

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



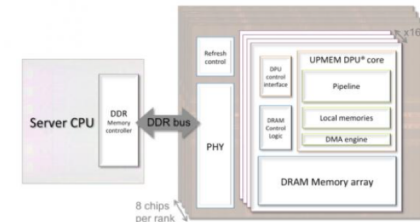
ABOUT FEATURED ARTICLES LEARNING CENTER NEWS

ABOUT About eeNews Europe Automotive Contact

Startup plans to embed processors in DRAM

October 13, 2016 // By Peter Clarke

Email print Share in Share reddit



Fabless chip company Upmem SAS (Grenoble, France), founded in January 2015, is developing a microprocessor for use in data-intensive applications in the datacenter that will sit embedded in DRAM to be close to the data.

Placing hundreds or thousands of processing elements in DRAM able to perform work for a controlling server CPU could have a revolutionary impact on how data

Samsung Function-in-Memory DRAM (2021)



Samsung Develops Industry's First High Bandwidth Memory with AI Processing Power

Korea on February 17, 2021

Audio



Share



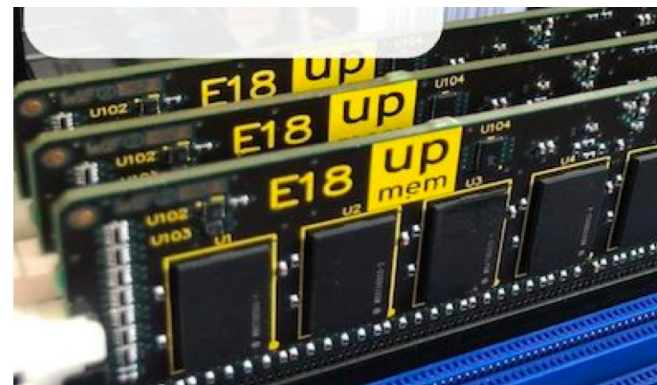
The new architecture will deliver over twice the system performance and reduce energy consumption by more than 70%

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."

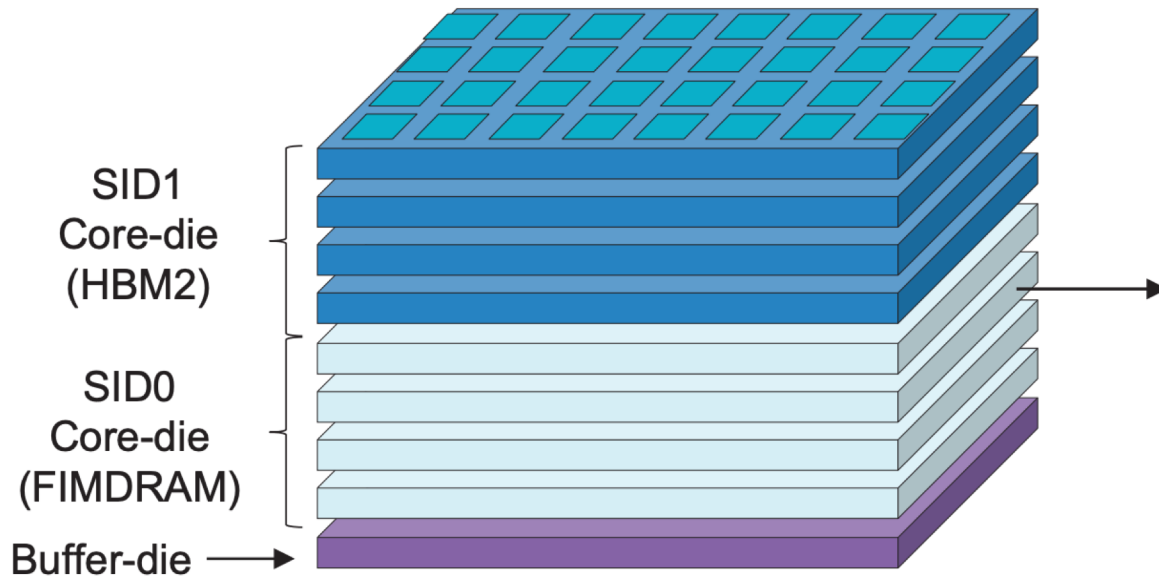
UPMEM Processing-in-DRAM Engine (2019)

- **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 Kwon¹, Suk Han Lee¹, Jaehoon Lee¹, Sang-Hyuk Kwon¹, Je Min Ryu¹, Jong-Pil Son¹, Seongil O¹, Hak-Soo Yu¹, Haesuk Lee¹, Soo Young Kim¹, Youngmin Cho¹, Jin Guk Kim¹, Jongyoon Choi¹, Hyun-Sung Shin¹, Jin Kim¹, BengSeng Phuah¹, HyoungMin Kim¹, Myeong Jun Song¹, Ahn Choi¹, Daeho Kim¹, SooYoung Kim¹, Eun-Bong Kim¹, David Wang², Shinhaeng Kang¹, Yuhwan Ro³, Seungwoo Seo³, JoonHo Song³, Jaeyoun Youn¹, Kyomin Sohn¹, Nam Sung Kim¹

¹Samsung Electronics, Hwaseong, Korea

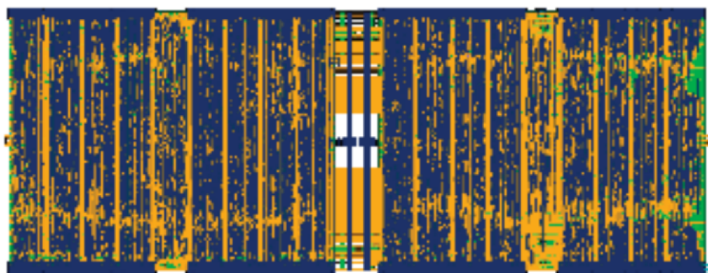
²Samsung Electronics, San Jose, CA

³Samsung Electronics, Suwon, Korea

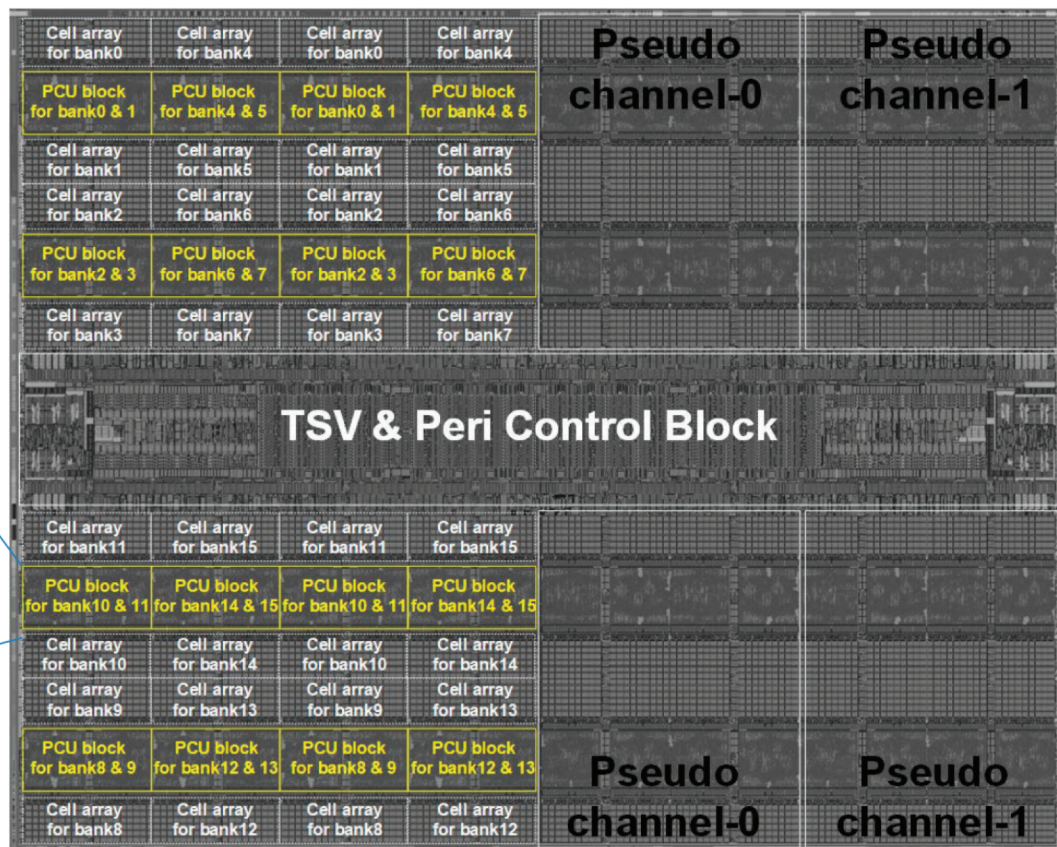
Samsung Function-in-Memory DRAM (2021)

Chip Implementation

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



[Digital RTL design for PCU block]



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 Kwon¹, Suk Han Lee¹, Jaehoon Lee¹, Sang-Hyuk Kwon¹, Je Min Ryu¹, Jong-Pil Son¹, Seongil O¹, Hak-Soo Yu¹, Haesuk Lee¹, Soo Young Kim¹, Youngmin Cho¹, Jin Guk Kim¹, Jongyeon Choi¹, Hyun-Sung Shim¹, Jin Kim¹, BengSeng Phuah¹, HyounMin Kim¹, Myeong Jun Song¹, Ahn Chai¹, Daeho Kim¹, Sooyoung Kim¹, Eun-Bong Kim¹, David Wang¹, Shinhaeng Kang¹, Yuhwan Ro¹, Seungwoo Seo¹, JoonHo Song¹, Jaeyoun Yoon¹, Kyomin Sohn¹, Nam Sung Kim¹

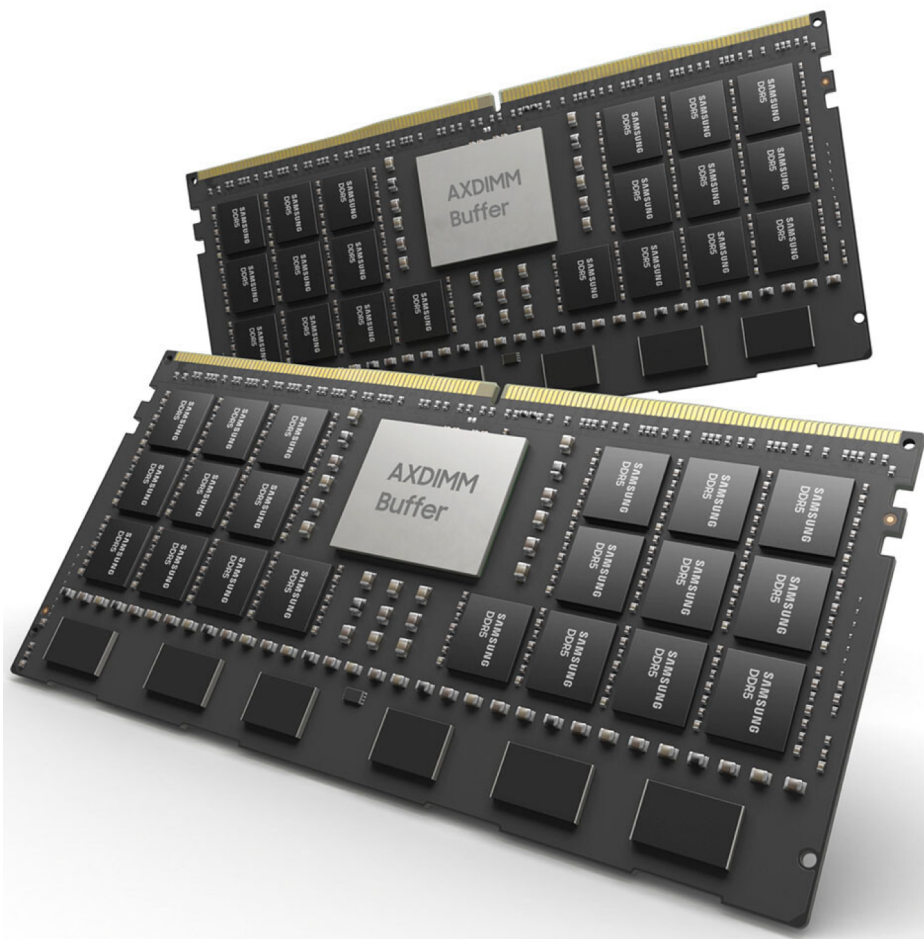
¹Samsung Electronics, Hwaseong, Korea

²Samsung Electronics, San Jose, CA

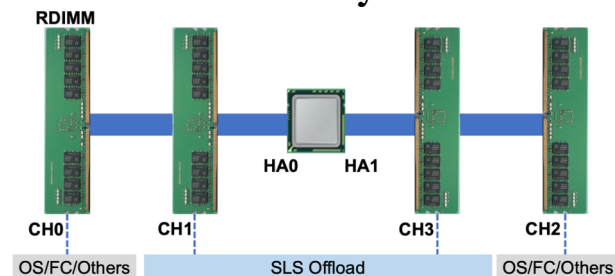
³Samsung Electronics, Suwon, Korea

Samsung AxDIMM (2021)

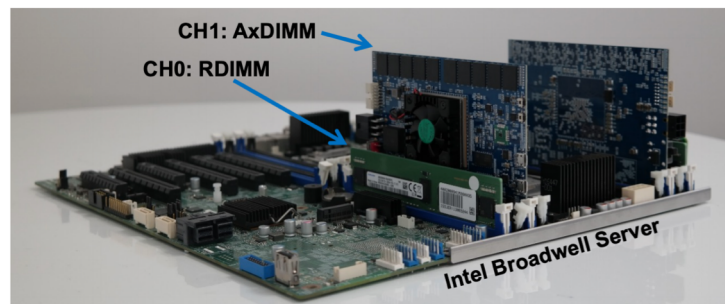
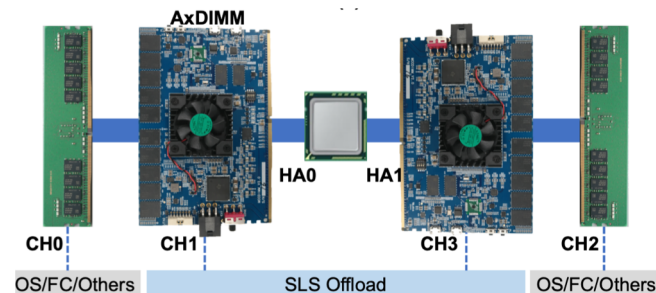
- DDR5-PIM
 - DLRM recommendation system



Baseline System



AxDIMM System

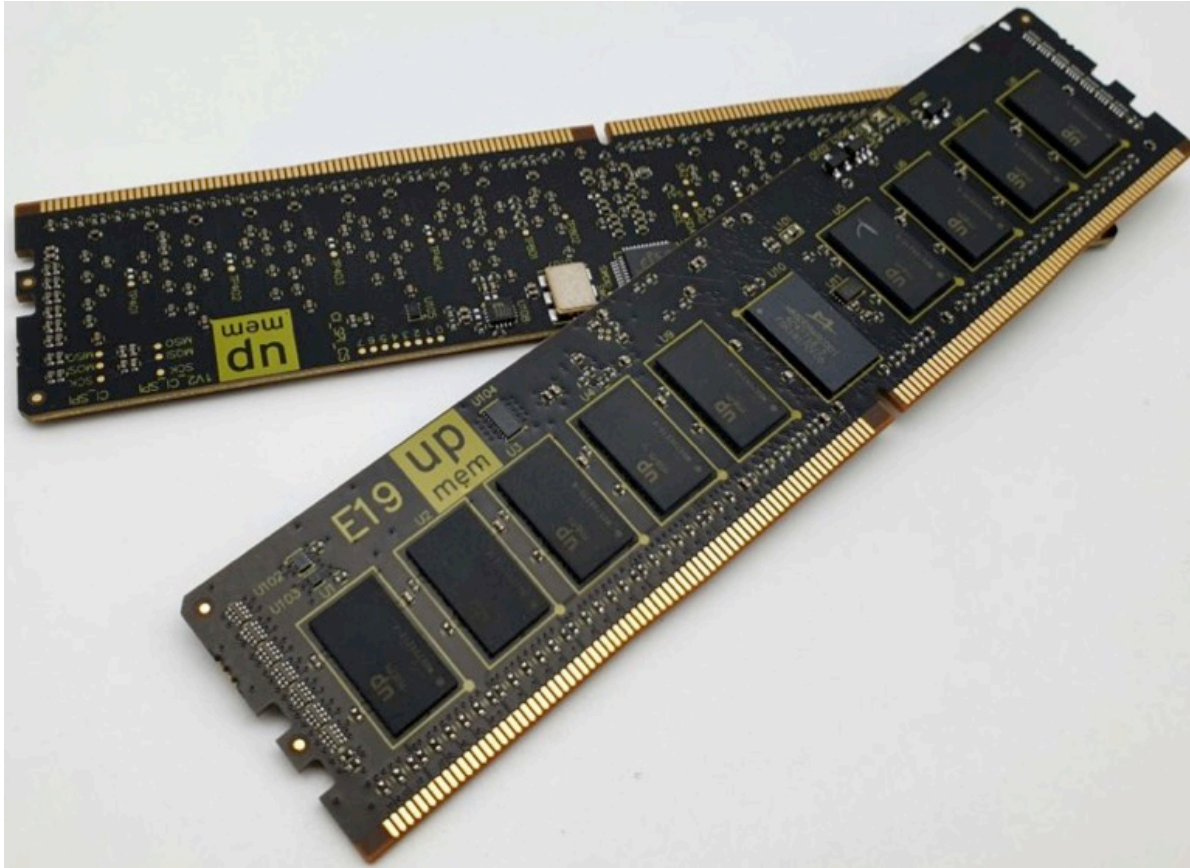


UPMEM PIM

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

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

**Exascale Computing Trends: Adjusting to the "New Normal" for Computer Architecture; John Shalf, Computing in Science & engineering, 2013*

Copyright UPMEM® 2019

Authorized licensed use limited to: ETH BIBLIOTHEK ZURICH. Downloaded on September 04, 2020 at 13:55:41 UTC from IEEE Xplore. Restrictions apply.

HOT CHIPS 31

up
mem

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

Authorized licensed use limited to: ETH BIBLIOTHEK ZURICH. Downloaded on September 04, 2020 at 13:55:41 UTC from IEEE Xplore. Restrictions apply.

HOT CHIPS 31

up
mem

UPMEM Patent

(12) United States Patent		(10) Patent No.: US 10,324,870 B2	
Devaux et al.		(45) Date of Patent: Jun. 18, 2019	
(54) MEMORY CIRCUIT WITH INTEGRATED PROCESSOR		(56) References Cited	
		U.S. PATENT DOCUMENTS	
(71) Applicant: UPMEM , Grenoble (FR)		5,666,485 A *	9/1997 Suresh G06F 13/1605 710/113
(72) Inventors: Fabrice Devaux , La Conversion (CH); Jean-François Roy , Grenoble (FR)		6,463,001 B1	10/2002 Williams
		7,349,277 B2 *	3/2008 Kinsley G11C 11/406 365/193
(73) Assignee: UPMEM , Grenoble (FR)		8,438,358 B1 *	5/2013 Kraipak G11C 7/04 711/167
(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.		(Continued)	
		FOREIGN PATENT DOCUMENTS	
(21) Appl. No.: 15/551,418		EP	0780768 A1 6/1997
		JP	H03109661 A 5/1991
(22) PCT Filed: Feb. 12, 2016		WO	2010/141221 A1 12/2010

(57)

ABSTRACT

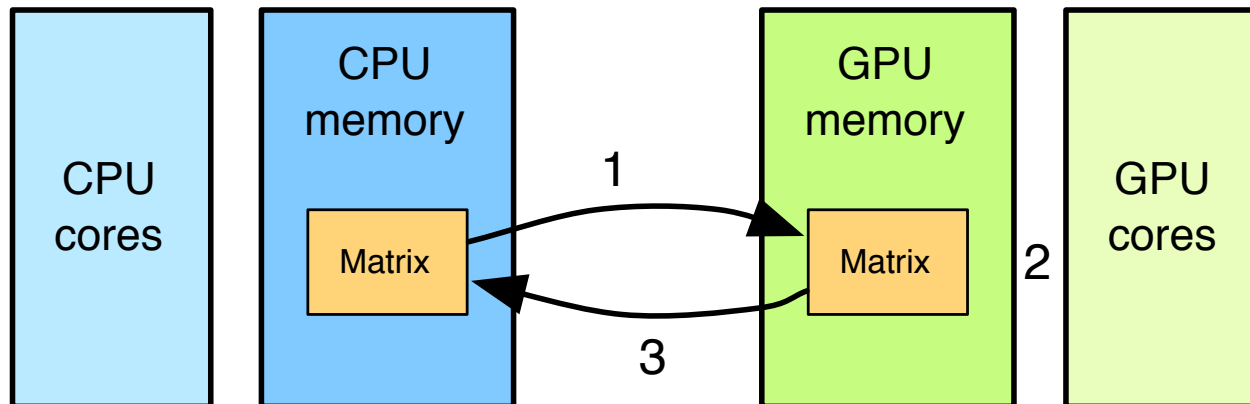
A memory circuit having: a memory array including one or more memory banks; a first processor; and a processor control interface for receiving data processing commands directed to the first processor from a central processor, the processor control interface being adapted to indicate to the central processor 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)

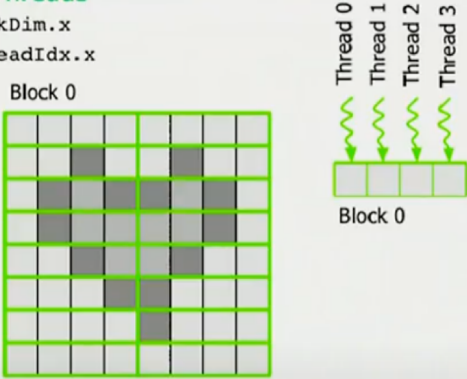


Lecture on GPU Programming

Indexing and Memory Access: 1D Grid

- One GPU thread per pixel
- Grid of Blocks of Threads
 - `gridDim.x`, `blockDim.x`
 - `blockIdx.x`, `threadIdx.x`

Block 0



Thread 0
Thread 1
Thread 2
Thread 3

Block 0


27

35:36 / 1:25:17

Design of Digital Circuits - Lecture 22: GPU Programming (ETH Zürich, Spring 2018)

2,072 views • May 24, 2018

24 1 SHARE SAVE ...

 **Onur Mutlu Lectures**
16.3K subscribers

SUBSCRIBED

P&S: Heterogeneous Systems

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.

Courses	Catalogue data	Performance assessment	Learning materials	Groups	Restrictions	Offered in	► Overview
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	<p>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).</p> <p>Despite the great advances in the adoption of heterogeneous systems in recent years, there are still many challenges to tackle, for example:</p> <ul style="list-style-type: none">- 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. <p>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.</p>						

Accelerator Model (II)

- 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

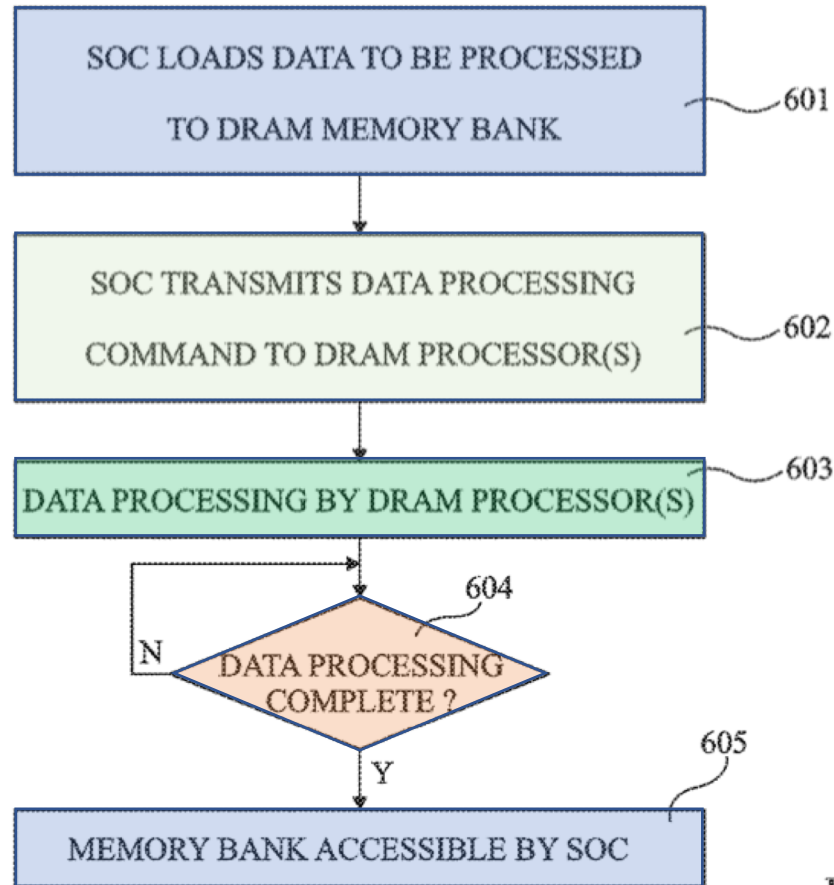


Fig 6

System Organization (I)

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

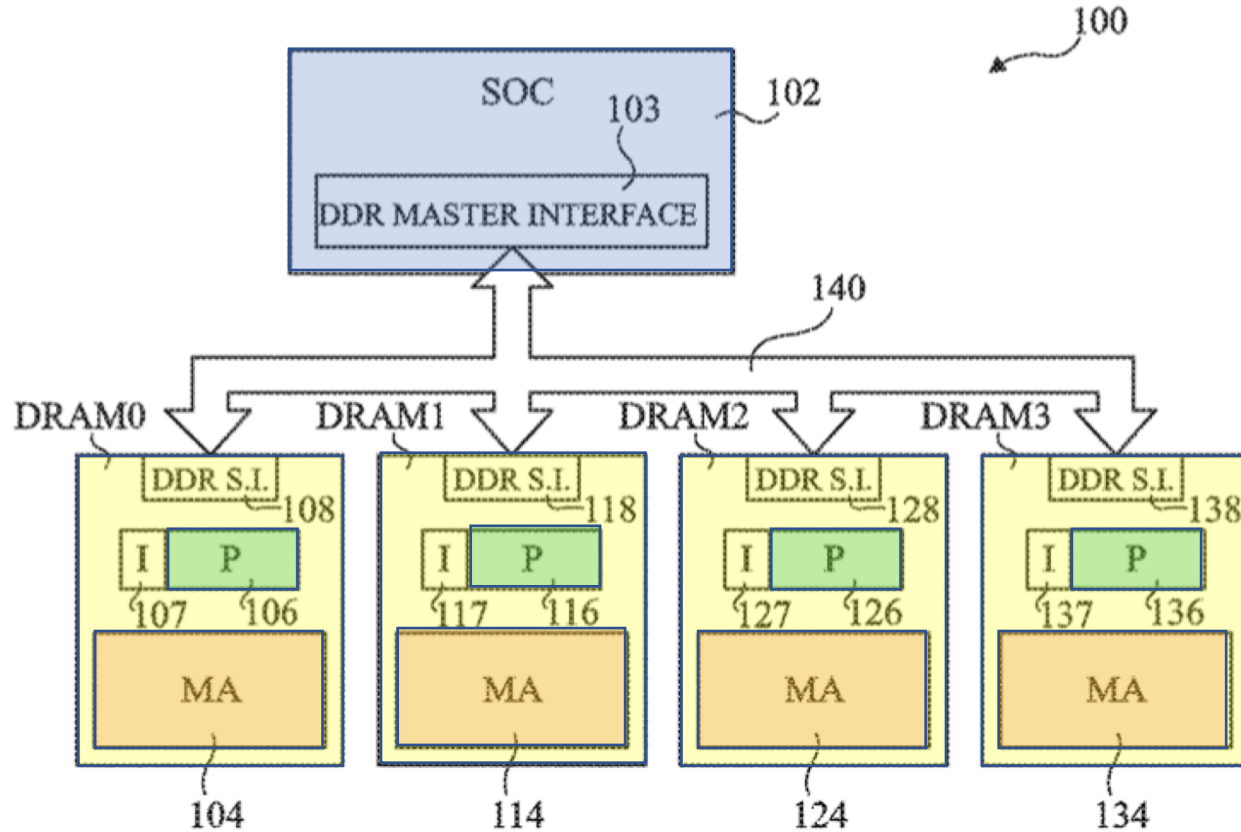
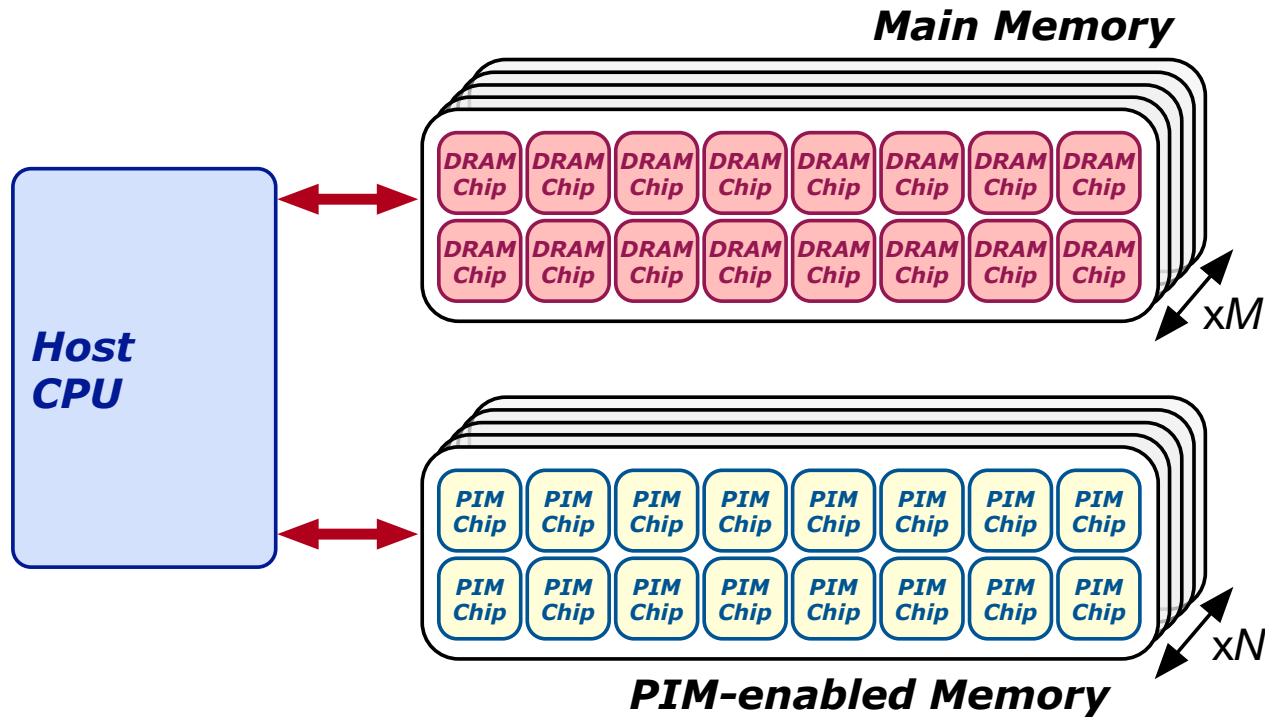


Fig 1

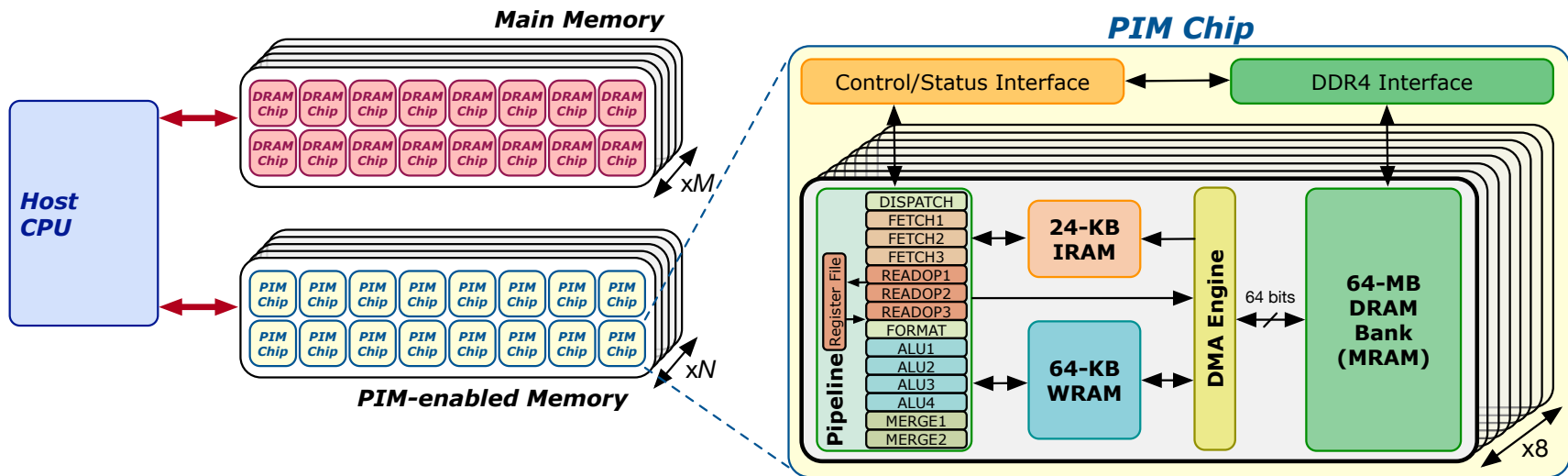
System Organization (II)

- In a UPMEM-based PIM system UPMEM DIMMs coexist with regular DDR4 DIMMs



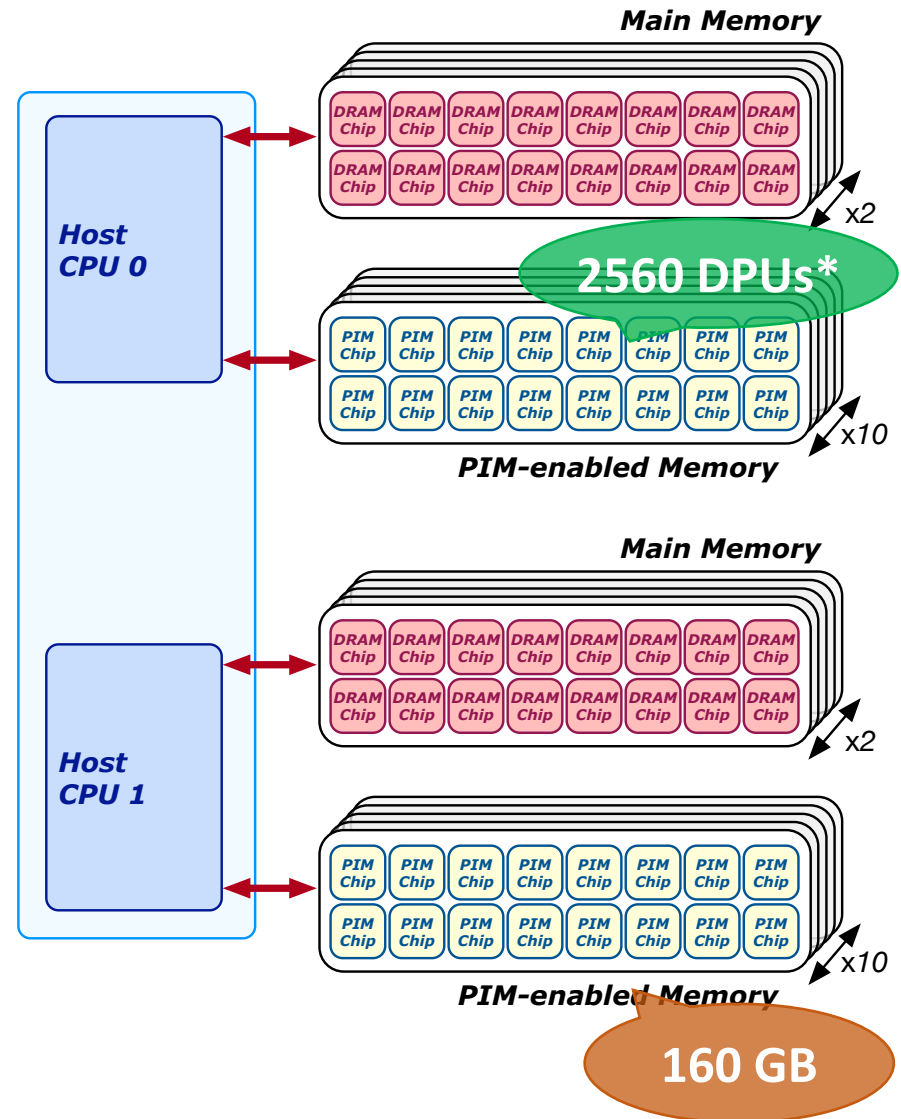
System Organization (III)

- 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

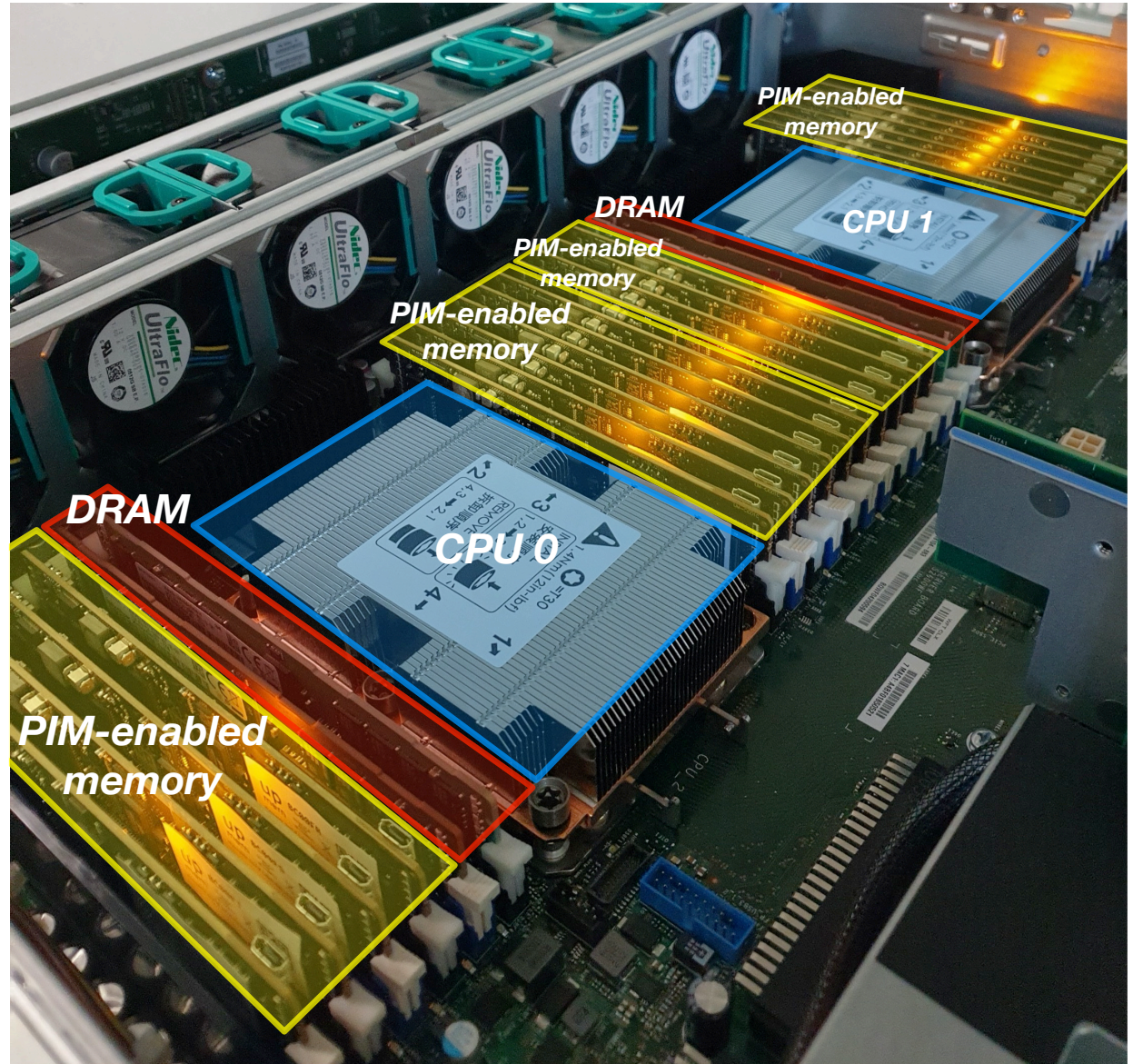
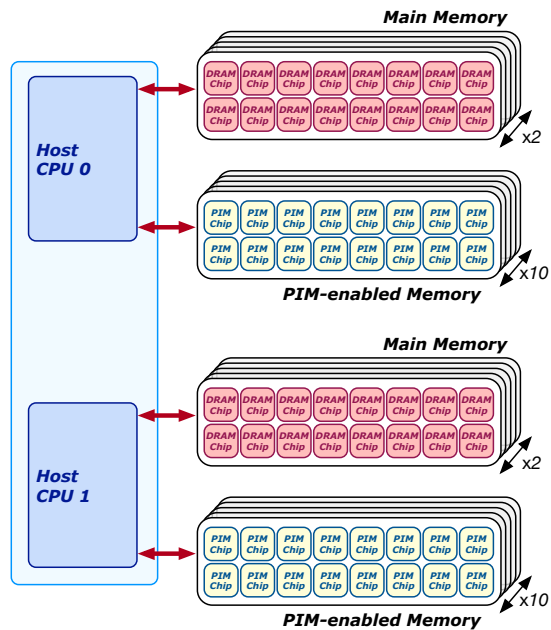


2,560-DPU System (I)

- 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

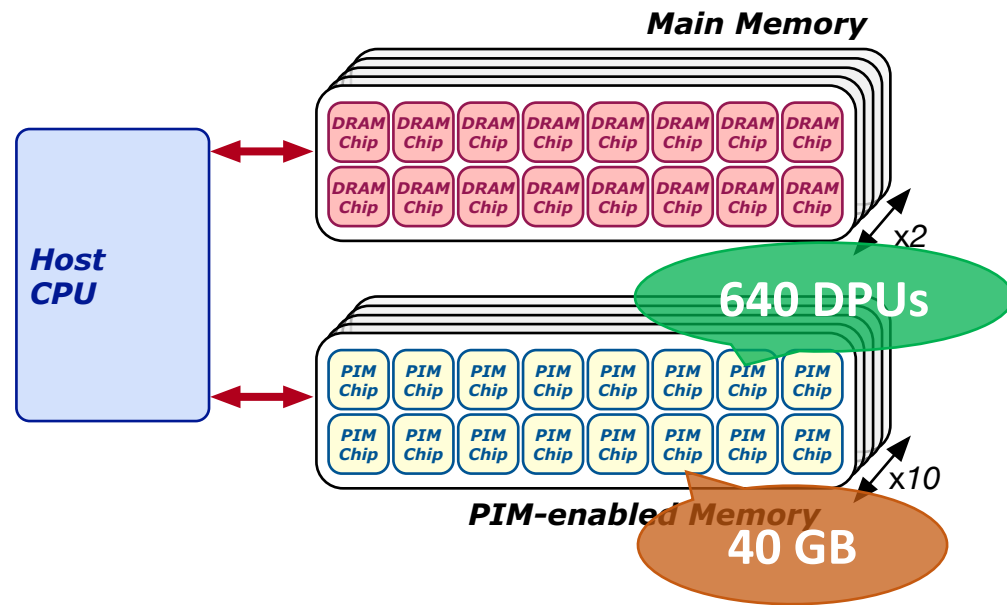


2,560-DPU System (II)



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

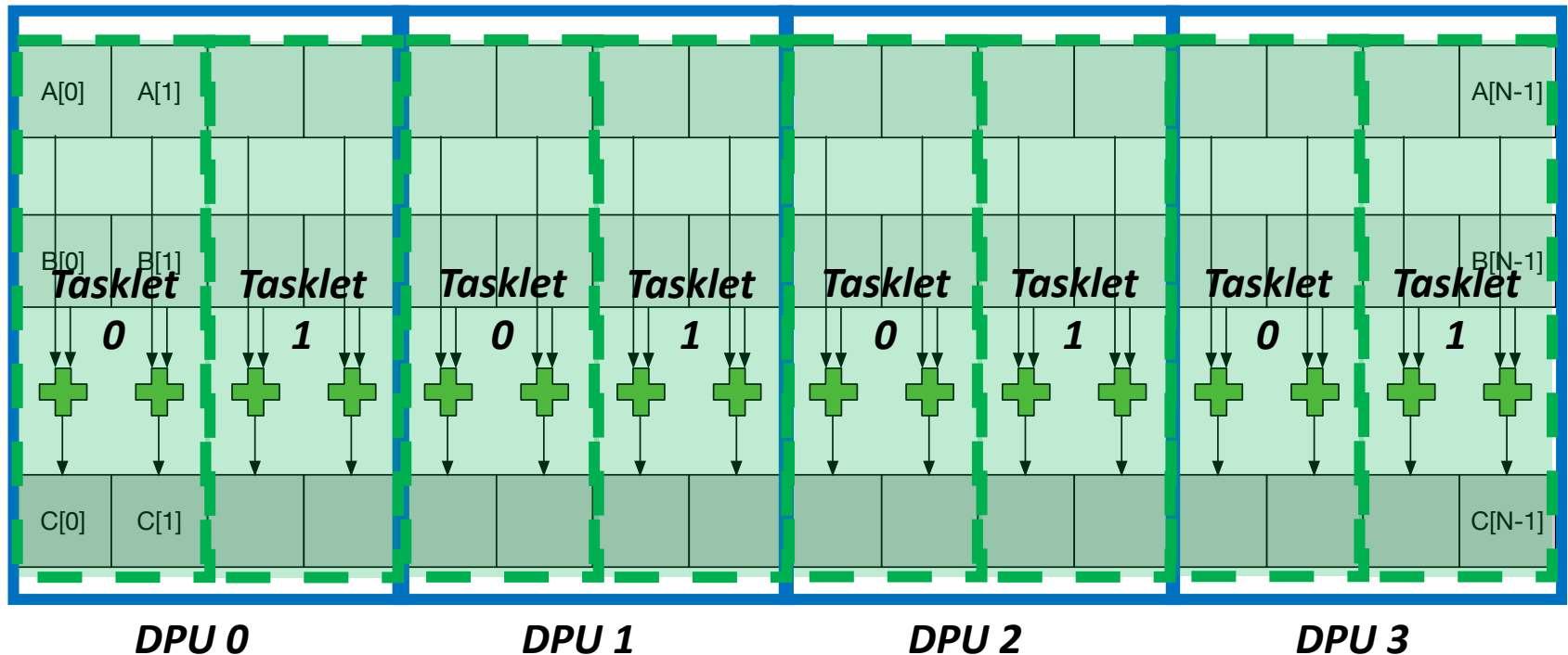


DPU Sharing? Security Implications?

- 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

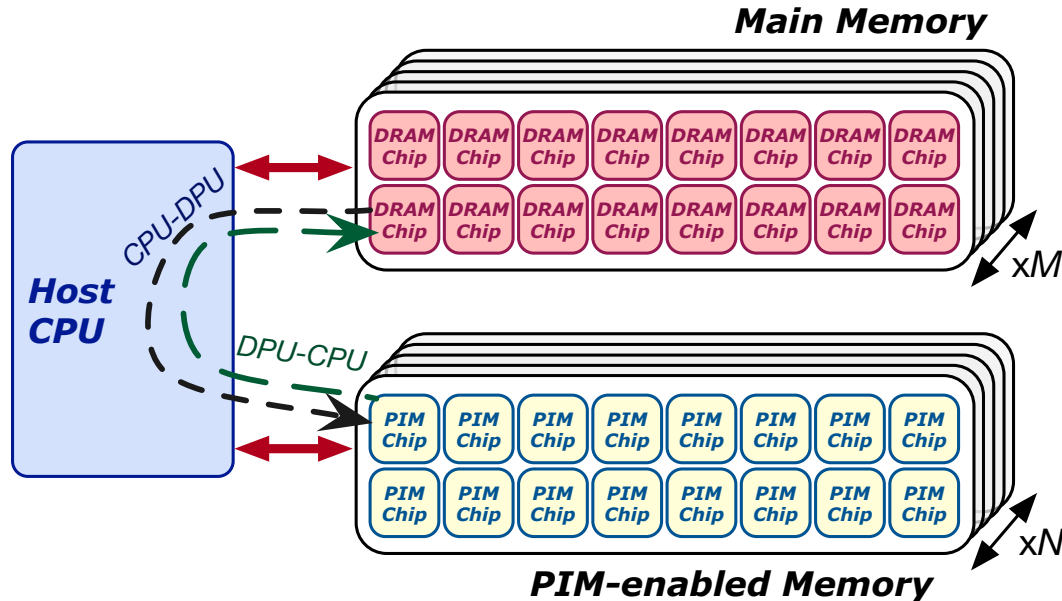
Vector Addition (VA)

- 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

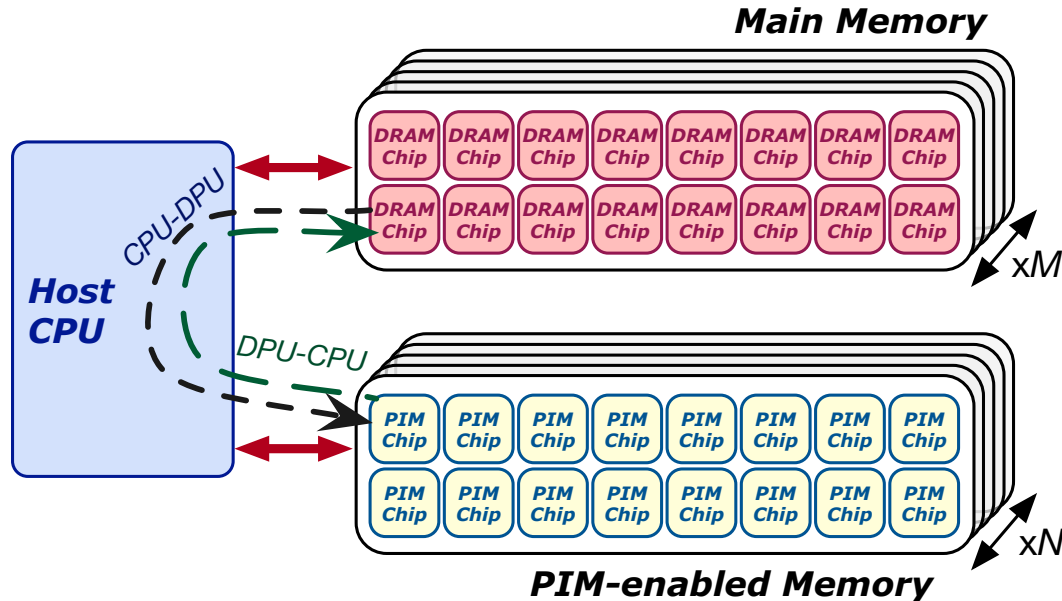
- CPU-DPU and DPU-CPU transfers
 - Between host CPU's main memory and DPUs' MRAM banks



- **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 place 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

DRAM Processing Unit (I)

- FIG. 4 schematically illustrates part of the computing system of FIG. 1 in more detail according to an example embodiment

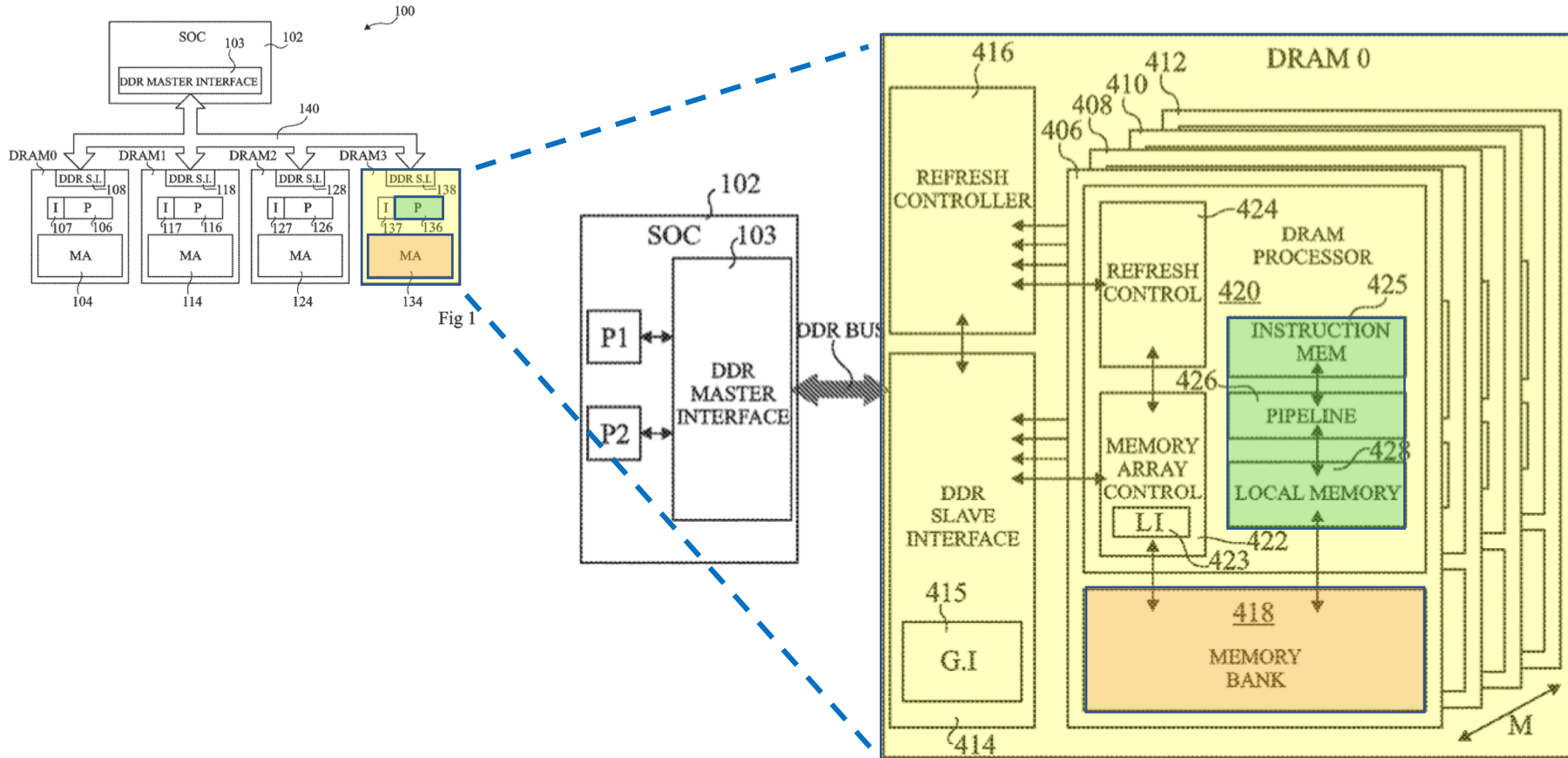
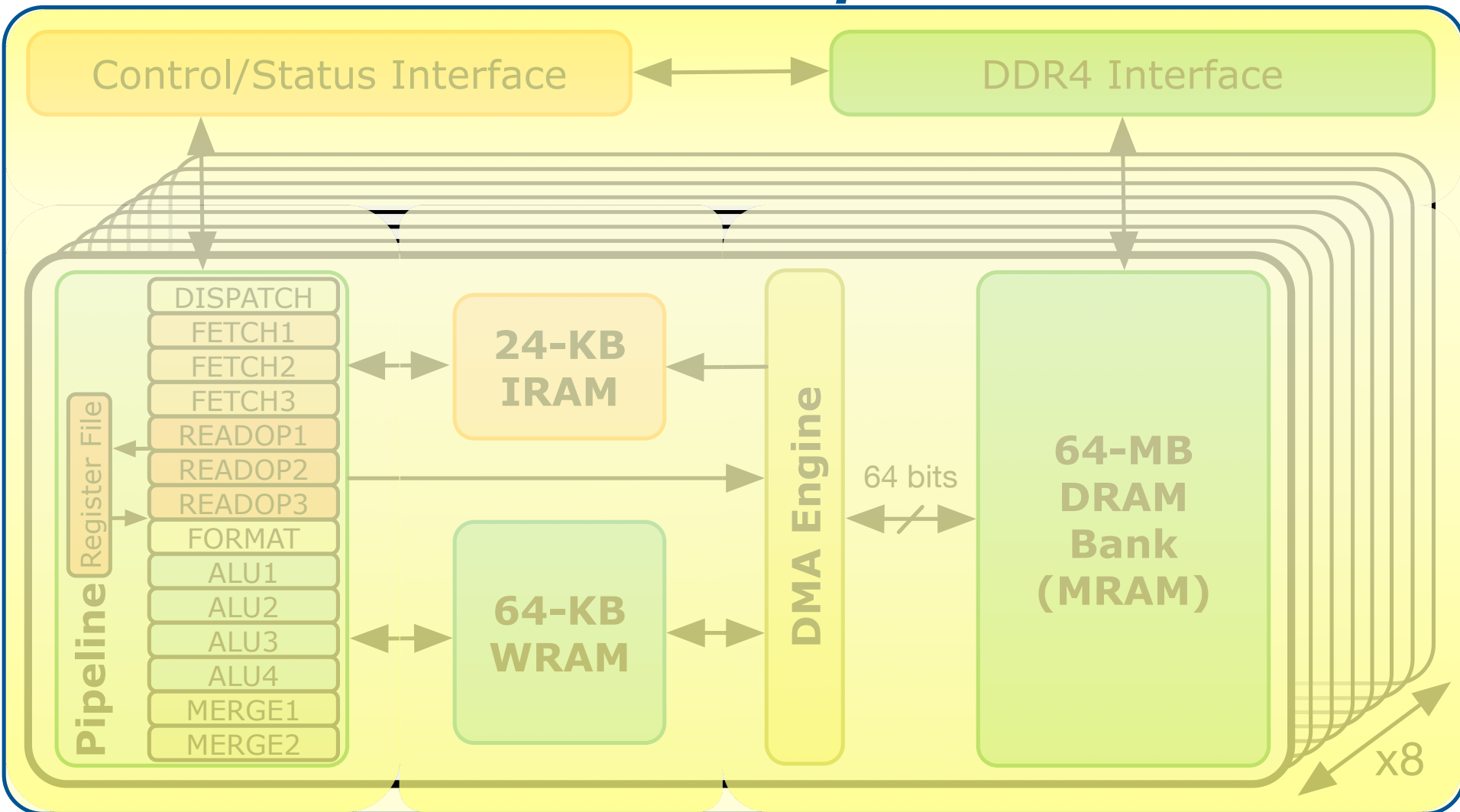


Fig 4

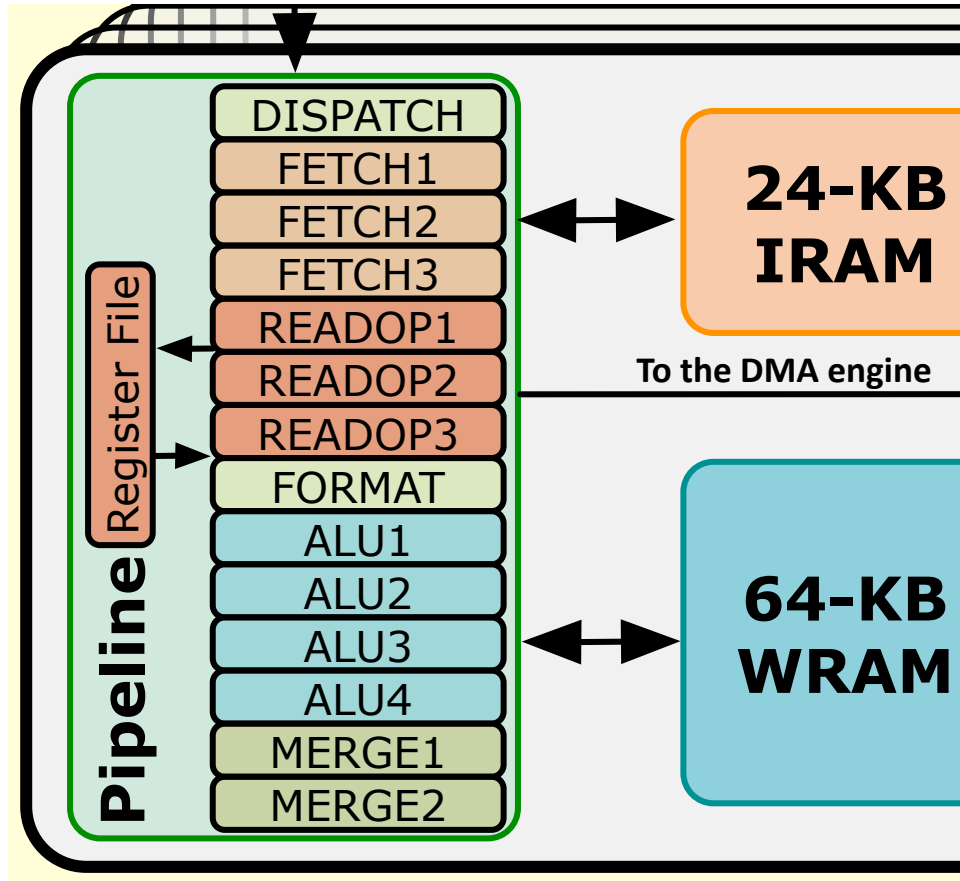
DRAM Processing Unit (II)

PIM Chip



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

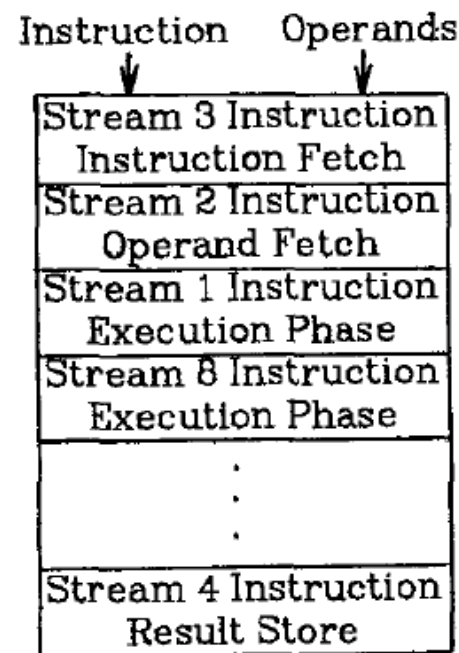


Fine-Grained Multithreading

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)

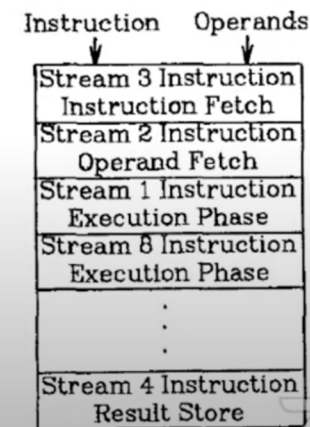
- Idea: Switch to another thread every cycle such that no two instructions from a thread are in the pipeline concurrently
- Tolerates the control and data dependence latencies by overlapping the latency with useful work from other threads
- Improves pipeline utilization by taking advantage of multiple threads
- Thornton, “Parallel Operation in the Control Data 6600,” AFIPS 1964.
- Smith, “A pipelined, shared resource MIMD computer,” ICPP 1978.

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



Onur Mutlu

zoom

Onur Mutlu - Digital Design & Comp Arch - Lecture 14: Pipelined Processor Design (Spring 2021)

1,193 views • Streamed live on Apr 22, 2021

42 0 SHARE SAVE ...



Onur Mutlu Lectures
16.2K subscribers

ANALYTICS

EDIT VIDEO

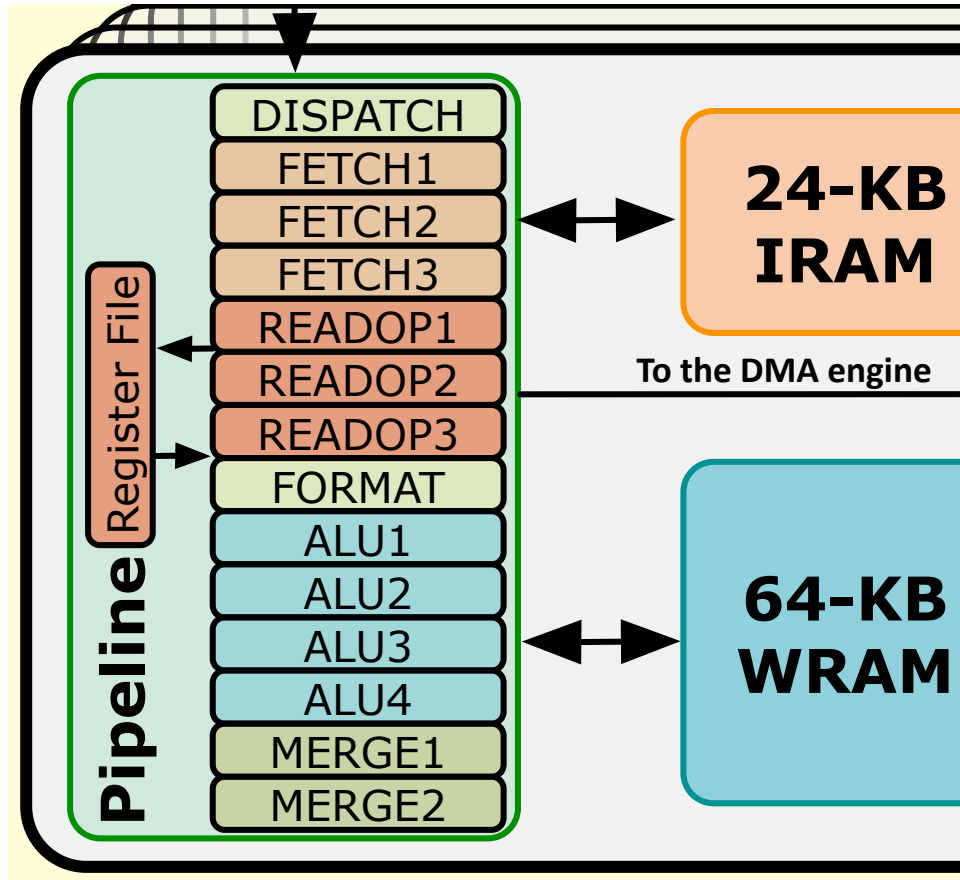
Lectures on Fine-Grained Multithreading

- Digital Design & Computer Architecture, Spring 2021, Lecture 14
 - 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

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

The screenshot shows a web page titled "Instruction Set Architecture — UPMEM DPU SDK 2021.2.0 Documentation". The page has a blue header with a hamburger menu icon and the text "UPMEM development tools documentation". Below the header, there is a breadcrumb trail: "» Instruction Set Architecture" with a "View page source" link. The main content area has a heading "Instruction Set Architecture" followed by a paragraph: "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." Below this is another paragraph: "Software developers should use this section as a reference manual to develop or debug assembly code." The next section is "Resources overview" followed by "Thread registers". A paragraph states: "The system is composed of 24 hardware threads. Each of them owns a set of private resources:". A bulleted list follows: "• 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.", and "• Two persistent flags, keeping information about the previous result of an arithmetic or logical instruction:". A sub-bullet for the ZF flag states: "◦ 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

```
1  #define SIZE 256
2  int* bufferA = mem_alloc(SIZE * sizeof(int));
3  for(int i = 0; i < SIZE; i++){
4      int temp = bufferA[i];
5      temp += scalar;
6      bufferA[i] = temp;
7  }
```

Compiled code
(UPMEM DPU ISA)

```
1  move r2, 0
2  .LBB0_1:                                // Loop header
3  lsl_add r3, r0, r2, 2                    // Address calculation
4  lw r4, r3, 0                            // Load from WRAM
5  add r4, r4, r1                           // Add
6  sw r3, 0, r4                            // Store to WRAM
7  add r2, r2, 1                           // Index update
8  jneq r2, 256, .LBB0_1                   // Conditional jump
```

Arithmetic Throughput: #Instructions

- Compiler explorer: <https://dpu.dev>

```
1  #define BLOCK_SIZE 1024
2
3  typedef int T;
4  void Benchmark__32bits(T *cache_A, T scalar) {
5      for (int i = 0; i < BLOCK_SIZE / sizeof(T); i++){
6          // WRAM READ
7          T temp = cache_A[i];
8
9          temp += scalar; // ADD
10
11         // WRAM WRITE
12         cache_A[i] = temp;
13     }
14 }
15
16 typedef long T_long;
17 void Benchmark__64bits(T_long *cache_A, T_long scalar) {
18     for (int i = 0; i < BLOCK_SIZE / sizeof(T_long); i++){
19         // WRAM READ
20         T_long temp = cache_A[i];
21
22         temp += scalar; // ADD
23     }
24
25
26
27
```

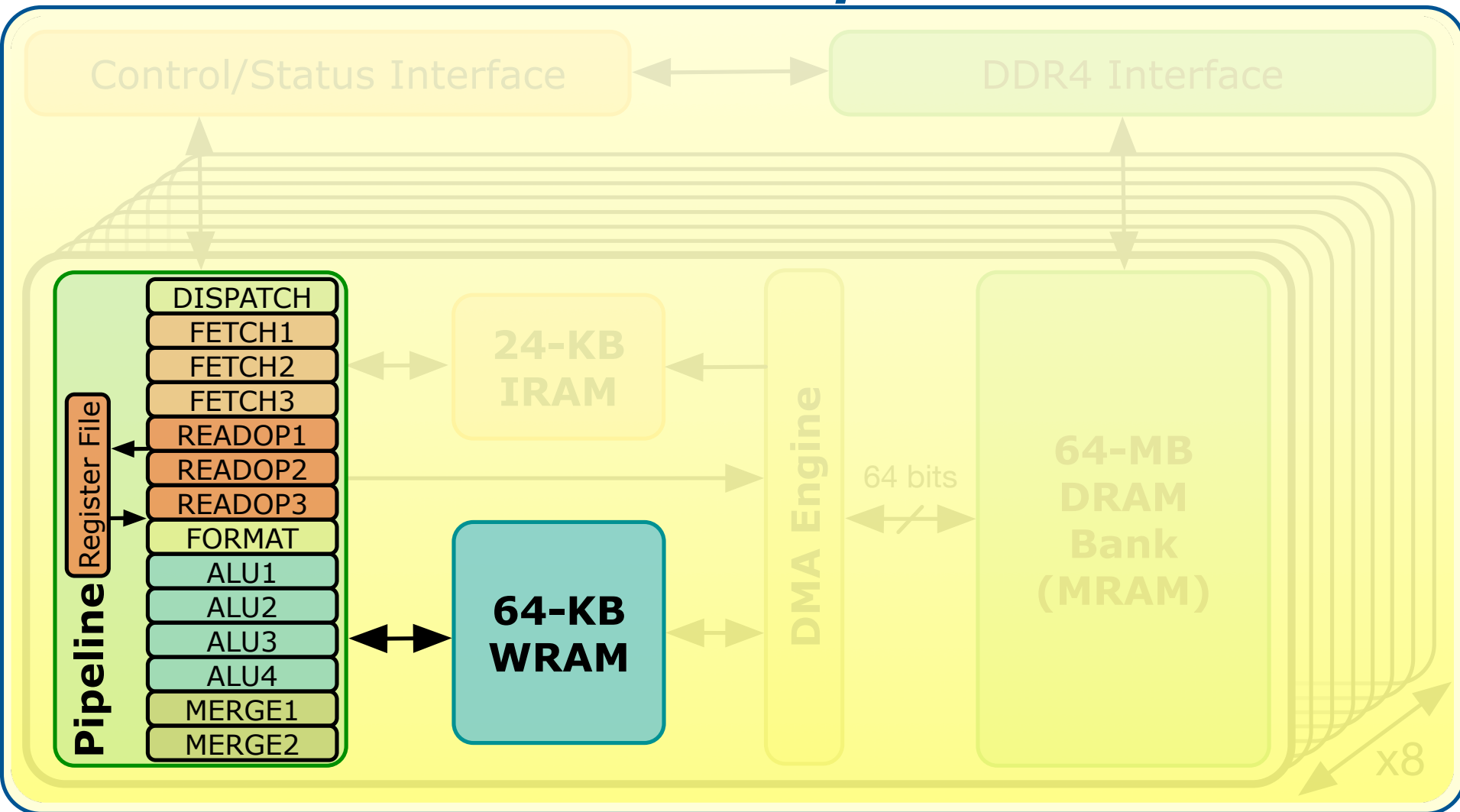
A ▾ ☐ 11010 ☐ ./a.out ☒ .LX0: ☒ .text ☒ // ☐ \

```
1 Benchmark__32bits:
2     move r2, 0
3 .LBB0_1:
4     lsl_add r3, r0, r2, 2
5     lw r4, r3, 0
6     add r4, r4, r1
7     sw r3, 0, r4
8     add r2, r2, 1
9     jneq r2, 256, .LBB0_1
10    jump r23
11 Benchmark__64bits:
12     move r1, 0
13 .LBB1_1:
14     lsl_add r4, r0, r1, 3
15     ld d6, r4, 0
16     add r7, r7, r3
17     addc r6, r6, r2
18     sd r4, 0, d6
19     add r1, r1, 1
20     jneq r1, 128, .LBB1_1
21    jump r23
```

6 instructions in the 32-bit ADD/SUB microbenchmark
7 instructions in the 64-bit ADD/SUB microbenchmark

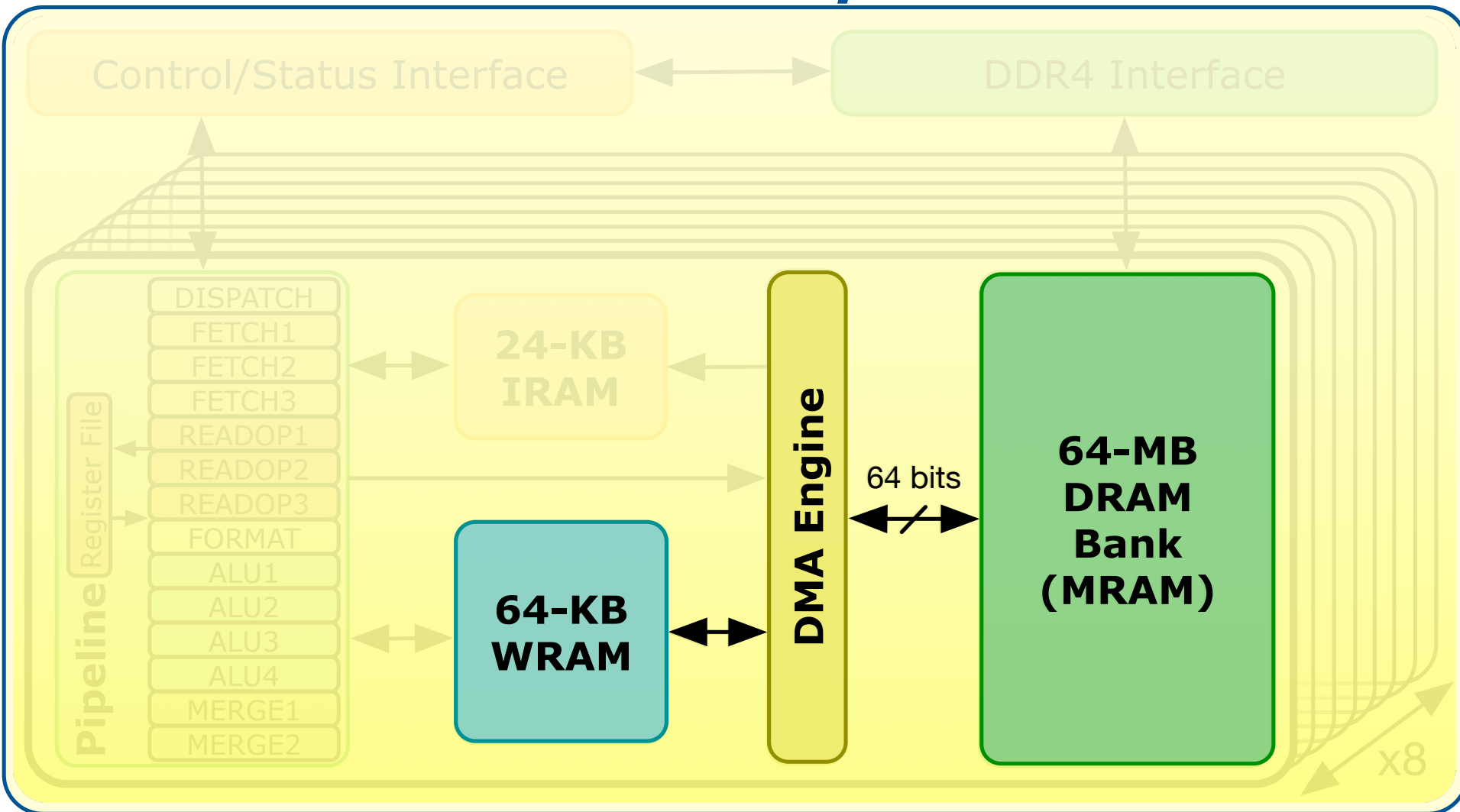
DPU: WRAM Bandwidth

PIM Chip



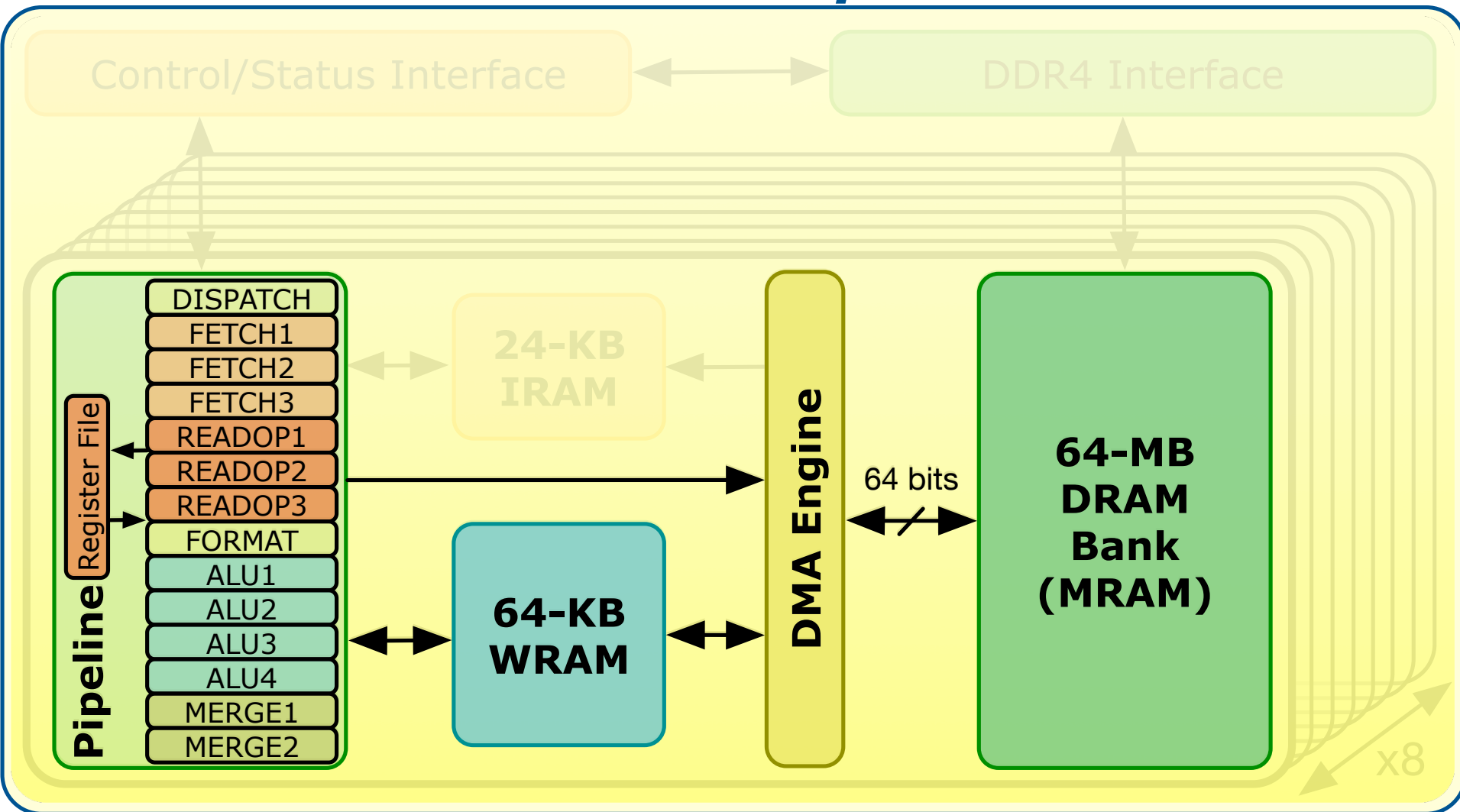
DPU: MRAM Latency and Bandwidth

PIM Chip



DPU: Arithmetic Throughput vs. Operational Intensity

PIM Chip



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 GIANNOULA, 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



The video player shows a lecture titled "Understanding a Modern Processing-in-Memory Architecture: Benchmarking and Experimental Characterization". The speaker is Juan Gómez Luna, Izzat El Hajj, Ivan Fernandez, Christina Giannoula, Geraldo F. Oliveira, and Onur Mutlu. The video is from the SAFARI Live Seminar series, hosted by ETH Zürich. The video player includes a progress bar at 2:26 / 2:57:10, a volume icon, and a play button. The video player also displays the title, view count (2,579 views), and the date it was streamed (Jul 12, 2021). The video player includes a like button (93 likes), a comment button (0 comments), a share button, a save button, and a subscribe button (SUBSCRIBED). The video player also displays the channel name "Onur Mutlu Lectures" and the subscriber count "18.7K subscribers".

Understanding a Modern Processing-in-Memory Architecture: Benchmarking and Experimental Characterization

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

<https://arxiv.org/pdf/2105.03814.pdf>
<https://github.com/CMU-SAFARI/prim-benchmarks>

ETH Zürich SAFARI

2:26 / 2:57:10

SAFARI Live Seminar: Understanding a Modern Processing-in-Memory Architecture

2,579 views • Streamed live on Jul 12, 2021

93 0 SHARE SAVE ...

 **Onur Mutlu Lectures**
18.7K subscribers

SUBSCRIBED

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