Detecting and Mitigating Data-Dependent DRAM Failures by Exploiting Current Memory Content (Micro'17)

Samira Khan, Chris Wilkerson, Zhe Wang,
Alaa R. Alameldeen, Donghyuk Lee, Onur Mutlu

Dec 14, 2017 @ Google Japan

Soramichi Akiyama

Artificial Intelligence Research Center,

National Institute of Advanced Industrial Science and Technology (AIST), Japan

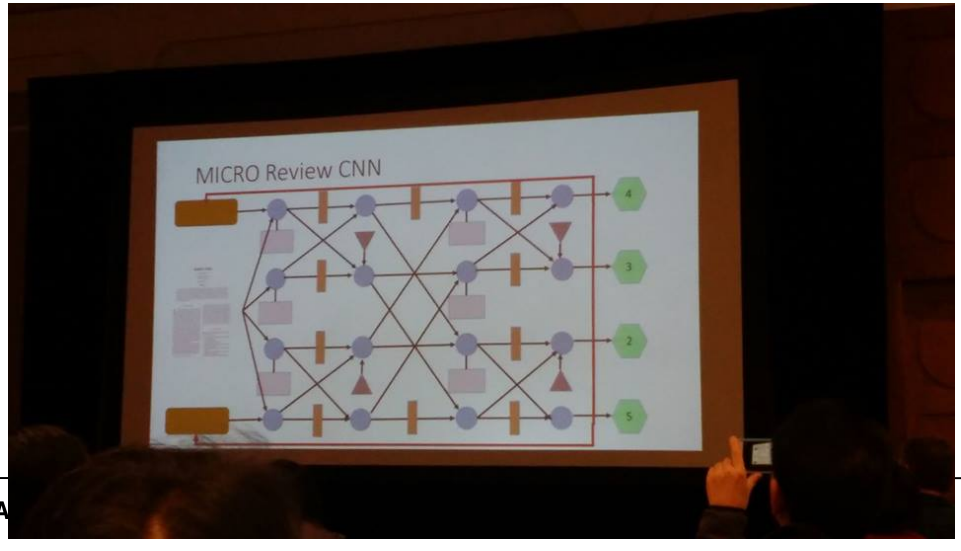
s.akiyama@aist.go.jp

DRAM-related papers from the same group

- "Low-Cost Inter-Linked Subarrays (LISA): Enabling fast inter-subarray data movement in DRAM", HPCA'16
  - Efficient data-movement between sub-arrays inside DRAM
- "ChargeCache: Reducing DRAM Latency by Exploiting Row Access Locality", HPCA'16
  - Reduce memory latency by shortening cell charge time
- "Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology", Micro'17
  - In-memory bitwise operation for DRAM
- "SoftMC: A Flexible and Practical Open-Sourced Infrastructure for Enabling Experimental DRAM Studies", HPCA'17
  - Open-sourced DRAM controller on an FPGA

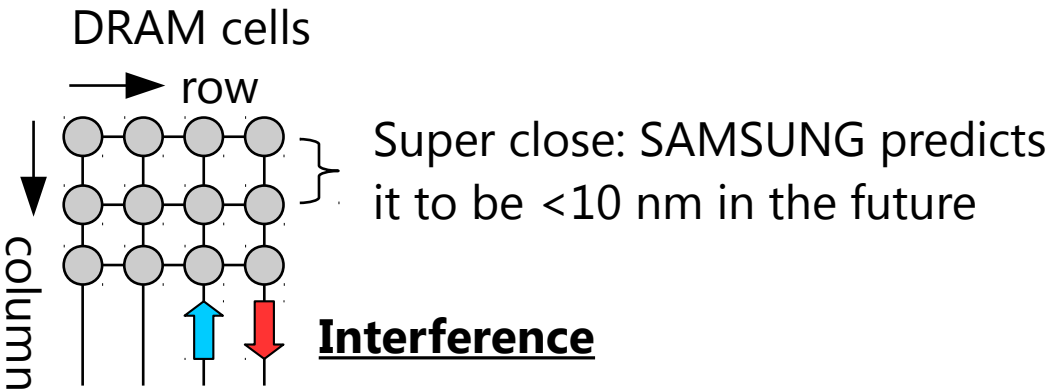
About Micro'17

- The **50th** Annual IEEE/ACM International Symposium on Microarchitecture, 2017
  - A top conference in computer architecture (along with ISCA, HPCA, ASPLOS)
  - **Sessions**: DRAM, Accelerators, GPUs-1, Non-Volatile Memory/Storage, In/Near Memory Computing, Security, Deep Learning, Prediction, Consistency/Coherency Translation, Energy, GPUs-2, OS and System Design, Unconventional Architectures, Compilers and Microarch.
- Number of Attendees: 350?
- Micro'18 will be in Fukuoka



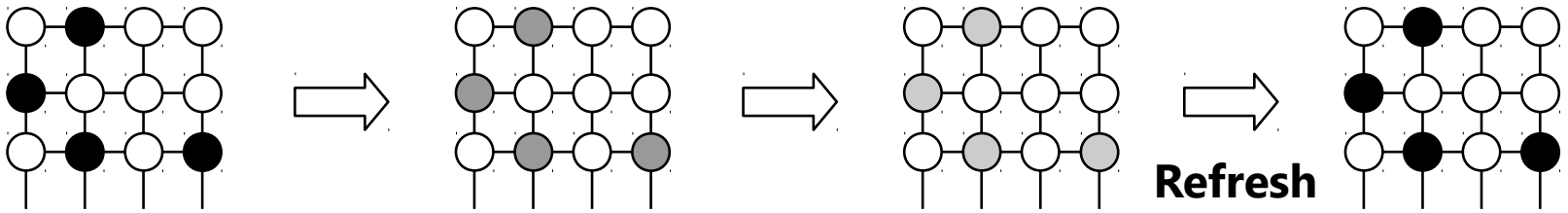
Background (1/2)

- Demand for more and more memory (DRAM)
  - Big data analysis, AI, ...
- Increased DRAM capacity → Higher density of DRAM cells
  - Capacity of 1 DIMM module: 1GB (2006?)→ 32GB (2016)
  - Same size → 32x density
- DRAM cells (capacitors) are super close to each other
  - Increased interference from neighboring cells



Background (2/2)

- More aggressive refresh to reduce interference
  - Refresh: key to keep DRAM contents consistent
  - Refresh interval: 64 ms (now) → 16 ms (predicted)



→ Higher overhead (performance, energy)

Trade-off related to Refresh interval

Refresh interval	Short	Long	Ideal
Interference	Low	High	Low
Overhead	High	Low	Low

Detecting and mitigating DRAM failures (bit flips) caused by interference is the key to achieve higher capacity with low overhead

Characteristics of Failures

1. Failures are data-dependent

- Interference comes from neighboring cells → Different data, different inference level

2. Failures are cell-dependent

- Due to variations in the manufacturing process
- "Cell X fails with a data pattern and a refresh rate" does not imply "Every cell fails with the same pattern and the same refresh rate"

  * This is not explicitly written, but assumed throughout the paper

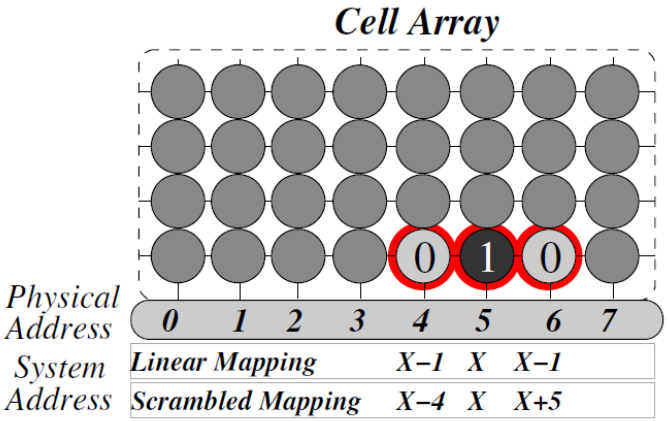
Room for optimization: some cells can work well even with shorter refresh intervals

Detecting Failures

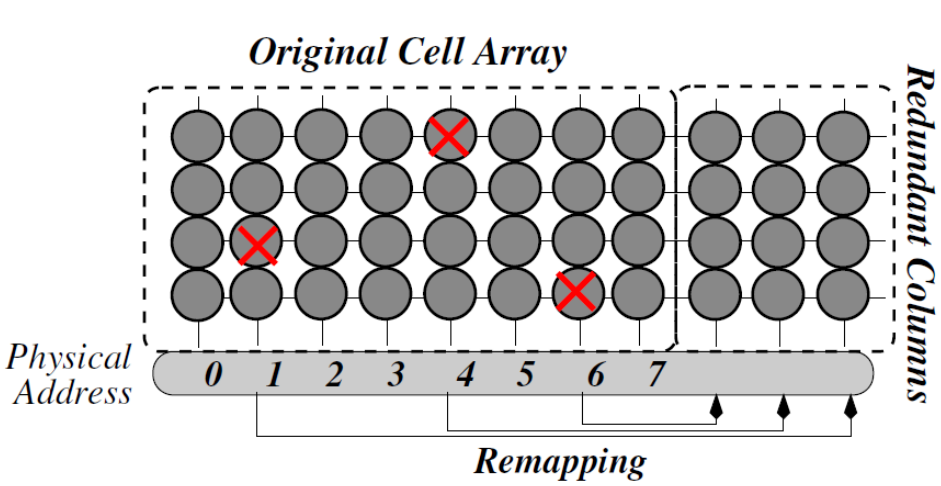
- Detecting faulty cells is the key
- Manufacturer testing
  - Exhaustively test all cells before the module is shipped
  - Not applicable to aggressive usages (e.g. shorten refresh interval)
- Online system-level testing
  - Detect all possible cells that are susceptible to failures for all possible contents at boot-time
  - Enables aggressive optimizations after DRAMs are shipped
  - Challenges in practice (next slide)

Challenges of Online Testing

1. Internal memory addresses are hidden and scrambled
→ Neighboring cells in the system address space are not real neighbors



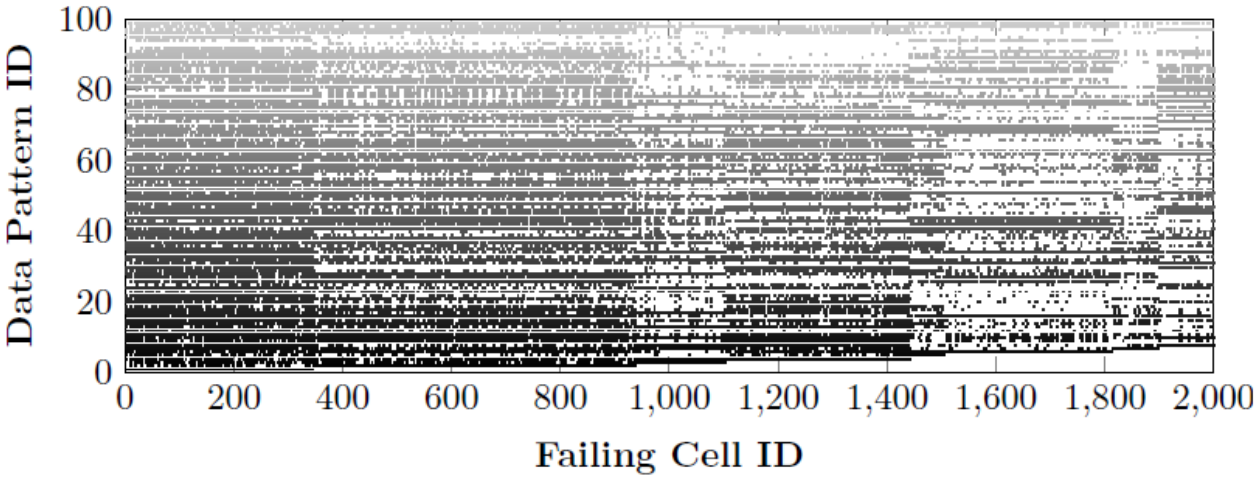
2. Faulty columns are remapped to a redundant column
→ Neighboring relationship may change time to time



Testing data-dependent failures online (w/o the proprietary hardware specification) is very challenging

Key Observations

- Detecting every-possible failures is a overkill
  - We only need possible failures with the current memory content



Failures are highly data-pattern dependent

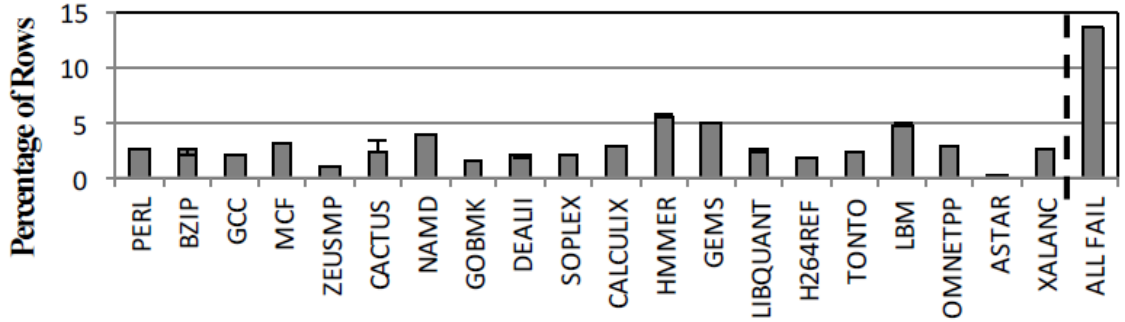


Figure 4: Percentage of rows that exhibit failures

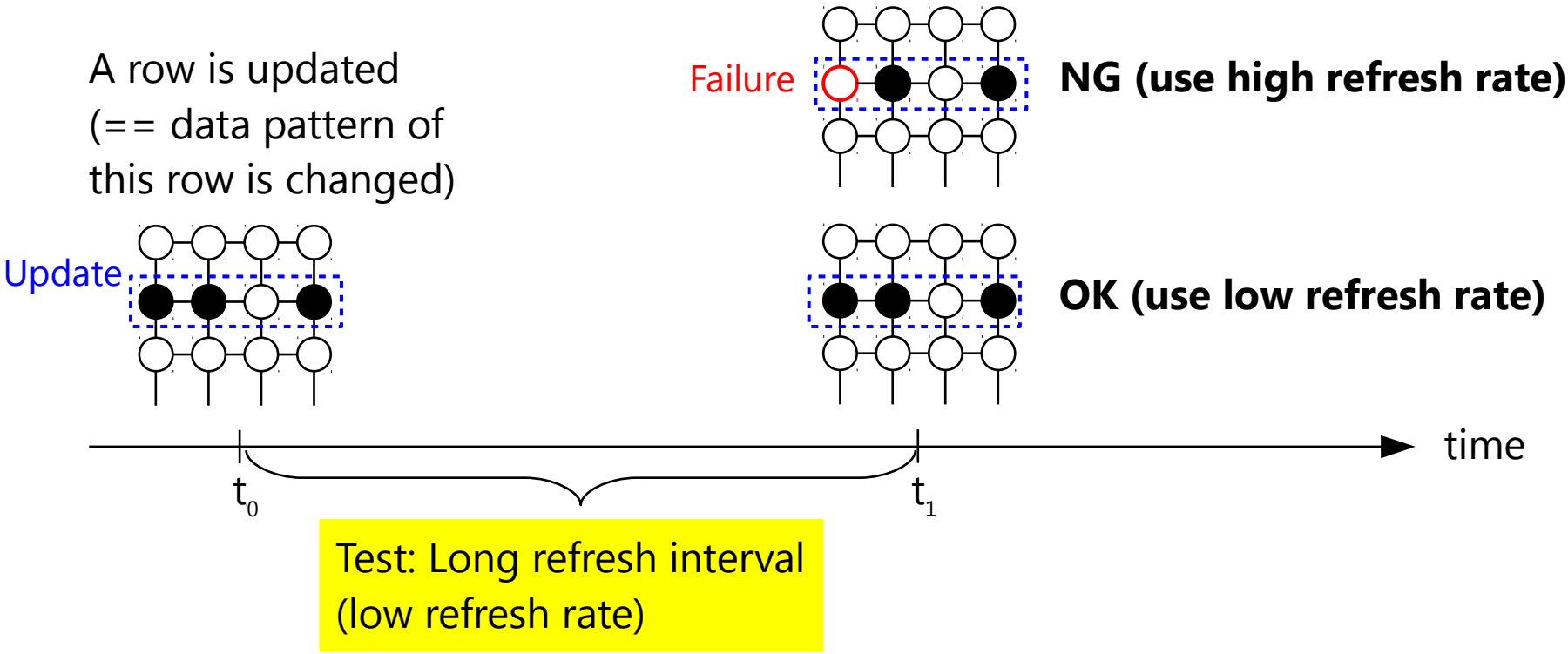
# of failed <u>rows</u> with the current content is 2.4X – 35.2X smaller than every possible data-dependent failure ("ALL FAIL")

* couldn't see how to do it. # of any possible data pattern for a row would be 2^8K (a row == 8K cells)

MEMCON: Design

- MEMCON: tests one row for the current memory content
  - Row update → Test the row w/ the new content & low refresh rate

    1. If no failures happen → refresh the row at the low refresh rate
    2. If failures do happen → refresh the row at the high refresh rate



MEMCON: Challenges

- A row cannot be accessed during a test
  - The content must be temporarily copied to a different region
  - to keep serving memory requests to that row
- Copy to a different region → extra read and write requests
  1. Copy the tested row into the temporal region
  2. Copy the tested row again to the temp region after the test

→ Extra memory bandwidth, interfere with critical program accesses

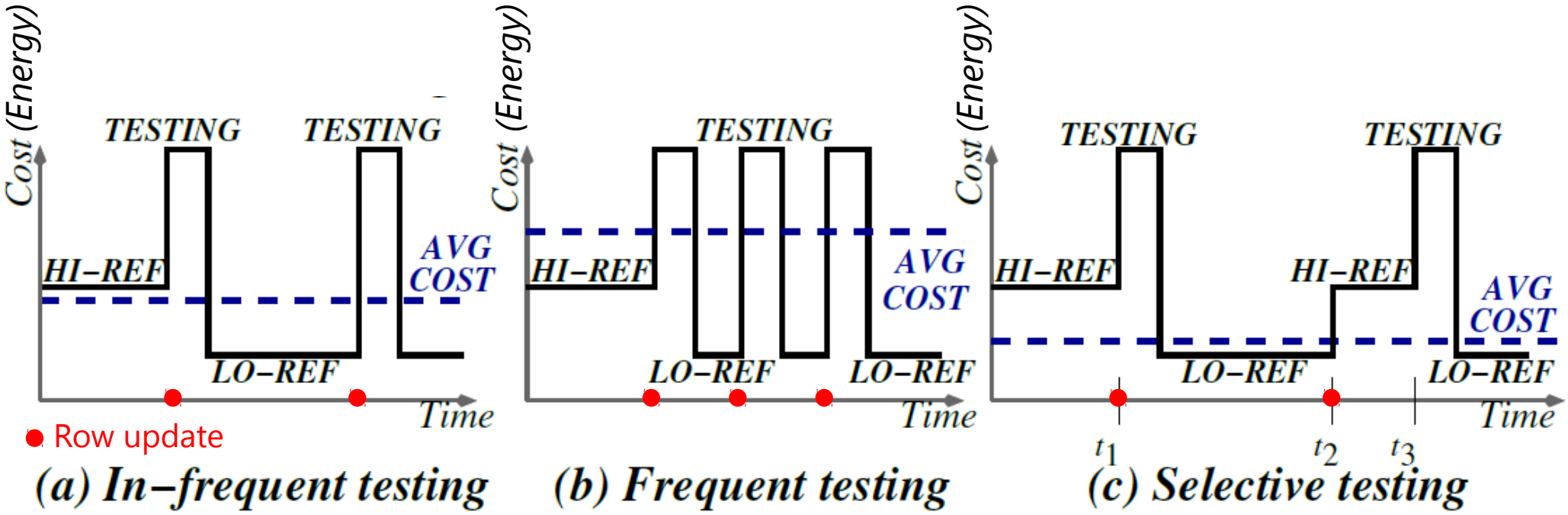
Design Challenge: How to minimize the overhead of testing?

Cost-Benefit Analysis (1/3)

- Overhead: Increased latency/energy due to extra copies

Option 1:
Test immediately after a write request
→ Cost vs. Benefit trade-off depends on the write frequency (not adjustable)

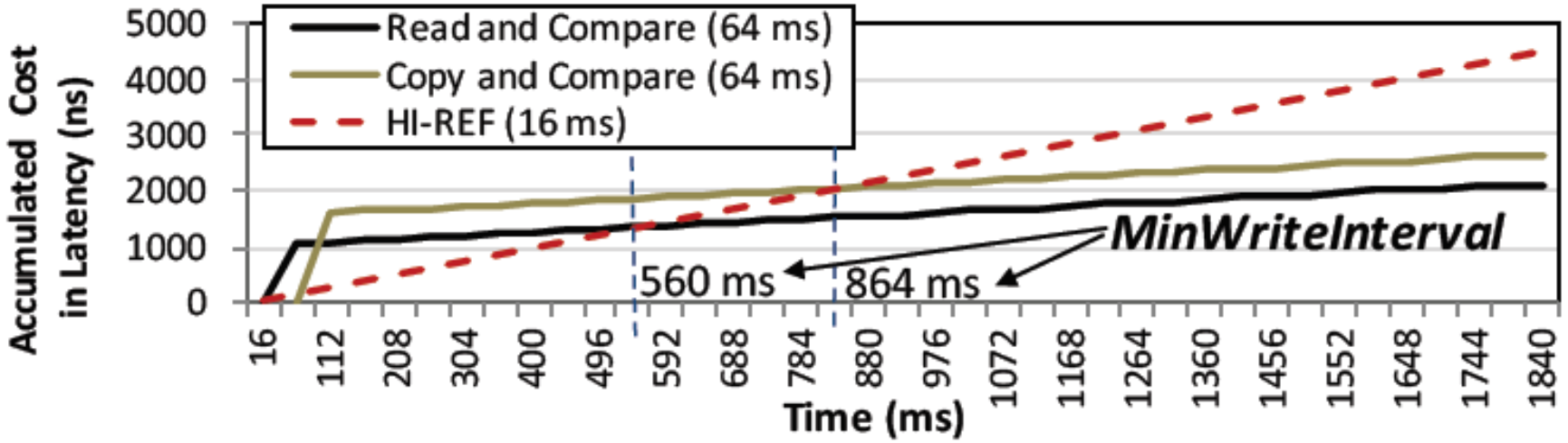
Option 2:
Selective testing → Cost vs. Benefit trade-off is adjustable



Cost-Benefit Analysis (2/3)

- How to selectively test? When should the test overhead should paid?
  - Cost with high refresh rate: $C_H(t) = \alpha t$
  - Cost with test + low refresh rate: $C_L(t) = C_T + \beta t$

$(\alpha > \beta)$



- Expr w/ real parameters: MinWriteInterval = 560ms, 864 ms

Predicting when the next write will come is the key

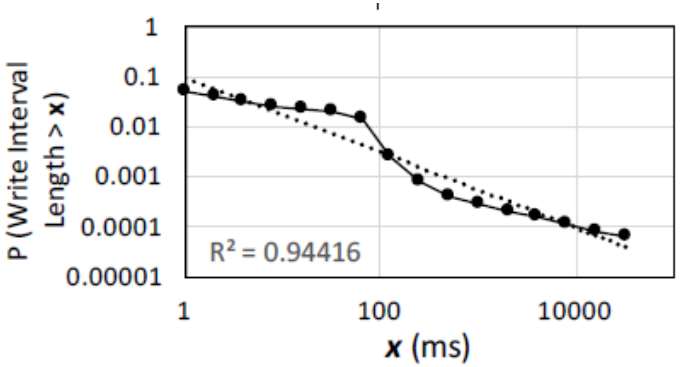
Cost-Benefit Analysis (3/3)

- How C_T is estimated from real values
- "Read and Compare" implementation
  - Copies the row to the memory controller and keep the original row idle for the test period
  - C_T = 1068 ns
  - cannot buffer many rows (due to space limitation)
- "Copy and Compare" mode
  - Copies the row to a redundant row and keep the original row idle for the test period
  - C_T == 1602 ns
  - can buffer many rows

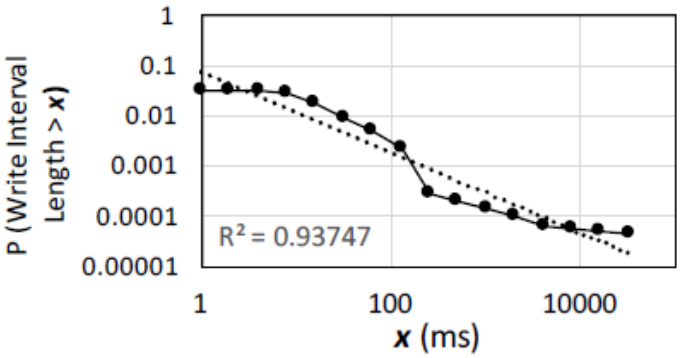
Write Interval Prediction (1/2)

- How to predict write interval for a given write?

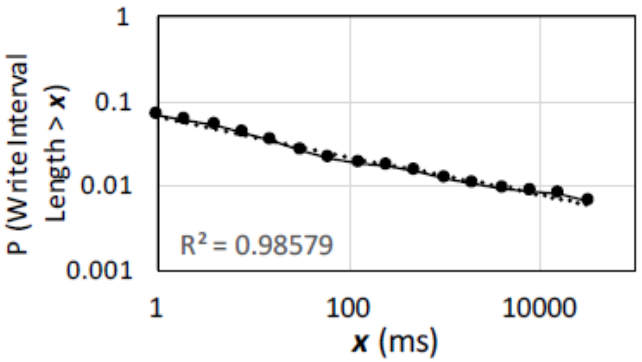
Traced write intervals of real applications using a FPGA-based memory controller



(a) *ACBrotherhood*



(b) *Netflix*



(c) *SystemMgt*

→ Write intervals of real apps obey Pareto distribution

Write Interval Prediction (2/2)

- Pareto distribution: decreasing hazard rate (DHR) property
  - longer a page is not written to, longer it is expected to remain idle

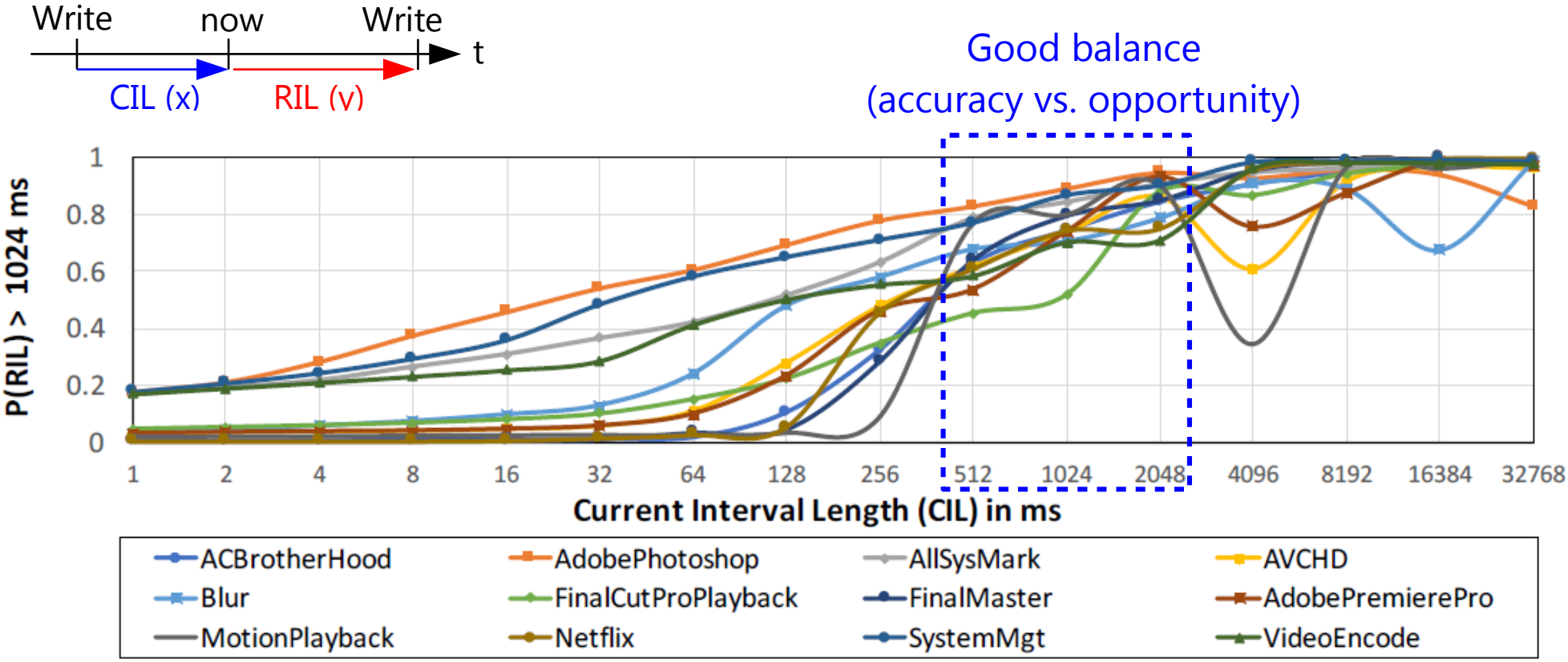


Figure 11: Probability that *RIL* is greater than 1024 ms, as a function of *CIL*

Evaluation Results

- Baseline: Always refresh with 16 ms interval (no failures occur)
- Upper bound: Always refresh with 64 ms interval (no failure mitigation)
- Evaluation based on Ramulator and memory trace using FPGA

Application	Type	Time (s)	Mem (GB)	Threads
ACBrotherHood	Game	209.1	2.8	8
AdobePhotoshop	Photo editing	149.2	3.0	4
AllSysMark	Media creation	2064	3.4	4
AVCHD	Video playback	217.3	5.2	2
BlurMotion	Image processing	93.4	0.2	2
FinalCutPro	Video editing	76.9	3.0	2
FinalMaster	Movie display	248.1	2.0	2
AdobePremiere	Video editing	298.8	5.0	2
MotionPlayBack	Video processing	233.9	5.6	2
Netflix	Video streaming	229.4	4.6	2
SystemMgt	Win 7 managing	466.2	7.6	2
VideoEncode	Video encoding	299.1	7.3	4

Table 1: Evaluated long-running workloads

Processor	1-4 cores, 4GHz, 4-wide, 128-entry instruction window
Last-Level Cache	64B cache-line, 16-way associative 512KB private cache-slice per core
Main Memory	8GB DIMM DDR3-1600 (800MHz clock rate, 1.25ns cycle time) Baseline (t_{REFI}/t_{RFC}): 1.95us/350ns MEMCON: t_{REFI}: LO-REF 7.8us, HI-REF 1.95us MEMCON: t_{RFC}: **530/890/1600ns (16/32/64Gb)**

Table 2: Evaluated system configuration

Evaluation Results (1/2)

- How many refreshes are reduced?

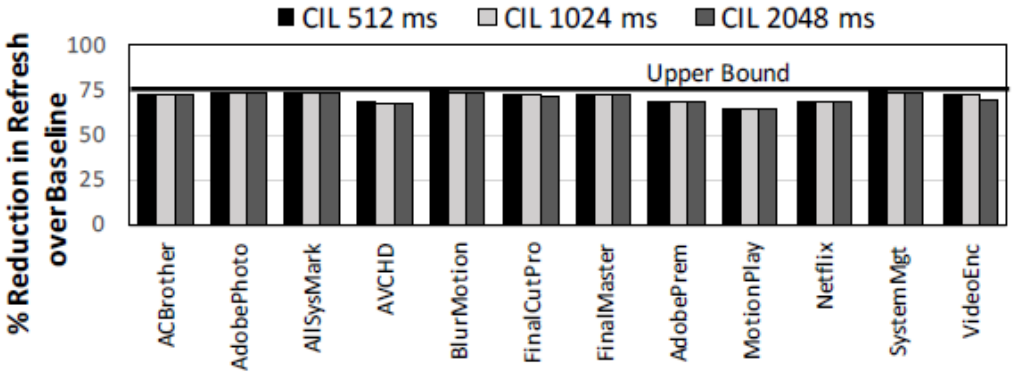


Figure 14: Reduction in refresh count with MEMCON

- Very close to the upper bound (a case when all refreshes are 64 ms but no error mitigation is applied)
- Different CILs do not affect the results that much

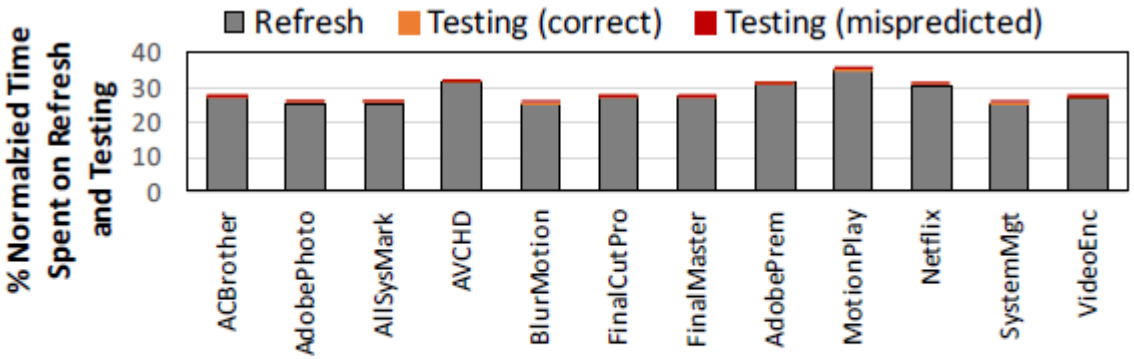
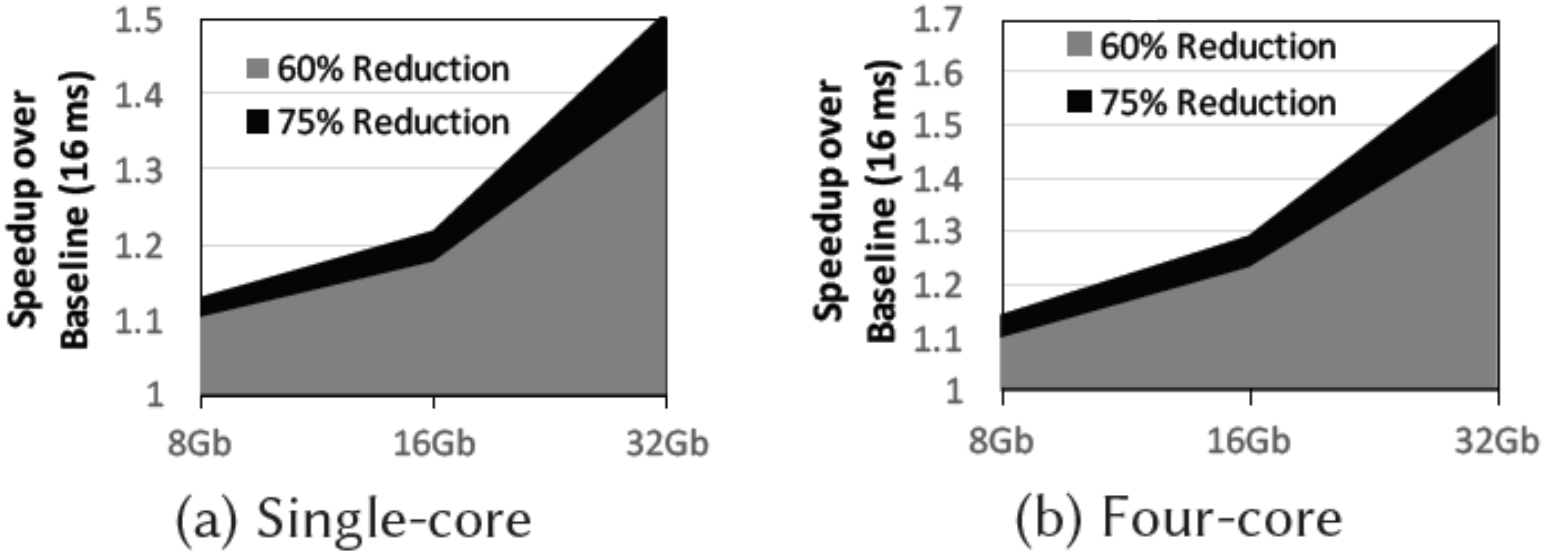


Figure 18: Fraction of time MEMCON spends on refresh operations and testing normalized to time spent on refresh in the baseline with 16 ms refresh

- Time spent for refresh reduced to 20 to 30 % of the baseline
- Mis-prediction overhead is very small

Evaluation Results (2/2)

- Performance Improvement thanks to MEMCON
  - Workload speeds to due to less frequen refresh (that incurs less resource contention inside DRAM)



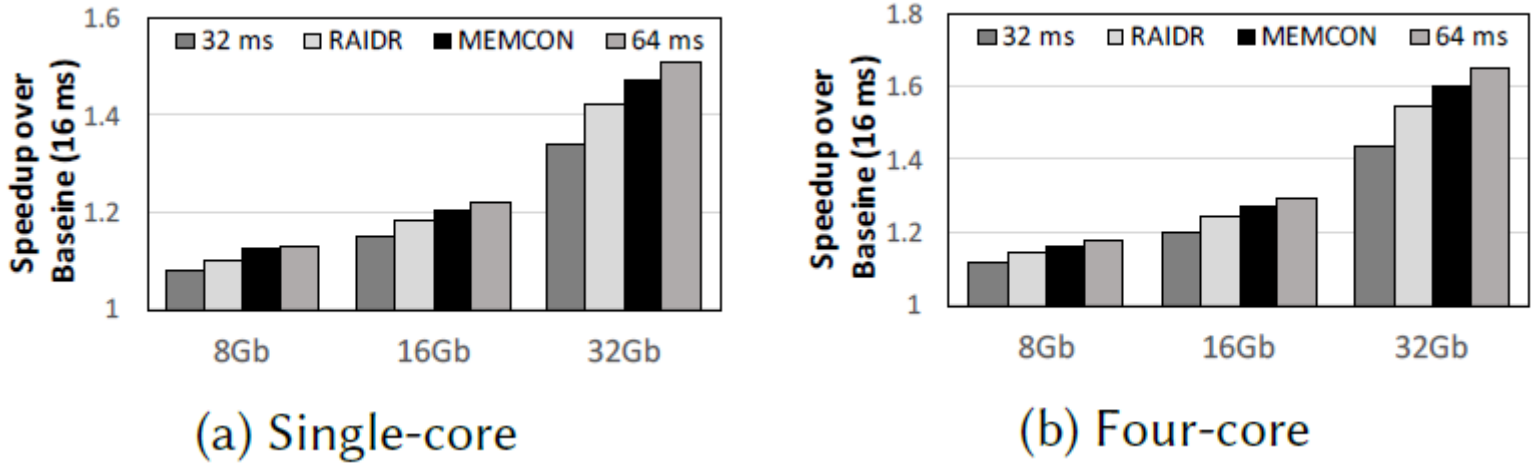
Refresh reduction vs. performance improvement

- MEMCON significantly improves performance thanks to reduced refreshes
- MEMCON's performance improvement increases with DRAM chip capacity

* Injected extra memory accesses to emulate the baseline (cycle accurate simulations infeasible)

Comparison with Other Techniques

- Other refresh optimization techniques
  - 32 ms: always refresh@32 ms interval (r iddle of long and short)
  - RAIDR: mitigate every possible failures (ıssuming that the DRAM internals are known)
  - 64 ms: always refresh@64 ms interval (no failure mitigation)



Dynamically detecting refresh interval for each row depending on the current content is the most effective

Related Work

- Reverse-Engineering of DRAM internal structures
  - [Jung et al., MEMSYS'16], [Khan et al., DSN'16], [Lee et al., SIGMETRICS'17]
  - MEMCON does not require DRAM internal structures such as how addresses are scrambled
- Multi-Rate DRAM Refresh
  - [Liu et al., ISCA'12], [Liu et al., ASPLOS'11], [Qureshi et al., DSN'15], [Venkastesan el al., HPCA'06]
  - Testing which cell to be refreshed with high/low rate is first done in MEMCON (existing works use simple tests)
- Refresh Optimization
  - [Chang et al., HPCA'14], [Isen et al., ISCA'09], [Mukundan et al., ISCA'13], [Nair et al., HPCA'13], [Steucheli et al., ISCA'10]
  - MEMCON is orthogonal and can be used on top of these works (?)