RowClone

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

Vivek Seshadri


SAFARI  Carnegie Mellon  Intel
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)
Memory Channel – Bottleneck

Limited Bandwidth

High Energy

Core
Cache
MC
Memory
Goal: Reduce Memory Bandwidth Demand

Reduce unnecessary data movement
Bulk Data Copy and Initialization

Bulk Data Copy

Bulk Data Initialization

```
src -- val -- dst
```

```
src -> dst
```

Bulk Data Copy and Initialization

The Impact of Architectural Trends on Operating System Performance

Mendel Rosenblum, Edouard Bugnion, Stephen Alan Herrod,
Emmett Witchel, and Anoop Gupta

Hardware Support for Bulk Data Movement in Server Platforms

Li Zhao†, Ravi Iyer‡ Srijana Makineni‡, Laxmi Bhuyan‡ and Don Newell‡
†Department of Computer Science and Engineering, University of California, Riverside, CA 92521
Email: {zhao, bhuyan}@cs.ucr.edu
‡Communications Technology Lab, Intel Corporation

Architecture Support for Improving Bulk Memory Copying and Initialization Performance

Xiaowei Jiang, Yan Solihin
Dept. of Electrical and Computer Engineering
North Carolina State University
Raleigh, USA

Li Zhao, Ravishankar Iyer
Intel Labs
Intel Corporation
Hillsboro, USA
Bulk Copy and Initialization – Applications

Forking

Zero initialization (e.g., security)

Checkpointing

VM Cloning

Deduplication

Page Migration

Many more
Shortcomings of Existing Approach

- High Energy
  (3600nJ to copy 4KB)

- High latency
  (1046ns to copy 4KB)

- Interference
Our Approach: In-DRAM Copy with Low Cost
Outline

✓ Introduction

- DRAM Background

- RowClone
  - Fast Parallel Mode
  - Pipelined Serial Mode

- End-to-end Design

- Evaluation
DRAM Chip Organization

- Chip I/O
- Bank
- Chip I/O
- Bank

- Subarray
- Bank I/O

- Memory Channel

- Row of DRAM Cells
- Row Buffer
DRAM Read Operation

**ACTIVATE**: Copy data from row to row buffer

**READ**: Transfer data to channel using the shared bus
DRAM Cell Operation

DRAM Cell

Sense Amplifier (Row Buffer)

$V_{DD}$

$V_{DD}/2$

$0$
DRAM Cell Operation

In the stable state, the sense amplifier drives the cell.

ACTIVATE

DRAM Cell

0

V_{DD}/2 \quad V_{DD}

V_{DD}/2 + \delta

Amplify the difference

Cell loses charge

Restore Cell Data

READ/WRITE
✓ Introduction
✓ DRAM Background
  ▪ RowClone
    • Fast Parallel Mode
    • Pipelined Serial Mode
▪ End-to-end Design
▪ Evaluation
RowClone: Fast Parallel Mode (FPM)

1. Source row to row buffer

2. Row buffer to destination row
Fast Parallel Mode: Implementation

Data gets copied

Ssense Amplifier
(Row Buffer)

Amplify the difference

$V_{DD}/2 \quad V_{DD}$

$V_{DD}/2 + \delta$

src 0

dst 0

$V_{DD}/2$ $\delta$

$V_{DD}/2$ $0$
Fast Parallel Mode: Implementation

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: Pipelined Serial Mode (PSM)

Overlap the latency of the read and the write
1.9X latency reduction, 3.2X energy reduction
Bulk Copy using RowClone

- **Inter subarray:** Use PSM twice
- **Inter bank:** Use PSM
- **Intra subarray:** Use FPM
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
Latency and Energy Benefits

Latency Reduction

<table>
<thead>
<tr>
<th>Copy</th>
<th>Zero</th>
<th>Intra-Subarray</th>
<th>Inter-Bank</th>
</tr>
</thead>
<tbody>
<tr>
<td>11.6x</td>
<td>1.9x</td>
<td>1.0x</td>
<td>6.0x</td>
</tr>
</tbody>
</table>

Energy Reduction

<table>
<thead>
<tr>
<th>Copy</th>
<th>Zero</th>
<th>Intra-Subarray</th>
<th>Inter-Bank</th>
</tr>
</thead>
<tbody>
<tr>
<td>74.4x</td>
<td>3.2x</td>
<td>1.5x</td>
<td>41.5x</td>
</tr>
</tbody>
</table>

Very low cost: 0.01% increase in die area
Outline

✓ Introduction
✓ DRAM Background
✓ RowClone
  • Fast Parallel Mode
  • Pipelined Serial Mode
  ▪ End-to-end Design
  ▪ Evaluation
How does the software communicate occurrences of bulk copy/initialization to hardware?

How to ensure cache coherence?

How to maximize use of the Fast Parallel Mode?

Handling data reuse after zero initialization?
1. Hardware/Software Interface

- Two new instructions
  - memcpy 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
✓ Introduction
✓ DRAM Background
✓ RowClone
  • Fast Parallel Mode
  • Pipelined Serial Mode
✓ End-to-end Design
  ▪ 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 bootstrap** (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)
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
Other Results and Discussion in the Paper

- Discussion on interleaving and copy granularity
- Detailed analysis of the fork benchmark
- Detailed multi-core results and analysis
- Results with the PSM mode
- Analysis of RowClone-ZI
- Comparison to memory-controller-based DMA
Conclusion

- 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% chip area overhead)
RowClone

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

Vivek Seshadri
Y. Kim, C. Fallin, D. Lee, R. Ausavarungnirun,
G. Pekhimenko, Y. Luo, O. Mutlu,
P. B. Gibbons, M. A. Kozuch, T. C. Mowry
## Multi-core Metrics

<table>
<thead>
<tr>
<th></th>
<th>2-core</th>
<th>4-core</th>
<th>8-core</th>
</tr>
</thead>
<tbody>
<tr>
<td># Workloads</td>
<td>138</td>
<td>50</td>
<td>40</td>
</tr>
<tr>
<td>Weighted Speedup</td>
<td>15%</td>
<td>20%</td>
<td>27%</td>
</tr>
<tr>
<td>Instruction Throughput</td>
<td>14%</td>
<td>15%</td>
<td>25%</td>
</tr>
<tr>
<td>Harmonic Speedup</td>
<td>13%</td>
<td>16%</td>
<td>29%</td>
</tr>
<tr>
<td>Max Slowdown Reduction</td>
<td>6%</td>
<td>12%</td>
<td>23%</td>
</tr>
<tr>
<td>Bandwidth/Instruction Reduction</td>
<td>29%</td>
<td>27%</td>
<td>28%</td>
</tr>
<tr>
<td>Energy/Instruction Reduction</td>
<td>19%</td>
<td>17%</td>
<td>17%</td>
</tr>
</tbody>
</table>
RowClone-ZI Single-Core

Instructions per Cycle

- bootup
- compile
- forkbench
- mcached
- mysql
- shell

Baseline  | RowClone  | RowClone-ZI
RowClone-ZI Multi-Core

[Graph showing normalized weighted speedup for Baseline, RowClone, and RowClone-ZI.]
Forkbench – Fraction of Memory Traffic

Number of Pages Updated

64MB
128MB
Forkbench – Performance

![Graph showing normalized IPC vs number of pages updated for 64MB and 128MB]
Forkbench – Energy

![Graph showing energy consumption for Baseline, RowClone-PSM, and RowClone-FPM.](Image)

- **Baseline**
- **RowClone-PSM**
- **RowClone-FPM**

**Y-Axis:** Normalized Energy

**X-Axis:** Number of Pages Updated (2, 4, 8, 16, 32, 64, 128, 256, 512, 1k, 2k, 4k, 8k, 16k)
Comparison to Prior Work

- Copy engines (Zhao et al. 2005, Jiang et al. 2009)
  - Addresses cache pollution, pipeline stalls due to copy
  - But requires data transfer over the memory channel

- IRAM (Patterson et al. 1997)
  - Compute + memory using same technology
  - Exploit high DRAM bandwidth
  - Goal: Wider range of SIMD operations
  - High cost
Why is FPM not done today?

- Copy/Initialization is important
  - But not well known

- Opportunity to perform in DRAM
  - Not well known

- This paper: Proof of concept
  - More challenges to be addressed