PiDRAM
A Framework for End-to-end Integration of Processing-using-memory

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

**Motivation:** Recent works propose in-DRAM computation primitives with great potential to improve performance and energy consumption of computing systems.

**Problem:** These works are developed in limited environments (e.g., simulators, characterization platforms) where many parts of the system are ignored.  
• The challenges in integrating these primitives into a system cannot be fully explored in these environments.

**Goal:** Develop a flexible platform to explore end-to-end implementations of current and future processing-in-memory (PuM) techniques.

**Key idea:** To build an FPGA-based infrastructure that supports in-DRAM operations and has system support.

**SAFARI**
Outline

• Background
  • DRAM Organization
  • Processing-using-Memory
  • Rocket Chip SoC Generator

• Overview of PiDRAM
  • Hardware & Software Components
  • Prototype

• Case Study #1 - RowClone
  • Challenges
  • Allocation Mechanism
  • Memory Coherence
  • Evaluation

• Installing and Using PiDRAM
Outline

• Background
  • DRAM Organization
  • Processing-using-Memory
  • Rocket Chip SoC Generator

• Overview of PiDRAM
  • Hardware & Software Components
  • Prototype

• Case Study #1 - RowClone
  • Challenges
  • Allocation Mechanism
  • Memory Coherence
  • Evaluation

• Installing and Using PiDRAM
DRAM Organization

[Image of DRAM organization with components such as Memory Controller, CPU, DDR Interface, Chip, DRAM Module, DRAM Bank, DRAM Subarray, Wordline Drivers, Sense Amplifiers, DRAM Row, Capacitor, and DRAM Cell.]

SAFARI
[Olgun+ ISCA'21]
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

\[ V_{DD} / 2 \]
DRAM Operation

![Diagram of DRAM operation](image)

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

**DRAM Command Sequence**

```
ACT R0  RD  RD  RD  PRE R0  ACT R1  RD  RD  RD  RD
```

**time**

[Kim+ HPCA'19]
Outline

• Background
  • DRAM Organization
  • Processing-using-Memory
  • Rocket Chip SoC Generator

• Overview of PiDRAM
  • Hardware & Software Components
  • Prototype

• Case Study #1 - RowClone
  • Challenges
  • Allocation Mechanism
  • Memory Coherence
  • Evaluation

• Installing and Using PiDRAM
• Take advantage of operational principles of memory to perform bulk data movement and computation in memory
  • Can exploit internal connectivity to move data
  • Can exploit analog computation capability

• Examples: RowClone, In-DRAM AND/OR, D-RaNGe, ...
  • RowClone: Fast and Efficient In-DRAM Copy and Initialization of Bulk Data (Seshadri et al., MICRO 2013)
  • “Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology” (Seshadri et al., MICRO 2017)
  • “D-RaNGe: Using Commodity DRAM Devices to Generate True Random Numbers with Low Latency and High Throughput” (Kim et al., HPCA 2019)
  • ...
• Take advantage of operational principles of memory to perform bulk data movement and computation in memory
  • Can exploit internal connectivity to move data
  • Can exploit analog computation capability

• Examples: RowClone, In-DRAM AND/OR, D-RaNGe, ...
  • RowClone: Fast and Efficient In-DRAM Copy and Initialization of Bulk Data (Seshadri et al., MICRO 2013)
  • “Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology” (Seshadri et al., MICRO 2017)
  • “D-RaNGe: Using Commodity DRAM Devices to Generate True Random Numbers with Low Latency and High Throughput” (Kim et al., HPCA 2019)
  • ...
RowClone: Can we do it in DRAM?

High Energy
(3600nJ to copy 4KB)

High latency
(1046ns to copy 4KB)
RowClone-FPM: Mechanism

1. Source row to row buffer
2. Row buffer to destination row
RowClone-FPM: Bitline Operation (I)

\[ V_{DD}/2 \delta \]

\[ V_{DD}/2 + \delta \]

Data gets copied

Sense Amplifier (Row Buffer)

\[ V_{DD}/2 \]

\[ 0 \]
Enable fast bulk-data copy with small overhead

Data gets copied

Sense Amplifier (Row Buffer)

RowClone-FPM: Bitline Operation (II)
In-DRAM Bitwise AND/OR

- We can support in-DRAM COPY, ZERO, AND, OR, NOT, MAJ
- At low cost
- Using inherent analog computation capability of DRAM
  - Idea: activating multiple rows performs computation
- 30-60X performance and energy improvement

In-DRAM AND/OR: Triple Row Activation

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

Final State

\[ AB + BC + AC \]

\[ C(A + B) + \sim C(AB) \]

Seshadri+, "Fast Bulk Bitwise AND and OR in DRAM", IEEE CAL 2015.
Processing-using-Memory in Real DRAM Chips

**ComputeDRAM** Demonstrates RowClone and AND/OR in real chips

- **Violate** DRAM timing parameters: tRAS, tRP
  - Induce undefined behavior

**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

Figure 4: Timeline for a single bit of a column in a row copy operation. The data in $R_1$ is loaded to the bit-line, and overwrites $R_2$.

Figure 5: Logical AND in ComputeDRAM. $R_1$ is loaded with constant zero, and $R_2$ and $R_3$ store operands (0 and 1). The result ($0 = 1 \land 0$) is finally set in all three rows.
Row Copy in ComputeDRAM

Bitline is above $V_{DD}/2$ when R2 is activated.
Lecture on PuM

Goal: Processing Inside Memory

- Many questions ... How do we design the:
  - compute-capable memory & controllers?
  - processor chip and in-memory units?
  - software and hardware interfaces?
  - system software, compilers, languages?
  - algorithms and theoretical foundations?
SIMDRAM Live Seminar

SIMDRAM: PuM Substrate

- SIMDRAM framework is built around a DRAM substrate that enables two techniques:

  1. Vertical data layout
     - most significant bit (MSB)
     - least significant bit (LSB)
  
  Pros compared to the conventional horizontal layout:
  
  - Implicit shift operation
  - Massive parallelism

  ![Diagram of vertical data layout]

  2. Majority-based computation
  
  \[ C_{out} = AB + AC_{in} + BC_{in} \]

  ![Diagram of majority-based computation]

  Pros compared to AND/OR/NOT-based computation:
  
  - Higher performance
  - Higher throughput
  - Lower energy consumption

SAFARI Live Seminar - Data-Centric & Data-Aware Frameworks for Fundamentally Efficient Data Handling

https://youtu.be/XIfPHtvA9rw
Outline

• Background
  • DRAM Organization
  • Processing-using-Memory
  • Rocket Chip SoC Generator

• Overview of PiDRAM
  • Hardware & Software Components
  • Prototype

• Case Study #1 - RowClone
  • Challenges
  • Allocation Mechanism
  • Memory Coherence
  • Evaluation

• Installing and Using PiDRAM
Rocket Chip

Open-source SoC design *generator*

Composed of many SoC component *generators*

- Generator: Chisel/Scala code that builds hardware

Outputs *synthesizable* Verilog RTL

Core Parameters
- Enable VM
- Use ISA Extensions
- Generate muldiv
- Generate FPU
• Background
  • DRAM Organization
  • Processing-using-Memory
  • Rocket Chip SoC Generator

• Overview of PiDRAM
  • Hardware & Software Components
  • Prototype

• Case Study #1 - RowClone
  • Challenges
  • Allocation Mechanism
  • Memory Coherence
  • Evaluation

• Installing and Using PiDRAM
Goal: Develop a flexible platform to explore end-to-end implementations of PuM techniques
• Enable rapid integration via key components
Goal: Develop a flexible platform to explore end-to-end implementations of PuM techniques
• Enable rapid integration via key components

Hardware
1. Easy-to-extend Memory Controller
2. ISA-transparent PuM Controller

Software
1. Extensible Software Library
2. Custom Supervisor Software
Current real PuM techniques require issuing DRAM command sequences with violated timings

• Extensible memory controller facilitates implementation of such DRAM command sequences through modular design

• New command sequences require only designing new state machines
Memory controller employs three submodules to further ease developer effort

1. **Periodic Operations Module (PerOps Module):**
   DRAM periodic refresh and bus maintenance operations

2. **Scheduler:**
   DRAM RD/WR operations, open-bank policy

3. **Configuration Register File (CRF):**
   Store timing information, accessed using LD/ST instructions from the CPU
PiDRAM: PuM Controller (I)

PuM Operations Controller (POC)

Provide ISA-transparent control for PuM operations

- Connected as a memory-mapped module
  - Hierarchically, resides within the memory controller
- Simple Interface: Offload 128-bit instructions
**PiDRAM: PuM Controller (II)**

Currently implements five instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Reserved</th>
<th>Copy</th>
<th>Dst. Row</th>
<th>Src. Row</th>
<th>Address</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RowClone</strong></td>
<td></td>
<td>69</td>
<td>68</td>
<td>64</td>
<td>63</td>
<td>32</td>
</tr>
<tr>
<td><strong>Activation Failure</strong></td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>69</td>
<td>68</td>
<td>64</td>
<td>63</td>
<td>32</td>
<td>31</td>
</tr>
<tr>
<td><strong>Read Random Number (RN)</strong></td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>69</td>
<td>68</td>
<td>64</td>
<td>63</td>
<td>32</td>
<td>31</td>
</tr>
<tr>
<td><strong>Read RN Buffer Size</strong></td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>69</td>
<td>68</td>
<td>64</td>
<td>63</td>
<td>32</td>
<td>31</td>
</tr>
<tr>
<td><strong>Write to configuration regs.</strong></td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>69</td>
<td>68</td>
<td>64</td>
<td>63</td>
<td>32</td>
<td>31</td>
</tr>
</tbody>
</table>

Reserved bits for other commands

- Instruction size is *configurable*
PiDRAM: Software Library

Pumolib: Expose PuM operations to the user while abstracting the hardware implementation details

```c
static inline void copy_row(char *source, char *target) {
    volatile uint64_t *ptr = (uint64_t*) IMOC_INST_UPPER;
    uint64_t imo_op = 0x1;
    uint32_t source_row_addr = (uint32_t) source;
    uint64_t target_row_addr = (uint32_t) target;
    uint64_t inst_lower = source_row_addr |
                         (target_row_addr << 32);
    uint64_t inst_upper = imo_op << (IMO_OP_OF2);

    *ptr = inst_upper;
    *(ptr+1) = inst_lower;
    *(ptr+2) = (uint64_t) 0x1;
    while(*ptr != 0x02);  
}
```
Simple, easy-to-hack OS to integrate PuM techniques end-to-end:

• Virtual memory management
• Memory allocation

OS based on RISC-V proxy kernel

Future work: Integrate pumolib into Linux
PiDRAM Workflow

1- User application interfaces with the OS via system calls
2- OS uses PuM Operations Library (pumolib) to convey operation related information to the hardware *using*

3- STORE instructions that target the memory mapped registers of the PuM Operations Controller (POC)

4- POC oversees the execution of a PuM operation (e.g., RowClone, bulk bitwise operations)

5- Scheduler arbitrates between regular (load, store) and PuM operations and issues DRAM commands with custom timings
PiDRAM FPGA Prototype

Xilinx ZC706

Single core RISC-V CPU @ 50MHz in-order, single-issue
16KB 4-way L1 D$
4KB I$

Compute-Enabled DIMM
RISC-V System
Outline

• Background
  • DRAM Organization
  • Processing-using-Memory
  • Rocket Chip SoC Generator

• Overview of PiDRAM
  • Hardware & Software Components
  • Prototype

• Case Study #1 - RowClone
  • Challenges
    • Allocation Mechanism
    • Memory Coherence
    • Evaluation

• Installing and Using PiDRAM
RowClone-FPM has memory mapping requirements

1-) **Alignment:** Operands must be placed at the same offset to their respective DRAM rows

2-) **Granularity:** Operands must occupy whole DRAM rows
RowClone-FPM has memory mapping requirements

3-) Mapping: Operands must be placed within the same subarray
(4) Satisfies all three requirements
Outline

• Background
  • DRAM Organization
  • Processing-using-Memory
  • Rocket Chip SoC Generator

• Overview of PiDRAM
  • Hardware & Software Components
  • Prototype

• Case Study #1 - RowClone
  • Challenges
    • Allocation Mechanism
    • Memory Coherence
    • Evaluation

• Installing and Using PiDRAM
New memory allocation mechanism to satisfy these three requirements: `alloc_align()`

```c
void* array = alloc_align(int size);
```

Optimize physical address allocation to `array` for `size` byte large copy operations so that RowClone can be used most effectively.

**Example (Row = 8 KiB)**

```c
char *A = alloc_align(A, 4096*3);
```

We can at least copy 8 KiB using RowClone.
OS has **full control** over VA → PA translation

**No control** over PA → DRAM address mapping

**Idea:** If we can control VA → PA and we know the PA → DRAM mappings, we can implement `alloc_align`
void* array = alloc_align(int size);

How to lay out an array onto DRAM?

I. Distribute the array over multiple banks while occupying rows as fully as possible

II. Fallback to malloc() for remaining data

Assumptions:

(i) We know the DDRX address mapping in our system
   • Reverse-engineer: (i) Check for RowClone, (ii) do Rowhammer

(ii) We know which DRAM rows are in the same subarray
   • Characterize pairs of rows for RowClone success rate
Other versions of alloc_align can be implemented to align multiple arrays in DRAM (for RowClone-Copy)

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

`id` = operand RowClone identifier

Operands with the `same id` are `placed into the same subarray`
PiDRAM Address Mapping

Our Configuration:
SODIMM: MT8JTF12864HZ-1G6G1
# of Rows = 16K
# of Banks = 8
# of Columns = 1K
Row Size = 8 KB
Total Size = 1 GB

### Physical to DRAM address mapping (configurable)

<table>
<thead>
<tr>
<th>Rows</th>
<th>Banks</th>
<th>Columns</th>
<th>000</th>
</tr>
</thead>
<tbody>
<tr>
<td>29</td>
<td>16</td>
<td>15</td>
<td>13</td>
</tr>
</tbody>
</table>

SAFARI
A = alloc_align(16*1024, 0);  B = alloc_align(16*1024, 0);

Virtual Addresses: 0x0000 0x1000 0x2000

Array A

16 KBs

4 KB

Bank 0  Bank 1  Bank 2

Row 0

Row 1

Array B

16 KBs

0x7000
SubArray Mapping Table (SAMT)

A table that collects DRAM rows that map to the same subarray under common entries as physical page address pairs

We assume that DRAM row \(\rightarrow\) DRAM subarray mapping is arbitrary
Reserve *one* address pair in every SAMT entry for initialization.

**SIM, Subarray Id Map**

Mapping from page numbers to subarray indices.

<table>
<thead>
<tr>
<th>Physical Page Number</th>
<th>Subarray ID (SAMT idx)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PPN 0</td>
<td>0</td>
</tr>
<tr>
<td>PPN 1</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>PPN 256</td>
<td>1</td>
</tr>
</tbody>
</table>
Alloc_align - Structures (III)

How to use SMT

**alloc_align(int size):**

1. Find out how many banks to utilize
   - Spread allocation across many banks
2. Split array into 4 KB blocks
   - 4 KB == 1 page
   - NB = # of 4 KB blocks in A
3. For all blocks A[i] where i < NB/2:
   1. Pick a bank.
   2. Select a SAMT entry (SAMTE) with $\geq 1$ free address pairs
   3. Select one pair from SAMTE
   4. Assign physical pages in the pair to virtual pages A[i] and A[i+NB/2]
RowClone-Initialize (RCI)

\[ \text{rci}(\text{char* } A, \text{ int } N) \rightarrow \text{Initialize } A \text{ with } N \text{ 0s.} \]

1. Split \( A \) into 4KB blocks

2. For all \( \text{blocks } A[i] \) where \( i < \text{NB}/2 \):
   1. Translate from \( A[i] \) to \( \text{PA1} \rightarrow \text{use as destination address} \)
   2. Access \( \text{SIM} \) with \( \text{PA1} \) to obtain its \text{subarray id} \)
   3. Access \( \text{SAMT} \) to get the address of the \text{all-zero} row, \( \text{PA}_{\text{zero}} \)
   4. Perform RowClone from \( \text{PA}_{\text{zero}} \) to \( \text{PA1} \)
      • This will automatically copy zeros to \( A[i+\text{NB}/2] \) too

\[ \text{(PA1) Physical Address} \rightarrow \text{Subarray ID Map} \rightarrow \text{Subarray Mapping Table} \rightarrow \text{(PA}_{\text{ZERO})} \text{ Source row that contains all 0s} \]
Outline

• Background
  • DRAM Organization
  • Processing-using-Memory
  • Rocket Chip SoC Generator

• Overview of PiDRAM
  • Hardware & Software Components
  • Prototype

• Case Study #1 - RowClone
  • Challenges
  • Allocation Mechanism
  • Memory Coherence
  • Evaluation

• Installing and Using PiDRAM
Memory Coherence (I)

RowClone, AMBIT operates on data in DRAM
Up-to-date data may be in caches → coherency

Implement CLFLUSH in RISC-V rocket

Pros:
• **Realistic**, supported in contemporary architectures
• Reads and writes can hit in the cache. Flush cache lines prior to in-DRAM operations

Cons:
• Instruction overhead: One instruction per cache block
Other mechanisms that can alleviate the overheads

Vivek Seshadri, Abhishek Bhowmick, Onur Mutlu, Phillip B. Gibbons, Michael A. Kozuch, and Todd C. Mowry,
"The Dirty-Block Index"
[Slides (pptx) (pdf)] [Lightning Session Slides (pptx) (pdf)] [Source Code]
Other mechanisms that can alleviate the overheads

Outline

• Background
  • DRAM Organization
  • Processing-using-Memory
  • Rocket Chip SoC Generator

• Overview of PiDRAM
  • Hardware & Software Components
  • Prototype

• Case Study #1 - RowClone
  • Challenges
  • Allocation Mechanism
  • Memory Coherence
  • Evaluation

• Installing and Using PiDRAM
Evaluation: Methodology

Table 2: PiDRAM system configuration

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

Test two configurations:

1. **Bare-Metal**: No address translation
2. **No Flush**: OS support, we assume data is always up-to-date in DRAM

Workloads:

- **Microbenchmarks**: CPU-Copy, RowClone-Copy
- **SPEC2006**: libquantum
Copy Throughput Improvement (I)

RowClone-Copy provides over 365x performance improvement over rocket CPU-Copy.
RCC and RCI (with system support) improve copy throughput by 119x and 89x, respectively.
CLFLUSH Overhead

CLFLUSH dramatically reduces the potential improvement

PiDRAM enables the study of better coherence mechanisms (e.g., DBI)
• PiDRAM can run real workloads end-to-end:
  • Replace libquantum calloc() with alloc_align() and rci(): 1.3% performance improvement
• RowClone-Copy can greatly improve bulk data copy performance with (119x) and without (365x) system support
• RowClone requires efficient coherency management mechanisms to achieve its potential copy throughput improvement
PiDRAM implements RowClone and D-RaNGe

**AMBIT, SIMDRAAM**
- Memory allocation and alignment
- Memory coherence
- Transposition (SIMDRAM places data vertically)

**DLPUF, QUAC-TRNG**
- Memory scheduling policies
- Efficient post-processing integration
Potential Case Studies (II)

<table>
<thead>
<tr>
<th>PuM Technique</th>
<th>Description</th>
<th>Integration Challenges</th>
</tr>
</thead>
<tbody>
<tr>
<td>RowClone [102]</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 [67]</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 [99]</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>SIMDAM [46]</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 [66]</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 [89]</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>
PiDRAM vs Other Platforms

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>×</td>
<td>×</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>SoftMC [48]</td>
<td>✓ (DDR3)</td>
<td>×</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>ComputeDRAM [36]</td>
<td>✓ (DDR3)</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>MEG [103]</td>
<td>✓ (HBM)</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Simulators [20, 65, 84, 87]</td>
<td>×</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>

PiDRAM enables end-to-end integration of PuM techniques via
(1) Interface with real DRAM
(2) Providing a flexible memory controller design
(3) Open source design
(4) Support for system software
Outline

• Background
  • DRAM Organization
  • Processing-using-Memory
  • Rocket Chip SoC Generator

• Overview of PiDRAM
  • Hardware & Software Components
  • Prototype

• Case Study #1 - RowClone
  • Challenges
  • Allocation Mechanism
  • Memory Coherence
  • Evaluation

• Installing and Using PiDRAM
BACKUP SLIDES
Alloc alloc_align and RCC

1. Characterize RowClone Success Rate
2. Allocate 128 KiB A and B to same subarray
   A = alloc_align(128*1024, θ);
   B = alloc_align(128*1024, θ);
3. Initialize Subarray Mapping Table
4. Allocation ID Table
   Bank 7
   0 → SA0
   1 → SA3
5. Subarray Mapping Table
   Bank 7
   Entry – SA0
   Entry – SA1
6. VA_{A00} VA_{A16} VA_{B00} VA_{B16}
7. Page Table
<table>
<thead>
<tr>
<th>Virt. Addr.</th>
<th>Physical Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>VA_{A00}</td>
<td>B0 SA0 ROW0</td>
</tr>
<tr>
<td>VA_{A01}</td>
<td>B1 SA0 ROW0</td>
</tr>
<tr>
<td>VA_{A02}</td>
<td>B2 SA0 ROW0</td>
</tr>
<tr>
<td>VA_{A16}</td>
<td>B0 SA0 ROW0</td>
</tr>
<tr>
<td>VA_{A17}</td>
<td>B1 SA0 ROW0</td>
</tr>
<tr>
<td>VA_{B00}</td>
<td>B0 SA0 ROW4</td>
</tr>
<tr>
<td>VA_{B01}</td>
<td>B1 SA0 ROW4</td>
</tr>
</tbody>
</table>
8. Copy 128 KiBs from A to B
   rcc(A, B, 128*1024);
9. Access page table to find source and destination DRAM rows
10. Consecutive blocks are assigned to DRAM rows in different DRAM banks

Arrays are split into 4KB blocks