P&S Processing-in-Memory

Real-World Processing-in-Memory Architectures

Dr. Juan Gómez Luna
Prof. Onur Mutlu

ETH Zürich
Fall 2021
12 October 2021
PIM Becomes Real

- **UPMEM**, founded in January 2015, announces the first real-world PIM architecture in 2016
- UPMEM’s PIM-enabled DIMMs start getting commercialized in 2019

- In early 2021, **Samsung announces FIMDRAM** at ISSCC conference
- Samsung’s LP-DDR5 and DDR5 announced a few months later

https://www.eenewsautomotive.com/news/startup-plans-embed-processors-dram-0#
Samsung Develops Industry’s First High Bandwidth Memory with AI Processing Power

Korea on February 17, 2021

The new architecture will deliver over twice the system performance and reduce energy consumption by more than 70%

Samsung Electronics, the world leader in advanced memory technology, today announced that it has developed the industry’s first High Bandwidth Memory (HBM) integrated with artificial intelligence (AI) processing power — the HBM-PIM. The new processing-in-memory (PIM) architecture brings powerful AI computing capabilities inside high-performance memory, to accelerate large-scale processing in data centers, high performance computing (HPC) systems and AI-enabled mobile applications.

Kwangil Park, senior vice president of Memory Product Planning at Samsung Electronics stated, “Our groundbreaking HBM-PIM is the industry’s first programmable PIM solution tailored for diverse AI-driven workloads such as HPC, training and inference. We plan to build upon this breakthrough by further collaborating with AI solution providers for even more advanced PIM-powered applications.”

UPMEM Processing-in-DRAM Engine (2019)

- Processing in DRAM Engine

- Includes **standard DIMM modules**, with a **large number of DPU processors** combined with DRAM chips.

- Replaces **standard** DIMMs
  - DDR4 R-DIMM modules
    - 8GB+128 DPUs (16 PIM chips)
    - Standard 2x-nm DRAM process
  - Large amounts of compute & memory bandwidth

Samsung Function-in-Memory DRAM (2021)

- FIMDRAM based on HBM2

![3D Chip Structure of HBM with FIMDRAM]

**Chip Specification**

- 128DQ / 8CH / 16 banks / BL4
- 32 PCU blocks (1 FIM block/2 banks)
- 1.2 TFLOPS (4H)
- FP16 ADD / Multiply (MUL) / Multiply-Accumulate (MAC) / Multiply-and- Add (MAD)

---

**ISSCC 2021 / SESSION 25 / DRAM / 25.4**

25.4 A 20nm 6GB Function-In-Memory DRAM, Based on HBM2 with a 1.2TFLOPS Programmable Computing Unit Using Bank-Level Parallelism, for Machine Learning Applications

Young-Cheon Kwon\(^1\), Suk Han Lee\(^2\), Jaehoon Lee\(^3\), Sang-Hyuk Kwon\(^4\), Je Min Ryu\(^1\), Jong-Pil Son\(^1\), Seongil O\(^5\), Hak-Soo Yu\(^6\), Haessuk Lee\(^1\), Soo Young Kim\(^1\), Youngmin Cho\(^1\), Jin Guk Kim\(^1\), Jongyoon Choi\(^1\), Hyun-Sung Shin\(^1\), Jin Kim\(^1\), BangSeng Phua\(^1\), HyoungMin Kim\(^1\), Myeong Jun Song\(^1\), Ahn Choi\(^1\), Daeho Kim\(^1\), SooYoung Kim\(^1\), Eun-Bong Kim\(^1\), David Wang\(^2\), Shinhueung Kang\(^1\), Yuhwan Ro\(^1\), Seungwoo Seo\(^1\), JoonHo Song\(^1\), Jaryoun Youn\(^1\), Kyomin Sohn\(^1\), Nam Sung Kim\(^1\)

\(^1\)Samsung Electronics, Hwasung, Korea  
\(^2\)Samsung Electronics, San Jose, CA 
\(^3\)Samsung Electronics, Suwon, Korea
Samsung Function-in-Memory DRAM (2021)

Chip Implementation

- Mixed design methodology to implement FIMDRAM
  - Full-custom + Digital RTL

---

[Digital RTL design for PCU block]
Samsung AxDIMM (2021)

- DDR5-PIM
  - DLRM recommendation system

UPMEM PIM
Microarchitecture and ISA
UPMEM DIMMs

- E19: 8 chips/DIMM (1 rank). DPUs @ 267 MHz
- P21: 16 chips/DIMM (2 ranks). DPUs @ 350 MHz
PIM’s Promises

UPMEM PIM massive benefits

- Massive speed-up
  - Massive additional compute & bandwidth
- Massive energy gains
  - Most data movement on chip
- Low cost
  - ~300$ of additional DRAM silicon
  - Affordable programming
- Massive ROI / TCO gains

<table>
<thead>
<tr>
<th>Energy efficiency when computing on or off memory chip</th>
<th>Server + PIM DRAM</th>
<th>Server + normal DRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRAM to processor 64-bit operand</td>
<td>pJ ~150</td>
<td>~3000*</td>
</tr>
<tr>
<td>Operation</td>
<td>pJ ~20</td>
<td>~10*</td>
</tr>
<tr>
<td>Server consumption</td>
<td>W ~700W</td>
<td>~300W</td>
</tr>
<tr>
<td>speed-up</td>
<td>~ x20</td>
<td>x1</td>
</tr>
<tr>
<td>energy gain</td>
<td>~ x10</td>
<td>x1</td>
</tr>
<tr>
<td>TCO gain</td>
<td>~ x10</td>
<td>x1</td>
</tr>
</tbody>
</table>

Technology Challenges

The Hurdles on the road to the Graal

- DRAM process highly constrained
  - 3x slower transistors than same node digital process
  - Logic 10 times less dense vs. ASIC process
  - Routing density dramatically lower
    - 3 metals only for routing (vs. 10+), pitch x4 larger
- Strong design choices mandatory

But the PIM Graal is worth it!

Take away

DRAM vs. ASIC
- Far less performing
- Wafers 2x cheaper vs. ASIC

Leapfrogging Moore’s law
- Total Energy efficiency x10
- Massive, scalable parallelism
- Very low cost

Copyright UPMEM® 2019

F. Devaux, "The true Processing In Memory accelerator," HotChips 2019. doi: 10.1109/HOTCHIPS.2019.8875680
A memory circuit having: a memory array including one or more memory banks; a first processor; and a processor control interface for receiving data processing commands directed to the first processor from a central processor, the processor control interface being adapted to indicate to the central processor when the first processor has finished accessing one or more of the memory banks of the memory array, these memory banks becoming accessible to the central processor.
Accelerator Model (I)

• UPMEM DIMMs coexist with conventional DIMMs

• Integration of UPMEM DIMMs in a system follows an accelerator model

• UPMEM DIMMs can be seen as a loosely coupled accelerator
  - Explicit data movement between the main processor (host CPU) and the accelerator (UPMEM)
  - Explicit kernel launch onto the UPMEM processors

• This resembles GPU computing
GPU Computing

- Computation is **offloaded to the GPU**
- Three steps
  - CPU-GPU data transfer (1)
  - GPU kernel execution (2)
  - GPU-CPU data transfer (3)

[Diagram showing CPU and GPU with data transfer steps]

https://www.youtube.com/watch?v=y40-tY5WJ8A
Lecture on GPU Programming

Indexing and Memory Access: 1D Grid
- One GPU thread per pixel
- Grid of Blocks of Threads
  - gridDim.x, blockDim.x
  - blockIdx.x, threadIdx.x


2,072 views • May 24, 2018

https://www.youtube.com/watch?v=y40-tY5WJ8A
227-0085-51L Projects & Seminars: Hands-on Acceleration on Heterogeneous Computing Systems

Semester: Autumn Semester 2021
Lecturers: O. Mutlu, J. Gómez Luna
Periodicity: every semester recurring course
Language of instruction: English
Comment: Only for Electrical Engineering and Information Technology BSc.

Course can only be registered for once. A repeatedly registration in a later semester is not chargeable.

Abstract
The category of "Laboratory Courses, Projects, Seminars" includes courses and laboratories in various formats designed to impart practical knowledge and skills. Moreover, these classes encourage independent experimentation and design, allow for explorative learning and teach the methodology of project work.

Objective
The increasing difficulty of scaling the performance and efficiency of CPUs every year has created the need for turning computers into heterogeneous systems, i.e., systems composed of multiple types of processors that can suit better different types of workloads or parts of them. More than a decade ago, Graphics Processing Units (GPUs) became general-purpose parallel processors, in order to make their outstanding processing capabilities available to many workloads beyond graphics. GPUs have been critical key to the recent rise of Machine Learning and Artificial Intelligence, which took unrealistic training times before the use of GPUs. Field-Programmable Gate Arrays (FPGAs) are another example computing device that can deliver impressive benefits in terms of performance and energy efficiency. More specific examples are (1) a plethora of specialized accelerators (e.g., Tensor Processing Units for neural networks), and (2) near-data processing architectures (i.e., placing compute capabilities near or inside memory/storage).

Despite the great advances in the adoption of heterogeneous systems in recent years, there are still many challenges to tackle, for example:
- Heterogeneous implementations (using GPUs, FPGAs, TPUs) of modern applications from important fields such as bioinformatics, machine learning, graph processing, medical imaging, personalized medicine, robotics, virtual reality, etc.
- Scheduling techniques for heterogeneous systems with different general-purpose processors and accelerators, e.g., kernel offloading, memory scheduling, etc.
- Workload characterization and programming tools that enable easier and more efficient use of heterogeneous systems.

If you are enthusiastic about working hands-on with different software, hardware, and architecture projects for heterogeneous systems, this is your P&S. You will have the opportunity to program heterogeneous systems with different types of devices (CPUs, GPUs, FPGAs, TPUs), propose algorithmic changes to important applications to better leverage the compute power of heterogeneous systems, understand different workloads and identify the most suitable device for their execution, design optimized scheduling techniques, etc. In general, the goal will be to reach the highest performance reported for a given important application.
• **FIG. 6** is a flow diagram representing operations in a method of delegating a processing task to a DRAM processor according to an example embodiment.
System Organization (I)

- FIG. 1 schematically illustrates a computing system comprising DRAM circuits having integrated processors according to an example embodiment.
System Organization (II)

- In a UPMEM-based PIM system UPMEM DIMMs coexist with regular DDR4 DIMMs
System Organization (III)

- A UPMEM DIMM contains 8 or 16 chips
  - Thus, 1 or 2 ranks of 8 chips each
- Inside each PIM chip there are:
  - 8 64MB banks per chip: Main RAM (MRAM) banks
  - 8 DRAM Processing Units (DPUs) in each chip, 64 DPUs per rank
2,560-DPU System (I)

- UPMEM-based PIM system with 20 UPMEM DIMMs of 16 chips each (40 ranks)
  - P21 DIMMs
  - Dual x86 socket
  - UPMEM DIMMs coexist with regular DDR4 DIMMs
  - 2 memory controllers/socket (3 channels each)
  - 2 conventional DDR4 DIMMs on one channel of one controller

* There are 4 faulty DPUs in the system that we use in our experiments. Thus, the maximum number of DPUs we can use is 2,556.
2,560-DPU System (II)
640-DPU System

- UPMEM-based PIM system with 10 UPMEM DIMMs of 8 chips each (10 ranks)
  - E19 DIMMs
  - x86 socket
    - 2 memory controllers (3 channels each)
    - 2 conventional DDR4 DIMMs on one channel of one controller
DPU Sharing? Security Implications?

• DPUs cannot be shared across multiple CPU processes
  - There are so many DPUs in the system that there is no need for sharing

• According to UPMEM, this assumption makes things simpler
  - No need for OS
  - Simplified security implications: No side channels
Vector Addition (VA)

• Our first programming example
• We partition the input arrays across:
  - DPUs
  - Tasklets, i.e., software threads running on a DPU
CPU-DPU/DPU-CPU Data Transfers

- CPU-DPU and DPU-CPU transfers
  - Between host CPU’s main memory and DPUs’ MRAM banks

- Serial CPU-DPU/DPU-CPU transfers:
  - A single DPU (i.e., 1 MRAM bank)

- Parallel CPU-DPU/DPU-CPU transfers:
  - Multiple DPUs (i.e., many MRAM banks)

- Broadcast CPU-DPU transfers:
  - Multiple DPUs with a single buffer
Inter-DPU Communication

- There is **no direct communication channel between DPUs**

- **Inter-DPU communication takes places via the host CPU** using CPU-DPU and DPU-CPU transfers

- **Example communication patterns:**
  - Merging of partial results to obtain the final result
    - Only DPU-CPU transfers
  - Redistribution of intermediate results for further computation
    - DPU-CPU transfers and CPU-DPU transfers
FIG. 4 schematically illustrates part of the computing system of FIG. 1 in more detail according to an example embodiment.
DRAM Processing Unit (II)

PIM Chip

Control/Status Interface

DDR4 Interface

24-KB IRAM

DMA Engine

64-KB WRAM

64-MB DRAM Bank (MRAM)

Register File

Pipeline

DISPATCH
FETCH1
FETCH2
FETCH3
READOP1
READOP2
READOP3
FORMAT
ALU1
ALU2
ALU3
ALU4
MERGE1
MERGE2
DPU Pipeline

• In-order pipeline
  - Up to 350 MHz

• Fine-grain multithreaded
  - 24 hardware threads

• 14 pipeline stages
  - DISPATCH: Thread selection
  - FETCH: Instruction fetch
  - READOP: Register file
  - FORMAT: Operand formatting
  - ALU: Operation and WRAM
  - MERGE: Result formatting
Fine-Grained Multithreading
Fine-Grained Multithreading

- **Idea:** Hardware has multiple thread contexts (PC+registers). Each cycle, fetch engine fetches from a different thread.
  - By the time the fetched branch/instruction resolves, no instruction is fetched from the same thread.
  - Branch/instruction resolution latency overlapped with execution of other threads’ instructions.

  + No logic needed for handling control and data dependences within a thread.
  -- Single thread performance suffers.
  -- Extra logic for keeping thread contexts.
  -- Does not overlap latency if not enough threads to cover the whole pipeline.
Fine-Grained Multithreading (II)

- **Idea:** Switch to another thread every cycle such that no two instructions from a thread are in the pipeline concurrently.
  - Tolerates the control and data dependence latencies by overlapping the latency with useful work from other threads.
  - Improves pipeline utilization by taking advantage of multiple threads.

Lecture on Fine-Grained Multithreading

Fine-Grained Multithreading

- **Idea:** Hardware has multiple thread contexts (PC+registers). Each cycle, fetch engine fetches from a different thread.
  - By the time the fetched branch/instruction resolves, no instruction is fetched from the same thread.
  - Branch/instruction resolution latency overlapped with execution of other threads’ instructions.

+ No logic needed for handling control and data dependences within a thread
  -- Single thread performance suffers
  -- Extra logic for keeping thread contexts
  -- Does not overlap latency if not enough threads to cover the whole pipeline

https://www.youtube.com/watch?v=6e5KZcCGBYw&list=PL5Q2soXY2Zi_uezj3aY39YB5pfW4Sj7LlN&index=16
Lectures on Fine-Grained Multithreading

  - Pipelined Processor Design (ETH, Spring 2021)
  - https://www.youtube.com/watch?v=6e5KZcCGBYw&list=PL5Q2soXY2Zi_uej3aY39YB5pfW4SJ7LIN&index=16

- Digital Design & Computer Architecture, Spring 2020, Lecture 18c
  - Fine-Grained Multithreading (ETH, Spring 2020)
  - https://www.youtube.com/watch?v=bu5dxKTvQVs&list=PL5Q2soXY2Zi_FRrloMa2fuYWPGiZUBQo2&index=26

https://www.youtube.com/onurmutlulectures
DPU Pipeline

• In-order pipeline
  - Up to 350 MHz

• Fine-grain multithreaded
  - 24 hardware threads

• 14 pipeline stages
  - DISPATCH: Thread selection
  - FETCH: Instruction fetch
  - READOP: Register file
  - FORMAT: Operand formatting
  - ALU: Operation and WRAM
  - MERGE: Result formatting
DPU Instruction Set Architecture

• Specific 32-bit ISA
  - Aiming at scalar, in-order, and multithreaded implementation
  - Allowing compilation of 64-bit C code
  - LLVM/Clang compiler

Instruction Set Architecture

This section covers the architecture concepts required to understand and use UPMEM DPU processor as a software developer. It is also providing an exhaustive list of the available processor instructions.

Software developers should use this section as a reference manual to develop or debug assembly code.

Resources overview

Thread registers

The system is composed of 24 hardware threads. Each of them owns a set of private resources:

- 24 general purpose 32-bits registers named $r0$ through $r23$
- A 16-bits wide program counter, named PC. Notice that the PC value does not address an instruction in memory, but the index of such an instruction directly. For example, a PC equal to 1 represents the second instruction in the DPU's program memory.
- Two persistent flags, keeping information about the previous result of an arithmetic or logical instruction:
  - ZF: last result is equal to zero

https://sdk.upmem.com/2021.2.0/201_IS.html#
Microbenchmark for INT32 ADD Throughput

C-based code

```c
#define SIZE 256
int* bufferA = mem_alloc(SIZE * sizeof(int));
for(int i = 0; i < SIZE; i++){
    int temp = bufferA[i];
    temp += scalar;
    bufferA[i] = temp;
}
```

Compiled code (UPMEM DPU ISA)

```assembly
move r2, 0
.LBB0_1:  // Loop header
    lsl_add r3, r0, r2, 2  // Address calculation
    lw r4, r3, 0           // Load from WRAM
    add r4, r4, r1         // Add
    sw r3, 0, r4           // Store to WRAM
    add r2, r2, 1          // Index update
    jneq r2, 256, .LBB0_1  // Conditional jump
```
Arithmetic Throughput: #Instructions

- Compiler explorer: https://dpu.dev

```c
#define BLOCK_SIZE 1024

typedef int T;

void Benchmark_32bits(T *cache_A, T scalar) {
    for (int i = 0; i < BLOCK_SIZE / sizeof(T); i++){
        // WRAM READ
        T temp = cache_A[i];

        temp += scalar; // ADD

        // WRAM WRITE
        cache_A[i] = temp;
    }
}

typedef long T_long;

void Benchmark_64bits(T_long *cache_A, T_long scalar) {
    for (int i = 0; i < BLOCK_SIZE / sizeof(T_long); i++){
        // WRAM READ
        T_long temp = cache_A[i];

        temp += scalar; // ADD
    }
}
```

6 instructions in the 32-bit ADD/SUB microbenchmark
7 instructions in the 64-bit ADD/SUB microbenchmark
DPU: WRAM Bandwidth

PIM Chip

Control/Status Interface → DDR4 Interface

- DISPATCH
- FETCH1
- FETCH2
- FETCH3
- READOP1
- READOP2
- READOP3
- FORMAT
- ALU1
- ALU2
- ALU3
- ALU4
- MERGE1
- MERGE2

24-KB IRAM

DMA Engine

64-MB DRAM Bank (MRAM)
DPU: MRAM Latency and Bandwidth

**PIM Chip**

Control/Status Interface  ---  DDR4 Interface

- DISPATCH
- FETCH1
- FETCH2
- FETCH3
- READOP1
- READOP2
- READOP3
- FORMAT
- ALU1
- ALU2
- ALU3
- ALU4
- MERGE1
- MERGE2

**Pipeline Register File**

- 24-KB IRAM
- 64-KB WRAM

**DMA Engine**

- 64-MB DRAM Bank (MRAM)

- 64 bits
DPU: Arithmetic Throughput vs. Operational Intensity

**PIM Chip**

- Control/Status Interface
- DDR4 Interface
- 24-KB IRAM
- 64-GB RAM Bank (MRAM)
- 64-KB WRAM
- DMA Engine
- 64 bits
Upcoming Lectures

- Microbenchmarking of the UPMEM DPU
  - Compute throughput
  - MRAM and WRAM bandwidth
  - Arithmetic intensity versus compute throughput
- Programming an UPMEM-based PIM system
- Introduction to Samsung’s PIM devices
Experimental Analysis of the UPMEM PIM Engine

Benchmarking a New Paradigm: An Experimental Analysis of a Real Processing-in-Memory Architecture

JUAN GÓMEZ-LUNA, ETH Zürich, Switzerland
IZZAT EL HAJJ, American University of Beirut, Lebanon
IVAN FERNANDEZ, ETH Zürich, Switzerland and University of Malaga, Spain
CHRISTINA GIANNOLA, ETH Zürich, Switzerland and NTUA, Greece
GERALDO F. OLIVEIRA, ETH Zürich, Switzerland
ONUR MUTLU, ETH Zürich, Switzerland

Many modern workloads, such as neural networks, databases, and graph processing, are fundamentally memory-bound. For such workloads, the data movement between main memory and CPU cores imposes a significant overhead in terms of both latency and energy. A major reason is that this communication happens through a narrow bus with high latency and limited bandwidth, and the low data reuse in memory-bound workloads is insufficient to amortize the cost of main memory access. Fundamentally addressing this data movement bottleneck requires a paradigm where the memory system assumes an active role in computing by integrating processing capabilities. This paradigm is known as processing-in-memory (PIM).

Recent research explores different forms of PIM architectures, motivated by the emergence of new 3D-stacked memory technologies that integrate memory with a logic layer where processing elements can be easily placed. Past works evaluate these architectures in simulation or, at best, with simplified hardware prototypes. In contrast, the UPMEM company has designed and manufactured the first publicly-available real-world PIM architecture. The UPMEM PIM architecture combines traditional DRAM memory arrays with general-purpose in-order cores, called DRAM Processing Units (DPUs), integrated in the same chip.

This paper provides the first comprehensive analysis of the first publicly-available real-world PIM architecture. We make two key contributions. First, we conduct an experimental characterization of the UPMEM-based PIM system using microbenchmarks to assess various architecture limits such as compute throughput and memory bandwidth, yielding new insights. Second, we present PrIM (Processing-In-Memory benchmarks), a benchmark suite of 16 workloads from different application domains (e.g., dense/sparse linear algebra, databases, data analytics, graph processing, neural networks, bioinformatics, image processing), which we identify as memory-bound. We evaluate the performance and scaling characteristics of PrIM benchmarks on the UPMEM PIM architecture, and compare their performance and energy consumption to their state-of-the-art CPU and GPU counterparts. Our extensive evaluation conducted on two real UPMEM-based PIM systems with 640 and 2,556 DPUs provides new insights about suitability of different workloads to the PIM system, programming recommendations for software designers, and suggestions and hints for hardware and architecture designers of future PIM systems.

Understanding a Modern PIM Architecture

Juan Gómez Luna, Izzat El Hajj, Ivan Fernandez, Christina Giannoula, Geraldo F. Oliveira, Onur Mutlu

https://github.com/CMU-SAFARI/prim-benchmarks

SAFARI Live Seminar: Understanding a Modern Processing-in-Memory Architecture
2,579 views • Streamed live on Jul 12, 2021

Onur Mutlu Lectures
18.7K subscribers

https://www.youtube.com/watch?v=D8Hyj2iU9I4&list=PL5Q2soXY2Zi_tOTAYm--dYByNPL7JhwR9
P&S Processing-in-Memory

Real-World Processing-in-Memory Architectures

Dr. Juan Gómez Luna
Prof. Onur Mutlu

ETH Zürich
Fall 2021
12 October 2021