Suggested Paper Discussion Format

- Problem & Goal
- Key Ideas/solution
- Novelty
- Mechanisms & Implementation
- Major Results
- Takeaways/Conclusions
- Strengths
- Weaknesses
- Alternatives
- New ideas/problems
- Brainstorming and Discussion

~20-25 minute Summary

~10 min Critique plus
~10 min Discussion
Presentation Schedule

- We will have 11 sessions of presentations

- 2 presentations in each of the 11 sessions
  - Max 50 minutes total for each presentation + discussion
  - We will take the entire 2 hours in each meeting

- Each presentation
  - One student presents one paper and leads discussion
  - Max 25 minute summary + analysis
  - Max 10 minute critique
  - Max 10 minute discussion + brainstorming + feedback
  - Should follow the suggested guidelines
Algorithm for Presentation Preparation

- Study Lecture 1b again for presentation guidelines
- Read and analyze your paper thoroughly
  - Discuss with anyone you wish + use any resources
- Prepare a draft presentation based on guidelines
- Meet mentor(s) and get feedback
  - Revise the presentation and delivery
- Meet mentor(s) again and get further feedback
  - Revise the presentation and delivery
- Meetings are mandatory – you have to schedule them with your assigned mentor(s). We may suggest meeting times.
- Practice, practice, practice
Example Paper Presentations
Learning by Example

- A great way of learning
- We will do at least one today
Structure of the Presentation

- Background, Problem & Goal
- Novelty
- Key Approach and Ideas
- Mechanisms (in some detail)
- Key Results: Methodology and Evaluation
- Summary
- Strengths
- Weaknesses
- Thoughts and Ideas
- Takeaways
- Open Discussion
Background, Problem & Goal
Novelty
Key Approach and Ideas
Mechanisms (in some detail)
Key Results:
Methodology and Evaluation
Summary
Strengths
Weaknesses
Thoughts and Ideas
Takeaways
Open Discussion
Example Paper Presentation
Let’s Review This Paper

- Vivek Seshadri, Yoongu Kim, Chris Fallin, Donghyuk Lee, Rachata Ausavarungnirun, Gennady Pekhimenko, Yixin Luo, Onur Mutlu, Michael A. Kozuch, Phillip B. Gibbons, and Todd C. Mowry,

"RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization"

Proceedings of the 46th International Symposium on Microarchitecture (MICRO), Davis, CA, December 2013. [Slides (pptx) (pdf)] [Lightning Session Slides (pptx) (pdf)] [Poster (pptx) (pdf)]
RowClone

Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization

Vivek Seshadri

Background, Problem & Goal
Memory Channel – Bottleneck

Limited Bandwidth

High Energy
Goal: Reduce Memory Bandwidth Demand

Reduce unnecessary data movement
Bulk Data Copy and Initialization

Bulk Data Copy

Bulk Data Initialization

Source (src) to Destination (dst)

Value (val) to Destination (dst)
Bulk Data Copy and Initialization
Bulk Data Copy and Initialization

\textit{memmove} \& \textit{memcpy}: 5\% cycles in Google’s datacenter [Kanev+ ISCA’15]

Forking

Zero initialization (e.g., security)

Checkpointing

VM Cloning

Deduplication

Page Migration

Many more
Shortcomings of Today’s Systems

1) High latency
2) High bandwidth utilization
3) Cache pollution
4) Unwanted data movement

1046ns, 3.6uJ (for 4KB page copy via DMA)
Novelty, Key Approach, and Ideas
RowClone: In-Memory Copy

1) Low latency

2) Low bandwidth utilization

3) No cache pollution

4) No unwanted data movement

1046ns, 3.6uJ  →  90ns, 0.04uJ
RowClone: In-DRAM Row Copy

Idea: Two consecutive ACTivates

Negligible HW cost

4 Kbytes

Step 1: Activate row A

Step 2: Activate row B

DRAM subarray

Row Buffer (4 Kbytes)

Transfer row

Transfer row

8 bits

Data Bus
Mechanisms (in some detail)
DRAM Chip Organization

- Memory Channel
- Chip I/O
- Bank
- Subarray
- Bank I/O
- Row of DRAM Cells
- Row Buffer
RowClone Types

- Intra-subarray RowClone (row granularity)
  - Fast Parallel Mode (FPM)

- Inter-bank RowClone (byte granularity)
  - Pipelined Serial Mode (PSM)

- Inter-subarray RowClone
RowClone: Fast Parallel Mode (FPM)

1. Source row to row buffer
2. Row buffer to destination row
RowClone: Intra-Subarray (I)

Data gets copied

Sense Amplifier (Row Buffer)

Amplify the difference
RowClone: Intra-Subarray (II)

1. **Activate** src row (copy data from src to row buffer)

2. **Activate** dst row (disconnect src from row buffer, connect dst – copy data from row buffer to dst)
Fast Parallel Mode: Benefits

Bulk Data Copy

Latency: 11x
1046ns to 90ns

Energy: 74x
3600nJ to 40nJ

No bandwidth consumption
Very little changes to the DRAM chip
Fast Parallel Mode: Constraints

- Location of source/destination
  - Both should be in the same subarray

- Size of the copy
  - Copies *all* the data from source row to destination
RowClone: Inter-Bank

Overlap the latency of the read and the write
1.9X latency reduction, 3.2X energy reduction
Generalized RowClone

Inter Subarray Copy (Use Inter-Bank Copy Twice)

Inter Bank Copy (Pipelined Internal RD/WR)

Intra Subarray Copy (2 ACTs)

0.01% area cost
RowClone: Fast Row Initialization

Fix a row at Zero
(0.5% loss in capacity)
RowClone: Bulk Initialization

- Initialization with arbitrary data
  - Initialize one row
  - Copy the data to other rows

- Zero initialization (most common)
  - Reserve a row in each subarray (always zero)
  - Copy data from reserved row (FPM mode)
  - 6.0x lower latency, 41.5x lower DRAM energy
  - 0.2% loss in capacity
RowClone: Latency & Energy Benefits

Latency Reduction

Energy Reduction

Very low cost: 0.01% increase in die area
System Design to Enable RowClone
End-to-End System Design

- Application
- Operating System
- ISA
- Microarchitecture
- DRAM (RowClone)

How to communicate occurrences of bulk copy/initialization across layers?

How to ensure cache coherence?

How to maximize latency and energy savings?

How to handle data reuse?
1. Hardware/Software Interface

- Two new instructions
  - memcopy and meminit
  - Similar instructions present in existing ISAs

- Microarchitecture Implementation
  - Checks if instructions can be sped up by RowClone
  - Export instructions to the memory controller
2. Managing Cache Coherence

- RowClone modifies data in memory
  - Need to maintain coherence of cached data

- Similar to DMA
  - Source and destination in memory
  - Can leverage hardware support for DMA

- Additional optimizations
3. Maximizing Use of the Fast Parallel Mode

- Make operating system subarray-aware

- Primitives amenable to use of FPM
  - Copy-on-Write
    - Allocate destination in same subarray as source
    - Use FPM to copy
  - Bulk Zeroing
    - Use FPM to copy data from reserved zero row
4. Handling Data Reuse After Zeroing

- Data reuse after zero initialization
  - Phase 1: OS zeroes out the page
  - Phase 2: Application uses cachelines of the page

- RowClone
  - Avoids misses in phase 1
  - But incurs misses in phase 2

- RowClone-Zero-Insert (RowClone-ZI)
  - Insert clean zero cachelines
Key Results:
Methodology and Evaluation
Methodology

- Out-of-order multi-core simulator
- 1MB/core last-level cache
- Cycle-accurate DDR3 DRAM simulator
- 6 Copy/Initialization intensive applications + SPEC CPU2006 for multi-core

Performance
- Instruction throughput for single-core
- Weighted Speedup for multi-core
Copy/Initialization Intensive Applications

- **System bootup** (Booting the Debian OS)
- **Compile** (GNU C compiler – executing cc1)
- **Forkbench** (A fork microbenchmark)
- **Memcached** (Inserting a large number of objects)
- **MySql** (Loading a database)
- **Shell** script (find with ls on each subdirectory)
Copy and Initialization in Workloads

![Diagram showing the fraction of memory traffic for different workloads: bootup, compile, forkbench, mcached, mysql, shell. The diagram uses colored bars to represent Zero, Copy, Write, and Read operations.](image)
Single-Core – Performance and Energy

Improvements correlate with fraction of memory traffic due to copy(initialization)
Multi-Core Systems

- Reduced bandwidth consumption benefits all applications.

- Run copy_INITIALIZATION intensive applications with memory intensive SPEC applications.

- Half the cores run copy_INITIALIZATION intensive applications. Remaining half run SPEC applications.
Multi-Core Results: Summary

- System Performance
- Memory Energy Efficiency

Consistent improvement in energy/instruction

Performance improvement increases with increasing core count.
Summary
Executive Summary

- Bulk data copy and initialization
  - Unnecessarily move data on the memory channel
  - Degrade system performance and energy efficiency

- **RowClone** – perform copy in DRAM with low cost
  - Uses row buffer to copy large quantity of data
  - **Source row → row buffer → destination row**
  - 11X lower latency and 74X lower energy for a bulk copy

- Accelerate Copy-on-Write and Bulk Zeroing
  - Forking, checkpointing, zeroing (security), VM cloning

- Improves performance and energy efficiency at low cost
  - 27% and 17% for 8-core systems (0.01% DRAM chip area)
Strengths
Strengths of the Paper

- Simple, novel mechanism to solve an important problem
- Effective and low hardware overhead
- Intuitive idea!
- Greatly improves performance and efficiency (assuming data is mapped nicely)
- Seems like a clear win for data initialization (without mapping requirements)
- Makes software designer’s life easier
  - If copies are 10x-100x cheaper, how to design software?

- Paper tackles many low-level and system-level issues
- Well-written, insightful paper
Weaknesses
Weaknesses

- Requires data to be mapped in the same subarray to deliver the largest benefits
  - Helps less if data movement is not within a subarray
  - Does not help if data movement is across DRAM channels
- Inter-subarray copy is very inefficient
- Causes many changes in the system stack
  - End-to-end design spans applications to circuits
  - Software-hardware cooperative solution might not always be easy to adopt
- Cache coherence and data reuse cause real overheads
- Evaluation is done solely in simulation
- Evaluation does not consider multi-chip systems
- Are these the best workloads to evaluate?
Recall: Try to Avoid Rat Holes

Performance Analysis Rat Holes

- Workload
- Metrics
- Configuration
- Details

Source: https://www.cse.wustl.edu/~jain/iuceed/ftp/k_10adp.pdf
Thoughts and Ideas
Extensions and Follow-Up Work

- Can this be improved to do faster inter-subarray copy?
  - Yes, see the LISA paper [Chang et al., HPCA 2016]

- Can we enable data movement at smaller granularities within a bank?
  - Yes, see the FIGARO paper [Wang et al., MICRO 2020]

- Can this be improved to do better inter-bank copy?
  - Yes, see the Network-on-Memory paper [CAL 2020]

- Can similar ideas and DRAM properties be used to perform computation on data?
  - Yes, see the Ambit paper [Seshadri et al., MICRO 2017]
LISA: Fast Inter-Subarray Data Movement

Kevin K. Chang, Prashant J. Nair, Saugata Ghose, Donghyuk Lee, Moinuddin K. Qureshi, and Onur Mutlu,
"Low-Cost Inter-Linked Subarrays (LISA): Enabling Fast Inter-Subarray Data Movement in DRAM"
[Slides (pptx) (pdf)]
[Source Code]

Low-Cost Inter-Linked Subarrays (LISA): Enabling Fast Inter-Subarray Data Movement in DRAM

Kevin K. Chang†, Prashant J. Nair*, Donghyuk Lee†, Saugata Ghose†, Moinuddin K. Qureshi*, and Onur Mutlu†

†Carnegie Mellon University    *Georgia Institute of Technology
Goal: Provide a new substrate to enable wide connectivity between subarrays
Key Idea and Applications

• **Low-cost Inter-linked subarrays (LISA)**
  - Fast bulk data movement between subarrays
  - **Wide datapath via isolation transistors:** 0.8% DRAM chip area

  ![Diagram of subarray connections](image)

• **LISA is a versatile substrate** → new applications
  - **Fast bulk data copy:** Copy latency 1.363ms → 0.148ms (9.2x)
    → 66% speedup, -55% DRAM energy
  - **In-DRAM caching:** Hot data access latency 48.7ns → 21.5ns (2.2x)
    → 5% speedup
  - **Fast precharge:** Precharge latency 13.1ns → 5.0ns (2.6x)
    → 8% speedup
More on LISA

Kevin K. Chang, Prashant J. Nair, Saugata Ghose, Donghyuk Lee, Moinuddin K. Qureshi, and Onur Mutlu,
"Low-Cost Inter-Linked Subarrays (LISA): Enabling Fast Inter-Subarray Data Movement in DRAM"
[Slides (pptx) (pdf)]
[Source Code]

Low-Cost Inter-Linked Subarrays (LISA): Enabling Fast Inter-Subarray Data Movement in DRAM

Kevin K. Chang†, Prashant J. Nair*, Donghyuk Lee†, Saugata Ghose†, Moinuddin K. Qureshi*, and Onur Mutlu†
†Carnegie Mellon University  *Georgia Institute of Technology
Network-On-Memory: Fast Inter-Bank Copy


**NoM: Network-on-Memory for Inter-Bank Data Transfer in Highly-Banked Memories**

Seyyed Hossein SeyyedAghaei Rezaei\(^1\)  
Mohammad Sadrosadati\(^3\)  
Mehdi Modarressi\(^{1,3}\)  
Onur Mutlu\(^4\)  
Rachata Ausavarungnirun\(^2\)  
Masoud Daneshtalab\(^5\)

\(^1\)University of Tehran  \(^2\)King Mongkut's University of Technology North Bangkok  \(^3\)Institute for Research in Fundamental Sciences  
\(^4\)ETH Zürich  \(^5\)Mälardalens University
In-DRAM Bulk Bitwise AND/OR

- Vivek Seshadri, Kevin Hsieh, Amirali Boroumand, Donghyuk Lee, Michael A. Kozuch, Onur Mutlu, Phillip B. Gibbons, and Todd C. Mowry,

"Fast Bulk Bitwise AND and OR in DRAM"

Fast Bulk Bitwise AND and OR in DRAM

Vivek Seshadri*, Kevin Hsieh*, Amirali Boroumand*, Donghyuk Lee*, Michael A. Kozuch†, Onur Mutlu*, Phillip B. Gibbons†, Todd C. Mowry*

*Carnegie Mellon University †Intel Pittsburgh
Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology

Proceedings of the 50th International Symposium on Microarchitecture (MICRO), Boston, MA, USA, October 2017.

[Slides (pptx) (pdf)] [Lightning Session Slides (pptx) (pdf)] [Poster (pptx) (pdf)]

Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology

Vivek Seshadri\textsuperscript{1,5} Donghyuk Lee\textsuperscript{2,5} Thomas Mullins\textsuperscript{3,5} Hasan Hassan\textsuperscript{4} Amirali Boroumand\textsuperscript{5} Jeremie Kim\textsuperscript{4,5} Michael A. Kozuch\textsuperscript{3} Onur Mutlu\textsuperscript{4,5} Phillip B. Gibbons\textsuperscript{5} Todd C. Mowry\textsuperscript{5}

\textsuperscript{1}Microsoft Research India \quad \textsuperscript{2}NVIDIA Research \quad \textsuperscript{3}Intel \quad \textsuperscript{4}ETH Zürich \quad \textsuperscript{5}Carnegie Mellon University
In-DRAM Bulk Bitwise Execution Paradigm

- Vivek Seshadri and Onur Mutlu, *"In-DRAM Bulk Bitwise Execution Engine"*  
  [*Preliminary arXiv version*]

---

**In-DRAM Bulk Bitwise Execution Engine**

Vivek Seshadri  
Microsoft Research India  
visesha@microsoft.com

Onur Mutlu  
ETH Zürich  
onur.mutlu@inf.ethz.ch
Extensions and Follow-Up Work (II)

- Can this idea be evaluated on a real system? How?
  - Yes, see the Compute DRAM paper [MICRO 2019]

- Can similar ideas be used in other types of memories? Phase Change Memory? RRAM? STT-MRAM?
  - Yes, see the Pinatubo paper [DAC 2016]

- Can we have more efficient solutions to
  - Cache coherence (minimize overhead)
  - Data reuse after copy and initialization
Pinatubo: A Processing-in-Memory Architecture for Bulk Bitwise Operations in Emerging Non-volatile Memories

Shuangchen Li\textsuperscript{1,*}, Cong Xu\textsuperscript{2}, Qiaosha Zou\textsuperscript{1,5}, Jishen Zhao\textsuperscript{3}, Yu Lu\textsuperscript{4}, and Yuan Xie\textsuperscript{1}

University of California, Santa Barbara\textsuperscript{1}, Hewlett Packard Labs\textsuperscript{2}
University of California, Santa Cruz\textsuperscript{3}, Qualcomm Inc.\textsuperscript{4}, Huawei Technologies Inc.\textsuperscript{5}
{shuangchenli, yuanxie}@ece.ucsb.edu\textsuperscript{1}
RowClone Demonstration in Real DRAM Chips

ComputeDRAM: In-Memory Compute Using Off-the-Shelf DRAMs

Fei Gao
feig@princeton.edu
Department of Electrical Engineering
Princeton University

Georgios Tziantzioulis
georgios.tziantzioulis@princeton.edu
Department of Electrical Engineering
Princeton University

David Wentzlaff
wentzlaf@princeton.edu
Department of Electrical Engineering
Princeton University

Takeaways
Key Takeaways

- A novel method to accelerate data copy and initialization
- Simple and effective
- Hardware/software cooperative
- Good potential for work building on it to extend it
  - To different granularities
  - To make things more efficient and effective
  - Many works have already built on the paper (see LISA, FIGARO, Ambit, ComputeDRAM, and other works in Google Scholar)
- Easy to read and understand paper
Open Discussion
Discussion Starters

- Thoughts on the previous ideas?
- How practical is this?
- Will the problem become bigger and more important over time?
- Will the solution become more important over time?
- Are other solutions better?
- Is this solution clearly advantageous or opposite in some cases?
General Issues

- Data mapping and interleaving
- Data coherence between caches and DRAM
- Data reuse

- All are issues with Processing in Memory
More on RowClone

- Vivek Seshadri, Yoongu Kim, Chris Fallin, Donghyuk Lee, Rachata Ausavarungnirun, Gennady Pekhimenko, Yixin Luo, Onur Mutlu, Michael A. Kozuch, Phillip B. Gibbons, and Todd C. Mowry,

"RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization"

Proceedings of the 46th International Symposium on Microarchitecture (MICRO), Davis, CA, December 2013. [Slides (pptx) (pdf)] [Lightning Session Slides (pptx) (pdf)] [Poster (pptx) (pdf)]
RowClone

Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization

Vivek Seshadri

Some History
Historical Perspective

- This work is perhaps the first example of “minimally changing DRAM chips” to perform data movement and computation
  - Surprising that it was done as late as 2013!

- It led to a body of work on in-DRAM (and in-NVM) computation with “hopefully small” changes

- Work building on RowClone still continues

- Initially, it was dismissed by some reviewers
  - Rejected from ISCA 2013 conference
**Paper Strengths**

The paper includes a well written background on DRAM organization/operation. The proposed technique is simple and elegant; it nicely exploits key circuit-level characteristics of DRAM designs and minimizes the changes necessary to commodity DRAM chips.

**Paper Weaknesses**

I am concerned on the applicability of the technique and found the evaluation to be uncompelling in terms of motivating the work as well as quantifying the potential benefit. Details on how to efficiently manage the coherence between the cache hierarchy and DRAM to enable the proposed technique are glossed over, but in my opinion are critical to the narrative.
Another Review and Rebuttal

Detailed Comments

The paper proposes a simple and not new idea, block copy in a DRAM, and the creates a complete

Reviewer B mentions that our idea is "not new". An explicit reference by the reviewer would be helpful here. While the reviewer may be referring to one of the patents that we cite in our paper (citations 2, 6, 25, 26, 27 in the paper), these patents are at a superficial level and do *not* provide a concrete mechanism. In contrast, we propose three concrete mechanisms and provide details on the most important architectural and microarchitectural modifications required at the DRAM chip, the memory controller, and the CPU to enable a system that supports the mechanisms. We also analyze their latency, hardware overhead, power, and performance in detail. We are not aware of any prior work that achieves this.
RowClone: Fast and Efficient In-DRAM Copy and Initialization of Bulk Data

Many programs initialize or copy large amounts of memory data. Initialization and copying are

V. Seshadri, Y. Kim, D. Lee, C. Fallin, R. Ausavarungnirun, G. Pekhimenko, Y. Luo, O. Mutlu,

Review #295A  3  4  5  3
Review #295B  4  3  4  3
Review #295C  3  4  4  3
Yet Later... in ISCA 2015...

Profiling a warehouse-scale computer

Svilen Kanev†
Harvard University

Juan Pablo Darago†
Universidad de Buenos Aires

Kim Hazelwood†
Yahoo Labs

Parthasarathy Ranganathan
Google

Tipp Moseley
Google

Gu-Yeon Wei
Harvard University

David Brooks
Harvard University

Figure 4: 22-27% of WSC cycles are spent in different components of “datacenter tax”.

Data movement In fact, RPCs are by far not the only code portions that do data movement. We also tracked all calls to the `memcpy()` and `memmove()` library functions to estimate the amount of time spent on explicit data movement (i.e., exposed through a simple API). This is a conservative estimate because it does not track inlined or explicit copies. Just the variants of these two library functions represent 4-5% of datacenter cycles. Recent work in performing data movement in DRAM [45] could optimize away this piece of tax.
RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization

You are an author of this paper.

+ Abstract
Bulk data copy and initialization operations are frequently triggered by several system level operations in modern systems. Despite the fact that these operations do not require

+ Authors

+ Topics

<table>
<thead>
<tr>
<th>OveMer</th>
<th>Nov</th>
<th>WriQua</th>
<th>RevExp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Review #206A</td>
<td>5</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Review #206B</td>
<td>4</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Review #206C</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Review #206D</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Review #206E</td>
<td>4</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>
Suggestions to Reviewers

- Be fair; you do not know it all
- Be open-minded; you do not know it all
- Be accepting of diverse research methods: there is no single way of doing research
- Be constructive, not destructive
- Do not have double standards...

Do not block or delay scientific progress for non-reasons