SIMDRAM: A Framework for Bit-Serial SIMD Processing using DRAM

Nastaran Hajinazar*  Geraldo F. Oliveira*
Sven Gregorio  Joao Ferreira  Nika Mansouri Ghiasi
Minesh Patel  Mohammed Alser  Saugata Ghose
Juan Gómez–Luna  Onur Mutlu

SAFARI  ETH Zürich  SFU  UNIVERSITY OF ILLINOIS
Data Movement Bottleneck

• Data movement is a major bottleneck

More than 60% of the total system energy is spent on data movement

Bandwidth-limited and power-hungry memory channel
Processing-in-Memory (PIM)

- **Processing-in-Memory**: moves computation closer to where the data resides
  - Reduces/eliminates the need to move data between processor and DRAM
Processing-using-Memory (PuM)

- **PuM**: Exploits analog operation principles of the memory circuitry to perform computation
  - Leverages the large internal bandwidth and parallelism available inside the memory arrays

- A common approach for PuM architectures is to perform bulk bitwise operations
  - Simple logical operations (e.g., AND, OR, XOR)
  - More complex operations (e.g., addition, multiplication)
Motivation, Goal, and Key Idea

• Existing PuM mechanisms are not widely applicable
  - Support only a limited and mainly basic set of operations
  - Lack the flexibility to support new operations
  - Require significant changes to the DRAM subarray

• **Goal:** Design a PuM framework that
  - Efficiently implements complex operations
  - Provides the flexibility to support new desired operations
  - Minimally changes the DRAM architecture

• **SIMDRAM:** An end-to-end processing-using-DRAM framework that provides the programming interface, the ISA, and the hardware support for:
  - Efficiently computing complex operations in DRAM
  - Providing the ability to implement arbitrary operations as required
  - Using an in-DRAM massively-parallel SIMD substrate that requires minimal changes to DRAM architecture
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)

2. Majority-based computation
   - Formula: $C_{out} = AB + AC_{in} + BC_{in}$

Pros compared to the conventional horizontal layout:
- Implicit shift operation
- Massive parallelism

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

**User Input**
- Desired operation
  - AND/OR/NOT logic

**Step 1: Generate MAJ logic**
- MAJ/NOT logic

**Step 2: Generate sequence of DRAM commands**
- ACT/PRE
- ACT/PRE
- ACT/PRE
- ACT/ACT/PRE
- done

**SIMDRAM Output**
- New SIMDRAM µProgram
- µProgram
- bbop_new
- New SIMDRAM instruction

**Main memory**

**User Input**
- SIMDRAM-enabled application

```plaintext
foo () {
    bbop_new
}
```

**Step 3: Execution according to µProgram**
- Control Unit
- µProgram
- done

**SIMDRAM Output**
- Instruction result in memory

**Memory Controller**
**SIMDRAM Framework: Overview**

Step 1:
- Builds an **efficient MAJ/NOT representation** of a given desired operation from its AND/OR/NOT-based implementation
SIMDRAM Framework: Overview

**Step 1:** Generate MAJ logic

**Step 2:** Generate sequence of DRAM commands

**New SIMDRAM μProgram**

- **μProgram**
- Main memory

- **bbop_new**
- New SIMDRAM instruction

**User Input**

- Desired operation
- AND/OR/NOT logic

**SIMDRAM Output**

- μProgram
- Instruction result in memory

**Step 2:**

- **Allocates DRAM** rows to the operation’s inputs and outputs
- Generates the **sequence of DRAM commands** (μProgram) to execute the desired operation
**Step 3:**

- **Executes the μProgram** to perform the operation
- **Uses a control unit** in the memory controller
SIMDRAM Framework: Overview

**User Input**

- Desired operation
- AND/OR/NOT logic

**Step 1: Generate MAJ logic**

- MAJ/NOT logic

**Step 2: Generate sequence of DRAM commands**

- ACT/PRE
- ACT/PRE
- ACT/PRE
- ACT/ACT/PRE
- done

**SIMDRAM Output**

- New SIMDRAM μProgram
- Main memory
- bbop_new
- New SIMDRAM instruction

**User Input**

- SIMDRAM-enabled application
  ```
  foo () {
    bbop_new
  }
  ```

**Step 3: Execution according to μProgram**

- Memory Controller

**SIMDRAM Output**

- Instruction result in memory
- ACT/PRE

SAFARI
Key Results

Evaluated on:
- 16 complex in-DRAM operations
- 7 commonly-used real-world applications

SIMDRAM provides:

- $88\times$ and $5.8\times$ the **throughput** of a **CPU** and a **high-end GPU**, respectively, over 16 operations

- $257\times$ and $31\times$ the **energy efficiency** of a **CPU** and a **high-end GPU**, respectively, over 16 operations

- $21\times$ and $2.1\times$ the **performance** of a **CPU** and a **high-end GPU**, over seven real-world applications

*SAFARI*
Conclusion

• **SIMDRAM:**
  - Enables *efficient* computation of a *flexible* set and wide range of operations in a PuM *massively parallel* SIMD substrate
  - Provides the hardware, programming, and ISA support, to:
    • Address key *system integration* challenges
    • Allow programmers to define and employ *new operations* without hardware changes

**SIMDRAM is a promising PuM framework**

• Can *ease the adoption* of processing-using-DRAM architectures
• Improve the *performance* and *efficiency* of processing-using-DRAM architectures
SIMDRAM: A Framework for Bit-Serial SIMD Processing using DRAM

Nastaran Hajinazar*
Sven Gregorio
Minesh Patel
Juan Gómez–Luna

Geraldo F. Oliveira*
Joao Ferreira
Mohammed Alser
Onur Mutlu

SafARI

SFU SIMON FRASER UNIVERSITY

ETH Zürich

UNIVERSITY OF ILLINOIS

URBANA–CHAMPAIGN