Exploiting DRAM Restore Time Variations in Deep Sub-micron Scaling

Xianwei Zhang*, Youtao Zhang*, Bruce R. Childers* and Jun Yang†

* Computer Science Department, University of Pittsburgh
{xianeizhang,zhangyt,childers}@cs.pitt.edu
†Electrical and Computer Engineering Department, University of Pittsburgh
juy9@pitt.edu

***Abstract*—Recent studies reveal that one of the major challenges in scaling DRAM in deep sub-micron regime is its significant variations on cell restore time, which affects timing constraints such as write recovery time tWR. Adopting traditional approaches results in either low yield rate or large performance degradation. In this paper, we propose schemes to expose the variations to the architectural level. By constructing memory chunks with different accessing speeds and, in particular, exploiting the performance benefits of fast chunks, a variation-aware memory controller can effectively compensate the performance loss due to relaxed timing constraints. Our experimental results show that, comparing to traditional designs such as row sparing and ECC, the proposed schemes help to improve system performance by up to 10.3% and 12.9%, respectively, for 20nm and 14nm tech nodes on a 4-core multiprocessor system.**

I. Introduction

DRAM density has been improved dramatically in the past four decades as the technology evolves along with CMOS process scaling. However, further scaling DRAM in deep sub-micron regime faces significant process variations, which results in more cells violating the standard timing constraints, degraded chip yield rate, and increasing manufacturing cost.

One representative issue in deep sub-micron scaling is that the time required to fully charge the storage capacitor (i.e., restore time) is expected to increase significantly [1]. For smaller tech nodes, more cells need longer restore time and 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 cost low. However, naively relaxing tWR introduces large performance loss. Adopting post-fabrication cell repair designs such as row sparing and ECC to rescue weak cells helps but only to a limited extend.

In this paper, 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 can create more fast chunks and reduce the number of slow chunks, which mitigates the scaling effect and improves the overall system performance.

In summary, we make the following contributions.

- We develop a detailed model to study the scaling and process variation effects on restore time, and perform Monte Carlo simulation to study the parameters and the distribution. We also study the trend of restore time under technology scaling.
- We propose fine-grained variation-aware scheduling schemes to address restore time variations. By exposing chunk-level time constrains and constructing fast chunks, we effectively mitigate the performance degradation due to timing constraint relaxation.
- We evaluate the proposed schemes on memory access latency and performance, and demonstrate their effectiveness over traditional approaches.

II. Background and Related Work

A. DRAM Cell Restoring Operation

A DRAM-based main memory system usually consists of several dual in-line memory modules (DIMMs) while each DIMM contains one or multiple ranks. One rank is often constructed using multiple DRAM chips, e.g., 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 maximize memory performance. Servicing a memory request starts with opening the target row, and then column read or write to fetch or write data. An opened row needs to be closed, i.e., restoring data back to device cells, before the corresponding memory bank can service a new request. Therefore, a change of restore time shall affect both read and write accesses.

B. Scaling effects

DRAM exhibits non-negligible scaling effects as it scales down to 20nm, 14nm [2] and further to 10nm [3], [4], [1]. This is because it becomes increasingly difficult to precisely control fabrication process in deep sub-micron regime. One effect is that memory cells at DIMM level have a wider range of variations. 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 [5].

978-3-9815370-4-8/DATE15/©2015 EDAA

Another scaling effect is that the timing constraints need to be relaxed in order to maintain an acceptable yield rate and to keep manufacturing cost under control [1]. 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 restoring time and shorter cell retention time, respectively.

The process variation on retention time has been well studied. Kong et al. [22] from IBM observed that the variation of retention time follows log-normal distribution. Kim et al. [23] 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 [24], [25], [26], [10]. For example, RAIDR [10] classifies DRAM rows into bins according to retention time, and applies different refresh rates to save power. Recently, Argrawal et al. [11] proposed Mosaic to divide the eDRAM module into regions, and refresh each region at a different rate.

For the variation on restoring time, Kang et al. [1] pointed out that the time to charge DRAM capacitor increases with technology scaling down. They proposed sub-array level parallelism (SALP) [21] to compensate the performance loss. Our design is orthogonal to SALP.

III. Motivation

A. Modeling PV Effects on Restoring Time

To study the effects of DRAM scaling, we model PV in deep sub-micron regime and study its impact on DRAM timing. Our model models both access transistor and storage capacitor in a DRAM cell. For DRAM scaling, access transistor can be either FinFET or VCAT [5], [6], [7], both of which feature gate-surrounded channel structure to suppress I_{off} while boosting I_{on}. In this paper, FinFET is picked for our circuit model, and BSIM-CMG version of Predictive Technology Model (PTM) is utilized for simulation. The trench capacitor is assumed to have a capacitance of 27fF[8], [6] 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, etc, and those components are simulated in HSPICE. The column circuits have generic topologies [9] and their transistor parameters are taken (and scaled) from a DRAM modeling tool from Rambus [8]. 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-1 case which is typically slower than writing a 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 curve fit using log-normal distribution. Similar to recent PV studies [10], [11], we include bulk distribution to depict the normal variation that dominates the majority of cells, and tail distribution to depict random manufacturing defects (Note: not all cells following the tail distribution are treated as defects. The worst ones are covered by conventional redundant repairs [11]). Table I summarizes the parameters for bulk and tail distributions after curve fitting with our cell samples.

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

To obtain the chip maps, we use the VARIUS [12] tool to incorporate both WID and D2D process variations for generating spatial maps. We focus on WID in this paper. Similar to previous PV studies [13], [11], we assume the same share of systematic and random components, and choose a small $\phi = 0.3$ meaning that the correlation range equals to 30% of the chip's side length, as shown in Table I.

B. Performance Degradation due to Timing Relaxation

Conventionally, each timing constraint for DRAM has a single fixed value, e.g. tWR keeps at 15ns in existing DRAM standard [14]. Given that more cells in deep sub-micron are likely to violate tWR, it is beneficial to relax it to allow most such cells finish restoring operations after a destructive read operation or a write operation, which helps to preserve high chip yield. A large tWR indicates longer bank occupancy and thus degrades chip performance.

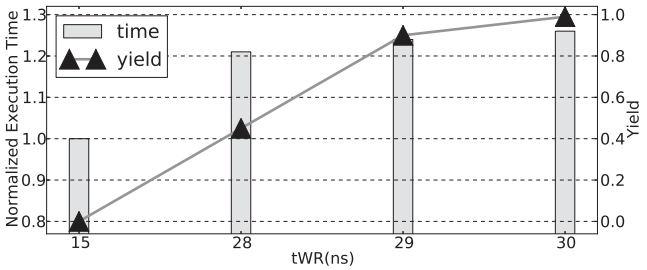


Fig. 1: Comparing performance and yield with different tWR.

Figure 1 compares performance and yield with different tWR relaxation at 20nm tech node. The scaling effect leads to that no chip can meet the existing specification, i.e., yield rate is 0% for tWR=15ns. At 20nm tech 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, e.g., 21%, can be observed when tWR is relaxed to 28ns. However, the yield is seriously lowered to 45%.

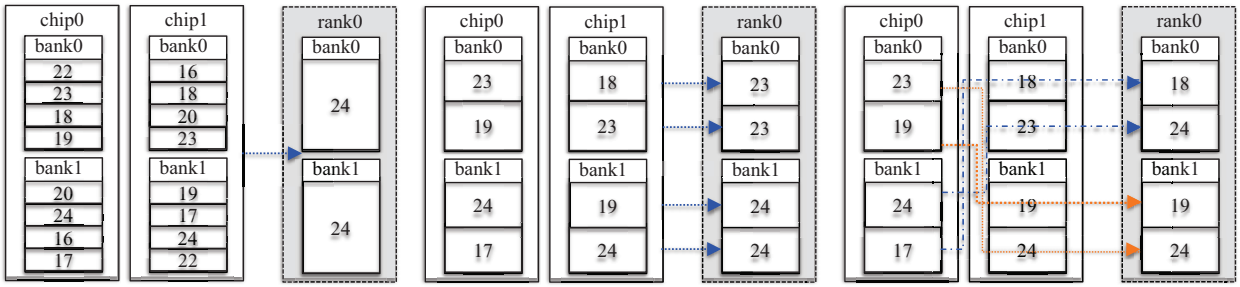
From the figure, it is challenging to achieve high chip yield while minimizing its impact on system performance.

IV. 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.

A. Chip-specific tWR Control

A simple enhancement to the approach that fixes one tWR in DRAM standard is to exploit chip variations and set different



(a) Chip-specific (b) Chunk-specific (c) Chunk-specific with remapping

Fig. 2: 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, i.e., the tWR of the weakest cell.

tWRs for different chips. For this purpose, a post-fabrication test process is performed by the manufacturer to 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 [1] that has the worst tWR of the entire DIMM (as shown in Figure 2(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.

B. Chunk-specific tWR Control

Even though tWR exhibits a wide range of variations when scaling in 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 chunks, set the chunk level tWR based on the worst chip-row within the chunk, and expose the chunk level tWR to the memory controller.

In Figure 2(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. Exposing chunk level tWR to the memory controller helps to speed up memory accesses when they fall into the fast chunks.

C. Constructing Fast Chunks through Chunk Remapping

In the previous design, a DIMM-chunk consists of multiple chip-chunks that are of the same chunk index from different chips, e.g., the 2nd DIMM-chunk is constructed from the 2nd chip-chunk from each chip. Since the chip-chunks that are of the same index exhibit significant tWR difference, it would be beneficial to form a chunk using chip-chunks that are of the same or similar tWRs.

As shown in Figure 2(c), if we form the first DIMM-chunk using the 4th chip-chunk from chip 0 and the 1st 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.

To maintain bank level parallelism, the chip-chunks from each bank are remapped as a group. In Figure 2(c), DIMM-chunk 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, i.e., 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.

D. Architectural Enhancements

To exploit chip-specific and chunk-specific restore time variations, a post-fabrication testing needs to be performed to detect restore times at either chip or chunk level. Given that cell restore time is thermal dependent — study showed that it becomes worse at low temperature [1], the manufacturer needs to record the worse ones under chip's allowed working conditions. The recorded timing constraints 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.

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.

[1] 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.

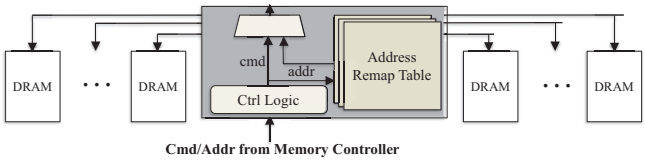


Fig. 3: The on-DIMM architectural enhancement.

To enable chunk re-organization, we need one extra chunk remapping table as shown in Figure 3. Similar as HP's MC-DIMM [15] and Mini-rank [16], our design integrates a bridge chip on-DIMM, which remaps the physical address sent from the memory controller to device row addresses in each chip. For the chunk remapping table, each entry maps the corresponding DIMM-chunk to the chip-chunk at each chip.

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

V. Experimental Setup

A. Configuration

To evaluate the effectiveness of our designs, we compared our designs to traditional repair solutions using an in-house chip-multiprocessor system simulator. We modeled a quad-core system with the parameters shown in Table II. The DRAM timing constraints follow Hynix DDR3 SDRAM data sheet [17] and are summarized in Table III. For the schemes exploiting chunk level timing constraints, we added two CPU cycles to access the timing table. For the scheme performing chunk-remapping, we added one extra DRAM cycle to access the mapping table.

TABLE II: 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	1channel, 2ranks/channel, 8banks/rank, 16K rows/bank, 8KB/row, 1066 MHz, tCK=0.935ns, width: x8

TABLE III: DRAM Timing Parameters

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

B. Workloads

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

TABLE IV: Benchmark Characteristics

Class	Workloads
spec high	429.mcf, 470.lbm, 450.sop, 433.mil, 471.omn, 459.Gem, 462.lib, 484.xal, 403.gcc, 482.sph, 410.bwa
spec med	437.les, 481.wrf, 436.cac, 434.zeu, 401.bzi, 473.ast, 447.dea, 456.hmm
spec low	400.per, 464.h264, 445.gob, 435.gro, 458.sje, 454.cal, 444.nam, 465.ton, 416.gam, 453.pov

VI. Results

We evaluated the following schemes:

— `Baseline`. The baseline sets tWR to 15ns, the same as existing DRAM products. 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 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 [9]. In our experiments, we set the spare density as high as 16 spare rows per 512-row block, which resides in the aggressive spectrum [18], [19]. Given that spares may be reserved for high-priority repairs, such as defects and retention failures, we testsed x=0, 2, 8, 16 spares out of 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 [20].

— `N-`x. This scheme implements the chunk-specific restore time control, with each bank being divided tino x chunks. Each DIMM chunk has its own timing constraints, which are exposed to the variation-aware memory controller.

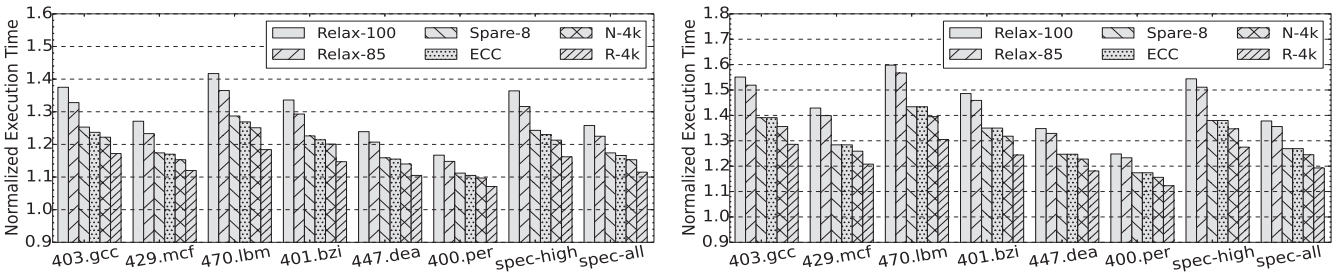
— `R-`x. This scheme implements the chunk-specific restore time control with chunk remapping, with each bank being divided into x chunks. Similar as `N-`x, the timing constraints of each chunk are exposed to the memory controller.

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

A. Impacts on Program Execution Time

Figure 4(a) compares the execution time under different schemes for 20nm technology node. The execution time is normalized to the ideal baseline. The figure reports the results from representative benchmarks, the highly memory-intensive subset (spec-high), and the full set (spec-all).

From Figure 4(a), we observed that (1) DRAM scaling has a large impact on restore time. To keep a high yield rate, the timing constraints have to be vastly relaxed from



(a) 20nm Normalized Average Execution Time

(b) 14nm Normalized Average Execution Time

Fig. 4: The execution time comparison of different scheme at 20nm and 14nm tech nodes. Representative applications and the geometric means for highly memory-intensive (spec-high) and all application (spec-all) are presented here.

16 cycles to over 30 cycles, which has a significant impact on performance. On average, `Relax-100` and `Relax-85` prolong the execution time by 25.8% and 22.5% respectively. Highly memory-intensively applications tend to have large degradation (i.e, over 30%). (2) Adding spare rows helps to mitigate performance losses: `Spare-8` is 17.4% 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) `N-4k` is 2% 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`. (5) `R-4k` achieves the best performance of all schemes because it helps to construct more fast chunks. On average, `R-4k` helps to reduce the performance loss from 25.8% in `Relax-100` to 11.5%, and 5.1% shorter compared to `N-4k` for spec-high.

Figure 4(b) shows the normalized execution time for 14nm tech node. From the figure, the performance loss from the ideal goes up as the tech node scales down. For example, `Relax-100` exhibits a 26% loss at 20nm node while a 38% loss at 14nm node. More slow cells are observed at 20nm, which weakens both ECC and Sparing effect, and narrow the gap between them. `ECC` and `Spare-8` have the same results because they have the same tWR after rescuing weak cells. In general, highly memory intensive applications, e.g., $400.per$, show large losses. Due to the fact that more memory cells violating timing constraints, it becomes increasingly difficult to mitigate performance loss at small tech nodes. However, `R-4k` shows a tendency of better improvement over others (e.g., `N-4k`) at 14nm tech node. For spec-high, `R-4k` reports a 11.5% shorter execution time compared to `Spare-8` and `ECC` (for $470.lbm$, the value is as high as 12.9%.).

B. Impacts on Memory Access Latency

Figure 5 compares the average memory access latencies under different schemes. Among these schemes, `R-4K` achieves the lowest latency, which is 6% lower than `N-4K`, 8.9% lower than `Spare-8`, and 18.8% lower than `Relax-100` for 20nm tech node. There is a clear latency increase for 14nm tech node, e.g., the average memory access latency of `R-4K` increased from 328ns to 367ns. This is mainly due to further relaxed timing constraints. In addition, `R-4K` achieves better improvement over other scheme at small tech node — the access latency of `N-4K` is 7.5%, 10.5%, and 21.6% lower than `N-4k`, `Spare-8` and `Relax-100`, respectively.

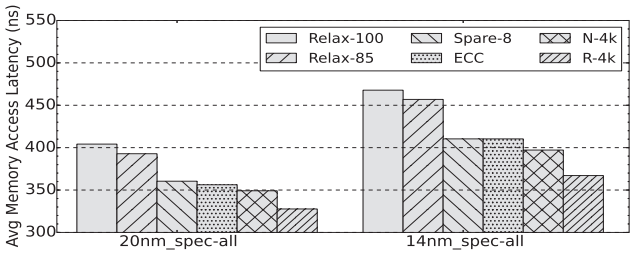


Fig. 5: Comparing memory latencies under different schemes.

C. Sensitivity Study

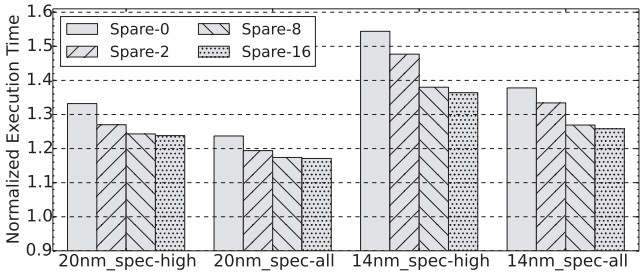
1) Varying Sparing Levels: Figure 6(a) compares the performance with different spare rows in each 512-row block. While better performance is observed with more spares, the improvement diminishes after using 8 spare rows. For example, the difference between `Spare-8` and `Spare-16` is only 0.3% at 20nm 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.

Note that `Spare-0` is different from `Relax-100` — while `Spare-0` places no spares, and use the chip-specific tWR, `Relax-100` sets tWR with respect to ensure all chips can work . Thus, `Spare-0` usually has a lower tWR.

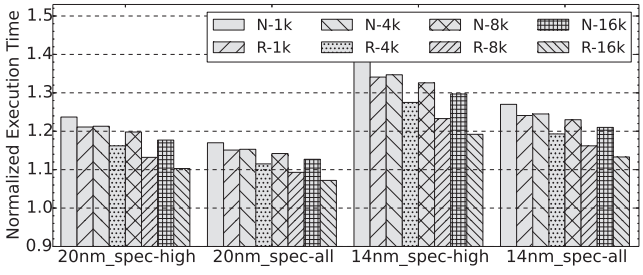
2) Varying the Number of Chunks: Figure 6(b) compares the performance with different numbers of chunks for chunk-specific restore time scheduling. From the figure, using a small number of chunks (e.g., 1024 chunks) fails to mitigate the performance loss effectively: for 20nm, `N-1k` is even worse than `ECC`. The performance improves with more chunks either with or without chunk reorganization. This is because splitting a slow chunk often leads to one slightly faster chunk, so that the overall memory access latency can be improved. The performance advantage of using chunk reorganization become bigger at smaller tech node.

D. Storage 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 V summarizes the storage overhead with different chunks.



(a) Sensitivity on Sparing Levels



(b) Sensitivity on Number of Chunks

Fig. 6: Sensitivity study using different spares and chunks.

TABLE V: Tested Chunk Configurations

#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

From Table V, `R-4k` requires 896KB DIMM storage to save the chunk mapping. The table is evenly divided among the banks in the DIMMs, i.e., 56KB per bank. We used CACTI 5.3 to model the table as a direct-mapped cache with 8B line size. For 32nm [2] technology, it has 0.348ns access latency, $0.229mm^2$ area overhead, 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 overehead, 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.

Based on the latency overheads calculated by CACTI, we charged 1 memory cycle and 2 CPU cycles to access the chunk mapping table and the timing table, respectively. We observed less than 1% performance overhead on average.

VII. 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 evaluated the proposed schemes with the experimental results showing that our schemes achieve up to 12.9% performance improvement over traditional solutions.

[2]Different from sub-20nm DRAM devices, caches are constructed with 32nm process technology, which is supported by CACTI tool.

Acknowledgment

The authors would like to thank Bo Zhao and Santiago Bock for providing process variation models and simulation infrastructure, respectively. We also thank the reviewers for their comments. This work was supported in part by NSF CCF-1422331 and CNS-1012070.

References

[1] "Co-architecting controllers and dram to enhance dram process scaling," in *The Memory Forum*, 2014.
[2] Samsung, "Strong 14nm finfet logic process and design infrastructure for advanced mobile soc applications," http://www.samsung.com/global/business/semiconductor/file/media/Samsung_Foundry_14nm_FinFET-0.pdf, 2013.
[3] R. Goering, "2014 tsmc technology symposium: Full speed ahead for 16nm finfet plus, 10nm, and 7nm," TSMC, Tech. Rep., April 2014.
[4] M. Mayberry, "Enabling breakthroughs in technology," Intel, Tech. Rep., June 2011.
[5] International Technology Roadmap for Semiconductors for Semiconductors, "Itrs report, 2012 edition," http://www.itrs.net.
[6] W. Mueller, G. Aichmayr, *et al.*, "Challenges for the dram cell scaling to 40nm," in *International Electron Devices Meeting*, 2005.
[7] K. Kim, "Technology for sub-50nm dram and nand flash manufacturing," in *International Electron Devices Meeting*, 2005.
[8] T. Vogelsang, "Understanding the energy consumption of dynamic random access memories," in *MICRO*, 2010.
[9] B. Jacob, S. Ng, *et al.*, "Memory systems: Cache, dram, disk," in *Morgan Kaufmann*, 2007.
[10] J. Liu, B. Jaiyen, *et al.*, "Raidr: Retention-aware intelligent dram refresh," in *ISCA*, 2012.
[11] A. Agrawal, A. Ansari, *et al.*, "Mosaic: Exploiting the spatial locality of process variation to reduce refresh energy in on-chip edram modules," in *ISCA*, 2014.
[12] S. R. Sarangi, B. Greskamp, *et al.*, "Varius: A model of process variation and resulting timing errors for microarchitects," *IEEE Transactions on Semiconductor Manufacturing*, 21(1):3–13, 2008.
[13] T. Karnik, S. Borkar, *et al.*, "Probalistic and variation-tolerant design: Key to continued moore's law," in *TAU Workshop*, 2004.
[14] Samsung, "What is twr." [Online]. Available: http://www.samsung.com/global/business/semiconductor/file/product/tWR-0.pdf
[15] J. H. Ahn, N. P. Jouppi, *et al.*, "Future scaling of processor-memory interfaces," in *International Conference for High Performance Computing, Networking, Storage and Analysis*, 2009.
[16] H. Zheng, J. Lin, *et al.*, "Mini-rank: Adaptive dram architecture for improving memory power efficiency," in *MICRO*, 2008.
[17] "2gb ddr3 sdram data sheet," Hynix Semiconductor, 2010.
[18] T. Kirihata, Y. Watanabe, *et al.*, "Fault-tolerant designs for 256 mb dram," *IEEE Journal of Solid-State Circuits*, 31(4):558–566, 1996.
[19] I. Koren and C. M. Krishna, *Fault-Tolerant Systems*. Morgan Kaufmann, 2010.
[20] C.-L. Su, Y.-T. Yeh, *et al.*, "An integrated ecc and redundancy repair scheme for memory reliability enhancement," in *IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, 2005.
[21] Y. Kim, V. Seshadri*et al.*, "A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM," in *ISCA*, 2012.
[22] W. Kong, P. Parries, *et al.*, "Analysis of Retention Time Distribution of Embedded DRAM - A New Method to Characterize Across-Chip Threshold Voltage Variation," in *International Test Conference*, 2008.
[23] K. Kim, J. Lee, "A New Investigation of Data Retention Time in Truly Nanoscaled DRAMs," in *IEEE Electron Device Letters*, 30(8):846–848, 2009.
[24] J. Kim, M. C. Papaefthymiou, "Block-based Multi-period Refresh for Energy Efficient Dynamic Memory," in *IEEE International ASIC/SOC Conference*, 2001.
[25] T. Ohsawa, K. Kai, *et al.*, "Optimizaing the DRAM Refresh Count for Merged DRAM/Logilsis," in *ISLPED*, 1998.
[26] R. K. Venkatesan, S. Herr, *et al.*, "Retention-aware Placement in DRAM (rapid): Software Methods for Quasi-non-volatile DRAM," in *HPCA*, 2006.