PiDRAM

An FPGA-based Framework for End-to-end Evaluation of Processing-in-DRAM Techniques

Ataberk Olgun
Juan Gomez Luna   Konstantinos Kanellopoulos   Behzad Salami
Hasan Hassan      Oğuz Ergin       Onur Mutlu

SAFARI   ETH zürich   kasırga   TOBB ETÜ
Executive Summary

**Motivation:** Commodity DRAM based PiM techniques improve the performance and energy efficiency of computing systems at no additional DRAM hardware cost.

**Problem:** Challenges of integrating these PiM techniques into real systems are not solved. General-purpose computing systems, special-purpose testing platforms, and system simulators cannot be used to efficiently study system integration challenges.

**Goal:** Design and implement a flexible framework that can be used to:
- Solve system integration challenges
- Analyze trade-offs of end-to-end implementations of commodity DRAM based PiM techniques

**Key idea:** PiDRAM, an FPGA-based framework that enables:
- System integration studies
- End-to-end evaluations of commodity DRAM based PiM techniques using real unmodified DRAM chips

**Evaluation:** End-to-end integration of two PiM techniques on PiDRAM’s FPGA prototype

**Case Study #1 – RowClone:** In-DRAM bulk data copy operations
- 119x speedup for copy operations compared to CPU-copy with system support
- 198 lines of Verilog and 565 lines of C++ code over PiDRAM’s flexible codebase

**Case Study #2 – D-RaNGe:** DRAM-based random number generation technique
- 8.30 Mb/s true random number generator (TRNG) throughput, 220 ns TRNG latency
- 190 lines of Verilog and 78 lines of C++ code over PiDRAM’s flexible codebase
Outline

Background
  DRAM Organization and Operation
  Commodity DRAM Based PiM Techniques

PiDRAM
  Overview
  Hardware & Software Components
  FPGA Prototype

Case Studies
  Case Study #1 – RowClone

Conclusion
Accessing a DRAM Cell

- wordline
- capacitor
- access transistor
- enable
- Sense Amp
- bitline
Accessing a DRAM Cell

1. Enable wordline
2. Connects cell to bitline
3. Cell loses charge to bitline
4. Deviation in bitline voltage
5. Enable sense amp
6. Cell charge restored

Sense Amp

\[ \frac{1}{2} V_{DD} \]

\[ V_{DD} + \delta \]

Wordline

Capacitor

Access Transistor

Bitline
DRAM Operation

**DRAM Command Sequence**

- **tRAS** (Activation Latency)
- **tRP** (Precharge Latency)
- **tRCD** (Access Latency)

**ACT R0** **RD** **RD** **RD** **PRE R0** **ACT R1** **RD** **RD** **RD**

*Time*

[Kim+ HPCA’19]
Outline

Background
- DRAM Organization and Operation
- Commodity DRAM Based PiM Techniques

PiDRAM
- Overview
- Hardware & Software Components
- FPGA Prototype

Case Studies
- Case Study #1 – RowClone

Conclusion
Use operational principles of commodity DRAM chips to perform **bulk data movement and computation in memory**

Two relevant examples:

1) **In-DRAM Copy**: In-DRAM bulk data copy (or initialization) at DRAM row granularity

2) **D-RaNGe**: In-DRAM true random number generation (TRNG) using access latency (tRCD) failures
In-DRAM Copy: Key Idea (RowClone)

1. Source row to row buffer
2. Row buffer to destination row

[Seshadri+ MICRO’13]
Outline

Background
DRAM Organization and Operation
Commodity DRAM Based PiM Techniques

PiDRAM
Overview
Hardware & Software Components
FPGA Prototype

Case Studies
Case Study #1 – RowClone

Conclusion
PiDRAM: Overview (I)

A flexible framework that can be used to:
• Solve system integration challenges
• Analyze trade-offs of end-to-end implementations of commodity DRAM based PiM techniques

Identify key components shared across PiM techniques

Implement customizable key components:
• Provide modularity, enhance extensibility of the framework

Common basis to enable system support for PiM techniques
PiDRAM: Overview (II)

Identify and develop four key hardware and software components

**Hardware**

1. Flexible PiM Ops. Controller
2. Easy-to-extend Memory Controller

**Software**

3. Extensible Software Library
4. Custom Supervisor Software
PiDRAM: System Design

Key components are attached to a real computing system

- PiM Ops. Controller and PiDRAM Memory Controller is implemented within the hardware system
- Custom supervisor software runs on the hardware system
- Extensible software library is used by the supervisor software
PiM Operations Controller (POC)

- Decode & execute PiDRAM instructions (e.g., in-DRAM copy)
- Receive instructions over memory-mapped interface (portable to other systems with different CPU ISAs)
- Simple interface to the PiDRAM memory controller:
  1. send request,
  2. wait until completion,
  3. read results
PiDRAM Memory Controller

Perform PiM operations by violating DRAM timing parameters

Support conventional memory operations (e.g., LOAD/STORE)
One state machine per operation (e.g., LOAD/STORE, in-DRAM copy)

Easily replicate a state machine to implement a new operation

Controls the physical DDR3 interface
Receives commands from command scheduler & operates DDR3 pins
PiM Operations Library (pimolib)

Contains customizable functions that interface with the POC Software interface for performing PiM operations

Executes LOAD & STORE requests to communicate with the POC
Custom Supervisor Software

- Exposes PiM operations to the user application via system calls
- Contains the necessary OS primitives to develop end-to-end PiM techniques (e.g., memory management and allocation for RowClone)
copy() function called by the user to perform a **RowClone-Copy** operation in DRAM

1. Application makes a system call: \texttt{copy(A, B, N \text{ bytes})}

2. Custom Supervisor Software calls the \texttt{copy()} pimolib function

\[ \text{Copy} (S, \ D) \]

- \( S \): source DRAM row
- \( D \): destination DRAM row
Copy \((S, D)\) executes two store instructions in the CPU

The first store updates the *instruction* register with Copy \((S, D)\)

The second store sets the “Start” flag in the *flag* register

Start \((S)\)

Start the execution of PiM operation
PiM Operation Execution Flow

6. POC instructs the memory controller to perform RowClone

7. POC resets the “Start” flag, and sets the “Ack” flag

8. PiDRAM memory controller issues commands with violated timing parameters to the DDR3 module
9. The memory controller sets the “Fin.” flag

10. Copy (S, D) periodically checks either “Ack” or “Fin.” flags using LOAD instructions

Copy (S, D) returns when the periodically checked flag is set
Data Register is not used in RowClone operations because the result is stored *in memory*

It is used to read true random numbers generated by D-RaNGe
PiDRAM Components Summary

Four key components orchestrate PiM operation execution

Four key components provide an extensible basis for end-to-end integration of PiM techniques
PiDRAM’s FPGA Prototype

Full system prototype on Xilinx ZC706 FPGA board

- **RISC-V System**: In-order, pipelined RISC-V Rocket CPU core, L1D/I$, TLB
- **PiM-Enabled DIMM**: Micron MT8JTF12864, 1 GiB, 8 banks
Outline

Background
- DRAM Organization and Operation
- Commodity DRAM Based PiM Techniques

PiDRAM
- Overview
- Hardware & Software Components
- FPGA Prototype

Case Studies
- Case Study #1 – RowClone

Conclusion
PiDRAM Case Studies

We conduct two case studies:
1. In-DRAM bulk data copy (RowClone)
2. In-DRAM true random number generation (D-RaNGe)

Demonstrate the flexibility and ease of use of PiDRAM
**RowClone Implementation (I)**

**Key Idea:** Perform in-DRAM copy operations by using carefully created DRAM command sequences

- ComputeDRAM [Gao+, MICRO’19] demonstrates in-DRAM copy operations in real DDR3 chips
- ACT → PRE → ACT command sequence with greatly reduced tRAS and tRP timing parameters

![Diagram showing DRAM subarray and rows with timing details](image-url)
RowClone Implementation (II)

1. Extend the PiDRAM memory controller to support the DRAM command sequence
2. Expose the operation to pimolib by implementing the `copy()` PiDRAM instruction

Only 198 lines of Verilog code

[Olgun+ ISCA'21]
RowClone System Integration

Two challenges in integrating RowClone end-to-end in a real system

① Memory allocation constraints (Intra-subarray operation)

② Memory coherency (Computation in DRAM)

Implement CLFLUSH instruction in the RISC-V CPU
Evict a cache block from the CPU caches to the DRAM module
RowClone Memory Allocation (I)

Memory allocation requirements

1. **Granularity:** Operands must occupy DRAM rows fully

Diagram:

- **BANK X:**
  - SA W:
    - Source 2
    - Target 2
  - SA Z:
    - Target 3

- **BANK Y:**
  - Source 1
  - Target 1
  - Source 3
  - Source 4
  - Target 4
RowClone Memory Allocation (I)

Memory allocation requirements

Alignment: Operands must be placed at the same offset
Memory allocation requirements

Mapping: Operands must be placed in the same subarray
RowClone Memory Allocation (I)

Memory allocation requirements

1. Source 1
2. Target 1
3. Source 3
4. Satisfies all three requirements
To overcome the memory allocation challenges implement a new memory allocation function

**Goal:** Allow programmers to allocate virtual memory pages that are mapped to the same DRAM subarray and aligned with each other

**Key Mechanism:** Distribute virtual pages to different banks while mapping them to DRAM rows in the same DRAM subarray

```c
alloc_align(int size, int id)
size: # of bytes allocated
id: allocations with the same id go to the same subarray
```
To overcome the memory allocation challenges, implement a new memory allocation function.

**Goal:** Allow programmers to allocate virtual memory pages that are mapped to the same DRAM subarray and aligned with each other.

**Key Mechanism:** Distribute virtual pages to different banks while mapping them to DRAM rows in the same DRAM subarray.

```c
alloc_align(int size, int id)
```

- **size**: # of bytes allocated
- **id**: allocations with the same id go to the same subarray

Table 2: PiDRAM system configuration

- **CPU**: 50 MHz; in-order Rocket core [16]; **TLB**: 4 entries DTLB; LRU policy
- **L1 Data Cache**: 16 KiB, 4-way; 64 B line; random replacement policy
- **DRAM Memory**: 1 GiB DDR3; 800MT/s; single rank; 8 KiB row size

**Microbenchmarks**
- CPU-Copy (using LOAD/STORE instructions)
- RowClone-Copy (using in-DRAM copy operations)

**Copy/Initialization Heavy Workloads**
- compile (initialization)
- forkbench (copy)

**SPEC2006** **libquantum**: replace “calloc()” with in-DRAM initialization
Microbenchmark Copy/Initialization Throughput Improvement

In-DRAM Copy and Initialization improve throughput by 119x and 89x, respectively
CLFLUSH dramatically reduces the potential throughput improvement.
Other Workloads

**forkbench** (copy-heavy workload)

- 9% execution time reduction by in-DRAM initialization
  - 17% of compile’s execution time is spent on initialization

**compile** (initialization-heavy workload)

**SPEC2006 libquantum**

- 1.3% end-to-end execution time reduction
  - 2.3% of libquantum’s time is spent on initialization
Outline

Background
- DRAM Organization and Operation
- Commodity DRAM Based PiM Techniques

PiDRAM
- Overview
- Hardware & Software Components
- FPGA Prototype

Case Studies
- Case Study #1 – RowClone

Conclusion
Executive Summary

**Motivation:** Commodity DRAM-based PiM techniques improve the performance and energy efficiency of computing systems at no additional DRAM hardware cost

**Problem:** Challenges of integrating these PiM techniques into real systems are not solved. General-purpose computing systems, special-purpose testing platforms, and system simulators *cannot* be used to efficiently study system integration challenges.

**Goal:** Design and implement a flexible framework that can be used to:
- Solve system integration challenges
- Analyze trade-offs of end-to-end implementations of commodity DRAM based PiM techniques

**Key idea:** FPGA-based framework that enables:
- System integration studies
- End-to-end evaluations of commodity DRAM based PiM techniques using real unmodified DRAM chips

**Evaluation:** End-to-end integration of two PiM techniques on PiDRAM’s FPGA prototype

**Case Study #1 – RowClone:** In-DRAM bulk data copy operations
- 118.5x speedup for copy operations compared to CPU-copy with system support
- 198 lines of Verilog and 565 lines of C++ code over PiDRAM’s flexible codebase

**Case Study #2 – D-RaNGe:** DRAM-based random number generation technique
- 8.30 Mb/s true random number generator (TRNG) throughput, 220 ns TRNG latency
- 190 lines of Verilog and 78 lines of C++ code over PiDRAM’s flexible codebase
PiDRAM is the first flexible end-to-end framework that enables system integration studies and evaluation of real Processing-using-Memory (PuM) techniques. PiDRAM, at a high level, comprises a RISC-V system and a custom memory controller that can perform PuM operations in real DDR3 chips. This repository contains all sources required to build PiDRAM and develop its prototype on the Xilinx ZC706 FPGA boards.
PiDRAM: A Holistic End-to-end FPGA-based Framework for Processing-in-DRAM

Ataberk Olgun, Juan Gómez Luna, Konstantinos Kanellopoulos, Behzad Salami, Hasan Hassan, Oğuz Ergin, Onur Mutlu

Processing-in-memory (PiM) techniques leverage the analog operation of memory cells to perform computation. Several recent works have demonstrated PiM techniques in off-the-shelf DRAM devices. Since DRAM is the dominant memory technology as main memory in current computing systems, these PiM techniques represent an opportunity for alleviating the data movement bottleneck at very low cost. However, system integration of PiM techniques imposes non-trivial challenges that are yet to be solved. Design space exploration of potential solutions to the PiM integration challenges requires appropriate tools to develop necessary hardware and software components. Unfortunately, current specialized DRAM-testing platforms, or system simulators do not provide the flexibility and/or the holistic system view that is necessary to deal with PiM integration challenges.

We design and develop PiDRAM, the first flexible end-to-end framework that enables system integration studies and evaluation of real PiM techniques. PiDRAM provides software and hardware components to rapidly integrate PiM techniques across the whole system software and hardware stack (e.g., necessary modifications in the operating system, memory controller). We implement PiDRAM on an FPGA-based platform along with an open-source RISC-V system. Using PiDRAM, we implement and evaluate two state-of-the-art PiM techniques: in-DRAM (i) copy and initialization, (ii) true random number generation. Our results show that the in-memory copy and initialization techniques can improve the performance of bulk copy operations by 12.6x and bulk initialization operations by 14.6x on a real system. Implementing the true random number generator requires only 190 lines of Verilog and 74 lines of C code using PiDRAM’s software and hardware components.

Extended Version on ArXiv

https://arxiv.org/abs/2111.00082
Long Talk + Tutorial on Youtube

https://youtu.be/s_z_S6FYpC8
PiDRAM
An FPGA-based Framework for End-to-end Evaluation of Processing-in-DRAM Techniques

Ataberk Olgun
Juan Gomez Luna  Konstantinos Kanellopoulos  Behzad Salami
Hasan Hassan  Oğuz Ergin  Onur Mutlu

SAFARI  kasırga
ETH zürich  TOBB ETÜ
BACKUP SLIDES
To enable alloc_align(), we maintain the SubArray Mapping Table (SAMT)

**alloc_align() function**

1. Retrieve a physical address pointing to a DRAM row in subarray 0
2. Update the page table to map programmer-allocated address to subarray 0
Initializing SAMT

https://arxiv.org/abs/2111.00082

Perform in-DRAM copy using every DRAM row address as source and destination rows

If the in-DRAM copy operation succeeds source and destination rows are in the same subarray
### Table 3: Comparing features of PiDRAM with related state-of-the-art hardware-software platforms

<table>
<thead>
<tr>
<th>Platforms</th>
<th>Interface with DRAM</th>
<th>Flexible MC for PuM</th>
<th>Open-source</th>
<th>System software support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Silent-PIM [57]</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>SoftMC [48]</td>
<td>✓ (DDR3)</td>
<td>X</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>ComputeDRAM [36]</td>
<td>✓ (DDR3)</td>
<td>X</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>MEG [103]</td>
<td>✓ (HBM)</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Simulators [20, 65, 84, 87]</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>PiDRAM (this work)</td>
<td>✓ (DDR3)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Allocate 128 KiB A and B to same subarray

\[ A = \text{alloc\_align}(128\times1024, 0); \]
\[ B = \text{alloc\_align}(128\times1024, 0); \]

Arrays are split into 4KB blocks

Allocate and RCC
## List of PuM Techniques That Can Be Studied Using PiDRAM

Table 1: PuM techniques that can be studied using PiDRAM. PuM techniques that we implement in this work are highlighted in bold

<table>
<thead>
<tr>
<th>PuM Technique</th>
<th>Description</th>
<th>Integration Challenges</th>
</tr>
</thead>
<tbody>
<tr>
<td>RowClone [91]</td>
<td>Bulk data-copy and initialization within DRAM</td>
<td>(i) memory allocation and alignment mechanisms that map source &amp; destination operands of a copy operation into same DRAM subarray; (ii) memory coherence, i.e., source &amp; destination operands must be up-to-date in DRAM.</td>
</tr>
<tr>
<td>D-RaNGe [62]</td>
<td>True random number generation using DRAM</td>
<td>(i) periodic generation of true random numbers; (ii) memory scheduling policies that minimize the interference caused by random number requests.</td>
</tr>
<tr>
<td>Ambit [89]</td>
<td>Bitwise operations in DRAM</td>
<td>(i) memory allocation and alignment mechanisms that map operands of a bitwise operation into same DRAM subarray; (ii) memory coherence, i.e., operands of the bitwise operations must be up-to-date in DRAM.</td>
</tr>
<tr>
<td>SIMDRAg [43]</td>
<td>Arithmetic operations in DRAM</td>
<td>(i) memory allocation and alignment mechanisms that map operands of an arithmetic operation into same DRAM subarray; (ii) memory coherence, i.e., operands of the arithmetic operations must be up-to-date in DRAM; (iii) bit transposition, i.e., operand bits must be laid out vertically in a single DRAM bitline.</td>
</tr>
<tr>
<td>DL-PUF [61]</td>
<td>Physical unclonable functions in DRAM</td>
<td>memory scheduling policies that minimize the interference caused by generating PUF responses.</td>
</tr>
<tr>
<td>QUAC-TRNG [82]</td>
<td>True random number generation using DRAM</td>
<td>(i) periodic generation of true random numbers; (ii) memory scheduling policies that minimize the interference caused by random number requests; (iii) efficient integration of the SHA-256 cryptographic hash function.</td>
</tr>
</tbody>
</table>
Outline

Background
  DRAM Organization and Operation
  Commodity DRAM Based PiM Techniques

PiDRAM
  Overview
  Hardware & Software Components
  FPGA Prototype

Case Studies
  Case Study #1 – RowClone
  Case Study #2 – D-RaNGe

Conclusion
In-DRAM TRNG: Key Idea (D-RaNGe)

- High % chance to fail with reduced $t_{RCD}$
- Low % chance to fail with reduced $t_{RCD}$

[Kim+ HPCA'19]
D-RaNGe Implementation

Identify four DRAM cells that fail randomly when accessed with a reduced $t_{\text{RCD}}$ (RNG Cell) in a cache block.
D-RaNGe Implementation

Periodically generate true random numbers by accessing the identified cache block with reduced tRCD

- 1 KiB random number buffer in POC
- Programmers read random numbers from the data register using the `rand_dram()` function call

190 lines of Verilog code
74 lines of C++ code
Evaluation

Methodology: Microbenchmark that reads true random numbers

PiDRAM D-RaNGe generates true random numbers at 8.30 Mb/s throughput