction sub-banking has restrictions which make it difficult to scale and manufacture. First, GBL signals are shared by all sub-banks and to avoid conflicts on a GBL, each sub-bank's SBL to GBL gating logic should be controlled independently. This independent control requires additional sub-bank selection signals, which increase DRAM chip size and access latency quite significantly [2]. The evaluation of area in previous work focuses more on the penalty from additional row address latches and SBL gating logic rather than these additional control wires. However, according to our analysis, the major area overhead of bitline-direction sub-banking comes from the additional wires for selecting address latches and gating logic. This is because these are bank-global signals with a wide wire pitch and their overhead accumulates as the number of sub-banks grows. Another constraint is that the sub-arrays at the boundary of every sub-bank require additional interleaving constraints because of the SA (sense-amp) arrays that are shared between tiles (Section II-A). To avoid this SA conflict, the memory controller should adopt a sub-array location-aware scheduling policy or extra SA arrays and dummy cell arrays should be placed at every sub-bank boundary. Neither of these approaches is discussed in prior work.

Finally, dividing a bank into many number bitline-direction sub-banks impacts DRAM manufacturability because of reduced DRAM row-repair flexibility. DRAM has spare wordlines placed across a bank, which are mapped to any faulty wordlines for increasing device manufacturing yield [20], [21]. With bitline direction sub-banking, repair mapping is constrained within each sub-bank range due to row address conflicts. This can potentially lower DRAM manufacturing yield, especially when DRAM cells have high defect probability [21], which is expected in future DRAM generations [22].

***Wordline-direction sub-banking.*** Wordline-direction sub-banking divides a wordline in a bank into multiple mini-wordline sections. Each mini-wordline is activated by a combination of MWL and one of multiple LWL_SEL signals. Such fine-granularity wordline activation was initially proposed to save activation power [23] and is also discussed as a sub-banking scheme [4], [3]. Ha et al [14] also proposed to utilize DDR4 half-page structure for power saving but they did not discuss about using their schemes for expanding memory parallelism. Like bitline direction sub-banking, wordline direction sub-banking requires extra row address latches and their control wires to maintain an active row in each vertical sub-bank.

Wordline direction sub-banking is also difficult to scale for three reasons. First, in each horizontal plane, sub-banks share a set of row address latches and sub-bank interleaving is restricted because of this resource conflict. When both sub-banks have memory accesses with high row address locality, such conflicts can significantly reduce effective bank parallelism. Second, in order to select the target mini-wordline, we need to increase the number of LWL_SEL wires proportionally to the number of sub-banks (Fig. 2b). This has significant impact on the very customized and constrained array design. Finally, the number of GBL wires should also grow with the number of sub-banks to transport data between sub-banks and write drivers and IOSA arrays. We can reduce GBL wire pitch or occupy additional area, but both options are critical to DRAM access latency and area and thus are difficult to implement. Splitting tiles to create just two sub-banks is free from this issue since both sub-banks share GBLs [4]. However, this mechanism still has to double SBL wires in order to transport twice the data from SA arrays to GBLs [14]. This doubling increases DRAM size by 1.4% [14], which is a fairly large increase for cost-sensitive DRAM design.

Specifically for GPU DRAM, Chatterjee et al [5] proposed *sub-channeling* which applies word-line direction sub-banking to HBM. Sub-channeling does not require extra GBL wires as it transfers the entire data via narrow per-mat channel in serialized fashion. This data serialization significantly increases memory transaction latency thus sub-channeling is only applicable to GPU system which is less sensitive to memory latency.

***Advantages of ERUCA.*** In ERUCA, we focus on how to avoid inter-sub-bank conflict on shared row address latches and global data path to maximize the effective memory parallelism efficiency. In terms of implementation, unlike prior work, we do not propose to split individual DRAM array tiles, but rather rely on the existing resources and half-page organization of DDR4 Combo DRAM so it is free from the 1.4% overhead to double SBL wires. ERUCA uses pseudo bitline direction sub-banking, and its purpose is to reduce row address latch resource conflicts between sub-banks thus it does not suffer from the additional area overhead from extra SA arrays. In summary, considering prior sub-banking mechanisms have scalability concerns

due to the area overhead, row-repair flexibility, and shared resource conflicts between sub-banks, in ERUCA, we choose to improve the efficiency of sub-banking parallelism with minimum cost.

### B. Data Bus Parallelism

For general purpose memory operation, data bus parallelism supported by our proposed scheme, DDB, is most beneficial when in-DRAM data transmission takes longer than external-channel communication burst. DDMA (Decoupled Direct Memory Access) also uses two switchable DRAM data buses, but the goal is to communicate with two different hosts [24]. DDMA has two sets of data ports, and each of the two data buses is connected to one set of ports. Therefore, the memory controller has to specify which data bus and I/O ports to use through a DRAM command. DDB is different from DDMA in its purpose, port organization, resources to build the dual bus structure, control mechanism and command scheduling policies.

There is other previous work utilizing the redundant data bus resources of Combo DRAM, but the usage is restricted to special purpose functions, such as constructing a separate data channel for communication within NVDIMM (Non-Volatile DIMM) [25] similar to the purpose of DDMA [26], [27]. DDB is different from the prior work in that we use the redundant data bus switchable with the regular bus to improve DRAM operation parallelism.

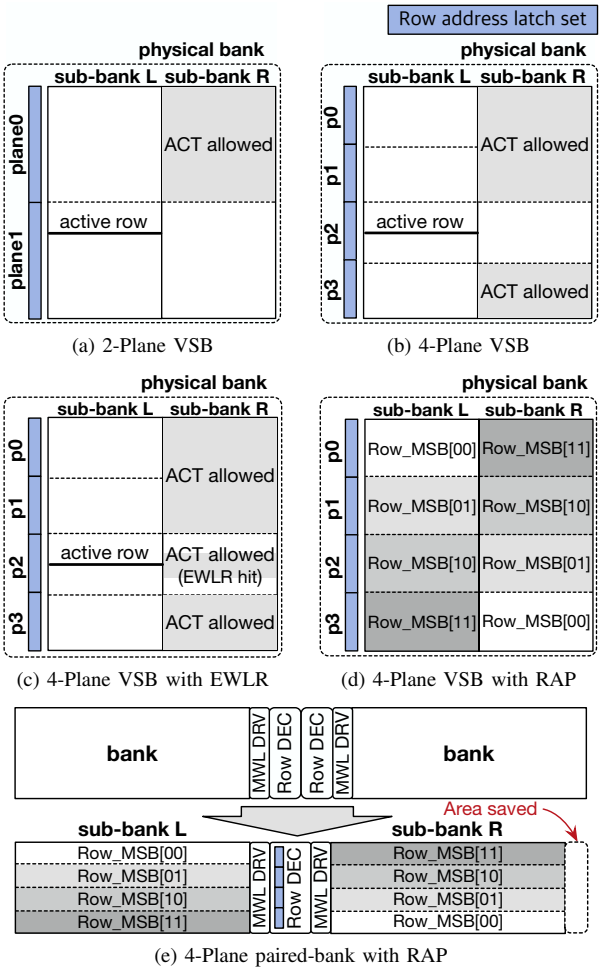
Previous work directly addresses the constraints of bank groups and proposes low-level, and costly, circuit techniques to allow for faster internal buses and avoid bank groups in the context of graphics memories [6].

## IV. Reducing Sub-bank Conflicts

One of the main goals of ERUCA is to maximize DRAM bank parallelism by reducing conflicts on inter-sub-bank shared resources. Before describing the conflict-reducing schemes, we first introduce our unique sub-banking implementation, which uses two mechanisms.

***Vertical sub-banks in Combo DRAM.*** The first mechanism, *vertical sub-banks* VSB, splits each bank into two sub-banks by exploiting already existing hardware resources. In current x4 DDR4 Combo DRAM chips, only one half of each bank is activated and accessed at any given time, as this enables a single design that maintains the same rank-level DRAM page size. VSB uses each half bank as an independent sub-bank by adding just extra row address latches that hold an active wordline address in each sub-bank. Fig. 3a shows this sub-bank structure, which has two row address latches positioned at plane0 and plane1, respectively. We define a *plane* as the portion of a bank that shares a row address latch. Each sub-bank is selected through the existing LWL_SEL_L or LWL_SEL_R signals, such that no additional wires are necessary.

One of the benefits of VSB over prior work [3], [4] is its dedicated column logic and data path, which it inherits from the Combo DRAM design. Because each sub-bank has dedicated GBLs, write drivers, and an IOSA array,




(a) 2-Plane VSB (b) 4-Plane VSB

(c) 4-Plane VSB with EWLR (d) 4-Plane VSB with RAP

(e) 4-Plane paired-bank with RAP

Fig. 3: Inter-sub-bank activation flexibility of different sub-banking schemes in ERUCA.

which are needed when operating in the x8 mode, we can overlap accesses to both sub-banks. This reduces sub-bank interleaving latency when prefetch width is insufficient to fully bridge the clock frequency gap between DRAM core and external channel, as with DDR4. Such aspects are not discussed in prior work [4], [3] and are unique to our design.

***VSB without Combo DRAM.*** Since our first sub-banking scheme is restricted to x4 Combo DRAM, we also propose a *paired-bank* structure for non-Combo DRAM (Fig. 3e). A *paired-bank* constructs a single bank from two adjacent banks by removing one of the row address decoders, retaining all main wordline drivers, and adding row address latches. The goal of VSB with paired-banks is to reduce DRAM area without losing performance, rather than improving performance. The row address decoders require significant area especially due to global address wires while the address latches of VSB are very small. In other words, the paired-bank design is a cheaper alternative to current DRAM that matches current performance by using the ERUCA techniques. Unless we note explicitly, discussions

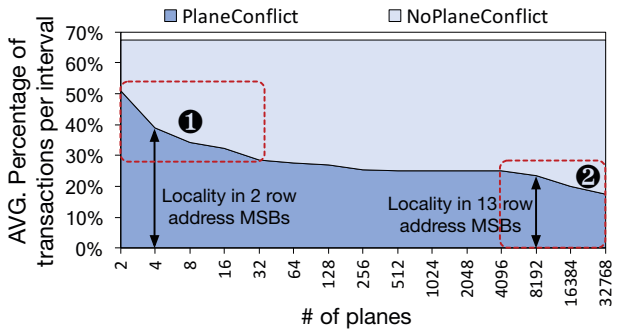



Fig. 4: Percentage of transactions in each timing interval which have PlaneConfclit or NoPlaneConflict

and evaluation below is for VSB with Combo DRAM.

***Plane conflicts.*** The performance of the two-plane VSB Fig. 3a is constrained by severe conflicts on the per-plane row address latches and MWLs which are shared between sub-banks—*plane conflicts*. When the left sub-bank has an active row in plane1 and the right sub-bank is idle, the right sub-bank can only be activated within plane0. If the right sub-bank attempts to activate plane1, the left sub-bank should first be precharged, causing inter-sub-bank row buffer thrashing which does not exist for full banks.

***Multiple planes.*** To alleviate plane conflicts, we can increase the number of planes which expands the available activation range. Fig. 3b shows the VSB structure with 4 planes. Since the row address latches of plane3 are available, plane conflicts in this address range are removed. Also, because SA arrays are not shared between sub-banks, adding more planes does not require additional placement of dummy SA array and cell array at the border of planes as needed by prior work [2], [3]. However, the benefit of additional planes diminishes as the number of planes increases. Also, like in prior work, each additional plane requires a set of row address latches and latch control wires and dividing a bank into too many planes impacts the faulty wordline repair flexibility (as discussed in III-A) and increases area overhead. Considering the benefits and the scaling problems of the many-plane VSB structure, plane count should remain low.

***Inter-sub-bank memory address locality.*** A large number of planes is only necessary if multiple accesses within a short time window have plane conflicts. The frequency of conflicts and their impact therefore depends on the memory access pattern and the function that maps physical addresses to DRAM locations. Specifically, conflicts arise when accesses within a time window all target the set of row addresses that fall in different sub-banks but within the same plane. Current DRAM address mapping schemes do not focus much on row addresses beyond utilizing row locality by deciding on column addresses within a row; the focus is on spreading accesses across channels, ranks, and banks to minimize bank conflicts and maximize overall throughput. Because of access locality in physical addresses, it is important to map those bits that change frequently to select those resources that must be parallelized. Thus, current DRAM mapping functions use address hashing schemes allocating physical memory address LSBs to parallel resources (i.e., channels, ranks and banks) to maximize parallelism and to DRAM columns to utilize row buffer locality. The address MSBs, which change less frequently, are usually allocated to DRAM rows [28], [29], [30]. Since plane ID is determined by DRAM row address in each sub-bank, row address locality between sub-banks affects plane conflicts; address locality refers to the similarity of row address between accesses within a short period of time.

We estimate the DRAM row address locality between sub-banks using memory access traces (with times and accurate physical addresses) extracted from a subset of SPEC2006 benchmarks (*mcf, lbm, gemsFDTD, omnetpp*) [31]. For each memory access, we evaluate whether it overlaps with another access to the same bank within a short time window. If it does overlap, we determine whether it causes a plane conflict or can utilize another sub-bank without a plane conflict. We use tRC (the minimum row activation time) as the time window for this experiment; accesses that are further apart will not be delayed because of the plane conflict.

Fig. 4 shows the fraction of transactions with or without plane conflicts in each time interval, averaged across all benchmarks. We ignore transactions that do not overlap with other transactions in the same bank. On average, 67% of the transactions per interval overlap with another transaction to the same bank and may exhibit a plane conflict. With 2 planes, 51% of transactions have plane conflicts, meaning they have the same plane ID but target a different sub-bank. As the number of planes increases, transactions with plane conflicts diminish. However, even with enough planes such that each plane is just a single row, 17% of transactions still exhibit plane conflicts. This shows that memory transactions to a sub-bank pair have high locality in row address bits; most locality, and hence conflicts, are mainly found in two address regions, regions ❶ and ❷ in Fig. 4.

In region ❶, only a few high-order row address bits have locality. This is mainly caused by the operating system (OS) page allocation policy. Modern OSs aggressively apply transparent huge page allocation and promotion [32] to avoid frequent TLB misses [33], [34]. When huge pages are used, there are 21 offset bits in an address and bits beyond the offset change infrequently. On the other hand, in region ❷, a wide range of row address bits have locality, as many applications exhibit significant spatial locality. We observed the similar address locality trend in other SPEC2006 application mixes.

Increasing the plane count can remove plane conflicts in region ❶ but the conflicts in region ❷ will remain. In addition, supporting a large number of planes requires significant additional hardware for latches and control signals. Therefore, we propose two mechanisms to mitigate plane conflicts effectively and with low cost by manipulating and exploiting address locality.

***EWLR (effective wordline range).*** The idea behind the

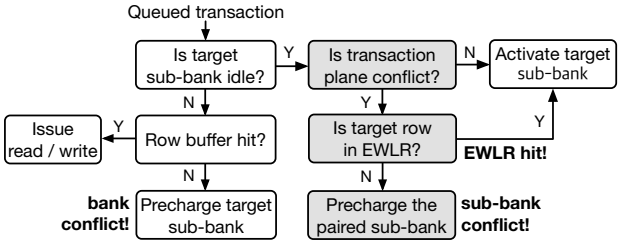



Fig. 5: Operation flow of ERUCA with EWLR

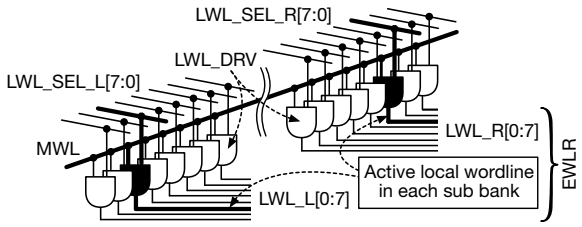
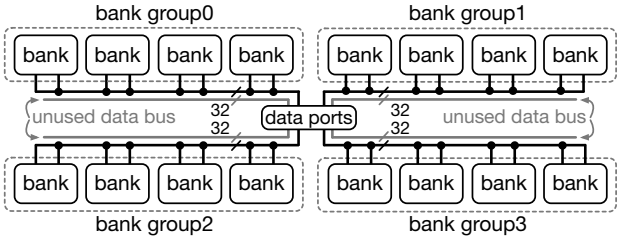
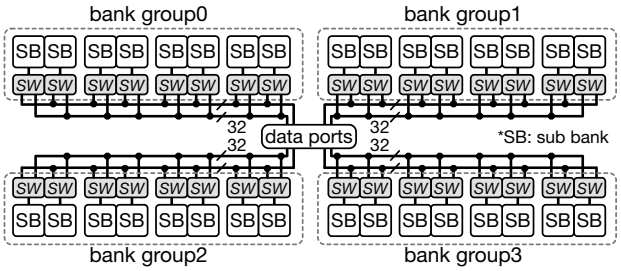



Fig. 6: Independent wordline activation in each sub-bank within one physical bank at EWLR hit.




(a) Bank group: Each bank group has a single dedicated bus




(b) DDB: Each bank group has dual data buses, and all banks are connected to DDB via cross-bar switches

Fig. 7: Global data bus organization comparison.

*effective wordline range* (EWLR) mechanism is that we can remove conflicts in region ❷, if each sub-bank can hold different active wordlines within a narrow row address range, even if there is a plane conflict. To do so, EWLR doubles a small number of row address latch bits—only enough latches for the LWL_SEL signals, which hold low address bits. The MWL address latches, which hold the high-order bits, must still be shared between all active rows in a plane. Thus, each sub-bank can maintain a somewhat different row address within the same plane, as long as all addresses differ only within values in the LWL_SEL latches (Fig. 6).

In other words, if an incoming row activation command matches the MWL address of an already active row in a different sub-bank—an *EWLR hit*—the sub-bank does not cause a plane conflict and can directly activate its target row without precharging other sub-banks (Fig. 3c). In fact, an EWLR hit not only eliminates a plane conflict but also saves activation power because it does not consume power to drive the already selected MWL signal. Since a charge pump is needed for MWL activation, an EWLR hit saves 18% of Vpp power during wordline activation (based on the power breakdown of the Rambus power model for a 55nm 2Gb DDR3 device [35]). Fig. 5 illustrates the operation flow of ERUCA with EWLR with the shaded stages only needed for handling bank conflicts.

To utilize EWLR, the physical DRAM address bits used for MWL offset need to be exposed to the memory controller. Also, two sets of row sub-address latches are inserted to maintain both LWL_SEL_L and LWL_SEL_R addresses independently. Given the observed high row address locality among transactions, EWLR effectively removes plane conflicts (see also Section VIII).

***RAP (row address permutation).*** The second ERUCA mechanism, *row address permutation* (RAP), also exploits row address locality. However, unlike EWLR which enables greater activation flexibility across sub-banks within the same plane, RAP aims to avoid plane address matches. To do so, RAP applies different plane ID mapping policies in each sub-bank (Fig. 3d). For example, consider the case of two sub-banks. RAP maps the high-order row address bits of the left sub-bank in descending order and those of the right sub-bank in the inverse order (by inverting their bit values). As a result, the memory transactions to each sub-bank are more likely to access different planes, reducing plane conflicts in region ❶ and ❷. RAP does not require any physical modification to DRAM and is very practical. Also, like EWLR, the DRAM physical address bits used for plane IDs need to be exposed to the memory controller.

## V. DDB: Dual Data Bus

To resolve the constraints on data bus parallelism caused by the increasing gap between in-DRAM and external channel frequencies, we propose the *dual data bus* (DDB). DDB doubles available bandwidth on the chip-global interconnect using a dual-bus structure with a set of small switches within the bus. DDB has very low area overhead because it reuses existing interconnect that is underutilized in the baseline Combo DRAM design. We also describe how DDB can be implemented in other DRAM architectures.

We develop DDB in the context of DDR4 DRAM, which exhibits data bus conflicts because the time required for transferring the data of a column command within the DRAM device exceeds the time required to traverse the DRAM channel. A Combo DRAM chip in x4 mode uses half of the available data path in the chip at any given time while the other half is left unused (Fig. 7a). We enable the use of both data buses by introducing very small cross-bar switches and interleave in-DRAM data transmissions within each bank group. DDB contributes to memory system

performance when a few bank groups are hot, if DRAM address hashing fails to interleave bank groups well.

Fig. 7b shows the layout of DDB with VSB. DRAM itself internally selects which bus half to use for any transfer using very simple logic to control the DDB switches. DDB supports back-to-back interleaved accesses for read and write commands both across and within bank groups as long as two conditions are met. First, the two consecutive accesses must not conflict on GBLs. GBLs are shared by a subset of tiles in each bank (Fig. 2a) and are occupied for one DRAM core clock per access. Second, one of the dual buses within the same bank group is available. A bus is occupied for one DRAM core clock while commands are synchronized with the bus clock. Current DRAMs operate with a core frequency of 200MHz while the bus frequency can be over 10 times higher [8].

It is worth noting that DDB is synergistic with VSB. VSB increases the number of effective banks per bank group and thus raises the pressure on the data bus. DDB mitigates the pressure by parallelizing bus utilization and we observe that VSB performs better with DDB (Section VIII).

***Controller changes.*** To prevent conflicts within DDB with growing channel frequency, we must prevent commands from violating the two conditions stated above. The first condition is already disallowed in existing designs because regardless of the core-to-bus frequency ratio, GBLs are shared within a bank (tCCD_L timing parameter). We handle the second condition by introducing a new timing parameter, the *two-command window* (tTCW), which defines the timing window in which only two column access commands are allowed. This is similar in concept to the *four activation window* (tFAW) timing parameter, but is motivated by bus conflicts rather than power constraints.

***Application to other DRAM types.*** Although we mainly develop DDB using the underutilized Combo DRAM interconnect as a cost efficient implementation mechanism, DDB is also applicable to other DRAM types. For none-Combo DRAM, the DDB switches can be extended to connect the buses of vertically-adjacent bank groups to mitigate bank-group timing constraints. Such DDB switches should connect the buses of bank groups 0 and 2 in Fig. 7a, for example. This can be applied to GDDR5 and HBM devices that also rely on bank groups for their high-frequency channel. For example, we conducted preliminary experiments with such a GDDR5 with a simulated GPGPU [36] and observed $10\%$ speedup on memory-intensive applications from the Rodinia benchmark suite [37]. We leave the evaluation of other applications of our ideas to future work.

## VI. Implementation Details

### A. Sub-bank Implementation

***Row address latches.*** Our sub-banking schemes (VSB and paired-bank) with $n$ planes require $n$ sets of row address latches to support independent wordline activation in each sub-bank (Fig. 8). For row address latch selection, each bank requires $n$ plane ID bits. We place the plane ID address

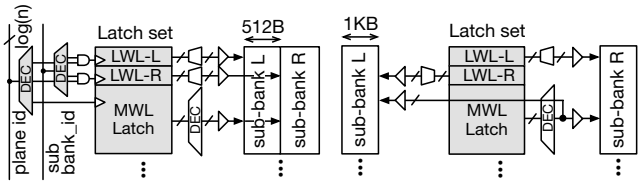



Fig. 8: Row address latches and selection signals in sub-banking schemes (latch selection omitted in paired-bank)

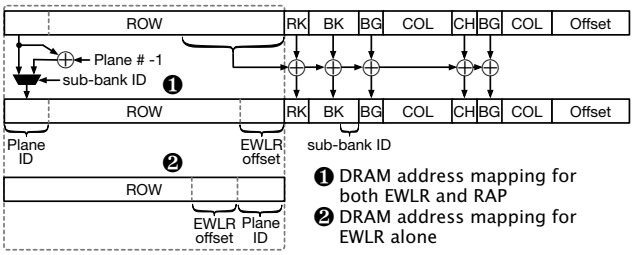


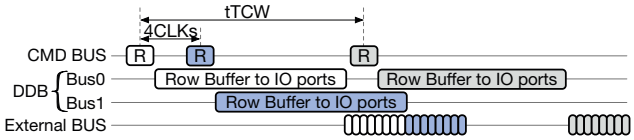

Fig. 9: DRAM address mappings for sub-banking with EWLR and RAP.

decoder near the address latches, thus reducing the number of global wires for plane ID to log(n). The same sub-bank selection mechanism is used for paired-bank, except that sub-banks are located at each side of each row decoder. For EWLR, additional row address latches are inserted to hold LWL_SEL_L and LWL_SEL_R address bits independently for each sub-bank. Also, a LWL_SEL latch selection wire along with a single bit decoder is added to choose the target LWL_SEL address latches.

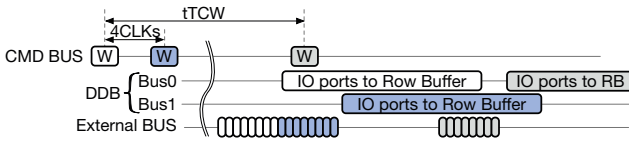
***Partial precharge.*** When both sub-banks hold an active row within the same EWLR, precharging one of the sub-banks should be performed without deactivating the shared MWL. We propose a partial precharge command which precharges only the logic blocks and data paths of one sub-bank. For this, the memory controller checks the EWLR hit status of both sub-banks. The partial precharge uses one of the unused address bits of the regular precharge command (precharge does not specify a row address) and DRAM simply regulates MWL driver inputs by this command without additional hardware.

***DRAM address mapping.*** Both EWLR and RAP require mindful selection of DRAM address hashing to maximize efficiency. When only EWLR is used, we use the row address LSBs for plane ID to give each sub-bank a higher chance to access different planes; the low-order row address bits change more frequently than the high-order bits (Fig. 9 ❷). The next 3 row address bits are allocated for the EWLR offset. Thus, because the high-order change infrequently, this mapping maximizes the likelihood that if a plane conflict does occur, it can be satisfied with an EWLR hit.

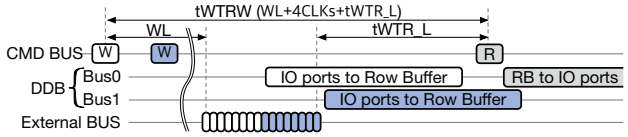
If RAP is used alone, we invert the row address MSBs with a bit-wise XOR and use the sub-bank ID to select between the original bits and the inverted bits for the final




(a) tTCW on consecutive read commands




(b) tTCW on consecutive write commands




(c) tTWTRW

Fig. 10: Command window for DDB to avoid global data bus conflict at fast bus clock frequency

plane ID. Address inversion proceeds in parallel with other address bit permutation and takes just two more gate delays for multiplexing by sub-bank ID.

Fig. 9 ❶ shows the DRAM address when EWLR and RAP are combined. We use the row address MSBs for plane ID and move the EWLR offset next to the plane ID. This decision is based on our observation that EWLR ID randomization using low-order row address bits is more effective because row address MSBs are frequently changed by RAP.

### B. DDB Implementation

***Cross-bar switch organization and control.*** To implement the set of switchable dual buses using the redundant data bus, we construct cross-bar switches between the two data buses and all banks in each bank group. We implement the switches using pass-transistors to minimize area overhead. The switch controller in each sub-bank generates the control signals using bank ID, sub-bank ID, and bus ID as inputs. Because bank and sub-bank IDs are already available in each bank, we only need to add two bus ID signals across the entire chip for both upper and lower bank groups. This is a small addition in the context of the already wide chip-global buses. At the other end of the DDB, we also have to modify the MUX and DEMUX logic to select the correct data from one of dual buses to transfer to the memory channel. Taking the existing MUX and DEMUX logic for bank groups as a base [7], we add just a single stage of a 32b 2-input MUX and DEMUX logic. This additional stage adds just two gate delays.

***Two-command window.*** The two-command window prevents more than two column accesses within a single DRAM core clock cycle time (5ns) in order to avoid a conflict on the DDB. The two-command window constraints consist of two timing parameters, tTCW (two column window) and

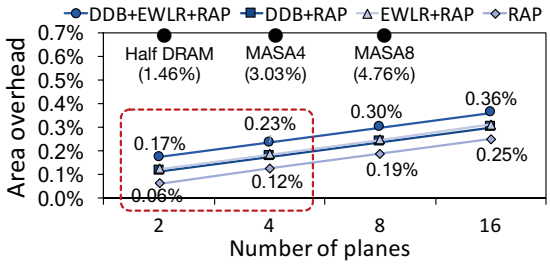



Fig. 11: DRAM area overhead comparison.

tTWTRW (two write to read window). The first parameter, tTCW, applies to consecutive read or write commands and blocks the third command until the window expires. Because the time that the data occupies the global data bus from the point of command issue is different for reads and writes, the memory controller has to keep track of two separate tTCW constraints for read and write commands, as depicted in Fig. 10a and Fig. 10b, respectively. Secondly, tTWTRW regulates the read command timing after two successive write commands; this is analogous to write-read turnaround. The value of tTWTRW is derived from the minimum timing margin needed between the first of the two write command and the read command to avoid conflicts on the DDB (Fig. 10c). Both tTCW and tTWTRW are managed independently for each bank group since each bank group possesses its own DDB. Again, two-command window constraints are applied only when the DRAM core clock cycle time is longer than twice the data burst time on the external DRAM channel.

### C. Overhead Estimation

We use 8Gb x4 DDR4 in 32nm technology as a baseline and estimate its size as $120.992mm^2$ ($8.98mm \times 13.47$mm) using CACTI-3DD [38]. To estimate the area overhead of our proposed schemes, we synthesize the additional logic blocks using the Synopsys SAED 32nm technology library. The latch selection wire pitch is conservatively estimated as $1\mu$m, which is three times the CSL wire pitch; each CSL wire has the width of 4 bitlines in 32nm DRAM technology [38].

VSB with 2 planes requires 40-bit latches in each latch set with $3:8$ row address pre-decoding. With EWLR, the number of bit latches per latch set increases to 48 due to the doubled LWL_SEL bits. As the number of planes increases, so does the number of latch sets, but the latch count in each latch set reduces slightly. Based on synthesis results, each 40b and 48b latch set occupies only $203\mu m^2$ and $244\mu m^2$, respectively. The overhead from the latch selection signals running across the row decoder grows as the number of planes increases, because the signals expend the width of the DRAM die. The latch selection signals are placed in the bitline direction. Therefore, we need to multiply the number of latch selection wires by the number of row decoders in the chip in the wordline direction. There are 8 such decoders in DDR4 [7]. Hence, every doubling of the number of planes expands the width of the DRAM die by $8\mu m$ ($1\mu m \times 8$).

EWLR adds another $2\mu m$ for the left and right sub-bank selection signals.

DDB requires $64$ switches and some switch control logic per sub-bank. Each switch is implemented as a CMOS pass-transistor pair. Based on our synthesis results, the switches and their control logic take $191\mu m^2$ per sub-bank. In addition, the 32b 2-input MUX and DEMUX logic blocks account for $674\mu m^2$. The four bus selection wires running across the DRAM die expand the chip height by just $4\mu m$.

Fig. 11 shows the DRAM area overhead with different combinations of ERUCA mechanisms. The key components of our proposed mechanisms are constructed using existing resources in Combo DRAM and the area overhead is thus very small. DDB incurs $0.05\%$ area overhead, $85\%$ of which is from the bus selection wires. ERUCA with 2-plane VSB with RAP alone increases DRAM die area by $0.06\%$ and EWLR further increases area by $0.06\%$. As the number of planes increases, the additional address latch selection wires expand DRAM size linearly. Considering we need to restrict the plane count due to lowering row repair flexibility, up to 4 planes the area overhead is less than $0.3\%$.

Paired-bank based ERUCA reduces DRAM area by $1.1\%$ even with all schemes added (EWLR, RAP, and DDB). This is because we can remove half of the row address decoders in the chip. Because we still have to maintain two MWL driver sets and row repair fuses, we assume $25\%$ row decoder width reduction.

Comparing the overhead of VSB based ERUCA to Half DRAM [4], [14], our approach incurs five times lower hardware overhead; Half DRAM is one of the cheaper options described in prior work and which also has two wordline direction sub-banks. MASA (the highest performing SALP scheme) has significant area overhead of 3.03% and 4.76% with 4 and 8 sub-array groups respectively due to the additional SA arrays and dummy mats between sub-array groups. The small area overhead of ERUCA makes its implementation more practical.

## VII. Evaluation Methodology

We use *USIMM* [39] as our baseline memory simulator and add support for ERUCA. To model performance, we use *Sniper* [40] with the processor configuration described in Tab. III. Tab. III also describes the DDR4-based memory system and timing parameters. We detail those parameters that govern the operation of an idealized DDR4 without bank groups, standard DDR4, and the use of DDB. Also, we use a single tRRD of 4 clocks assuming tFAW constrains the power consumption. The rest of DRAM timing parameters follow the DDR4 standard [8].

We evaluated our proposed schemes on the high and medium memory intensity applications of the *SPEC2006* benchmark suite [31]. We also ran all experiments with low memory intensity benchmarks but omit their results because they are simply not impacted by the memory system performance. We form 4-program mixes from the chosen applications as shown in Tab. III). We simulate $200M$ representative memory-intensive CPU instructions from each mix.

| System configuration | |
|---|---|
| Processor | 4-core OoO x86, 4GHz, Fetch/Issue width (8), LSQ (32), ROB (192) |
| TLB | I-TLB:128, D-TLB:64, Associativity (4) |
| L1 | 32KB, Associativity (L1I: 4, L1D: 8), LRU |
| LLC | 1MB per core, Associativity (16), LRU |
| DRAM | DDR4, 1.33GHz (18-18-18), 2channels × 1rank, FR-FCFS, Adaptive-open page [42], Power (Micron DDR4 8Gb [13]), Intel Skylake address mapping [30], [28] (Fig. 9) |

| DRAM timing parameters | | | |
|---|---|---|---|
| Parameter (Value) | Ideal | Bank groups | DDB |
| tCCD_S (4CLKs) | diff banks | diff BGs | diff banks |
| tCCD_L (5ns) | same bank | same BG | same bank |
| tWTR_S (2.5ns) | diff banks | diff BGs | diff banks |
| tWTR_L (7.5ns) | same bank | same BG | same bank |
| tTCW (5ns) | - | - | same BG |
| tTWTRW (WL+4CLKs+tWTR) | - | - | same BG |

| | Benchmarks | MPKI |
|---|---|---|
| mix0 | mcf:lbm:omnetpp:gemsFDTD | H:H:H:H |
| mix1 | mcf:lbm:gemsFDTD:soplex | H:H:H:H |
| mix2 | lbm:omnetpp:gemsFDTD:soplex | H:H:H:H |
| mix3 | omnetpp:gemsFDTD:soplex:milc | H:H:H:M |
| mix4 | gemsFDTD:soplex:milc:bwaves | H:H:M:M |
| mix5 | soplex:milc:bwaves:leslie3d | H:M:M:M |
| mix6 | milc:bwaves:astar:leslie3d | M:M:M:M |
| mix7 | milc:bwaves:astar:cactusADM | M:M:M:M |
| mix8 | bwaves:leslie3d:astar:cactusADM | M:M:M:M |

Tab. III: Evaluation parameters.

All simulations are conducted while accurately capturing physical memory addresses allocated by Linux 3.19.0-32-generic. Physical addresses depend on the fragmentation level of physical memory. We therefore use a tool [34] to control the level of fragmentation. Fragmentation is quantified by the *free memory fragmentation index* (FMFI) [41]. We experiment with both $50\%$ and $10\%$ fragmentation levels.

## VIII. Evaluation Results

***General performance.*** Fig. 12 shows the weighted speedup [43] achieved by different ERUCA configurations over baseline DDR4 with 16 banks and 4 bank groups. We use 4 planes for this experiment and evaluate plane-count sensitivity later. We also compare with an idealized DDR4 that has 32 banks and enough buses to avoid bank grouping and to a configuration with 32 banks with bank groups and their associated timing constraints ($bg32$). Note that neither of these configurations is practical because doubling the number of full banks adds $11\%$ more die area according to the Rambus power model [35]. All configurations improve performance and their impact is greater when memory pressure is larger (the mixes are ordered left to right by decreasing memory access intensity). ERUCA based on 4-plane VSB without any proposed techniques to avoid plane conflicts improves the overall speedup by 10%.

DDB improves performance by an additional 2%. This is surprisingly low given that DDB works as an ideal data bus at the 1333MHz DRAM channel frequency. We find that

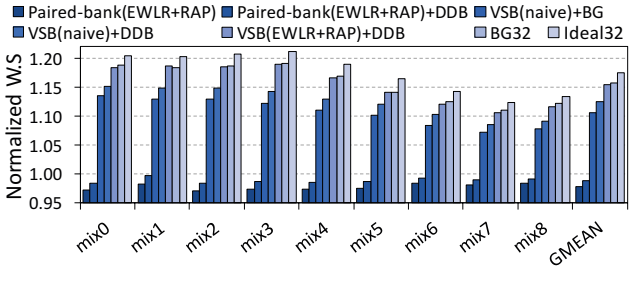



Fig. 12: Normalized weighted speedup over DDR4.

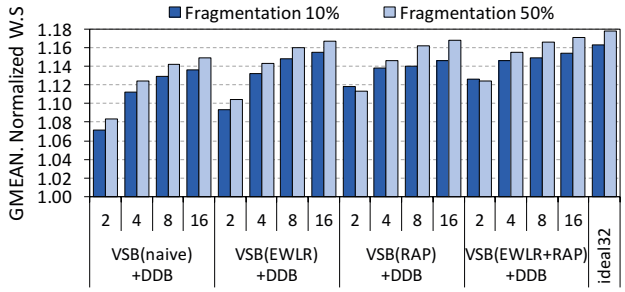
the DRAM address mapping of Skylake (Fig. 9) successfully distributes accesses to different bank groups such that baseline performance is not degraded much by bank groups. We later show that DDB is much more beneficial when bus frequency is higher.

EWLR and RAP provide a substantial improvement of 15% mean weighted speedup over baseline. When both EWLR and RAP are used, the incremental benefit of EWLR on top of RAP is small because of the low address locality in the higher-order address bits, which EWLR targets. For all the mixes, 4-plane VSB with, EWLR, RAP, and DDB provides almost similar performance to DDR4 with a full 32 banks, despite requiring less than 0.3% DRAM area compared to 11% for full banks. Although 4-plane VSB with EWLR and RAP does not perfectly resolve plane conflicts, with the efficient bus parallelism by DDB, it has similar parallelism of naively scaled DDR4 and comes within 2% of the performance of an idealized design. The memory transactions of mix3 and mix4 have high row address locality across many address bits accentuating the benefits of EWLR and RAP. Although not included in Fig. 12, the four memory non-intensive benchmark mixes have 1-6% speedup with 4P VSB over baseline, which approaches the speedup of the ideal case with relatively low plane conflicts.

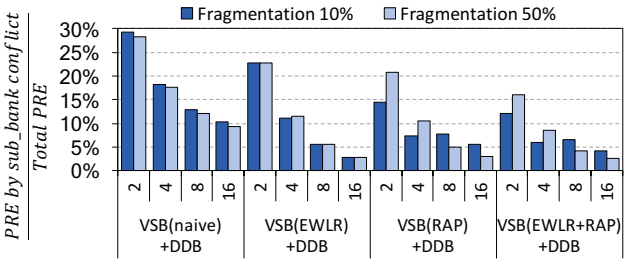
*Paired-bank*-based ERUCA with EWLR+RAP loses 2% performance (GMEAN) compared to baseline DDR4. Adding DDB reduces the loss to 1%. Paired-bank ERUCA enables a 1.1% DRAM area savings (cf. 1% profit margin), demonstrating the power of our novel ERUCA mechanisms.

***Plane-count sensitivity.*** Fig. 13a compares the weighted speedup of VSB with DDB as we vary the plane count and plane conflict avoidance scheme. This sensitivity evaluation is conducted with both 10% and 50% physical memory fragmentation to observe the potential impact of lower memory address spatial locality.

Increasing the plane count improves the performance with all EWRL and RAP combinations. As expected, the relative benefits diminish as plane count increases. With 16 planes, all configurations except for the naive VSB have similar speedup, which is within 0.5% of the performance of an idealized DRAM with 32 full banks and no bank groups. Naive VSB with 16 planes has lower performance because it suffers from plane conflicts that are removed most-effectively by EWLR. This is also expected as RAP targets an optimal distribution across a small number of




(a) Normalized weighted speedup; results are normalized separately to DDR4 with 10% and 50% memory fragmentation.




(b) Fraction of precharges triggered by plane conflicts.

Fig. 13: Conflict-avoidance and plane count sensitivity.

planes and EWLR is designed to take advantage of address locality even when plane count is high; indeed with just 2 planes, RAP provide greater benefit than EWLR. VSB with both EWLR and RAP is less sensitive plane count, and exhibits just a 4% performance variation between 2 and 16 planes.

We find that when both techniques are used together, most of the penalty of plane conflicts is avoided with only two planes, coming within 4% of ideal performance. This is an important result because the area overhead with 2 planes is 27% smaller, and row repair is twice more effective than with 4 planes.

When memory fragmentation is higher, increasing bank parallelism has a stronger impact on performance. This is because the baseline DDR4 with 16 banks exhibits lower row locality, and hence, more bank conflicts when memory is highly fragmented. The impact of memory fragmentation is more noticeable for VSB with RAP only. RAP applies different DRAM address mappings to the high-order row address bits to avoid plane conflicts under the assumption that row locality in higher-order bits of the address is larger. Locality is lower when memory is highly fragmented, which decreases the effectiveness of RAP. Plane conflicts involved with RAP are more frequent with fewer planes because RAP has only two candidates to remap, and diminish as remapping flexibility increases with additional planes.

Fig. 13b compares the fraction of precharge commands issued as a result of plane conflicts. This metric shows the degree of plane conflicts in all pairs of sub-banks. The results are highly correlated with the speedup trends. As with the performance trends (Fig. 12), the combination of EWLR and RAP is effective at reducing dependence on high plane

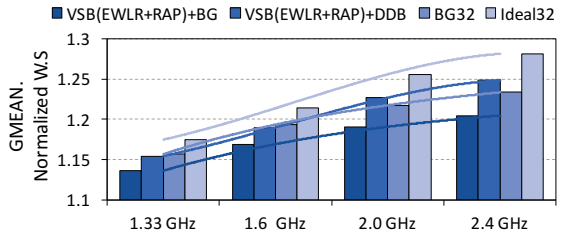



Fig. 14: DDB speedup at different frequencies.

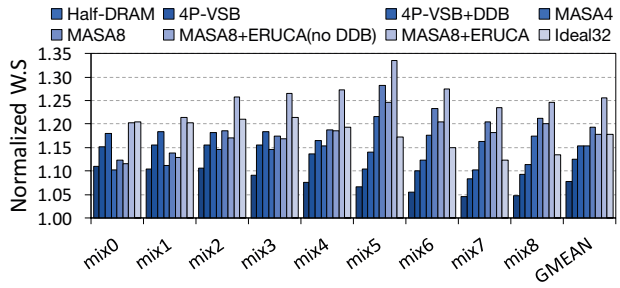
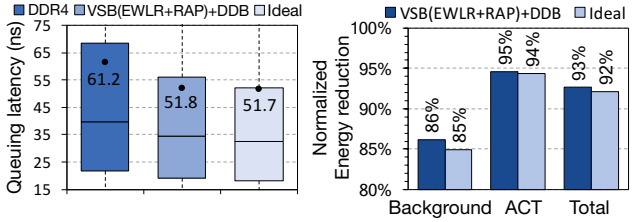



Fig. 15: Performance comparison to prior work. Weighted speedups are normalized to baseline DDR4. Both EWLR and RAP are applied to VSB.

counts.

***Sensitivity to memory channel frequency.*** Fig. 14 compares the speedup achieved by DDB as channel frequency increases. We also scale up the CPU clock frequency proportionally to maintain the memory access intensity across experiments. Because DRAM internal frequency is fixed (200MHz [8]), the disparity in frequencies increases for higher channel frequencies. As a result, the channel delay caused by bus conflicts increases with growing channel clock frequency as well. The speedup of the two configurations using bank grouping saturates due to the growing timing delay (tCCD_L) because accesses are no perfectly interleaved across bank groups. On the other hand, VSB with DDB shows a similar performance growth trend to that of an idealized DRAM. VSB with DDB exhibits 5% higher speedup compared to VSB without DDB at 2.4GHz channel frequency, for example. Note that DDB has to use command window parameters from 2Ghz due to DDB resource conflicts. This shows that DDB supports scalable data bus parallelism and does not hinder the performance gained by a faster channel.

***Comparison to prior work.*** Fig. 15 compares ERUCA to prior DRAM sub-banking schemes. The performance gain of Half DRAM is limited to 8% as it suffers from conflicts on the shared row address latches. With ERUCA and the similar (yet cheaper) 4-plane VSB +DDB, performance improvement doubles to 15% over baseline. Both MASA4 and MASA8 with 4 and 8 sub-array groups respectively, outperform VSB+DDB and ideal32 at medium memory intensity because they offer a larger number of effective banks. However, with high memory intensity, the performance benefit of MASA is restricted by the shared GBL




(a) Read quieting latency: dot(mean), box(1st, median, 3rd quartile)

(b) Normalized average energy reduction compared to DDR4

Fig. 16: Latency and energy consumption comparison.

wires between sub-array groups. This sharing requires an additional latency (tSA) [2] to interleave accesses to different sub-array groups.

Combining VSB(EWLR+RAP)+DDB and MASA provides significant performance synergy with a larger number of effective banks, row-locality utilization, and chip-level data bus parallelism. Our evaluation shows that MASA8 with VSB and VSB+DDB have 7% and 10% performance gains compared to MASA8 only, and 26% and 29% gain compared to baseline DDR4.

***Latency and power analysis.*** Fig. 16 a compares the average read queueing latency of the different memory configurations. ERUCA with VSB (EWLR+RAP+DDB, 4 planes) significantly reduces all statistical read queueing latency parameters and all but one parameter essentially match those of the ideal case. Importantly, average latency decreases by 15% from DDR4 and is within 1% of ideal. The third quartile latency is higher than ideal because ERUCA suffers from rare plane conflicts that delay DRAM accesses within the memory scheduling queue. This explains the small difference in performance between ERUCA and ideal.

Fig. 16 b shows the average energy reduction compared to standard DDR4. Background energy is saved mainly by improving overall performance and execution time. ERUCA reduces activation energy by 6% compared to the baseline by making more effective banks available and increasing the opportunity to exploit DRAM page locality. In addition, every EWLR hit saves the energy associated with precharging a row. Again, the difference between ERUCA and ideal is minimal—under 1% of expected energy consumption.

## IX. Conclusion

In this paper, we show how delicate manipulation of DRAM access mechanisms and DRAM internal structures leads to significant performance improvements of $\sim 15\%$ with less than a $0.3\%$ DRAM area overhead. These benefits are the result of maximizing available parallelism within a DRAM chip, both in terms of enabling efficient sub-banking and increasing internal global-chip interconnect bandwidth. While prior work targeted similar benefits [2], [3], [4], [14], [6], ERUCA is the first to recognize the inter-sub-bank relation and its impact on parallelism efficiency. EWLR takes advantage of the partitioning of signaling into a main wordline and local select signals to increase sub-banking

flexibility at low cost. The RAP hashing scheme better utilizes sub-banks and avoids unnecessary resource conflicts.

Another important insight in our work is that with more effective banks and with growing disparity between core and bus clock rates, additional internal global-chip bandwidth is necessary. We design a light-weight modification to the existing bank-group bus that practically doubles its bandwidth and leads to performance gains. Again, we believe we are the first to make this observation and exploit it.

We show how ERUCA provides nearly-free performance gains by exploiting underutilized resources of Combo DRAM when configured for x4 interfaces and by carefully avoiding overheads of prior sub-banking schemes. At the same time, we demonstrate that the ERUCA techniques are effective for those other sub-banking schemes as well. In addition to Combo DRAM, we show that ERUCA can be applied to other DRAM designs to save cost while maintaining performance.

## ACKNOWLEDGMENT

The authors appreciate the reviewers for their comments and feedback and acknowledge Texas Advanced Computing Center for providing HPC resources.

## REFERENCES

[1] B. Keeth, R. J. Baker, B. Johnson, and F. Lin, *DRAM Circuit Design: Fundamental and High-Speed Topics*. Wiley-IEEE Press, 2nd ed., 2007.

[2] Y. Kim, V. Seshadri, D. Lee, J. Liu, and O. Mutlu, "A case for exploiting subarray-level parallelism (SALP) in DRAM," in *Computer Architecture (ISCA), 2012 39th Annual International Symposium on*, pp. 368–379, IEEE, 2012.

[3] Y. H. Son, O. Seongil, H. Yang, D. Jung, J. H. Ahn, J. Kim, J. Kim, and J. W. Lee, "Microbank: Architecting through-silicon interposer-based main memory systems," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '14, (Piscataway, NJ, USA), pp. 1059–1070, IEEE Press, 2014.

[4] T. Zhang, K. Chen, C. Xu, G. Sun, T. Wang, and Y. Xie, "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)*, pp. 349–360, IEEE, 2014.

[5] N. Chatterjee, M. OConnor, D. Lee, D. R. Johnson, S. W. Keckler, M. Rhu, and W. J. Dally, "Architecting an energy-efficient dram system for gpus," pp. 73–84, 2017.

[6] T.-Y. Oh, Y.-S. Sohn, S.-J. Bae, M.-S. Park, J.-H. Lim, Y.-K. Cho, D.-H. Kim, D.-M. Kim, H.-R. Kim, H.-J. Kim, *et al.*, "A 7 Gb/s/pin 1 Gbit GDDR5 SDRAM with 2.5 ns bank to bank active time and no bank group restriction," *IEEE Journal of Solid-State Circuits*, vol. 46, no. 1, pp. 107–118, 2011.

[7] K. Koo, S. Ok, Y. Kang, S. Kim, C. Song, H. Lee, H. Kim, Y. Kim, J. Lee, S. Oak, Y. Lee, J. Lee, J. Lee, H. Lee, J. Jang, J. Jung, B. Choi, Y.-J. Kim, Y. Hur, Y. Kim, B.-T. Chung, and Y. Kim, "A 1.2v 38nm 2.4gb/s/pin 2gb ddr4 sdram with bank group and 4 half-page architecture.," in *ISSCC*, pp. 40–41, IEEE, 2012.

[8] Joint Electron Device Engineering Council, *DDR4 SDRAM STANDARD, JESD79-4*, Sep. 2012.

[9] Joint Electron Device Engineering Council, *Graphics Double Data Rate (GDDR5) SGRAM Standard, JESD212B.01*, Dec. 2013.

[10] Joint Electron Device Engineering Council, *High Bandwidth Memory (HBM) DRAM, JESD235*, Oct. 2013.

[11] Samsung Electronics Co., "8Gb B-die DDR4 SDRAM." http://www.samsung.com/semiconductor/global/file/product/2017/02/8G_B_DDR4_Samsung_Spec_Rev2_1_Feb_17-0.pdf, 2017.

[12] SK hynix Inc., "8Gb DDR4 SDRAM." https://www.skhynix.com/product/filedata/fileDownload.do?seq=7128, 2017.

[13] Micron Technology Co., "8Gb DDR4 SDRAM." https://www.micron.com/~/media/documents/products/data-sheet/dram/ddr4/8gb_auto_ddr4_dram.pdf, 2017.

[14] H. Ha, A. Pedram, S. Richardson, S. Kvatinsky, and M. Horowitz, "Improving Energy Efficiency of DRAM by Exploiting Half Page Row Access," in *Proceedings of the 49th International Symposium on Microarchitecture*, 2016.

[15] T. J. Dell, "A white paper on the benefits of chipkill-correct ECC for PC server main memory," *IBM Microelectronics Division*, pp. 1–23, 1997.

[16] B. Schroeder, E. Pinheiro, and W.-D. Weber, "DRAM errors in the wild: a large-scale field study," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 37, pp. 193–204, ACM, 2009.

[17] A. Norwood, "Forecast analysis: Dram, worldwide, 3q16 update," 2016. Gartner, Document ID: G00296517.

[18] F. Wang and A. P. Eppich, "6f2 dram cell design with 3f-pitch folded digitline sense amplifier," Mar. 25 2008. US Patent 7,349,232.

[19] K. Koo, S. Ok, Y. Kang, S. Kim, C. Song, H. Lee, H. Kim, Y. Kim, J. Lee, S. Oak, Y. Lee, J. Lee, J. Lee, H. Lee, J. Jang, J. Jung, B. Choi, Y.-J. Kim, Y. Hur, Y. Kim, B.-T. Chung, and Y. Kim, "A 1.2v 38nm 2.4gb/s/pin 2gb ddr4 sdram with bank group and 4 half-page architecture.," in *ISSCC*, pp. 40–41, IEEE, 2012.

[20] M. Horiguchi, J. Etoh, M. Aoki, K. Itoh, and T. Matsumoto, "A flexible redundancy technique for high-density drams," *IEEE Journal of Solid-State Circuits*, vol. 26, no. 1, pp. 12–17, 1991.

[21] M. Horiguchi, "Redundancy techniques for high-density drams," in *Innovative Systems in Silicon, 1997. Proceedings., Second Annual IEEE International Conference on*, pp. 22–29, IEEE, 1997.

[22] P. J. Nair, D.-H. Kim, and M. K. Qureshi, "Archshield: Architectural framework for assisting dram scaling by tolerating high error rates," in *ACM SIGARCH Computer Architecture News*, vol. 41, pp. 72–83, ACM, 2013.

[23] E. Cooper-Balis and B. Jacob, "Fine-grained activation for power reduction in dram," *IEEE Micro*, vol. 30, no. 3, pp. 34–47, 2010.

[24] D. Lee, L. Subramanian, R. Ausavarungnirun, J. Choi, and O. Mutlu, "Decoupled direct memory access: Isolating cpu and io traffic by leveraging a dual-data-port dram," in *Proceedings of the 2015 International Conference on Parallel Architecture and Compilation (PACT)*, PACT '15, (Washington, DC, USA), pp. 174–187, IEEE Computer Society, 2015.

[25] A. Proctor, "Nv-dimm: Fastest tier in your storage strategy," 2012.

[26] S. Lym, "Semiconductor device including a non-volatile memory preserving data stored in a volatile memory when powered off," May 31 2016. US Patent 9,355,723.

[27] C. Song, "Memory and memory module including the same," Mar. 29 2016. US Patent 9,298,558.

[28] Z. Z. Zhao Zhang, Xiaodong Zhang, "A permutation-based page interleaving scheme to reduce row-buffer conflicts and exploit data locality," *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, vol. 00, no. undefined, p. 32, 2000.

[29] D. Kaseridis, J. Stuecheli, and L. K. John, "Minimalist open-page: A dram page-mode scheduling policy for the many-core era," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 24–35, ACM, 2011.

[30] P. Pessl, D. Gruss, C. Maurice, and S. Mangard, "Reverse engineering intel DRAM addressing and exploitation," *CoRR*, vol. abs/1511.08756, 2015.

[31] J. L. Henning, "Spec cpu2006 benchmark descriptions," *ACM SIGARCH Computer Architecture News*, vol. 34, no. 4, pp. 1–17, 2006.

[32] A. Arcangeli, "Transparent hugepage support," in *KVM forum*, vol. 9, 2010.

[33] A. Basu, J. Gandhi, J. Chang, M. D. Hill, and M. M. Swift, "Efficient virtual memory for big memory servers," in *ACM SIGARCH Computer Architecture News*, vol. 41, pp. 237–248, ACM, 2013.

[34] Y. Kwon, H. Yu, S. Peter, C. J. Rossbach, and E. Witchel, "Coordinated and efficient huge page management with ingens," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pp. 705–721, USENIX Association, 2016.

[35] T. Vogelsang, "Understanding the energy consumption of dynamic random access memories," in *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 363–374, IEEE Computer Society, 2010.

[36] T. Aaamodt and A. Boktor, "Gpgpu-sim 3. x: A performance simulator for many-core accelerator research," in *International Symposium on Computer Architecture (ISCA), http://www. gpgpu-sim. org/isca2012-tutorial*, 2012.

[37] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*, pp. 44–54, Ieee, 2009.

[38] K. Chen, S. Li, N. Muralimanohar, J. H. Ahn, J. B. Brockman, and N. P. Jouppi, "Cacti-3dd: Architecture-level modeling for 3d die-stacked dram main memory," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2012*, pp. 33–38, IEEE, 2012.

[39] N. Chatterjee, R. Balasubramonian, M. Shevgoor, S. Pugsley, A. Udipi, A. Shafiee, K. Sudan, M. Awasthi, and Z. Chishti, "Usimm: the utah simulated memory module," *University of Utah, Tech. Rep*, 2012.

[40] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: exploring the level of abstraction for scalable and accurate parallel multi-core simulation," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, p. 52, ACM, 2011.

[41] M. Gorman and A. Whitcroft, "The what, the why and the where to of anti-fragmentation," in *Ottawa Linux Symposium*, vol. 1, pp. 369–384, Citeseer, 2006.

[42] J. M. Dodd, "Adaptive page management," July 11 2006. US Patent 7,076,617.

[43] A. Snavely and D. M. Tullsen, "Symbiotic jobscheduling for a simultaneous mutlithreading processor," *ACM SIGPLAN Notices*, vol. 35, no. 11, pp. 234–244, 2000.