On the Restore Time Variations of Future DRAM Memory

XIANWEI ZHANG, YOUTAO ZHANG, BRUCE R. CHILDERS, and JUN YANG,
University of Pittsburgh

As the *de facto* main memory standard, DRAM (Dynamic Random Access Memory) has achieved dramatic density improvement in the past four decades, along with the advancements in process technology. Recent studies reveal that one of the major challenges in scaling DRAM into the deep sub-micron regime is its significant variations on cell restore time, which affect timing constraints such as write recovery time. Adopting traditional approaches results in either low yield rate or large performance degradation. In this article, we propose schemes to expose the variations to the architectural level. By constructing memory chunks with different access speeds and, in particular, exploiting the performance benefits of fast chunks, a variation-aware memory controller can effectively mitigate the performance loss due to relaxed timing constraints. We then proposed restore-time-aware rank construction and page allocation schemes to make better use of fast chunks. Our experimental results show that, compared to traditional designs such as row sparing and Error Correcting Codes, the proposed schemes help to improve system performance by about 16% and 20%, respectively, for 20nm and 14nm technology nodes on a four-core multiprocessor system.

Categories and Subject Descriptors: B.3.1 [**Memory Structures**]: Semiconductor Memories—*Dynamic Memory (DRAM)*; B.3.3 [**Memory Structures**]: Performance Analysis and Design Aids

General Terms: Design, Performance

Additional Key Words and Phrases: DRAM, memory access, restore, process variation, further scaling, remapping

ACM Reference Format:
Xianwei Zhang, Youtao Zhang, Bruce R. Childers, and Jun Yang. 2017. On the restore time variations of future DRAM memory. ACM Trans. Des. Autom. Electron. Syst. 22, 2, Article 26 (February 2017), 24 pages.
DOI: http://dx.doi.org/10.1145/2967609

1. INTRODUCTION

DRAM (Dynamic Random Access Memory) is the *de facto* memory technology for constructing main memory in modern computing systems. Due to fast technology advances, DRAM has achieved significant density improvements in the past four decades. With the wide adoption of chip multiprocessors and the fast-growing data-intensive applications, for example, graphics and social network applications [Luo et al. 2014; Wu et al. 2013], future computers demand even larger main memory, which needs to scale DRAM further for improved density and energy consumption. However, scaling DRAM in the deep sub-micron regime is subject to significant process variations (PVs) [Asadinia et al. 2015; Agrawal et al. 2014; Mandelman et al. 2002] such that a growing

This work was supported by the National Science Foundation, under grants CCF-1422331, CNS-1012070, CCF-1535755 and CCF-1617071.
Authors' addresses: X. Zhang, Y. Zhang, and B. R. Childers are with Department of Computer Science, University of Pittsburgh, Pittsburgh, PA 15260; emails: {xianeizhang, zhangyt, childers}@cs.pitt.edu; J. Yang is with Department of Electrical and Computer Engineering, University of Pittsburgh, Pittsburgh, PA 15261; email: juy9@pitt.edu.

DOI: http://dx.doi.org/10.1145/2967609

number of cells in a chip shall not be able to meet the standard timing constraints, leading to degraded chip yield rate and increased manufacturing cost.

One representative issue in deep sub-micron scaling is that the time required to fully charge the storage capacitor, that is, restore time, is expected to increase significantly [Kang et al. 2014; Zhang et al. 2015]. For smaller technology nodes, more cells in a chip need longer restore time and thus are likely to violate the standard timing constraints. It is inevitable to relax the timing parameters, such as write recovery time (tWR), in order to maintain an acceptable chip yield rate and to keep manufacturing costs low. However, naively relaxing tWR introduces large performance loss. Adopting post-fabrication cell repair designs such as row sparing and Error Correcting Codes (ECC) to rescue weak cells helps but only to a limited extent.

In this article, we propose to exploit the restore time variations at fine granularity. In particular, we partition memory banks into chunks and expose timing differences of different chunks to the memory controller. By re-organizing device chunks to form logical chunks, we are able to identify a set of fast chunks for performance improvement.

In summary, we make the following contributions.

—We model the scaling and process variation effects on restore time and perform Monte Carlo simulation to study the impact of varying parameters and distributions on performance, which reveals the trend of restore time under technology scaling.
—We propose fine-grained variation-aware scheduling schemes to address restore time variations. By exposing restore time difference at the chunk level, we are able to construct a set of fast chunks that effectively mitigates the performance degradation due to relaxed timing constraints.
—We propose restore-time-aware rank construction and page allocation schemes to fully exploit chunk-level restore time difference. The former groups similar chips together such that a better rank set can be formed from given chips. The latter allocates frequently accessed virtual pages to fast device chunks such that the average memory access latency can be reduced.
—We evaluate the proposed schemes and compare them with existing solutions. Our experimental results show that, compared to traditional designs such as row sparing and ECC, the proposed schemes help to improve system performance by about 16% and 20%, respectively, for 20nm and 14nm technology nodes on a four-core multiprocessor system.

The remainder of the article is organized as follows: Section 2 introduces the DRAM background. Section 3 models the process variations on restore time and motivates the article. Section 4 elaborates the proposed chunk-level mitigation designs. Section 5 and Section 6 present the experimental methodology and analyze the results, respectively. We discuss the related work in Section 7 and conclude the article in Section 8.

2. BACKGROUND

In this section, we discuss the basics of DRAM structure and operations and the scaling effects on restore timing.

2.1. DRAM Cell Restore Operation

A DRAM-based main memory system usually consists of several dual in-line memory modules (DIMMs). One DIMM contains one or multiple ranks while one rank is often constructed using multiple DRAM chips, for example, eight (without ECC) or nine (with ECC) chips, and contains multiple banks that can be operated independently.

A memory controller receives read and write requests from processors and translates requests to device commands. The commands are sent to DRAM modules sequentially following strict timing constraints to ensure reliability and to maximize memory

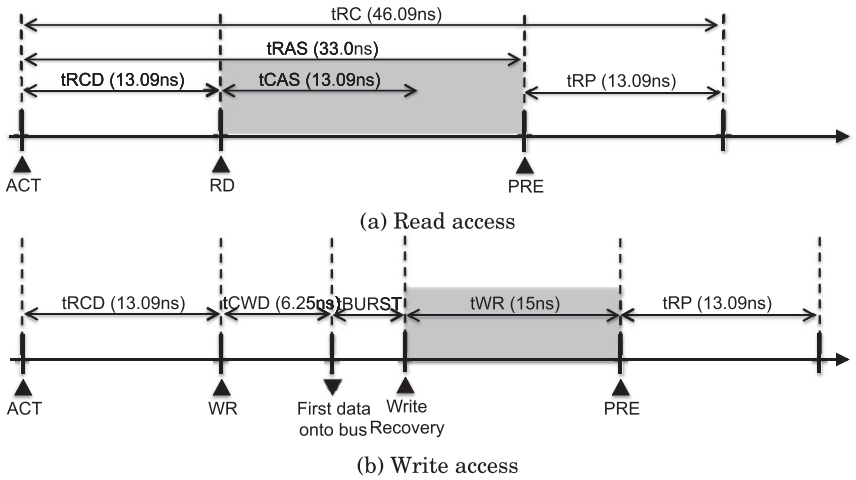


Fig. 1. Commands involved in DRAM access.

performance. We briefly summarize the commands and timing constraints involved in read and write operations. A more comprehensive discussion can be found in Jacob et al. [2007].

READ: as illustrated in Figure 1(a), read access starts with an ACTIVATE (ACT) command to bring the required row into the sense amplifiers (SAs, also referred to as row buffer); then a READ (RD) command is issued to fetch data from the row buffer. The interval between ACT and RD is constrained by tRCD. A DRAM read is destructive, and hence the charge in the storage capacitors needs to be restored. The restore operation is performed concurrently with RD, and a row cannot be closed until restoring completes, which is determined by tRAS-tRCD. Once the row is closed, a PRECHARGE (PRE) can be issued to prepare for a new row access. PRE is constrained by timing tRP. The time for the whole read process is tRC=tRAS+tRP.

WRITE: write works similarly to read, with ACT as the first command to be performed. After tRCD has been elapsed, a WRITE (WR) is issued to overwrite the content in the row buffer and then update (restore) the value into the DRAM cells. Before issuing PRE, the new data overwritten in the sense amps must be restored into the target bank, taking tWR time.

To summarize, both ACT and WR commands involve the restoring operation, and hence a change in restore time shall affect both DRAM read and write accesses.

2.2. Scaling Effects

DRAM scaling has been the major drive for capacity growth in past decades. However, DRAM exhibits non-negligible scaling effects as it scales down to 20nm, 14nm [Samsung 2013], and further to 10nm [Goering 2014; Mayberry 2011; Kang et al. 2014]. When the feature size scales down, the physical geometry of the devices also needs to be downsized, including gate length, transistor supply voltage, and trench capacitor size. It becomes increasingly difficult to precisely control the fabrication process at small feature technologies.

One DIMM, given that it consists of multiple DRAM chips, exhibits both within-die (WID) and die-to-die (D2D) variations, which can be categorized to systematic and random components. While systematic variations are mostly introduced by lithographic aberrations, and show high spatial correlation, random variations are caused by random doping fluctuation and are essentially unpredictable [ITRS 2012].

Table I. Modeling Parameters

tech node	μ_{bulk}	σ_{bulk}	μ_{tail}	σ_{tail}	ϕ	random weight
20nm	2.031	0.21	3.081	0.063	0.3	0.5
14nm	2.048	0.247	3.283	0.0735	0.3	0.5

Because of the process variation, future DRAM behavior will be more statistical than deterministic, and thus more cells are expected to violate current timing constraints. For example, smaller cells tend to require longer time to fully charge the storage capacitor and hold the charge for shorter period of time, which results in longer cell restore time and shorter cell retention time, respectively. To maintain an acceptable yield rate and to keep manufacturing cost under control [Kang et al. 2014], the timing constraints need to be relaxed, which results in performance degradation.

3. MOTIVATION

3.1. Modeling PV Effects on Restoring Time

To study the effects of DRAM scaling, we model PV in the deep sub-micron regime and study its impact on DRAM timing. Our model studies both access transistor and storage capacitor in a DRAM cell. For DRAM scaling, the access transistor can be either FinFET or VCAT [ITRS 2012; Mueller et al. 2005; Kim 2005], both of which feature gate-surrounded channel structure to suppress I_{off} while boosting I_{on}. In this article, FinFET is picked for our circuit model, and the BSIM-CMG version of Predictive Technology Model (PTM) is utilized for simulation. The trench capacitor is assumed to have a capacitance of 27fF [Vogelsang 2010; Mueller et al. 2005] and stays constant in all simulated technology sizes. The core circuits of a DRAM array are built, including cell, sense amplifier, write driver (WD) and column mix, and so on, and those components are simulated in HSPICE. The column circuits have generic topologies [Jacob et al. 2007] and their transistor parameters are taken (and scaled) from a DRAM modeling tool from Rambus [Vogelsang 2010]. Bitline (BL) capacitance due to wire and transistor parasitic is also modeled. In the simulation, the wordline is kept at boosted V_{PP} (e.g., 2.4V at 20nm), and BL and cell are initially grounded by SA. Next, WD overpowers SA, which then pulls up the BL. Meanwhile, the BL voltage is gradually forced onto the cell capacitor by charging through the access transistor. We simulate the write-bit-1 case, which is typically slower than writing bit 0, because the access transistor is gradually source degenerated as the storage capacitor is charged up. The simulation is repeatedly performed on 20nm and 14nm technology nodes.

Using the above cell model, we generate 100K samples and perform curve fitting using log-normal distribution. Similar to recent PV studies [Liu et al. 2012; Agrawal et al. 2014], we include bulk distribution to depict the normal variation that dominates the majority of cells and tail distribution to depict random manufacturing defects.[1] Table I summarizes the parameters for bulk and tail distributions after curve fitting with our cell samples.

To obtain the chip maps, we use the VARIUS tool [Sarangi et al. 2008], which is a novel microarchitecture-aware model for both WID and D2D process variations. VARIUS works by taking a variable's distribution parameters, including mean (μ), standard variance (σ), and spatial correlation (ϕ) to output its variation map. For D2D variations,[2] the μ difference among different dies follow normal distribution, and we borrow the setting of Zhao et al. [2013], Bowman et al. [2002], and Liang et al. [2007] to model the D2D variation. As mentioned, WID variation can be further divided into

[1]Note that not all cells following the tail distribution are treated as defects. The worst ones are covered by conventional redundant repairs [Agrawal et al. 2014].

[2]Differing from Zhang et al. [2015], we take D2D variation into consideration in this article.

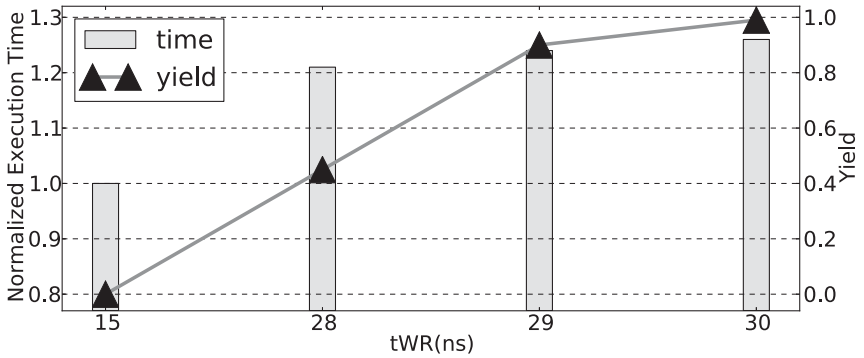


Fig. 2. Comparing performance and yield with different tWR values.

systematic and random effects [Teodorescu and Torrellas 2008; Agrawal et al. 2014]. In VARIUS, systematic variation is modeled using a multivariate normal distribution with a spherical correlation structure [Sarangi et al. 2008]. Moreover, random variation is also modeled with normal distribution. The two distributions are independent and each has its own sigma, denoted as σ_{sys} and σ_{rand}, respectively; hence, the overall normal distribution can be achieved by superposing both components, that is, the overall standard variation is $\sigma_{total} = \sqrt{\sigma_{sys}^2 + \sigma_{rand}^2}$.

Similar to previous PV studies [Karnik et al. 2004; Agrawal et al. 2014], we assume the same share of systematic and random components and choose $\phi = 0.3$, meaning that the correlation range equals to 30% of the chip's side length, as shown in Table I.

3.2. Performance Degradation due to Timing Relaxation

Conventionally, each timing constraint for DRAM has a single fixed value, for example, tWR remains at 15ns in the existing DRAM standard [Samsung 2001]. Given that more cells in the deep sub-micron regime are likely to violate tWR, it is beneficial to relax it to allow most of these cells to finish restoring operations after a destructive read or write operation, which helps to preserve high chip yield. A larger tWR indicates longer bank occupancy and lower bank throughput.

Figure 2 compares the performance and yield with different relaxed tWR values at the 20nm technology node. If tWR is set to 15ns, then the scaling effect would lead to no chip satisfying the existing specification, that is, yield rate is 0%. At the 20nm technology node, the majority of chips have large tWR values in a tight range (28–29). To achieve 99% yield rate, tWR has to be relaxed to 30ns, which prolongs the execution by over 25%; A smaller degradation, for example, 21%, can be observed when tWR is relaxed to 28ns. However, the yield is seriously lowered to 45%.

From the figure, we see that it is challenging to achieve high chip yield while minimizing its impact on system performance.

4. THE PROPOSED DESIGNS

In this section, we elaborate the proposed designs. For discussion purpose, we focus on tWR relaxation while tRAS is relaxed accordingly in each design.

4.1. Chip-Specific tWR Control

We start with a simple enhancement to the current DRAM standard of adopting one tWR—by exposing chip variations, we may set different tWRs for different chips. For this purpose, a post-fabrication test process is performed by the manufacturer to

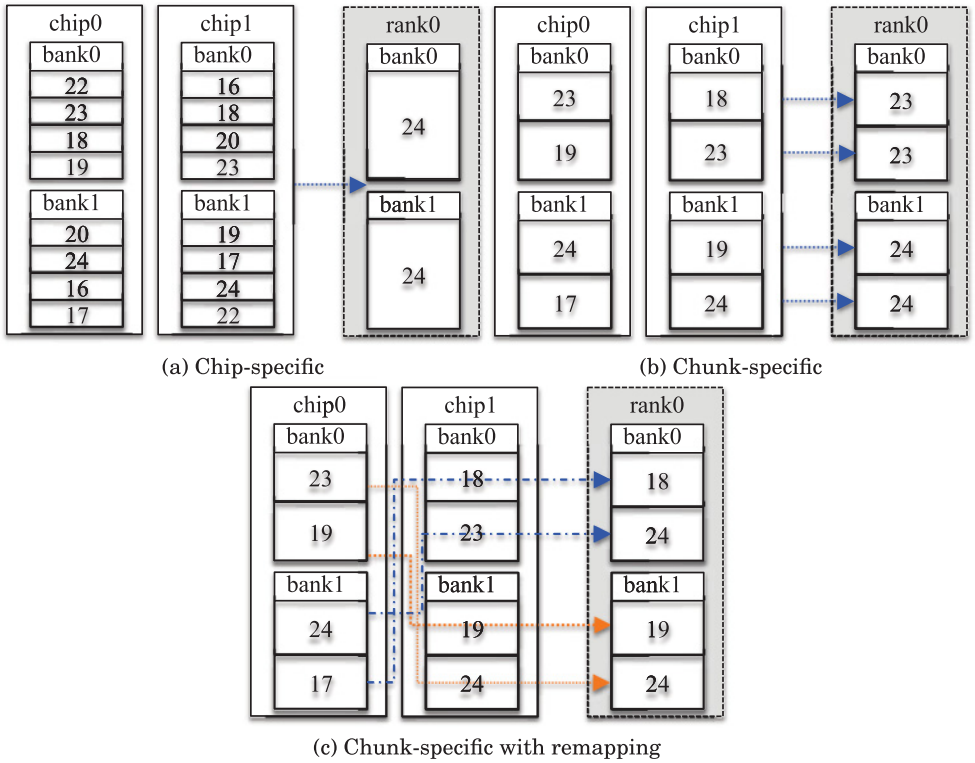


Fig. 3. Comparison of different schemes: (a) the chip-specific tWR; (b) the chunk-specific tWR; (c) the chunk-specific tWR with chunk remapping. For illustration purpose, each rank consists of two chips while each chip contains two four-row banks. One **DIMM-row** (i.e., the row exposed to the OS) consists of two **chip-row** segments—the number in each chip-row indicates its corresponding tWR, that is, the tWR of the weakest cell.

determine the tWR of each chip while a DIMM is then constructed using chips with the same or similar tWRs. Each DIMM derives its tWR from the chip-row[3] that has the worst tWR of the entire DIMM (as shown in Figure 3(a)) or the worst one after adopting a small number of spares to rescue those slowest chip-rows.

The chip-specific tWR design helps to improve chip yield rate, as otherwise a chip with tWR = 24ns would be discarded if tWR is set as 23ns or less in the standard. While technically all fabricated chips can now be treated as good ones, those with very large tWR (e.g., twice as large as the expected tWR) should still be marked as failed chips, as DIMMs constructed from them tend to have very low performance.

4.2. Chunk-Specific tWR Control

Even though tWR exhibits a wide range of variations when scaling in the deep sub-micron regime, only a small number of cells need long recovery time. Setting a DIMM's tWR based on the chip-row that has the worst tWR is often too pessimistic. We therefore propose to partition each memory bank into a number of smaller chunks and set the chunk-level tWR based on the worst chip-row within the chunk. The chunk-level tWR is then exposed to the memory controller for better scheduling.

[3]A chip-row refers to the portion of cells of a row that reside on one chip. A DIMM-row refers to all the cells of a row.

In Figure 3(b), one chunk consists of two rows. Since the first chunk has 23ns and 18ns tWRs for its two chip-rows, its chunk tWR is set to 23ns. By taking advantage of these fast chunks, a chunk-tWR-aware memory controller can speed up memory accesses that fall into the fast chunks.

For discussion purpose, in this article, a *chip-chunk* is referred to as one chunk within one chip; a *DIMM-chunk* is referred to as the set of same-index chip-chunks from different chips of the DIMM. For example, the second DIMM-chunk consists of the second chip-chunk from each chip.

4.3. Constructing Fast Chunks through Chunk Remapping

The previous design can only form a DIMM-chunk from the same-index chip-chunks, which can be optimized to further reduce tWR values. This is because the chip-chunks those are of the same index may exhibit significant tWR difference. It would be beneficial to form a chunk using chip-chunks that are of the same or similar tWRs.

For the example in Figure 3(c), if we form the first DIMM-chunk using the fourth chip-chunk from chip 0 and the first chip-chunk from chip 1, the tWR of this chunk can be as low as 18ns. Constructing a number of such fast chunks helps to speed up the average row access time of the given DIMM.

The chunk remapping is done in two steps: (1) after detecting the tWR for each chip-chunk, we compute the averaged tWR for each chip-bank and sort chip-banks independently on each chip. A **DIMM-bank** consists of chip-banks that are of the same index on the sorted list. (2) For chip-chunks within each chip-bank, we sort them again such that each **DIMM-chunk** consists of chip-chunks that are of the same index on the sorted list.

While only one access is allowed to access one bank at any time, the multiple banks in a DIMM can be accessed simultaneously. To maintain the same bank-level parallelism, we treat the chip-chunks from one bank as a group in chunk remapping. In Figure 3(c), DIMM-chunks 0 and 1 belong the DIMM-bank 0. Since DIMM-chunk 0 is constructed using chip-chunk 3 on chip 0, DIMM-chunk 1 needs to use chunks from the same group, that is, chip-chunk 2 on chip 0. In this way, simultaneously accessing two different DIMM-banks will never compete for the same chip-bank on any chip.

4.4. Restore-Time-Aware Rank Construction

A DIMM rank is composed of multiple chips, which work in lockstep fashion. The access speed of one logical row is determined by its worst chip-row. While chunk-remapping does not have to form a DIMM-row using the chip rows that of the same physical index, it may still be ineffective when one of the chips that form a rank contains many slow rows. A bad chip would lead to a slow rank no matter how the chunks are remapped.

In this article, we propose to construct DRAM ranks using similar chips rather than random chip selection in the baseline design. Algorithm 1 illustrates the details. Given N DRAM chips, our goal is to construct a better rank set (and each rank contains R chips). The rows in each chip are divided into K chunks, and we use M bins to assist rank construction in the algorithm.

We first compute the average chip-level tWR, which uses the chunk-level tWR values of each chip. The latter can be collected during post-fabrication testing. We sort the chips based on their average tWR values and choose M seed chips, that is, the chips on the sorted chip list whose indice can be divided by $\lfloor N/M \rfloor$. The seed chips are distributed to M bins.

We then place the rest of chips into M bins based on their similarity to the seed chip of each bin. The chunk-level tWR values of each chip are treated as a K-item vector. The similarity of two chips is calculated using the Hamming distance of the two K-item vectors. The candidate chip is placed in the bin whose seed chip has the smallest

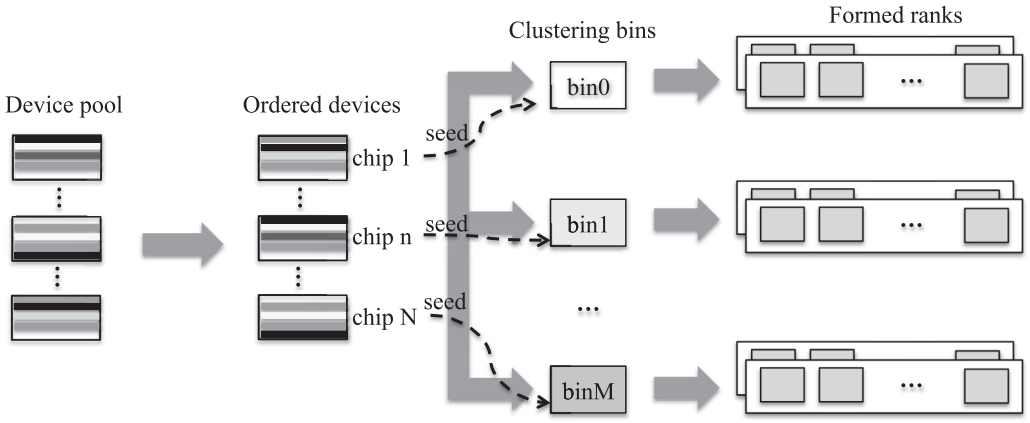


Fig. 4. Rank construction consists of three steps: (1) chip sorting and seed chip selection; (2) distributing chips to bins; (3) constructing DRAM ranks using chips from each bin.

ALGORITHM 1: Bin-Based Rank Construction

Input: Chunk-level tWR values of all candidate chips
Arguments : N-total number of candidate chips; K-chunks in each chip; M-total clustering bins; R-chips in a rank;
Output: N/R formed ranks
1. Preprocessing: calculate the average tWR of each chip, and order the chips;
2. Seed pick-up: assign one seed chip per bin by selecting one every N/M chips;
3. Scan the remaining non-seed chips, and put each into a bin;

$chipID = 0$; bin size $= \lfloor N/M/R \rfloor \times R$;
create one extra bin, Bin_{M+1};
repeat
  fetch the $Chip_{chipID}$ from the pool;
  min_dist = IMAX; min_id $= -1$;
  for $id = 0$; $id < M$; id++; **do**
    if (Bin_{id} *is full*) **then**
      continue;
    end
    $dist =$ Hamming_distance(seed chip of Bin_{id}, $Chip_{chipID}$);
    if ($dist < min\_dist$) **then**
      $min\_dist = dist$; $min\_id = id$;
    end
  end
  if (min_id ! $= -1$) **then**
    put the chip into Bin_{min_id};
  else
    put the chip into Bin_{M+1};
  end
  $chipID$ ++;
until $chipID >= N - M$;
4. Sequentially scan the classified chips of each bin, and form R-chip ranks;

Hamming distance, that is, the highest similarity, to the candidate chip. Once a bin reaches its size limit, that is, $n \times R$, where $n = \lfloor N/M/R \rfloor$ and $n \times R \leq N/M$, it can no longer accept new chips. In the algorithm, an extra bin Bin_{M+1} is used to hold the leftover chips. When filling chips to each bin, we construct a rank if a bin has R chips (the seed chip is used to form a rank in the last batch).

Table II. Comparison of Average tWRs for 20nm and 14nm Nodes (10 Ranks Were Constructed and Each Chip Has 4,096 Chunks)

rankID	Baseline[1] (ns)	Chunk[2] (ns)	ChunkBin[3] (ns)	ChunkSortBin[4] (ns)
0	25.9	22.4	25.9	27.3
1	30.1	27.1	30.7	27.3
2	25.9	22.2	25.9	22.4
3	30.7	27.2	25.9	22.2
4	28.8	27	31.8	22.2
5	28.8	27	25.9	22.2
6	28.8	27.1	25.9	22.2
7	28.8	27.1	25.9	22.3
8	25.9	22.3	25.9	22.2
9	28.8	27.0	25.9	22.2
avg(20nm)	28.25	25.64	26.97	23.25
avg(14nm)	35.04	31.71	33.33	28.73

The table lists the rank-level tWR values and their averages for 20nm node; and the average tWR values for 14nm node.
Note:
[1]`Baseline` is the baseline that constructs ranks using random grouping.
[2]`Chunk` is the scheme that conducts chunk remapping on the basis of `Baseline`.
[3]`ChunkBin` is the scheme that adopts our proposed restore-time-aware rank construction. Chunks within each chip are not remapped.
[4]`ChunkSortBin` is the scheme that remaps chunks after the rank construction.

Since Algorithm 1 needs to scan each chip and compute its similarity with all seed chips, the time complexity is $O(N \times M \times K)$. Here M and K are constant. M is usually small ($M << N$) while K can be relatively large, for example, $K = 1{,}024$. Therefore, the time complexity is linear to the number of candidate chips. This is a light weight rank construction scheme. As a comparison, the recently proposed rank construction scheme [Wang et al. 2015] needs to sort the candidate chips continuously, which results in time complexity up to $O(N^3)$. Our experiments show that the two algorithms achieve similar rank-level tWR results.

Table II compares the average tWR values when using different rank construction algorithms. We constructed 10 ranks, and the rows in each bank are divided into 4,096 chunks. We list the rank tWR values and the average tWR values at 20nm technology node and the average tWR values at 14nm technology node. From the table, we find that, without chunk remapping, the slow rows can significantly affect the rank-level tWR values. For example, for `Baseline` at 20nm node, the average tWR of all ranks is 28.25ns while the average tWR of rank-1 is 30.1ns. Performing chunk remapping after bin-based rank construction is the most effective scheme. In the experiments, we set the number of bins to 5. Varying the bin counts from 2 to 10 shows similar results. In addition, we compare the results with the scheme that performs heavy weight rank construction [Wang et al. 2015] and observe similar average tWR values of all ranks.

4.5. Restore-Time-Aware Page Allocation

The translation of virtual to physical address is supported in hardware by a Memory Management Unit (MMU), and the virtual-physical mapping is determined by the operating system (OS). Traditional page allocation is restore time oblivious, as all physical pages have the same access latency. However, when a set of fast DRAM chunks are constructed and exposed to the memory controller, it is beneficial to exploit the access latency difference to speed up program execution.

Clearly, the memory system can be more effective if fast chunks are assigned to service performance-critical pages. In this article, the page criticality is estimated using its

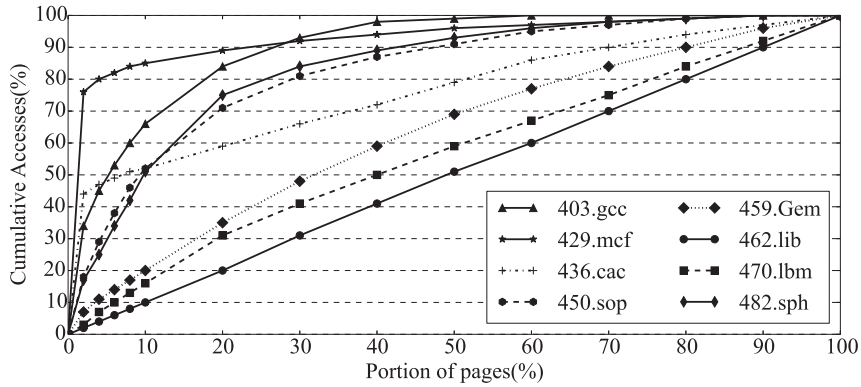


Fig. 5. The page access distributions in SPEC CPU2006.

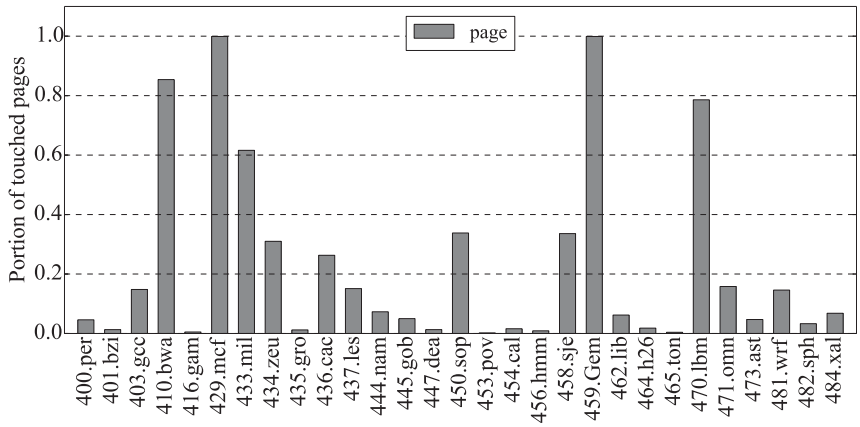


Fig. 6. The portion of touched pages for each benchmark. The memory capacity is 2GB, which can be divided into 52K 4KB-page.

access frequency [Son et al. 2013; Lee et al. 2001]. Studies have shown that it is usually a small subset of pages, referred to as hot pages, that are frequently accessed in modern applications [Bhattacharjee and Martonosi 2009; Ramos et al. 2011; Ayoub et al. 2013]. We adopt the offline profiling approach as in Son et al. [2013] to identify hot pages.

Figure 5 studies the page access distribution of a set of SPEC CPU2006 applications. The figure shows that different applications have very different access behaviors: for some workloads, for example, 459.*Gem* and 470.*lbm*, accesses are evenly distributed such that the number of accumulative requests grows linearly with the number of touched pages; for some other applications, for example, 429.*mcf* and 403.*gcc*, most memory accesses come from a small subset of hot pages. The hot pages are the ones to be allocated in fast DRAM chunks.

The benefit of restore-time-aware page allocation also depends on the number of hot pages, that is, whether the hot page set can all be allocated in the fast chunks. Figure 6 compares the number of touched pages of different benchmarks. From the figure, the majority touch less than one-eighth of the total memory space, while some benchmarks (i.e., 459.*lbm* and 429.*mcf*) use up all available space.

In this article, our goal is to illustrate that a restore-time-aware page allocator can take advantage of the latency difference of the DRAM chunks. For this purpose, we

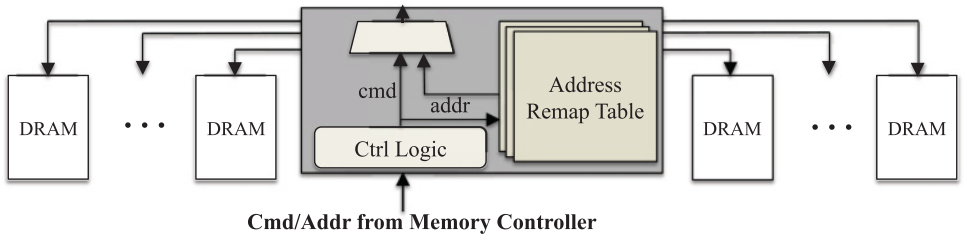


Fig. 7. The on-DIMM architectural enhancement.

Table III. Remap Table

DIMM_chunk	chip0_chunk	chip1_chunk	...	chip7_chunk
...	...	...	...	...
10	1220	124	...	256
...	...			

adopt a simple strategy that profiles program execution offline and statically allocates hot pages to fast chunks. In the case where profiles are not accurate, we may need to design and enable more flexible strategies, such as the detailed behavioral synthesis [J. Cong and Zou 2011] and data migration and compression [Ozturk and Kandemir 2008]. We leave this as our future work.

4.6. Architectural Enhancements

In order to exploit restore time variations at either chip- or chunk-level, post-fabrication testing needs to be performed to detect restore time at fine granularity. Given that cell restore time is thermal dependent, study has shown that it becomes worse at low temperature [Kang et al. 2014], the manufacturer needs to examine different types of data patterns [Venkatesan et al. 2006; Liu et al. 2012, 2013] to record the worse timing constraints under a chip's allowed working conditions. The values are organized as a table (with each entry in the table recording affected timing constraints tWR/tRAS of its corresponding DIMM chunk) and saved in non-volatile storage in the DIMM [Seshadri et al. 2013].

The memory controller loads this table at boot time and schedules memory accesses accordingly to maximize bandwidth and avoid conflicts. As an example, two READ operations cannot be scheduled back to back to a DIMM bank if the later one accesses a fast chunk and shall compete with the preceding READ for using the data bus.

To enable chunk re-organization, we need one extra chunk remapping table as shown in Figure 7. Similarly as HP's MC-DIMM [Ahn et al. 2009] and Mini-rank [Zheng et al. 2008], our design integrates an enhanced registering clock driver (RCD) [JEDEC 2009] to remap and redrive the physical address. Differing from rank-subsetting designs where only partial chips are involved for each memory access, our proposed design follows the chips' original lockstep working fashion, but only requires each chip to receive a dedicated row address. Compared to conventional DIMM RCD, an additional register, together with some pins, and DIMM PCB traces [Micron 2008] are needed to implement address remapping. For the chunk remapping table, each entry maps the corresponding DIMM-chunk to the chip-chunk at each chip. Given the following Table III, when the bridge chip receives a request asking for data in the 10th DIMM-chunk, it translates the requests to asking for segment data from the 1,220th chunk from chip 0, the 124th chunk from chip 1, and so on.

Table IV. System Configuration

Processor	Four 3.2GHz cores; four-issue; 128 ROB size
Cache	L1(private): 64KB, 4-way, 3 cycles L2(shared): 2MB, 6-way, 32 cycles 64B cacheline
Memory Controller	Bus frequency: 1066 MHz 128-entry queue; close page
DRAM	1 channel, 2 ranks/channel, 8 banks/rank, 16K rows/bank, 8KB/row, 1066 MHz, tCK = 0.935ns, width: x8

Table V. System Configuration

Timing Parameters	Time(ns)	DRAM Cycles(CLK)
CL	13.09	14
tRCD	13.09	14
tRC	46.09	50
tRAS	33.0	36
tRP	13.09	14

5. EXPERIMENTAL SETUP

5.1. Configuration

To evaluate the effectiveness of our designs, we compared them to traditional repair solutions using an in-house chip-multiprocessor system simulator. We modeled a quad-core system with the parameters shown in Table IV.

For both 20nm and 14nm technology nodes, we used VARIUS to generate 90 chips and then form ranks in a different fashion, discussed in Table II. The memory system to be simulated is composed of two ranks. We constructed five rank pairs and tested the proposed designs with all pairs.

The DRAM timing constraints follow the Hynix DDR3 SDRAM data sheet [Hynix 2010] and are summarized in Table V. For the schemes exploiting chunk-level timing constraints, we added two CPU cycles to access the timing table. For the schemes performing chunk-remapping, we added one extra DRAM cycle to access the mapping table.

5.2. Workloads

We used SPEC CPU2006 and simulated 1 billion instructions after skipping the warm-up phase of each benchmark. The Read and Write memory accesses per kilo instructions (MPKI) for each workload is profiled to indicate the memory intensiveness. Based on MPKI, the applications are classified into three categories (Spec-High/Spec-Med/Spec-Low), as shown in Table VI. The workloads are running in rate mode, where all cores execute the same task.

We performed timing simulation until all cores finish the execution and averaged the execution time of all the four cores. We constructed five rank pairs, that is, DIMMs. One simulation run used one DIMM while the reported results are the average of the runs using different DIMMs.

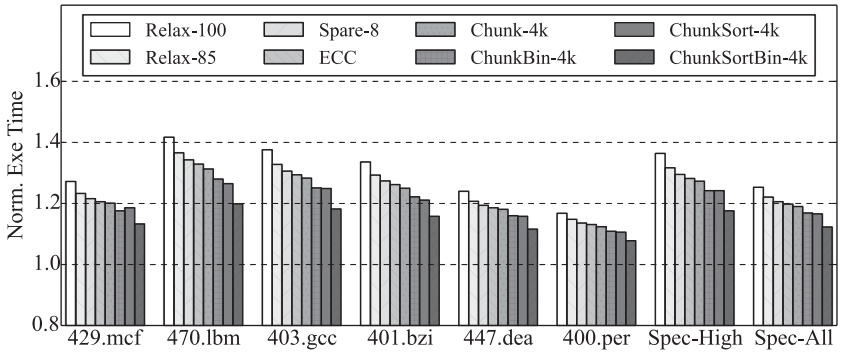
6. RESULTS

We evaluated the following schemes:

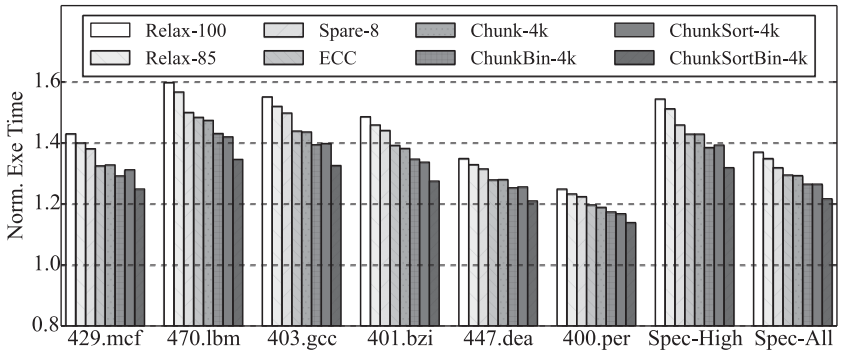
Table VI. Benchmark Characteristics

Class	Workload	Read MPKI	Write MPKI
Spec-High	429.mcf	58.07	4.56
	470.lbm	31.31	23.38
	450.sop	26.77	2.69
	433.mil	24.85	8.38
	471.omn	19.26	0.04
	459.Gem	19.04	2.66
	462.lib	17.48	0.52
	484.xal	16.89	0.45
	403.gcc	14.69	0.52
	482.sph	14.14	1.11
	410.bwa	10.04	1.67
Spec -Med	437.les	8.70	2.48
	481.wrf	5.18	1.40
	436.cac	5.13	1.52
	434.zeu	4.69	1.22
	401.bzi	3.92	1.72
	473.ast	3.53	0.96
	447.dea	3.15	0.20
	456.hmm	3.11	2.88
Spec -Low	400.per	1.73	0.22
	464.h26	1.47	0.66
	445.gob	1.27	0.92
	435.gro	1.06	0.28
	458.sje	0.90	0.45
	454.cal	0.66	0.35
	444.nam	0.65	0.62
	465.ton	0.30	0.03
	416.gam	0.18	0.04
	453.pov	0.01	0

—`Baseline`. The baseline sets tWR to 15ns, the same as existing DRAM specification. It is the ideal baseline due to scaling. The results of other schemes are normalized to the baseline for comparison.
—`Relax-`x. Given that scaling in the deep sub-micron regime leads to worse timing, this scheme relaxes time constraints to achieve x% yield. We relaxed tWR and set tRAS/tRC accordingly. We tested $x = 85$ and $x = 100$, respectively.
—`Spare-`x. One commonly adopted post-fabrication repair approach is to integrate sparing rows/columns to mitigate performance and yield loss. It is implemented by using a laser programmable link to disconnect the abnormal rows/columns and connect the spare ones [Jacob et al. 2007]. In our experiments, we set the spare density as high as 16 spare rows per 512-row block, which resides in the aggressive spectrum [Kirihata et al. 1996; Koren and Krishna 2010]. Given that spares may be reserved for high-priority repairs, such as defects and retention failures, we tested $x = 0, 2, 8, 16$ spares from each 512-row block, respectively.
—`ECC`. ECC is implemented by placing one extra ECC chip to correct errors in data chips. Though ECC is conventionally used to correct soft errors, it can be potentially used to tolerate weak cells. Exploiting ECC chips to rescue slow rows sacrifices soft error resilience and hurts reliability [Su et al. 2005].



(a) 20nm normalized average execution time



(b) 14nm normalized average execution time

Fig. 8. The execution time comparison of different schemes for 20nm and 14nm technology nodes under random page allocation policy. Representative applications and the geometric means for highly memory-intensive (Spec-High) and all applications (Spec-All) are presented here.

—`Chunk-`x. This scheme implements the chunk-specific restore time control, with each bank being divided into x chunks. Each DIMM chunk has its own timing constraints, which are exposed to the variation-aware memory controller.
—`ChunkSort-`x. This scheme implements the chunk-specific restore time control with chunk remapping, with each bank being divided into x chunks. Similarly as `Chunk-`x, the timing constraints of each chunk are exposed to the memory controller.
—`ChunkBin-`x. This schemes is similar as `Chunk-`x. The difference is that it constructs ranks using the proposed bin-based matching scheme.
—`ChunkSortBin-`x. This schemes is similar as `ChunkSort-`x. The difference is that it constructs ranks using the proposed bin-based matching scheme.

We compared these schemes on memory access latency and system performance, and studied their sensitivity to different system configurations.

6.1. Impacts on Program Execution Time

Figure 8 compares the execution time with random page allocation policy under different schemes for 20nm (Figure 8(a)) and 14nm (Figure 8(b)) technology nodes. The execution time is normalized to the ideal baseline, that is, tWR is 15ns. The figure reports the results from representative benchmarks of all classified categories (e.g., `Spec-High` is the set of highly memory-intensive subset) and the full set (`Spec-All`).

6.1.1. 20nm Technology Node. From Figure 8(a), we observed that (1) DRAM scaling has a large impact on restore time. To maintain a high yield rate, the timing constraints have to be vastly relaxed from 16 cycles to over 30 cycles, which significantly hurts performance. On average, `Relax-100` and `Relax-85` prolong the execution time by 25.3% and 22.1%, respectively. Highly memory-intensive applications tend to have large degradation (i.e., over 30%). (2) Adding spare rows helps to mitigate performance losses: `Spare-8` is 20.6% worse than the ideal. (3) `ECC` works only slightly better than `Spare-8`. This is because SEC-DED ECC can only correct one bit in each 64-bit block. Since there might be multiple cells violating timing constraints within such a 64-bit block, ECC lacks the ability to effectively adapt restore time variations. (4) `Chunk-4k` is 1% better than `ECC` as it exposes chunk-level restore time variations. There are a small number of chunks that have smaller tWRs than the single tWR in `ECC`. Due to random page allocation policy, the exposed fast chunks cannot be fully exploited, and thus the performance improvement is pretty limited. (5) `ChunkSort-4k` works better than `Chunk-4k` because it helps to construct more fast chunks. On average, `ChunkSort-4k` helps to reduce the performance loss from 25.3% in `Relax-100` to 16.6% and 3% better than `Chunk-4k` for `Spec-High`.

In addition, restore-time-aware rank construction helps to reduce tWR: `ChunkBin-4k` is 2% better than `Chunk-4k` while `ChunkSortBin-4k` is 4.3% better than `ChunkSort-4k`. Interestingly, `ChunkBin-4k` and `ChunkSort-4k` achieve comparable performance improvements over the baseline. While both schemes require post-chip-fabrication testing to extract chunk-level tWR values, the former needs rank clustering, which imposes an extra step and cost during fabrication; the latter needs to embed mapping table and thus introduces extra runtime overhead. `ChunkSortBin-4k` achieves the best performance while it incurs both extra fabrication cost and runtime overhead.

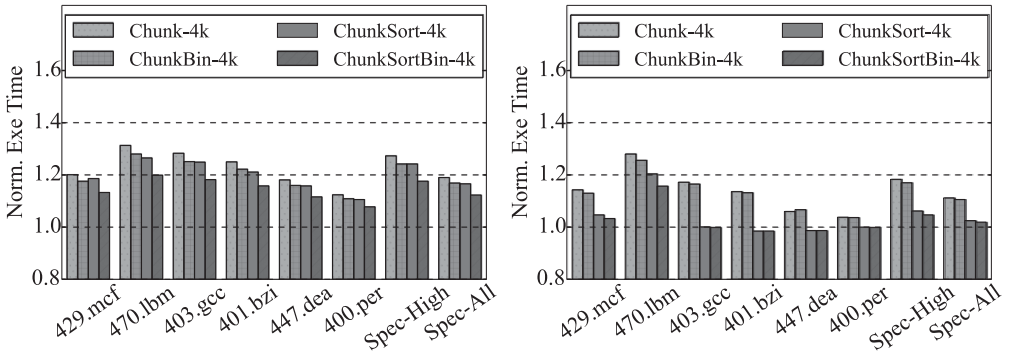
6.1.2. 14nm Technology Node. Figure 8(b) shows the normalized execution time for 14nm technology node. Comparing with Figure 8(a), the performance difference from the ideal increases as the technology node scales down. For example, `Relax-100` exhibits a 25% loss at 20nm node while a 37% loss at 14nm node. In general, highly memory intensive applications, for example, 403.*gcc*, show large losses. Due to the fact that more memory cells violating timing constraints, it becomes increasingly difficult to mitigate performance loss at small technology nodes.

With more slow cells at 14nm, both ECC and Sparing become less effective and their impacts on performance become smaller. `ChunkSort-4k` shows a tendency of better improvement over traditional solutions and `Chunk-4k`. For Spec-High, `ChunkSort-4k` executes 7% faster comparing to `Spare-8` and `ECC`. `ChunkSortBin-4k` achieves over 20% performance improvements over `Spare-8` and `ECC` and 27.5% over `Relax-100`.

6.2. Restore-Time-Aware Page Allocation

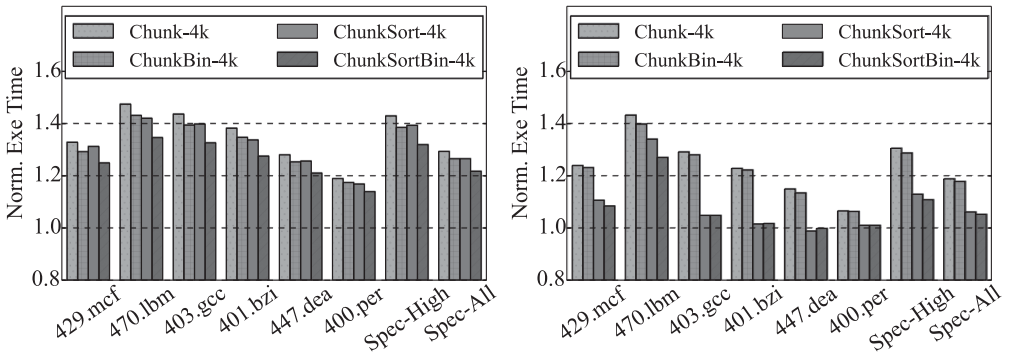
Figures 9 and 10 compare the results using random and restore-time-aware page allocation schemes at 20nm and 14nm technology nodes, respectively. From the figure, by making better use of fast chunks, restore-time-aware page allocation speeds up the execution of all chunk based schemes, for example, for `ChunkSortBin-4k`, restore-time-aware allocation achieves 10% and 15% improvement over random allocation for 20nm and 14nm nodes, respectively. Restore-time-aware allocation is very effective for most benchmark programs—on average, `ChunkSortBin-4k` is only 2% worse than the ideal `Baseline`.

In the figure, 470.*lbm* achieves small improvement because it evenly accesses a large number of memory pages and lacks very hot pages. Given that a small number of chunks have shorter than 15ns tWR values, it is not surprising to find that some



(a) With random page allocation (b) With restore-time-aware page allocation

Fig. 9. The execution time comparison of different schemes at 20nm technology node.



(a) With random page allocation (b) With restore-time-aware page allocation

Fig. 10. The execution time comparison of different schemes at 14nm technology node.

benchmark programs, for example, 403.*gcc* and 400.*per*, have their hot pages fit in these fast chunks and thus outperform `Baseline`.

Also in the figure, we observed that the effectiveness of restore-time-aware rank construction diminishes after adopting restore-time-aware allocation. For example, on average, `ChunkSort-4k` and `ChunkSortBin-4k` have a less than 1% difference when using restore-time-aware allocation at the 20nm node. Nevertheless, those benchmarks with a large footprint and relatively uniform access pattern, for example, 470.*lbm*, can still achieve distinct benefits.

6.3. Impacts on Memory Access Latency

Figure 11 compares the average memory access latencies under different schemes. Among these schemes, `ChunkSortBin-4K` achieves the lowest latency, which is 7% lower than `ChunkBin-4K`, about 12% lower than `Spare-8` and `ECC`, and 17% lower than `Relax-100` for the 20nm technology node with random page allocation. There is a clear latency increase for the 14nm technology node, for example, the average memory access latency of `ChunkSortBin-4K` increases from 280ns to 434ns. This is mainly due to further relaxed timing constraints. In addition, restore-time-aware allocation offers great help in lowering the latency for chunk-based schemes.

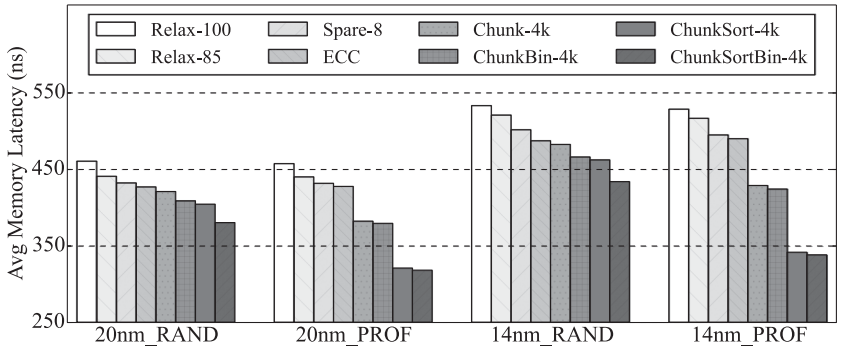


Fig. 11. Comparing memory latencies under different schemes (values are averaged over all SPEC benchmarks).

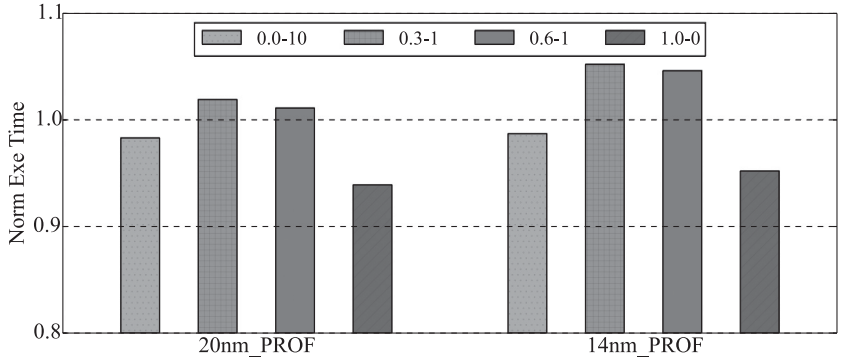


Fig. 12. Sensitivity study of scheme *ChunkSortBin-4k* for both the 20nm and 14nm tech nodes, under different cases (ϕ-w), in terms of the spatial correlation parameter (ϕ) and the ratio of systematic and random distributions (w).

6.4. Sensitivity Study

The effectiveness of the conventional Sparing technique strongly depends the sparing levels being used; the proposed chunk-based schemes depends on the chunk granularity. We conducted the sensitivity studies on these two key parameters.

6.4.1. Varying Variation Correlation. As discussed in Section 3.1, restore variation is the combination of systematic and random components: The systematic part is characterized by the spatial correlation, which is depicted by ϕ; random variation is reflected by the weight of sigmas, that is, $w = \sigma_{rand}/\sigma_{sys}$. To study the correlation effects, we sweep over different combinations of σ-w, as reported in Figure 12. As ϕ decreases and w increases, cells' restoring becomes more randomly distributed; in contrast, combination $1.0 - 0$ gives the extreme cases that all cells in the chip are correlated and the distribution is solely systematic.

Figure 12 shows that our proposed scheme is always efficient because of the exposed cell variation, which agrees with the existing observation that manufacturing defects always provide some weak cells [Agrawal et al. 2014]. In addition, the achieved results are even better under some extreme cases, for example, 0.0-10 and 1.0-0. The reason is that both provide more fast cells and thus larger remapping opportunities to expose fast regions to improve performance.

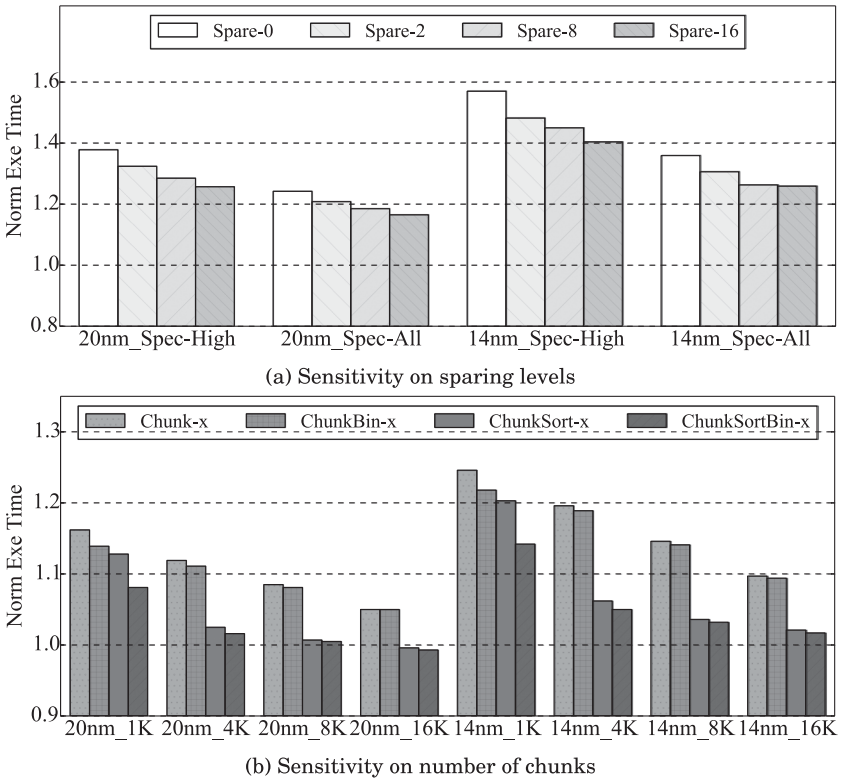


Fig. 13. Sensitivity study using different spares and chunks under hot page allocation policy.

6.4.2. Varying Sparing Levels. Figure 13(a) compares the performance with different spare rows[4] in each 512-row block. The X-axis shows the results for `Spec-High` and `Spec-All` benchmark sets at 20nm and 14nm nodes. While better performance can be achieved with more spares, the improvement gradually diminishes. For example, the difference between `Spare-8` and `Spare-16` is only 0.4% at the 14nm node. The reason is that after rescuing a very small number of super slow rows, we expect to encounter a relatively large number of slow rows (due to scaling effect), which is beyond the ability that a sparing approach has.

6.4.3. Varying the Number of Chunks. Figure 13(b) compares the performance with different numbers of chunks for chunk-specific restore time scheduling. The X-axis `Anm_B` indicates the results for the schemes using `B` chunks at `A`-nm technology node. The results are normalized to the ideal `Baseline`. From the figure, all schemes show better performance when using more chunks. For example, ranging from 1K to 16K chunks, `ChunkSortBin-16K` reduces the performance losses from 15% to 2% at the 14nm node. `ChunkSortBin-16K` achieves better than `Baseline` performance at the 20nm node because exposing tWR at fine-granularity leads to more faster chunks, which reduces the average memory access latencies.

We observed that the performance gap between `Chunk` and `ChunkBin` reduces significantly when there are more than 4K chunks. This is because restore-time-aware

[4]Note that `Spare-0` differs from `Relax-100`, while `Spare-0` places no spares and uses the chip-specific tWR, `Relax-100` sets tWR to ensure all chips can work. As a result, `Spare-0` usually has a lower tWR.

Table VII. Tested Chunk Configuration

#CKs	MC area (KB)	DIMM area (KB)	total area (KB)
1k	24	224	248
4k	96	896	992
8k	192	1792	1984
16k	384	3584	3968

allocation makes better use of fast chunks, and exposing tWR at 8K granularity is often sufficient to identify the set of fast chunks to service the hot page set of the tested benchmark programs. The gap between `ChunkSort` and `ChunkSortBin` reduces similarly.

The performance gap enlarges between `Chunk` and `ChunkSort` with more chunks. This is because remapping finer granularity chunks helps to form a number of faster chunks, which brings down the average memory access latency when they are fully exploited through restore-time-aware allocation. From the figure, we also found that `ChunkSort-4K` is better than `Chunk-8K` while `ChunkSortBin-4K` is better than `ChunkSort-8K`, showing that chunk remapping is very effective in reducing average memory access latency.

6.5. Testing Overhead

The proposed method requires memory manufacturers to test the restoring time of each chunk[5] and then to cluster chips into different ranks/DIMMs. Hence, the manufacturing time will be definitely lengthened. To avoid dramatical increase, chunk-level timing measurement can be performed in a binary search fashion, that is, starting from a middle value (e.g., 22ns), and then halving to the left (e.g., 16-21ns) or right part (e.g., 23-30ns). Further, the number of chunks is fixed to be smaller than that of rows, and thus the testing time of each chip is $O(1)$.

For rank construction, thousands of chips should be clustered into different bins, as discussed in Section 4.4, for later rank formation. To make the algorithm more practical, we propose to run the algorithm for a group of, for example, 10K, already-manufactured chips, instead of running until after 1-day production. Apparently, the algorithm cost would not be too high given the moderate number of chips, and the chip production would have tolerable impact.

6.6. Storage, Area, and Latency Overhead

To enable variation-aware memory scheduling, we added two tables: one is for extracting the timing constraints of each chunk while the other is for chunk address remapping. Table VII summarizes the storage overhead with different chunks.

From Table VII, `ChunkSort-4k` requires 896KB DIMM storage to save the chunk mapping. The table is evenly divided among the banks in the DIMMs, that is, 56KB per bank. We used CACTI 5.3 to model the table as a direct-mapped cache with 8B line size. For 32nm[6] technology, it has 0.348ns access latency, 0.229mm^2 area overhead, and 0.0181W standby leakage power and consumes 0.012nJ dynamic energy per access. Similarly, for the 96KB cache at the memory controller side, we have 0.414ns access latency, 0.349mm^2 area overhead, 0.015nJ energy per access, and 0.03W standby leakage power. The area, power, and energy overheads are very moderate for the DIMM and the memory controller.

The remap table occupies about 1.83mm^2 area in total, which is trivial compared to conventional RCD area of 108mm^2 [JEDEC 2009] and DIMM area of over 4000mm^2 [Micron 2009]. The overheads of added register and DIMM PCB traces are moderate

[5]As in Wang et al. [2015], manufacturing testing results are unnecessary to be saved inside DRAM.

[6]Differing from sub-20nm DRAM chips, caches are constructed with 32nm process technology, which is supported by the CACTI tool.

[Ahn et al. 2008], and the overheads can be even smaller by combining adjacent chips into groups [Ahn et al. 2008]. Based on the latency overheads calculated by CACTI, we charged one memory cycle[7] and two CPU cycles to access the chunk mapping table and the timing table, respectively. We observed less than 1% performance overhead on average.

7. RELATED WORK

7.1. Process Variation

Further scaling DRAM is subject to challenging process variations on write recovery (tWR) and refresh [Kang et al. 2014]. The process variation on retention time has been well studied. Kong et al. [2008] from IBM observed that the variation of retention time follows log-normal distribution. Kim and Lee [2009] investigated the main and tail distributions of retention time. By exploiting the non-uniformity of cell retention time, different refresh schemes have been devised in the literature [Kim and Papaefthymiou 2001; Ohsawu et al. 1998; Venkatesan et al. 2006; Liu et al. 2012]. RAIDR [Liu et al. 2012] classifies DRAM rows into bins according to retention time and applies different refresh rates to save power. Recently, Agrawal et al. [2014] proposed Mosaic to model the process variation of retention time in eDRAM and divide the module into regions to refresh each region at a different rate. Only limited works addressed the process variations on restore time. Kang et al. [2014] pointed out that the time to charge DRAM capacitor increases with technology scaling down. They proposed sub-array level parallelism (SALP) [Kim et al. 2012] to compensate the performance loss. Our design is orthogonal to SALP.

7.2. Timing Reduction

Reducing timing constraint values can significantly improve memory system performance. By segmenting a long bitline using an isolation transistor, TL-DRAM [Lee et al. 2013] creates a segment of rows with low ACT and PRE latencies. CHARM [Son et al. 2013] achieves the latency reductions by attaching fewer DRAM cells to each bitline. NUAT [Shin et al. 2014] exploits the electric charge variation to speed up the charge sharing and sensing operations. Usually, DRAM vendors incorporate excessive timing margins to ensure high yield and reliability, which indicates that DRAM devices are possible to function better than the standard timing constraints. For this purpose, Chandrasekar et al. [2014] proposed to identify the excess in process-margins to save timing constraints. Lee et al. [2015] took advantages of the large margins to improve performance.

The schemes proposed in this article differ from the above designs. We focus on relaxed restore timing constraints due to scaling. It is orthogonal to the schemes that exploits excessive margins in fabrication.

7.3. Fault Tolerance

Due to the fast scaling in memory process technology, we expect to see large process variations such that the number of weak bits tends to increase in future DRAM chips [Wang 2014; Wang et al. 2014]. Different schemes have been proposed to rescue weak cells. Sparing is one commonly used approach to disconnect the abnormal cells and connect to the spare ones [Jacob et al. 2007]. Another widely approach is to tolerate weak cells using ECC, which is implemented by placing one extra ECC chip onto the commodity DIMMs to handle errors (mainly transient soft errors) [Schechter et al.

[7]One memory cycle is enough because only one-eighth of the table is being looked up for each access, whereas the total size of the remap table is hundreds of KBs.

2010]. Reviriego et al. [2010] provided two models to enable quick evaluation of mean time to failure of single event upsets (SEUs). Maestro et al. [2011] exploited the locality of errors within an multiple cell upset to correct most double errors. Both Sparing and ECC can be used to cover slow cells, but Sparing must be reserved for higher priority repairs (e.g., defects and retention failures) and using ECC to tolerate faulty cells sacrifice soft error resilience and neither is able to work efficiently under the case of high bit error rate because of the pricy storage and latency overhead [Schechter et al. 2010].

Our experiments show that Sparing and ECC offer little help to mitigate the restore time variations. The schemes proposed in this article are designed to alleviate the performance degradation caused by slow tWR.

7.4. Wafer/Die Matching

Carefully matching wafers and dies during fabrication can effectively improve yield. Smith et al. [2007], Verbree et al. [2010], and Taoul et al. [2015] investigated wafer matching in 3D Stacked IC to avoid stacking of good dies on bad dies and further to improve the compound yield. As for conventional 2D DRAM, Wang et al. [2015] proposed to integrate *compatible* devices to alleviate refresh overhead. A drawback of the scheme is its up to $O(N^3)$ time complexity. The rank construction scheme in this article is linear to the number of candidate chips, that is, $O(N)$ time complexity.

8. CONCLUSION

In this work, we studied DRAM scaling effects on restore time and showed that future DRAM chips need relaxed timing constraints to maintain high yield rates and to keep the manufacturing cost low. Existing approaches are ineffective to address the performance losses. We proposed schemes to expose restore time variations at chunk level and devised architectural enhancements to enable find-grained variation-aware scheduling. We then proposed restore-time-aware rank construction and criticality-aware page allocation schemes to make better use of fast chunks. We evaluated the proposed schemes with the experimental results showing that our schemes achieve as high as 25% average performance improvement over traditional solutions.

ACKNOWLEDGMENTS

The authors thank Bo Zhao and Santiago Bock for providing process variation models and simulation infrastructure, respectively. We also thank the reviewers for their valuable comments.

REFERENCES

Aditya Agrawal, Amin Ansari, and Josep Torrellas. 2014. Mosaic: Exploiting the spatial locality of process variation to reduce refresh energy in on-chip eDRAM modules. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*. 84–95.

Jung Ho Ahn, Norman P. Jouppi, Christos Kozyrakis, Jacob Leverich, and Robert S. Schreiber. 2009. Future scaling of processor-memory interfaces. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–12.

Jung Ho Ahn, Jacob Leverich, Robert S. Schreiber, and Norman P. Jouppi. 2008. Multicore DIMM: An energy efficient memory module with independently controlled DRAMs. *IEEE Comput. Architect. Lett.* 8, 1 (January–June 2009), 5–8.

Marjan Asadinia, Mohammad Arjomand, and Hamid Sabrazi Azad. 2015. Prolonging lifetime of PCM-based main memories through on-demand page pairing. *ACM Trans. Des. Autom. Electron. Syst.* 20, 2 (Feb. 2015), 1–24.

Raid Ayoub, Rajib Nath, and Tajana Simunic Rosing. 2013. CoMETC: Coordinated management of energy/thermal/cooling in servers. *ACM Trans. Des. Autom. Electron. Syst.* 19, 1 (December 2013), 1–28.

Abhishek Bhattacharjee and Margaret Martonosi. 2009. Thread criticality predictors for dynamic performance, power and resource management in chip multiprocessors. In *Proceedings of International Symposium on Computer Architecture (ISCA)*. 290–301.

K. A. Bowman, S. G. Duvall, and J. D. Meindl. 2002. Impact of die-to-die and within-die parameter fluctuations on the meximum clock frequency distribution for gigascale integration. *J. Solid-State Circ.* 37, 2 (Feb. 2002), 183–190.

Karthik Chandrasekar, Sven Goossens, Christian Weis, Martijin Koedam, Benny Akesson, Norbert Wehn, and Kees Goossens. 2014. Exploiting expendable process-margins in DRAMs for run-time performance optimization. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 1–6.

Richard Goering. 2014. *TSMC Technology Symposim: Full Speed Ahead for 16nm FinFET Plus, 10nm, and 7nm*. Technical Report. TSMC.

Hynix. 2010. 2Gb DDR3 SDRAM Data Sheet. Hynix Semiconductor. Retrieved from https://www.skhynix.com/product/filedata/fileDownload.do?seq=2390.

ITRS. 2012. International Technology Roadmap for Semiconductors (ITRS) Report. Retrieved from http://www.itrs.net.

B. Liu J. Cong, W. Jiang and Y. Zou. 2011. Automatic memory partitioning and scheduling for throughput and power optimization. *ACM Trans. Des. Autom. Electron. Syst.* 16, 2 (Mar. 2011), 1–25.

Bruce Jacob, Spencer Ng, and David Wang. 2007. *Memory Systems: Cache, DRAM, Disk*. Morgan Kaufmann, San Francisco, CA.

JEDEC. 2009. Definition of the SSTE32882 Registering Clock Driver. Retrieved from https://www.jedec.org/standards-documents/docs/jesd82-29a.

Uksong Kang, Hak soo Yu, Churoo Park, Hongzhong Zheng, John Halbert, Kuljit Bains, SeongJin Jang, and Joo Sun Choi. 2014. Co-architecting controllers and DRAM to enhance DRAM process scaling. In *The Memory Forum*. 1–4.

Tanay Karnik, Shekhar Borkar, and Vivek De. 2004. Statistical design for variation tolerance: Key to continued Moore's law. In *Proceedings of the International Conference on Integrated Circuit Design and Technology*. 175–176.

Joohee Kim and Marios C. Papaefthymiou. 2001. Block-based multi-period refresh for energy efficient dynamic memory. In *Proceedings of the IEEE International ASIC/SOC Conference*. 193–197.

Kinam Kim. 2005. Technology for sub-50nm DRAM and NAND flash manufacturing. In *Proceedings of the International Electron Devices Meeting*. 323–326.

K. Kim and J. Lee. 2009. A new investigation of data retention time in truly nanoscaled DRAMs. *IEEE Electron Device Lett.* 30, 8 (August 2009), 846–848.

Yoongu Kim, Vivek Seshadri, Donghyuk Lee, Jamie Liu, and Onur Mutlu. 2012. A case for exploiting subarray-level parallelism (SALP) in DRAM. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*. 368–379.

Toshiaki Kirihata, Yohji Watanabe, Hing Wong, John K. DeBrosse, Munehiro Yoshida, Daisuke Kato, Shuso Fujii, Matthew R. Wordeman, Peter Poechmueller, Stephen A. Parke, and Yoshiaki Asso. 1996. Fault-tolerant designs for 256 Mb DRAM. *IEEE J. Solid-State Circ.* 31, 4 (Apr. 1996), 558–566.

W. Kong, P. C. Parries, G. Wang, and S. S. Iyer. 2008. Analysis of retention time distribution of embedded DRAM – a new method to characterize across-chip threshold voltage variation. In *Proceedings of the International Test Conference*. 1–7.

Israel Koren and C. Mani Krishna. 2010. *Fault-Tolerant Systems*. Morgan Kaufmann, San Francisco, CA.

Donghee Lee, Jongmoo Choi, Jong-Hun Kim, Sam H. Noh, Sang Lyul Min, Yookun Cho, and Chong Sang Kim. 2001. LRFU: A spectrum of policies that subsumes the least recently used and least frequently used policies. *IEEE Trans. Comput.* 50, 12 (Dec. 2001), 1352–1361.

Donghyuk Lee, Yoongu Kim, Gennady Pekhimenko, Samira Khan, Vivek Seshadri, Kevin Chang, and Onur Mutlu. 2015. Adaptive-latency DRAM: Optimizing DRAM timing for the common-case. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*. 489–501.

Donghyuk Lee, Yoongu Kim, Vivek Seshadri, Jamie Liu, Lavanya Subramanian, and Onur Mutlu. 2013. Tiered-latency DRAM: A low latency and low cost DRAM architecture. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*. 615–626.

Xiaoyao Liang, Ramon Canal, Gu-Yeon Wei, and David Brooks. 2007. Process variation tolerant 3T1D-based cache architectures. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*. 15–26.

J. Liu, B. Jaiyen, Y. Kim, C. Wilkerson, and O. Mutlu. 2013. An experimental study of data retention behavior in modern DRAM devices: Implications for retention time profiling mechanisms. In *ISCA*. 60–71.

Jamie Liu, Ben Jaiyen, Richard Veras, and Onur Mutlu. 2012. RAIDR: Retention-aware intelligent DRAM refresh. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*. 1–12.

Y. Luo, S. Govindan, B. Sharma, M. Santaniello, J. Meza, A. Kansal, J. Liu, B. Khessib, K. Vaid, and O. Mutlu. 2014. Characterizing application memory error vulnerability to optimize datacenter cost via heterogeneous-reliability memory. In *Proceedings of Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 467–478.

J. A. Maestro, P. Reviriego, S. Baeg, S. Wen, and R. Wong. 2011. Mitigating the effects of large multiple cell upsets (*MCU*) in memories. *ACM Trans. Des. Autom. Electron. Syst.* 16, 4 (October 2011), 1–10.

J. A. Mandelman, R. H. Dennard, G. B. Bronner, J. K. DeBrosse, R. Divakaruni, Y. Li, and C. J. Radens. 2002. Challenges and future directions for the scaling of dynamic random-access memory (DRAM). *IBM J. Res. Dev.* 46, 2.3 (March 2002), 187–212.

Mike Mayberry. 2011. *Enabling Breakthroughs in Technology*. Technical Report. Intel.

Micron. 2008. TN-46-14: Hardware tips for point-to-point system design: Termination, layout, and routing. Technical Note. (June 2008).

Micron. 2009. TN-04-55: DRAM Module Form Factors. Retrieved from https://www.micron.com/~/media/documents/products/.../dram-modules/tn0455.pdf.

W. Mueller, G. Aichmayr, W. Bergner, E. Erben, T. Hecht, C. Kapteyn, A. Kersch, S. Kudelka, F. Lau, J. Luetzen, A. Orth, J. Nuetzel, T. Schloesser, A. Scholz, U. Schroeder, A. Sieck, A. Spitzer, M. Strasser, P.-F. Wang, S. Wege, and R. Weis. 2005. Challenges for the DRAM cell scaling to 40nm. In *Proceedings of the International Electron Devices Meeting*. 336–339.

Taku Ohsawu, Koji Kai, and Kazuaki Murakami. 1998. Optimizing the DRAM refresh count for merged DRAM/logic LSIs. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*. 82–87.

O. Ozturk and M. Kandemir. 2008. ILP-based energy minimization techniques for banked memories. *ACM Trans. Des. Autom. Electron. Syst.* 13, 2 (July 2008), 1–40.

Luiz E. Ramos, Eugene Gorbatov, and Richard Bianchini. 2011. Page placement in hybrid memory systems. In *Proceedings of the International Conference on Supercomputing*. 85–95.

P. Reviriego, J. A. Maestro, and C. J. Bleakley. 2010. Reliability analysis of memories protected with BICS and a per-word parity bit. *ACM Trans. Des. Autom. Electron. Syst.* 15, 2 (February 2010), 1–15.

Samsung. 2001. What is tWR. Retrieved from http://www.samsung.com/global/business/semiconductor/file/product/tWR-0.pdf.

Samsung. 2013. Strong 14nm FinFET logic Process and Design Infrastructure for Advanced Mobile SOC Applications. Retrieved from http://www.samsung.com/global/business/semiconductor/file/media/Samsung_Foundry_14nm_FinFET-0.pdf.

S. R. Sarangi, B. Greskamp, R. Teodorescu, J. Nakano, A. Tiwari, and J. Torrellas. 2008. VARIUS: A model of process variation and resulting timing errors for microarchitects. *IEEE Trans. Semicond. Manufact.* 21, 1 (February 2008), 3–13.

Stuart Schechter, Gabriel H. Loh, Karin Straus, and Doug Burger. 2010. Use ECP, not ECC, for hard failures in resistive memories. In *Proceedings of International Symposium on Computer Architecture (ISCA)*. 141–152.

Vivek Seshadri, Y. Kim, C. Fallin, D. Lee, R. Ausavarungnirun, G. Pekhimenko, Y. Luo, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry. 2013. RowClone: Fast and energy-efficient in-DRAM bulk data copy and initialization. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*. 185–197.

Wongyu Shin, Jeongmin Yang, Jungwhan Choi, and Lee-Sup Kim. 2014. NUAT: A non-uniform access time memory controller. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*. 464–475.

G. Smith, L. Smith, H. Hosali, and S. Arkalgud. 2007. Yield considerations in the choice of 3D technology. In *Proceedings of the International Symposium on Semiconductor Manufacturing*. 1–3.

Young Hoon Son, O. Seongil, Yuhwan Ro, Jae W. Lee, and Jung Ho Ahn. 2013. Reducing memory access latency with asymmetric DRAM bank organization. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*. 380–391.

Chin-Lung Su, Yi-Ting Yeh, and Cheng wen Wu. 2005. An integrated ECC and redundancy repair scheme for memory reliability enhancement. In *Proceedings of the IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*. 81–89.

M. Taoul, S. Hamdioui, and E. J. Marinissen. 2015. Yield improvement for 3D wafer-to-wafer stacked ICs using wafer matching. *ACM Trans. Des. Autom. Electron. Syst.* 20, 2 (Feb. 2015), 1–23.

Radu Teodorescu and Josep Torrellas. 2008. Variation-aware application scheduling and power management for chip multiprocessors. In *Proceedings of International Symposium on Computer Architecture (ISCA)*. 363–374.

Ravi K. Venkatesan, Stephen Herr, and Eric Rotenberg. 2006. Retention-aware placement in DRAM(RAPID): Software methods for quasi-non-volatile DRAM. In *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*. 155–165.

Jouke Verbree, Erik Jan Marinissen, Philippe Roussel, and Dimitrios Velenis. 2010. On the cost-effectiveness of matching repositories of pre-tested wafers for wafer-to-wafer 3D chip stacking. In *Proceedings of the IEEE European Test Symposium (ETS)*. 36–41.

Thomas Vogelsang. 2010. Understanding the energy consumption of dynamic random access memories. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*. 363–374.

David Wang. 2014. Replacement of A Faulty Memory Cell with A Spare Cell for A Memory Circuit. Patent. (April 2014).

Jue Wang, Xiangyu Dong, and Yuan Xie. 2014. ProactiveDRAM: A DRAM-initiated retention management scheme. In *Proceedings of the IEEE International Conference on Computer Design (ICCD)*. 22–27.

Ying Wang, Yinhe Han, Cheng Wang, Huawei Li, and Xiaowei Li. 2015. RADAR: A case for retention-aware DRAM assembly and repair in future FGR DRAM memory. In *Proceedings of the Design Automation Conference (DAC)*. 1–6.

L. Wu, R. J. Barker, M. A. Kim, and K. A. Ross. 2013. Navigating big data with high-throughput, energy-efficient data partitioning. In *Proceedings of International Symposium on Computer Architecture (ISCA)*. 249–260.

Xianwei Zhang, Youtao Zhang, Bruce R. Childers, and Jun Yang. 2015. Exploiting DRAM restore time variations in deep sub-micron scaling. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 477–482.

Bo Zhao, Yu Du, Jun Yang, and Youtao Zhang. 2013. Process variation-aware nonuniform cache management in a 3D die-stacked multicore processor. *IEEE Trans. Comput.* 62, 11 (Sep. 2013), 2252–2265.

Hongzhong Zheng, Jiang Lin, Zhao Zhang, Eugene Gorbatov, Howard David, and Zhichun Zhu. 2008. Mini-rank: Adaptive DRAM architecture for improving memory power efficiency. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*. 210–221.

Received September 2015; revised March 2016; accepted June 2016