Near-memory computing/processing

Asif Ali Khan

Fall Semester 2024

Department of Computer Systems Engineering

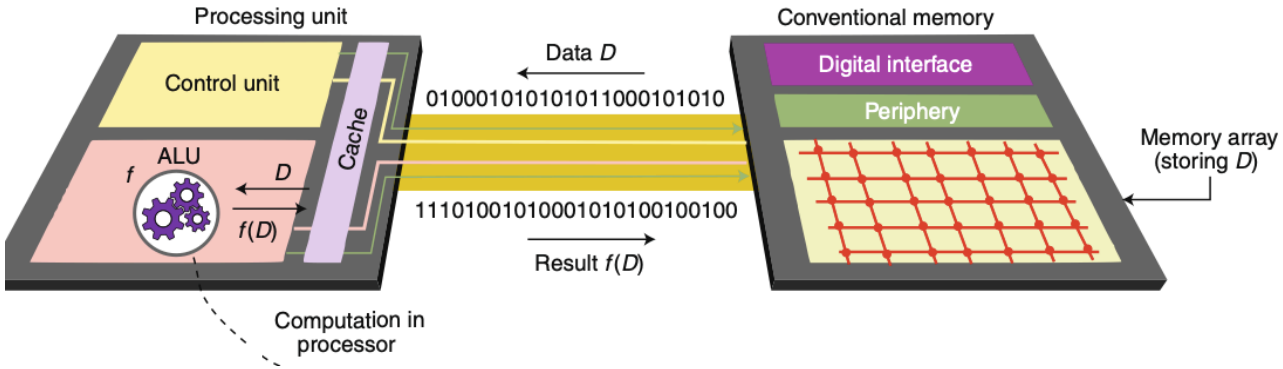
UET Peshawar, Pakistan

Oct 31, 2024

Recap: Domain-specific architectures

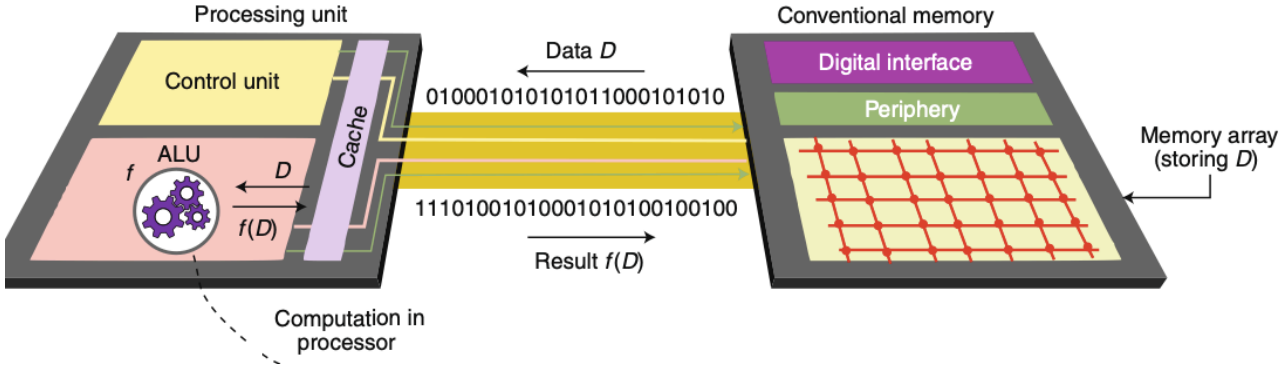
- Domain-specific architectures follow four key principles
  - Specialization
  - Parallelism
  - Efficient memory system organization
  - Reduced control overhead
- Specialization can be in data and can be hardware
- Parallelism can be of SIMD, MIMD, or some other type
- Small and local memories around compute units to avoid data movement over the external bus (as much as possible)
- Typically no PC and other control circuitry used in Von-Neumann systems

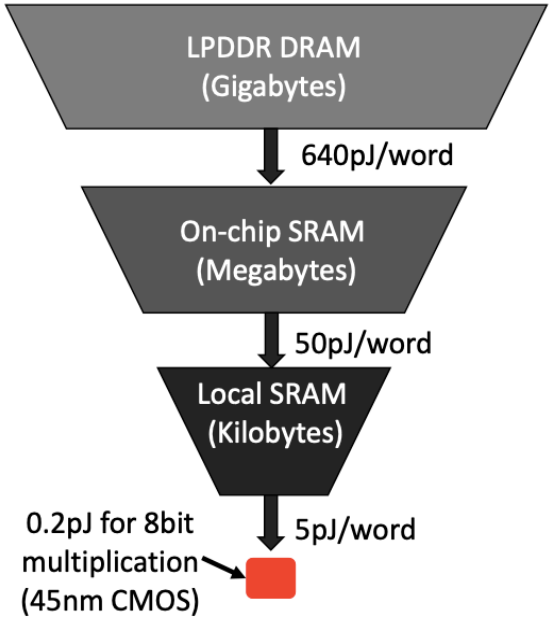
Non-Von-Neumann computing paradigm



Abu Sebastian et al, Nature Nanotechnology, 2020

Non-Von-Neumann computing paradigm

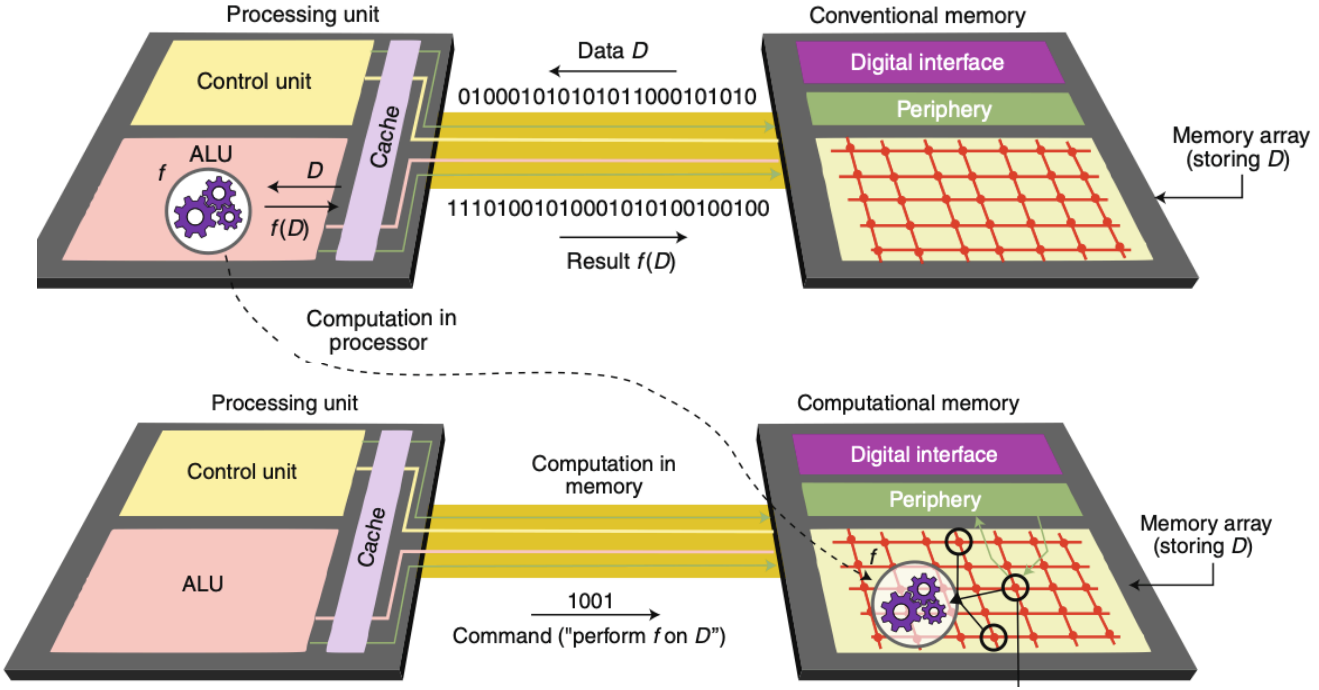


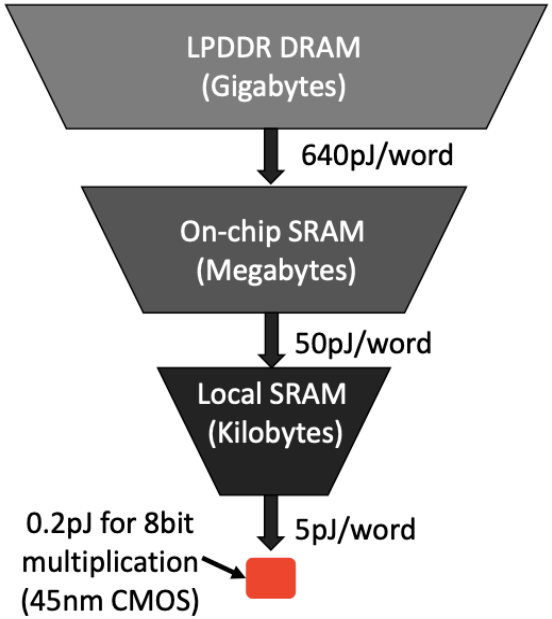


Bill Dally, MICRO, 2019

Abu Sebastian et al, Nature Nanotechnology, 2020

Non-Von-Neumann computing paradigm

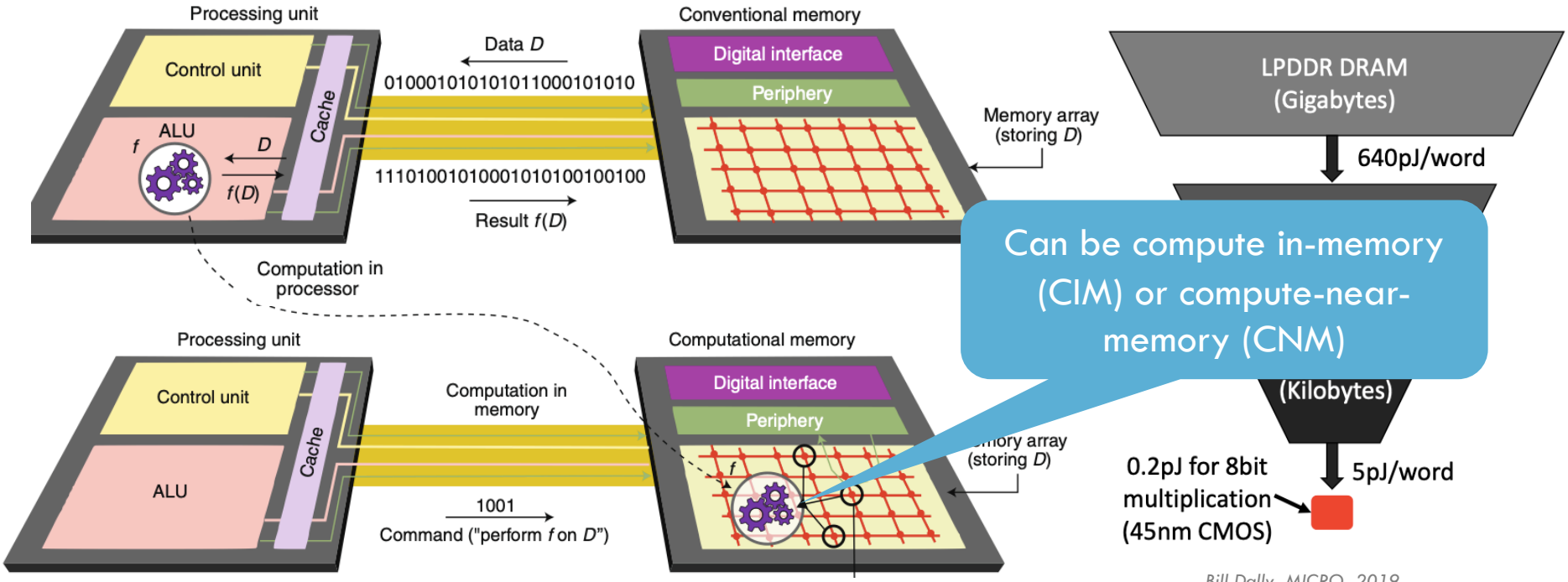




Bill Dally, MICRO, 2019

Abu Sebastian et al, Nature Nanotechnology, 2020

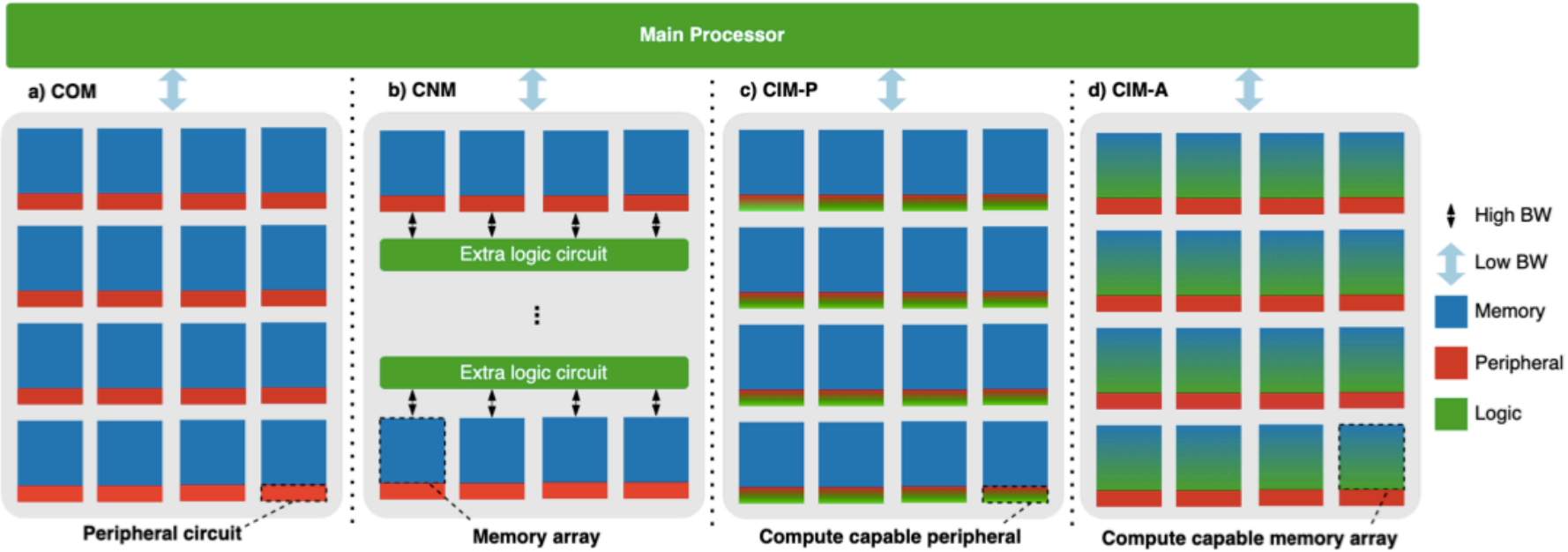
Non-Von-Neumann computing paradigm



Bill Dally, MICRO, 2019

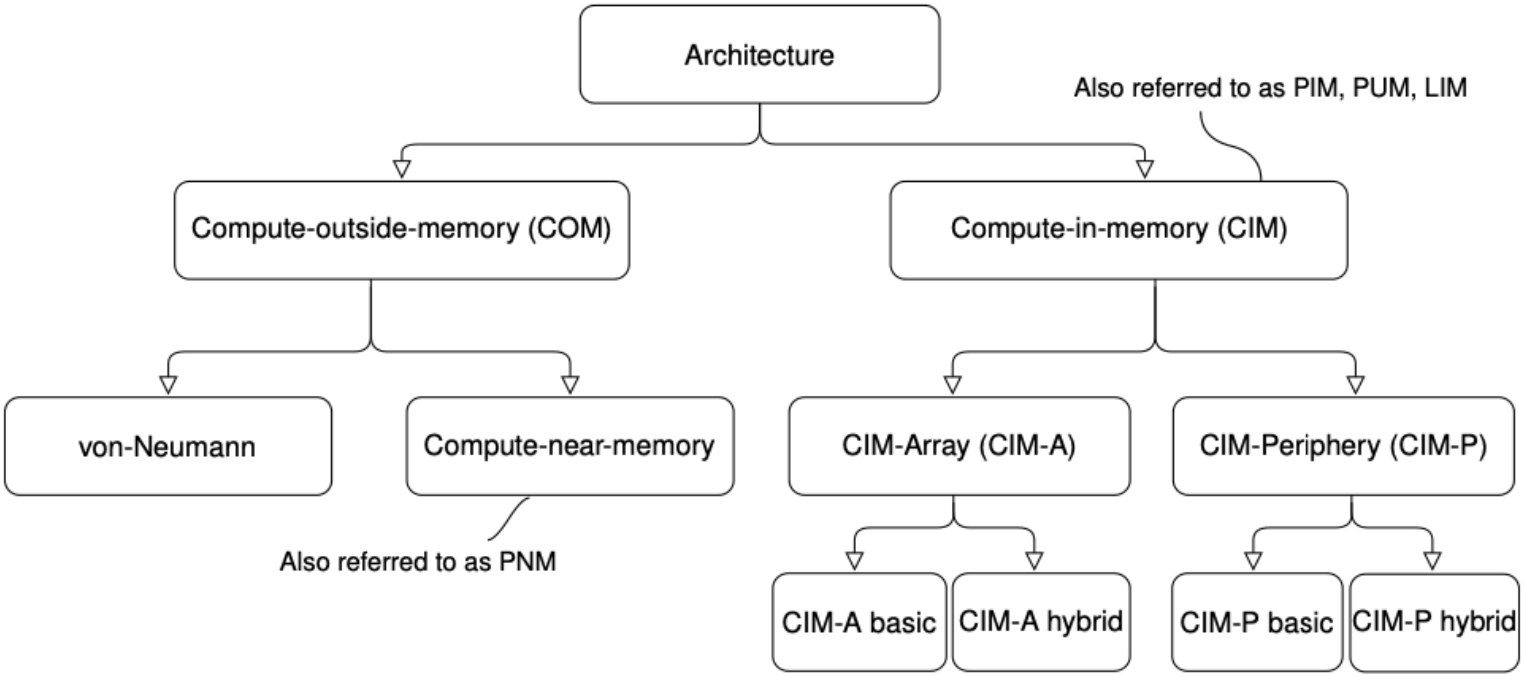
Abu Sebastian et al, Nature Nanotechnology, 2020

CIM / CNM types



Khan et al, Arxiv, 2024

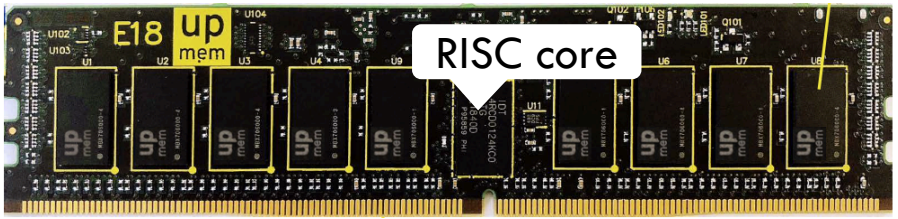
Terminology overview



Do read: Khan et al., "The Landscape of Compute-near-memory and Compute-in-memory: A Research and Commercial Overview", Arxiv 2024

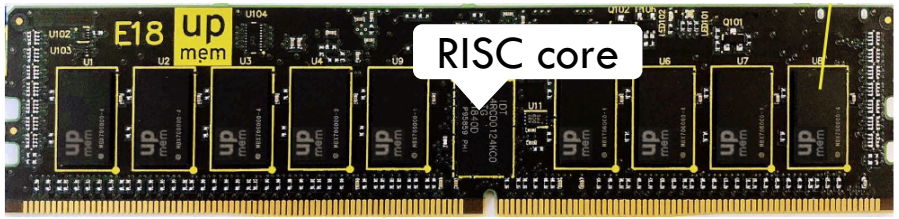
Khan et al, Arxiv, 2024

Computation near-memory (CNM)

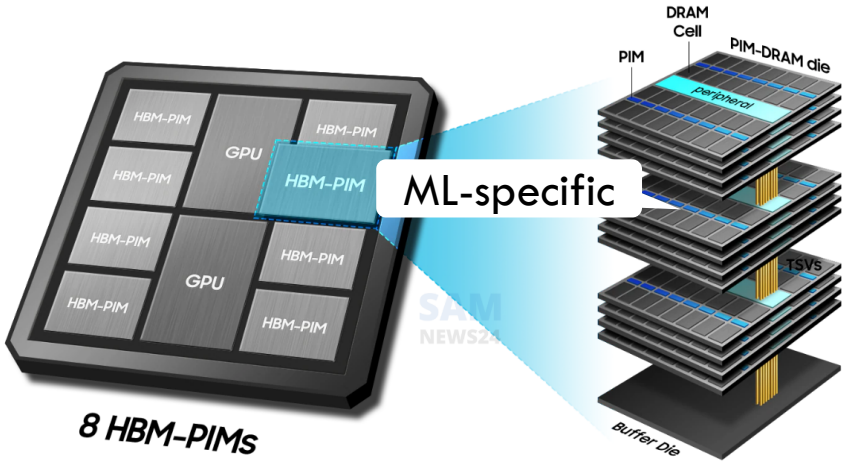


J. G. Luna et al, IEEE Access 2022

Computation near-memory (CNM)

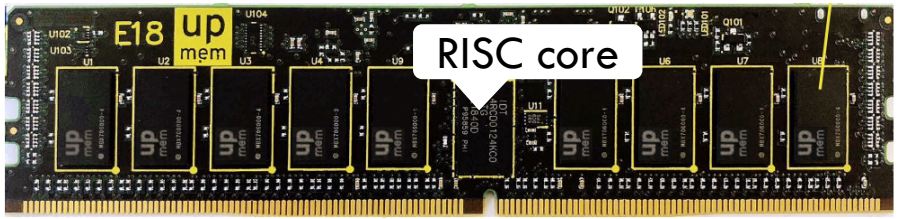


J. G. Luna et al, IEEE Access 2022

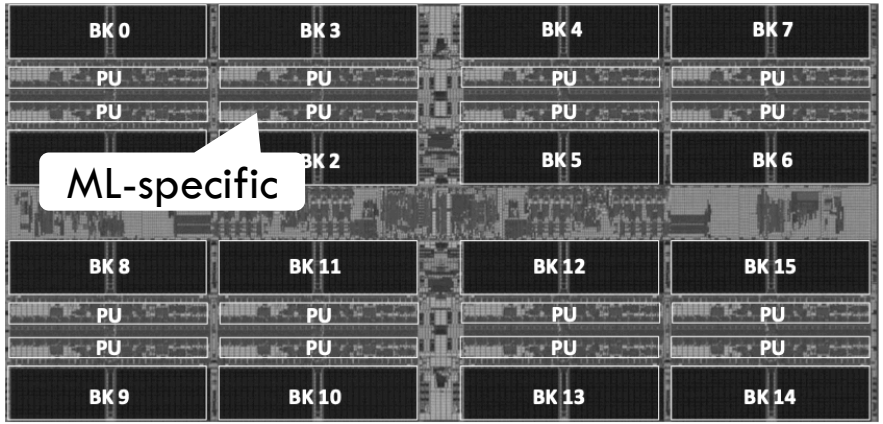


Samsung news

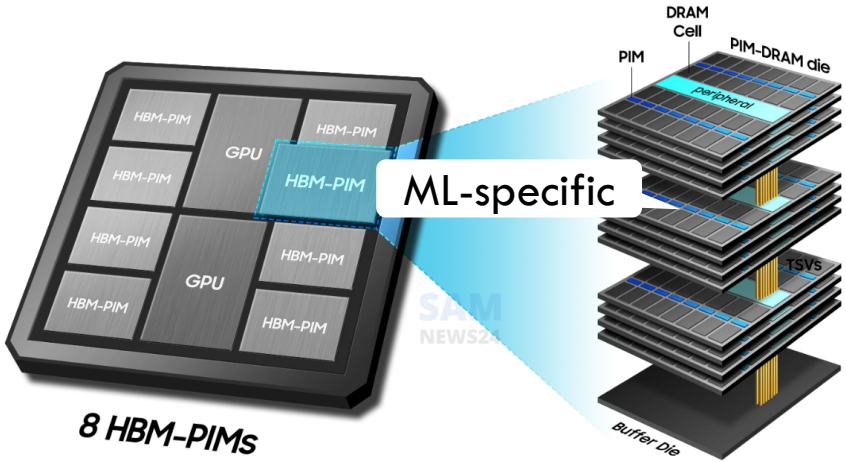
Computation near-memory (CNM)



J. G. Luna et al, IEEE Access 2022

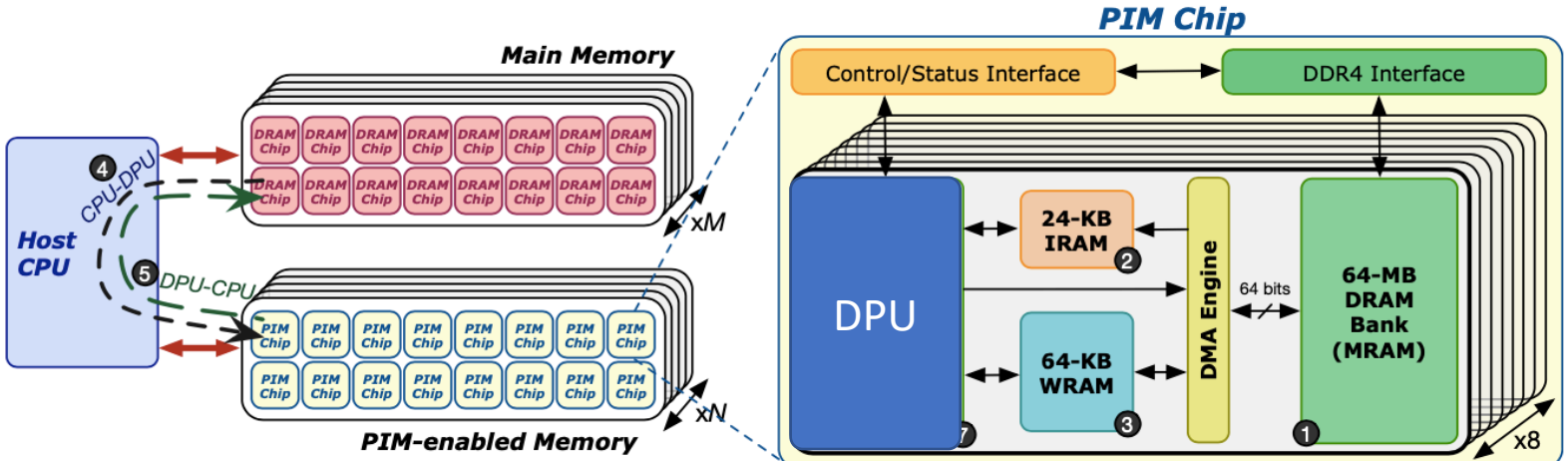


S. Lee et al., ISSCC 2022



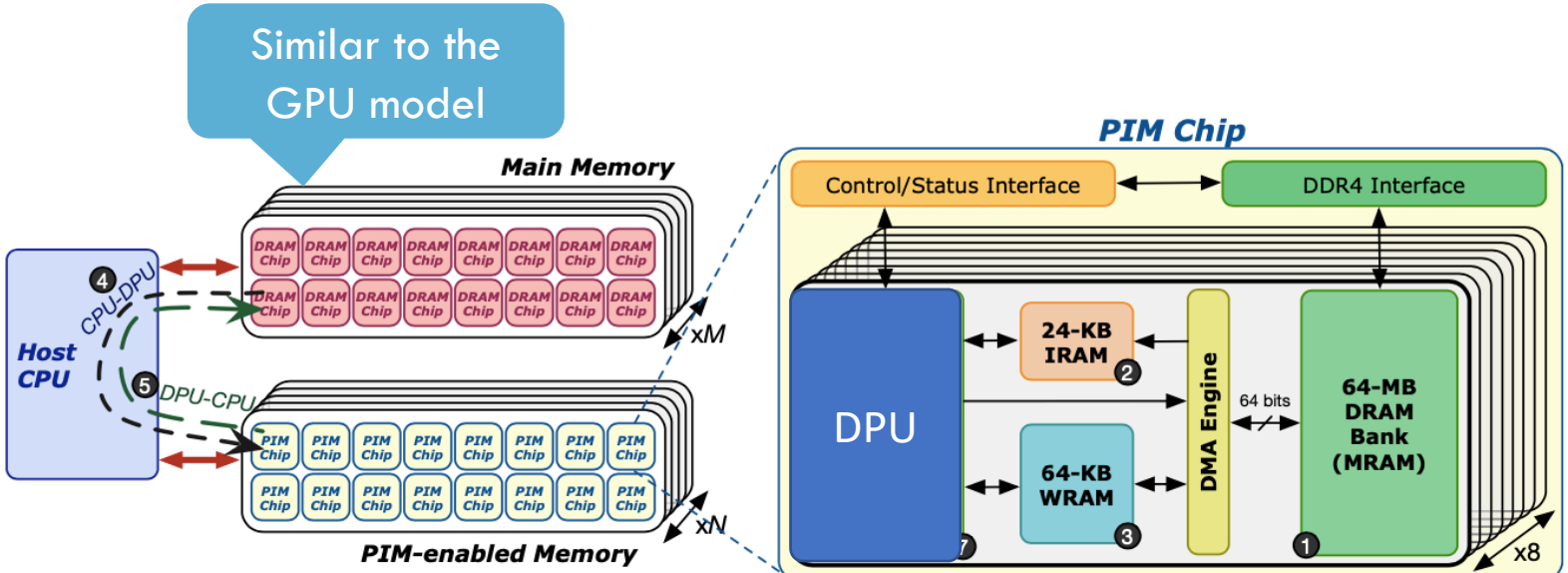
Samsung news

The UPMEM architecture



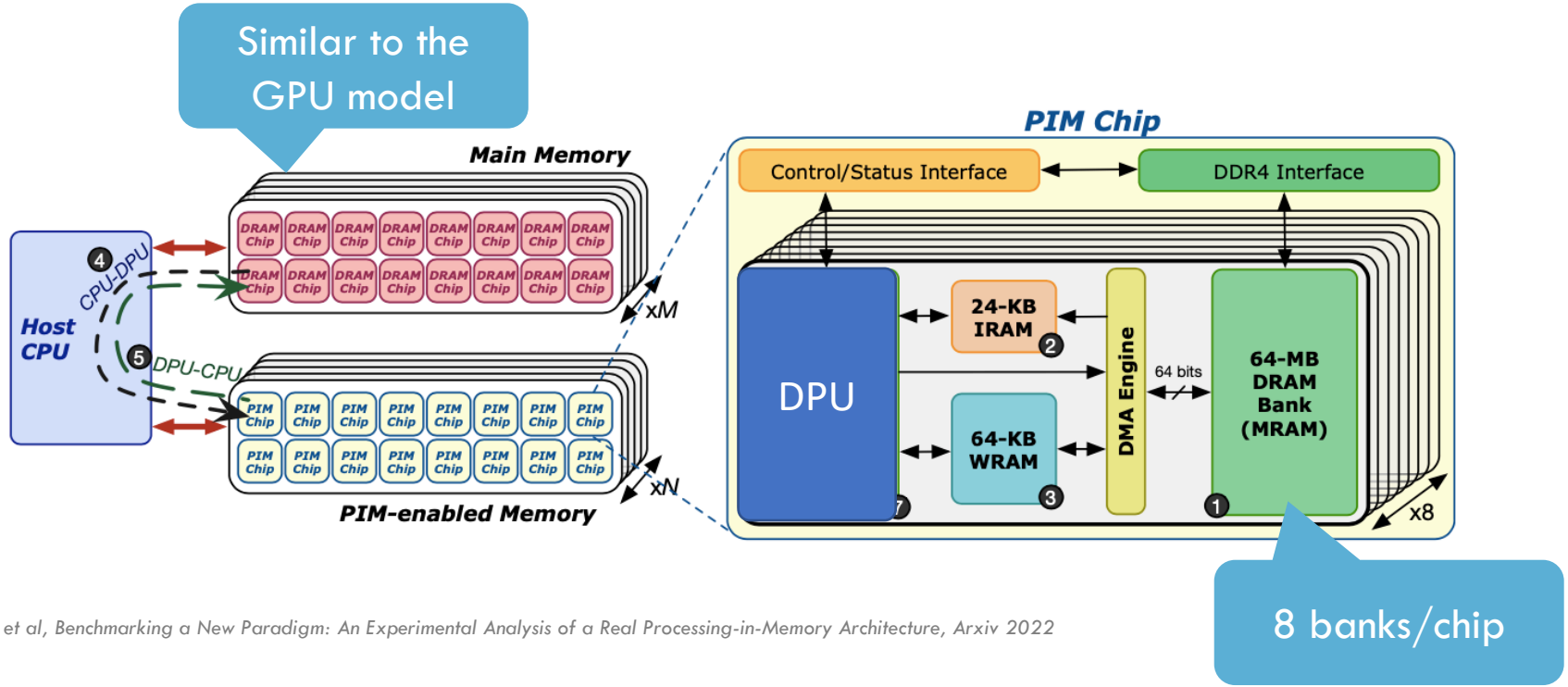
Luna et al, Benchmarking a New Paradigm: An Experimental Analysis of a Real Processing-in-Memory Architecture, Arxiv 2022

The UPMEM architecture



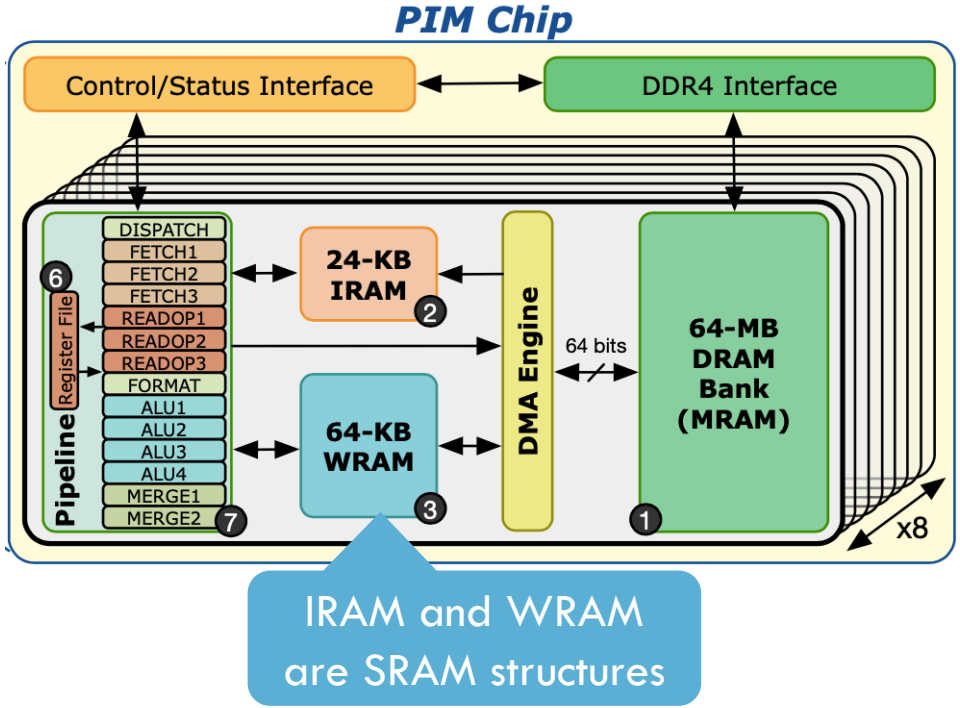
Luna et al, Benchmarking a New Paradigm: An Experimental Analysis of a Real Processing-in-Memory Architecture, Arxiv 2022

The UPMEM architecture



Luna et al, Benchmarking a New Paradigm: An Experimental Analysis of a Real Processing-in-Memory Architecture, Arxiv 2022

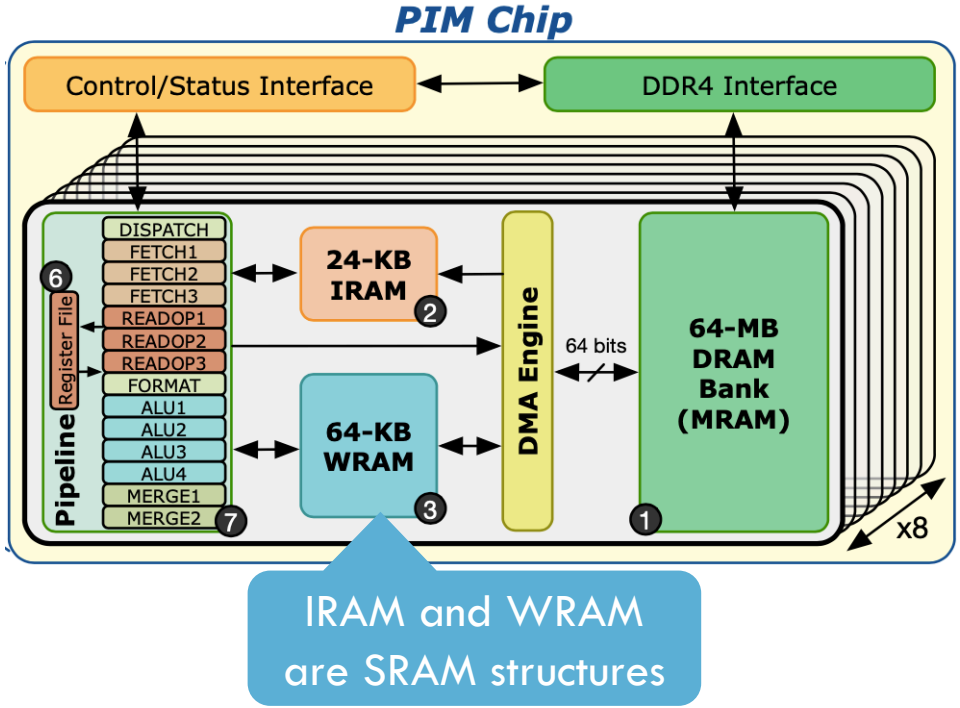
The UPMEM architecture



- Organization: 2 ranks per DIMM, 8 chips per rank, 8 DPUs/banks per chip

Luna et al, Arxiv 2022

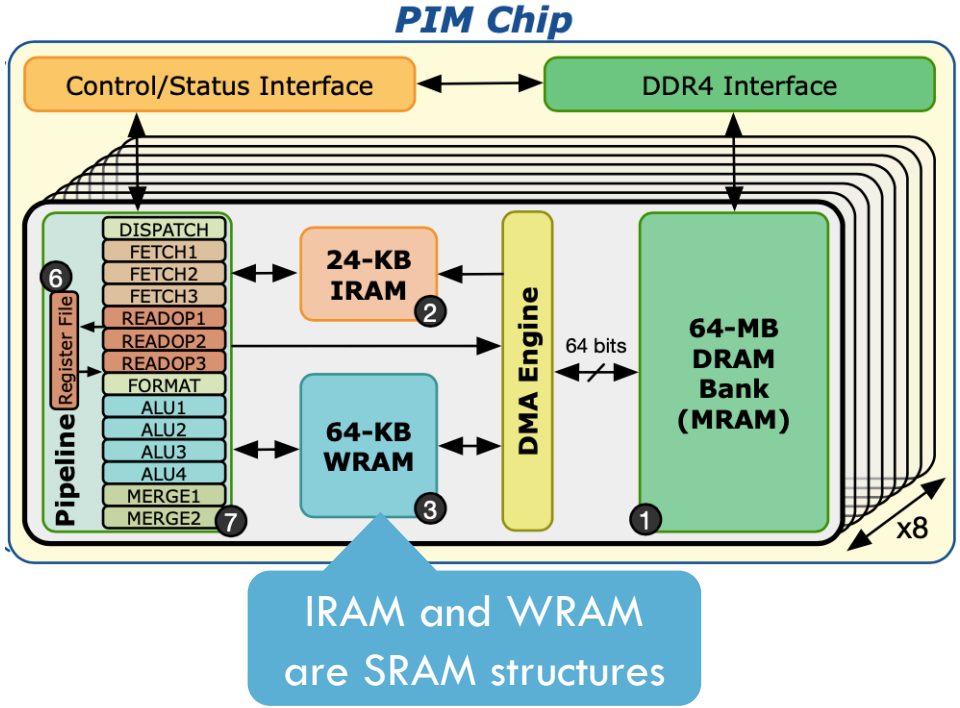
The UPMEM architecture



- Organization: 2 ranks per DIMM, 8 chips per rank, 8 DPUs/banks per chip
- Each DPU has a 64MB DRAM (MRAM) and a 64KB working RAM (WRAM)

Luna et al, Arxiv 2022

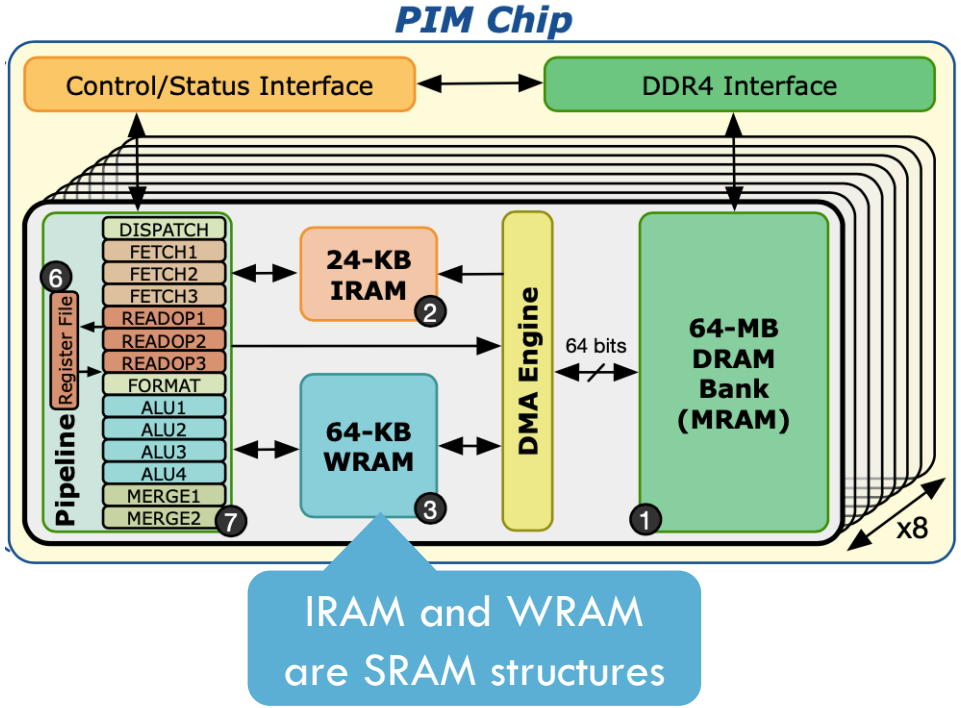
The UPMEM architecture



- Organization: 2 ranks per DIMM, 8 chips per rank, 8 DPUs/banks per chip
- Each DPU has a 64MB DRAM (MRAM) and a 64KB working RAM (WRAM)
- Working RAM is a scratchpad
  - Management still critical for perf

Luna et al, Arxiv 2022

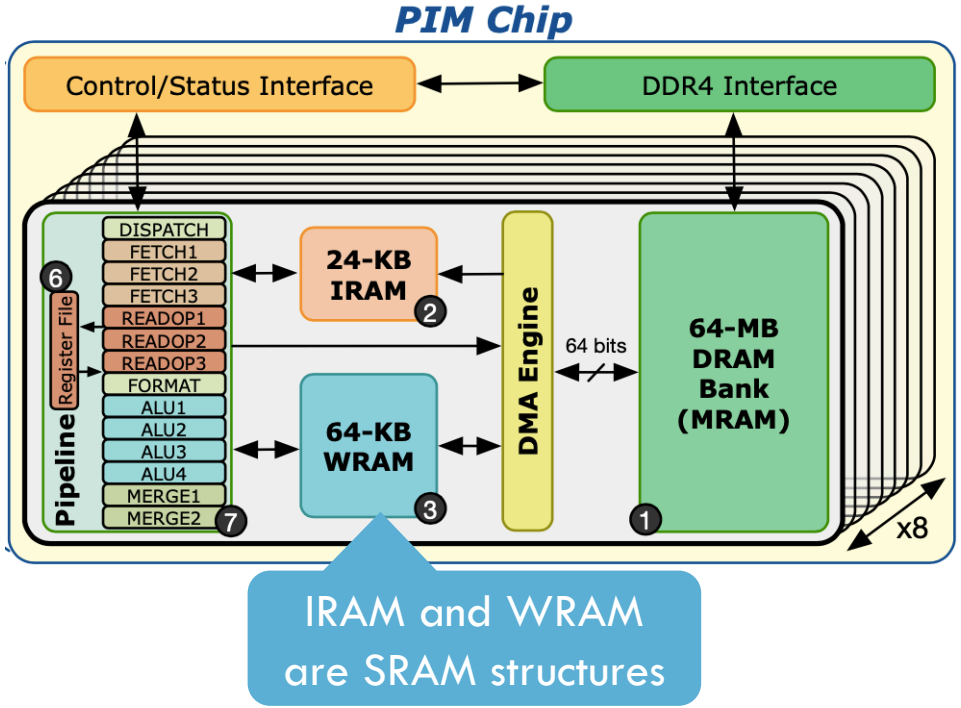
The UPMEM architecture



- Organization: 2 ranks per DIMM, 8 chips per rank, 8 DPUs/banks per chip
- Each DPU has a 64MB DRAM (MRAM) and a 64KB working RAM (WRAM)
- Working RAM is a scratchpad
  - Management still critical for perf
- DPU pipeline
  - 14 stages, 24 threads, 450MHz

Luna et al, Arxiv 2022

The UPMEM architecture



- Organization: 2 ranks per DIMM, 8 chips per rank, 8 DPUs/banks per chip
- Each DPU has a 64MB DRAM (MRAM) and a 64KB working RAM (WRAM)
- Working RAM is a scratchpad
  - Management still critical for perf
- DPU pipeline
  - 14 stages, 24 threads, 450MHz
- DPUs can not communicate directly
  - Data sharing via host

Luna et al, Arxiv 2022

The DPU ISA

- RISC ISA

- Operators (non-exhaustive):
  - Arithmetic: ADD, SUB, AND, OR, XOR
  - Loads/Stores: SD, SW, SH, SB, LD, LW, LH, LB
  - Shift/Rotate: LSL, LSR, ROL, ROR
  - Count bits: CLZ, CLO, CLS, CAO
  - Multiplication/Division: MUL_STEP, DIV_STEP, MUL
  - DMA loads/stores: SDMA, LDMA, LDMAI

The DPU ISA

- RISC ISA
- No FPUs
  - FPs are software-emulated

- Operators (non-exhaustive):
  - Arithmetic: ADD, SUB, AND, OR, XOR
  - Loads/Stores: SD, SW, SH, SB, LD, LW, LH, LB
  - Shift/Rotate: LSL, LSR, ROL, ROR
  - Count bits: CLZ, CLO, CLS, CAO
  - Multiplication/Division: MUL_STEP, DIV_STEP, MUL
  - DMA loads/stores: SDMA, LDMA, LDMAI

The DPU ISA

- RISC ISA
- No FPUs
  - FPs are software-emulated
- The earlier versions also didn't have a multiplier
  - Was also software emulated
  - The recent version have smaller (8b?) multipliers

- Operators (non-exhaustive):
  - Arithmetic: ADD, SUB, AND, OR, XOR
  - Loads/Stores: SD, SW, SH, SB, LD, LW, LH, LB
  - Shift/Rotate: LSL, LSR, ROL, ROR
  - Count bits: CLZ, CLO, CLS, CAO
  - Multiplication/Division: MUL_STEP, DIV_STEP, MUL
  - DMA loads/stores: SDMA, LDMA, LDMAI

The DPU ISA

- RISC ISA
- No FPUs
  - FPs are software-emulated
- The earlier versions also didn't have a multiplier
  - Was also software emulated
  - The recent version have smaller (8b?) multipliers
- No vector instructions

- Operators (non-exhaustive):
  - Arithmetic: ADD, SUB, AND, OR, XOR
  - Loads/Stores: SD, SW, SH, SB, LD, LW, LH, LB
  - Shift/Rotate: LSL, LSR, ROL, ROR
  - Count bits: CLZ, CLO, CLS, CAO
  - Multiplication/Division: MUL_STEP, DIV_STEP, MUL
  - DMA loads/stores: SDMA, LDMA, LDMAI

Programmability of the UPMEM System

- Comes with its own SDK

Programmability of the UPMEM System

- Comes with its own SDK
- Application must have a host part and a device part

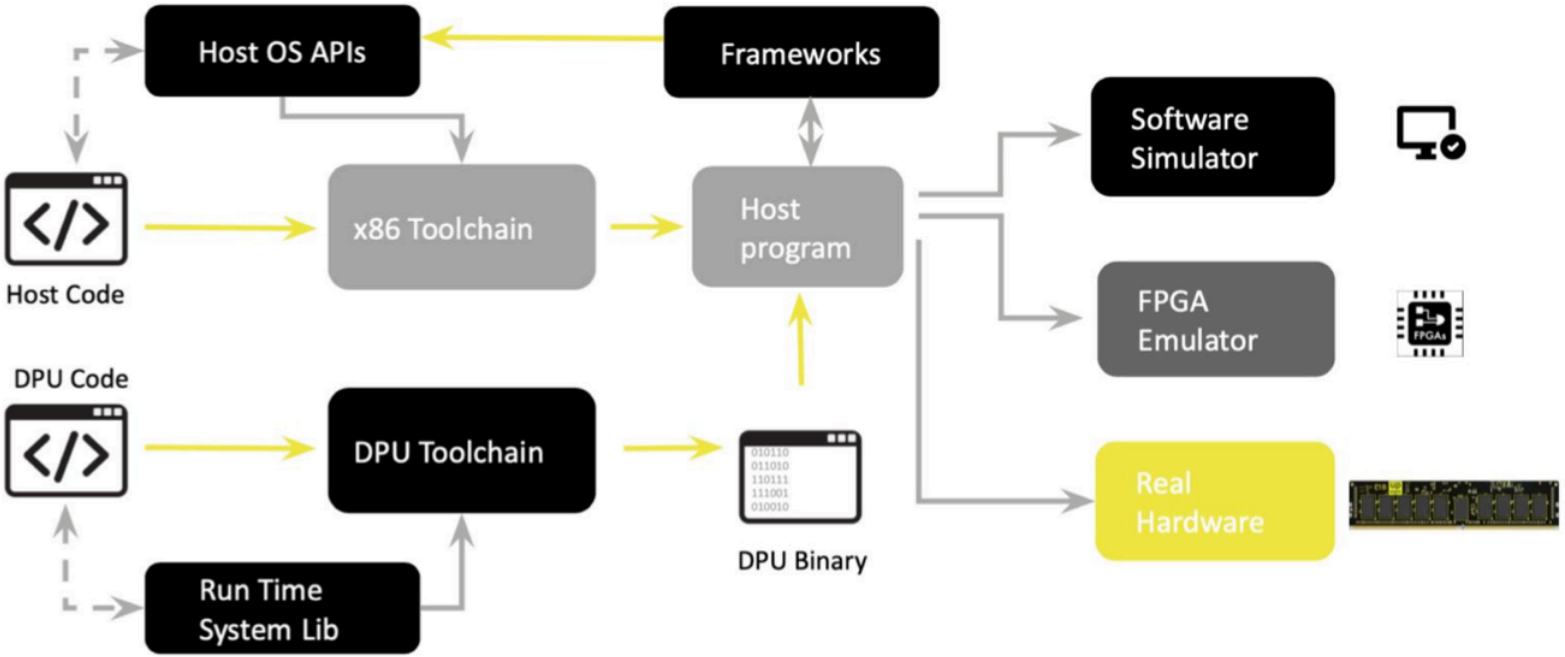
Programmability of the UPMEM System

- Comes with its own SDK
- Application must have a host part and a device part
- Host program
  - Compiled with the x86 toolchain
  - Uses UPMEM's host library
  - Includes DPU allocation, program loading, data copying etc.

Programmability of the UPMEM System

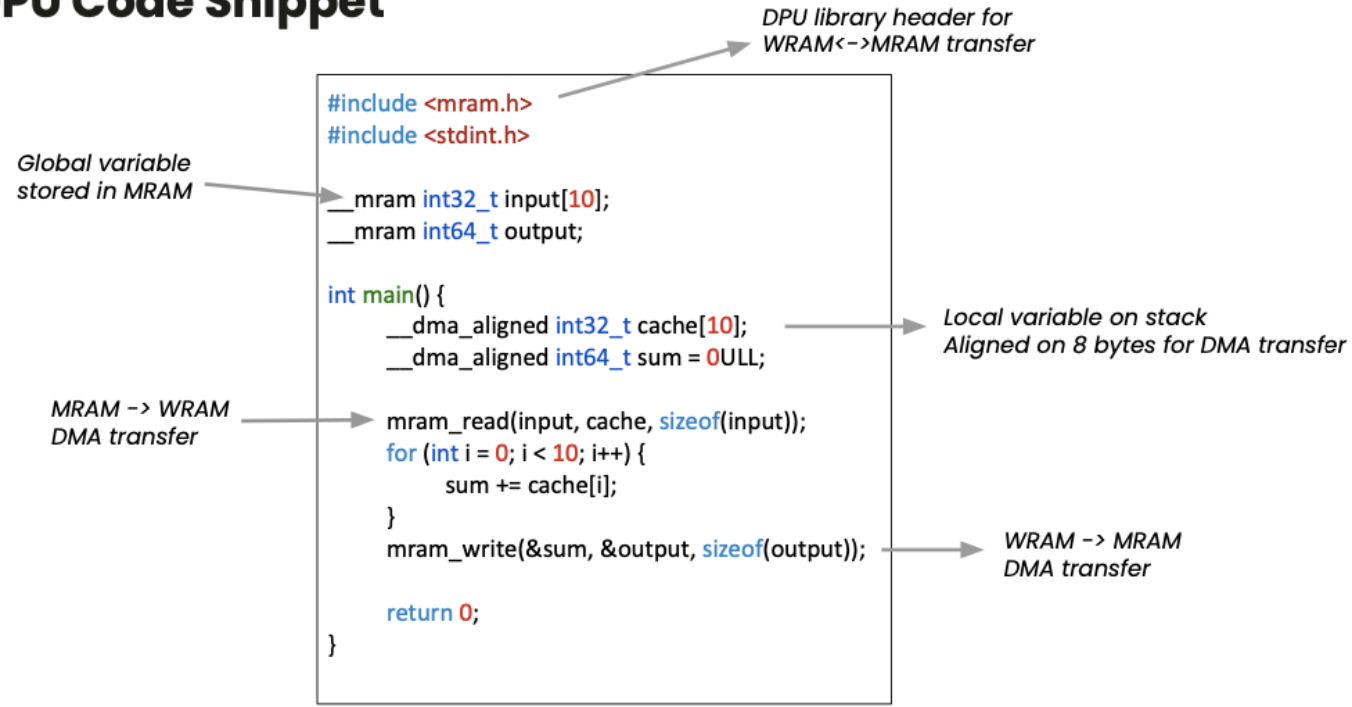
- Comes with its own SDK
- Application must have a host part and a device part
- Host program
  - Compiled with the x86 toolchain
  - Uses UPMEM's host library
  - Includes DPU allocation, program loading, data copying etc.
- Device program
  - Compiled using dpu-clang (LLVM based compiler for DPUs)
  - Uses DPU libraries for primitives such as synchronization (mutex locks), MRAM/WRAM management, DMA calls etc.

Programmability of the UPMEM System



Programmability of the UPMEM System

DPU Code Snippet



Programmability of the UPMEM System

Host Code Snippet

Include C Host library header

```
#include <dpu.h>

int array[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};

int main() {
    struct dpu_set_t dpu_set, dpu;
    DPU_ASSERT(dpu_alloc(1, "", dpu_set));
    DPU_ASSERT(dpu_load(dpu_set, "dpu_binary", NULL));

    DPU_FOREACH(dpu_set, dpu) {
        DPU_ASSERT(dpu_copy_to(dpu, "input", 0, array, sizeof(array)));
    }

    DPU_ASSERT(dpu_launch(set, DPU_SYNCHRONOUS));

    int64_t sum;
    DPU_FOREACH(dpu_set, dpu) {
        DPU_ASSERT(dpu_copy_from(dpu, "output", 0, &sum, sizeof(sum)));
    }
    return 0;
}
```

Allocate 1 DPU

Load DPU program

Copy the array content to the DPU variable named "input"

Boot the DPU program and wait for it to finish

Copy the DPU variable named "output" to the host variable sum

Programmability of the UPMEM System

- Programmability, as per UPMEM, is easy

Programmability of the UPMEM System

- Programmability, as per UPMEM, is easy

BUT

Programmability of the UPMEM System

- Programmability, as per UPMEM, is easy

BUT

- Programmer is responsible for
  - Load balancing in thousands of DPUs
  - Explicit data movement between
    - Main memory and MRAM
    - MRAM and WRAM

Programmability of the UPMEM System

- Programmability, as per UPMEM, is easy

BUT

- Programmer is responsible for
  - Load balancing in thousands of DPUs
  - Explicit data movement between
    - Main memory and MRAM
    - MRAM and WRAM
  - Data coherency

Programmability of the UPMEM System

- Programmability, as per UPMEM, is easy

BUT

- Programmer is responsible for
  - Load balancing in thousands of DPUs
  - Explicit data movement between
    - Main memory and MRAM
    - MRAM and WRAM
  - Data coherency

- Summary

- Simple
- Local computations
- Data communications
- Application specific
- Programmability

Programmability of the UPMEM System

- UPMEM SDK still requires you to know all device details

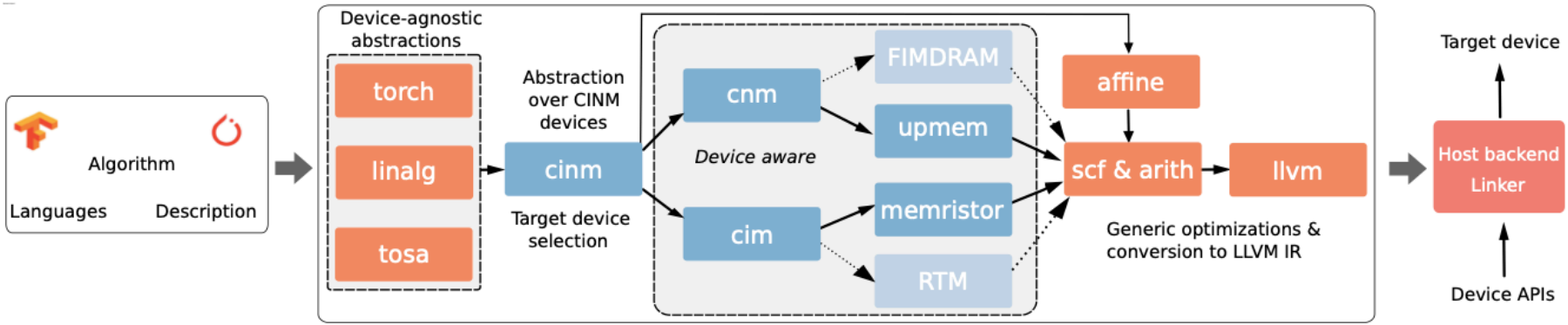
Programmability of the UPMEM System

- UPMEM SDK still requires you to know all device details
- There are high-level compilation frameworks that accept device-agnostic representations

Programmability of the UPMEM System

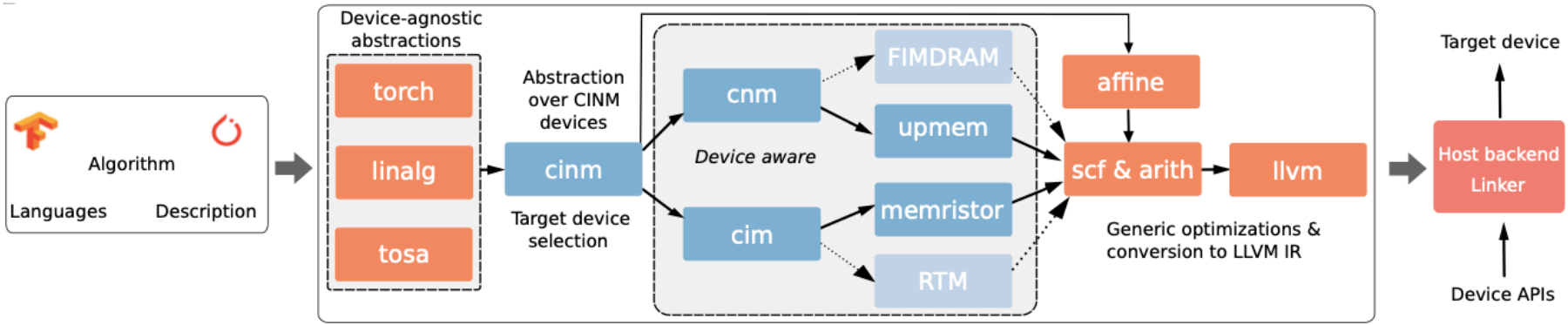
- UPMEM SDK still requires you to know all device details
- There are high-level compilation frameworks that accept device-agnostic representations
- .. and take care of all other critical jobs, e.g., efficient scheduling and load-balancing

The Cinammon flow



Khan et al., Arxiv 2024

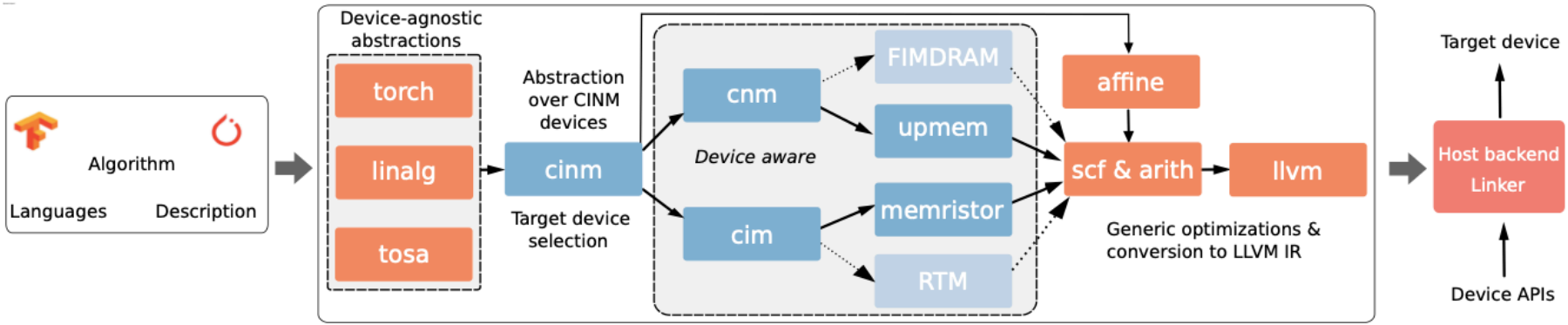
The Cinammon flow



- Automatically takes care of all the lowerings and device calls

Khan et al., Arxiv 2024

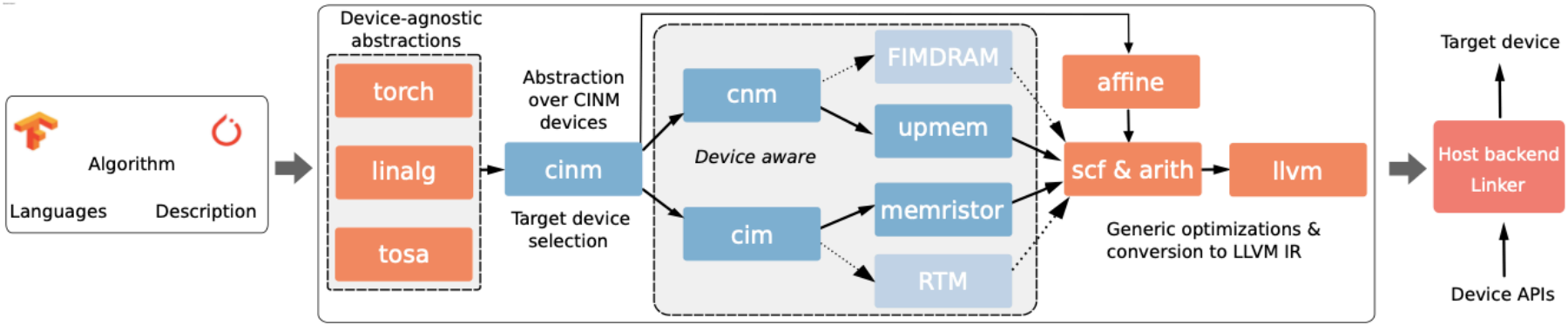
The Cinammon flow



- Automatically takes care of all the lowerings and device calls
- Performs device-specific optimizations

Khan et al., Arxiv 2024

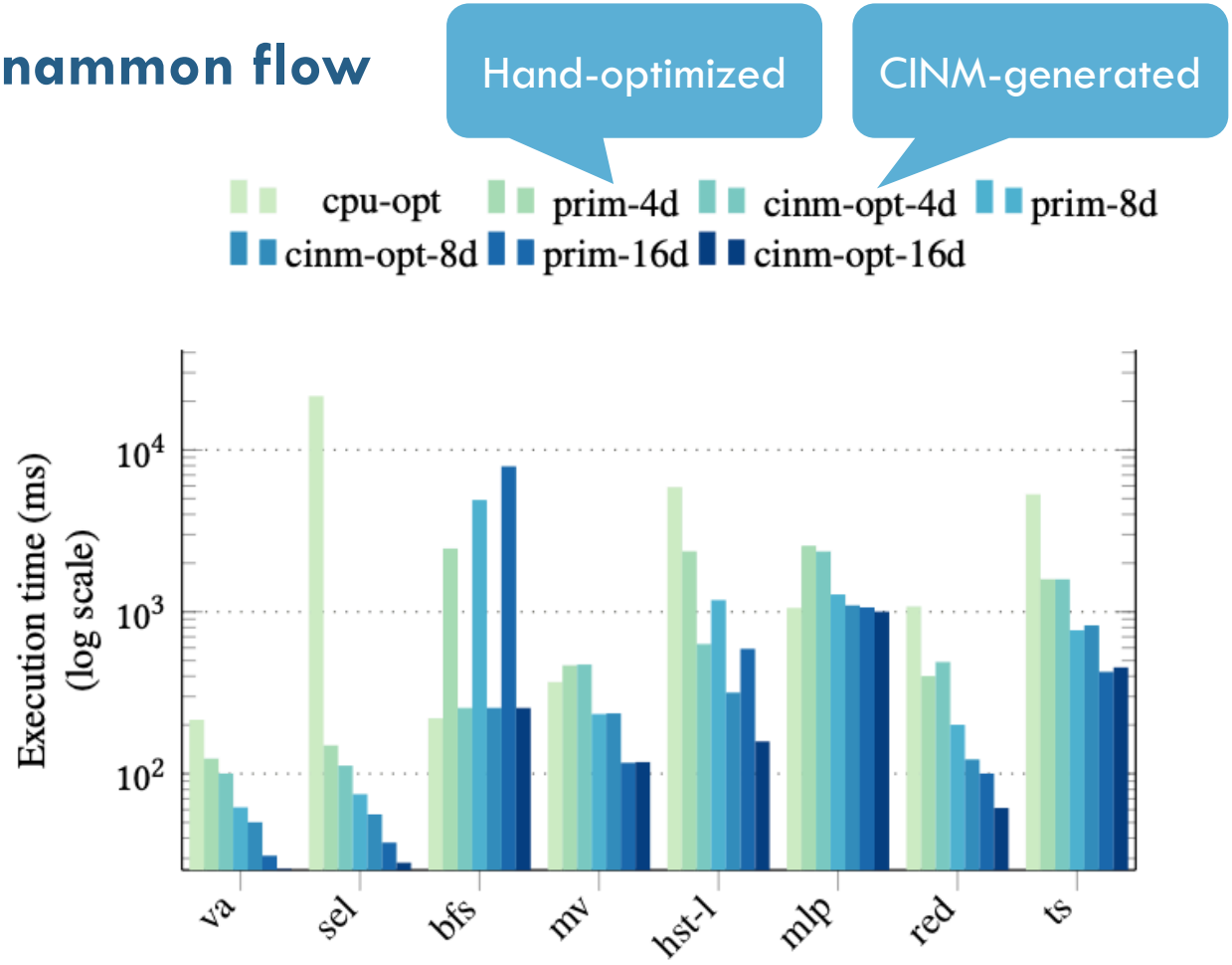
The Cinammon flow



- Automatically takes care of all the lowerings and device calls
- Performs device-specific optimizations
- Generates code that is better/comparable to hand-optimized codes

Khan et al., Arxiv 2024

The Cinammon flow

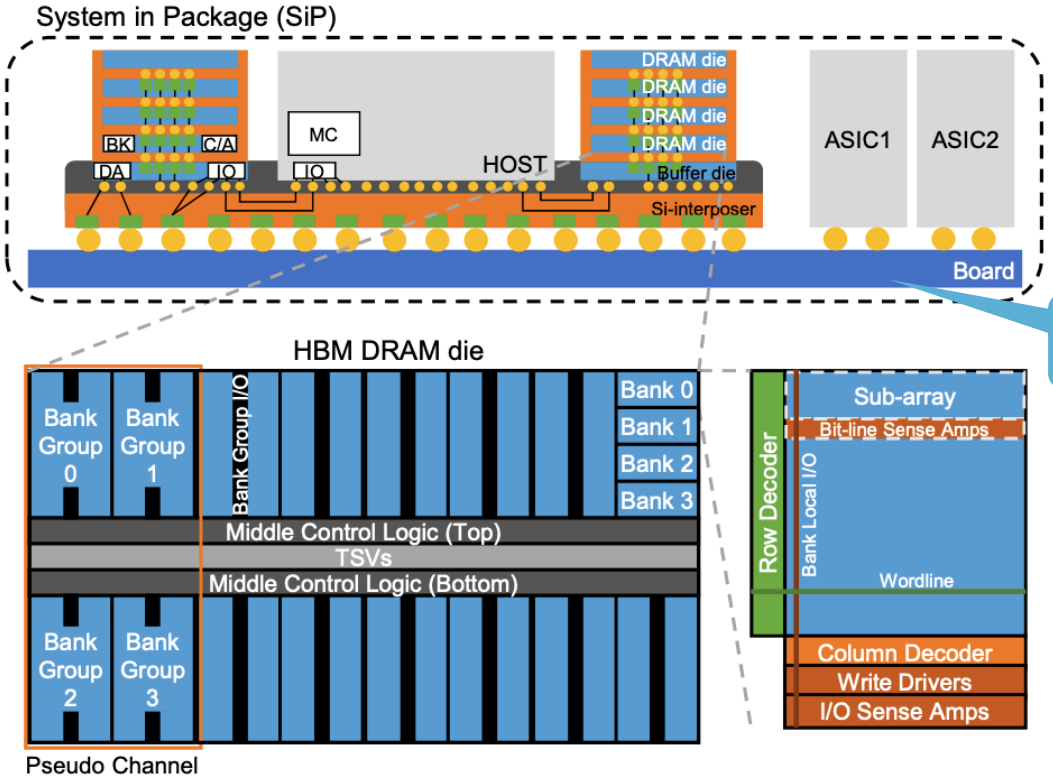


Khan et al., Arxiv 2024

UPMEM resources

- Readings:
  - Luna et al., "Benchmarking Memory-Centric Computing Systems: Analysis of Real Processing-in-Memory Hardware", Arxiv 2023
- Simulators:
  - UPMEM SDK: https://www.upmem.com/developer/
  - uPIMulator: Hyun et al., "Pathfinding Future PIM Architectures by Demystifying a Commercial PIM Technology", Arxiv 2024
- Benchmark suites
  - https://github.com/CMU-SAFARI/prim-benchmarks (PrIM benchmarks)

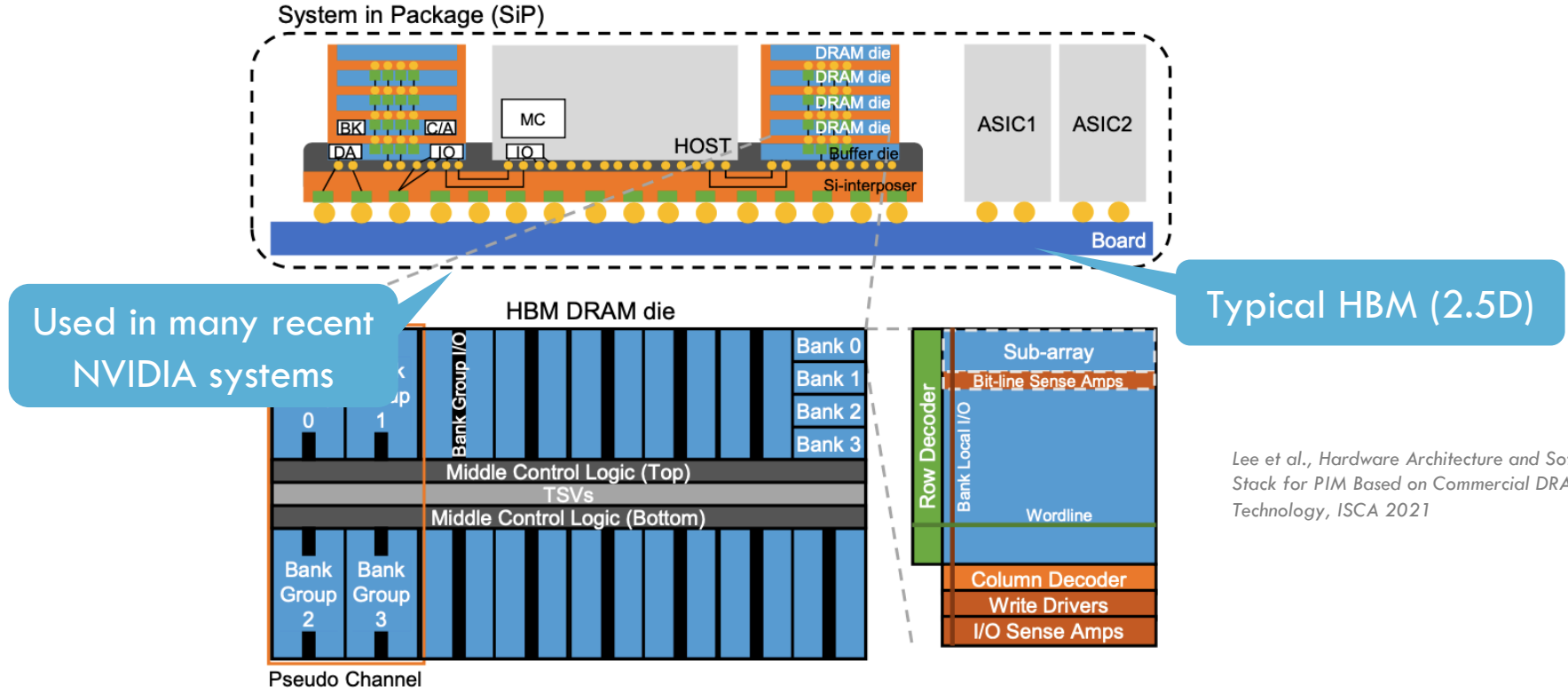
Samsung's FIMDRAM architecture



Typical HBM (2.5D)

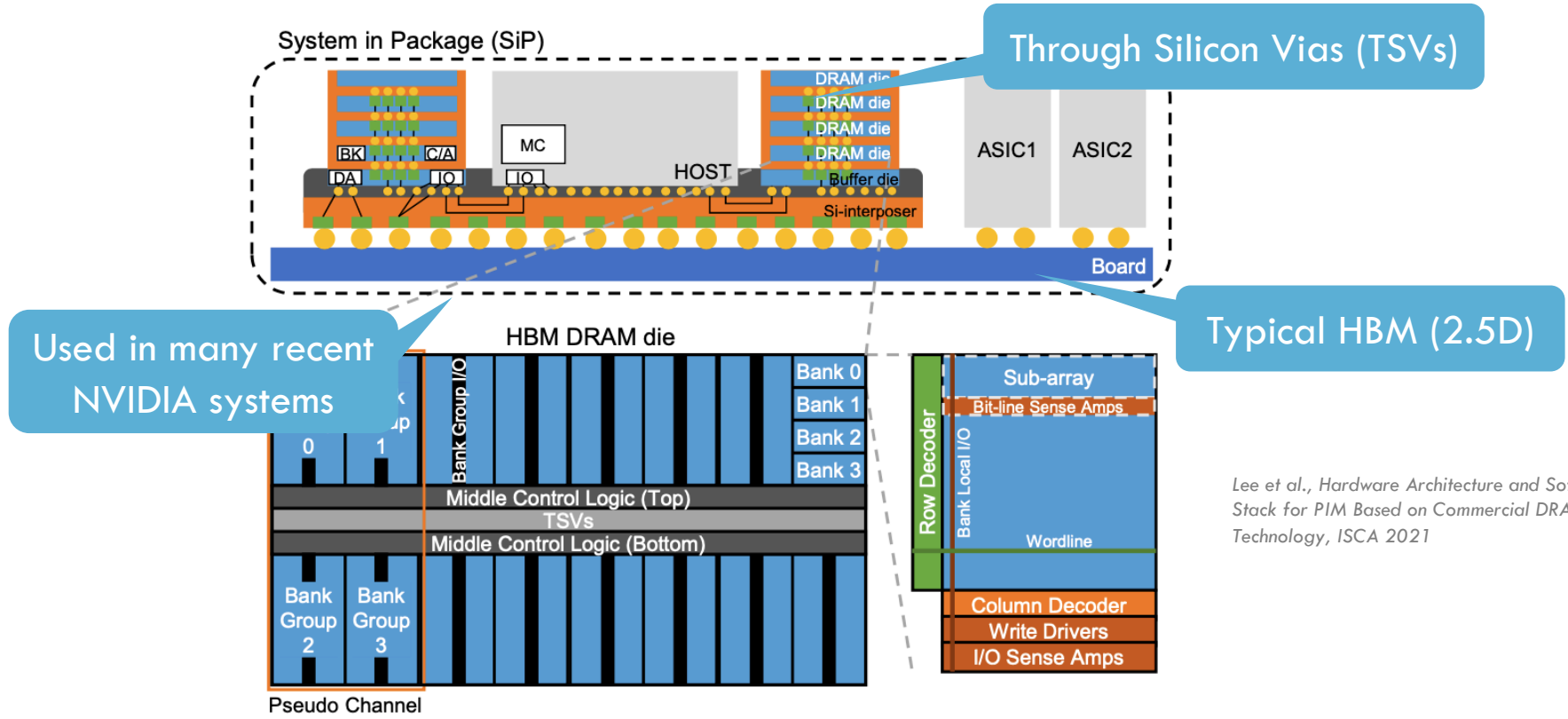
Lee et al., Hardware Architecture and Software Stack for PIM Based on Commercial DRAM Technology, ISCA 2021

Samsung's FIMDRAM architecture



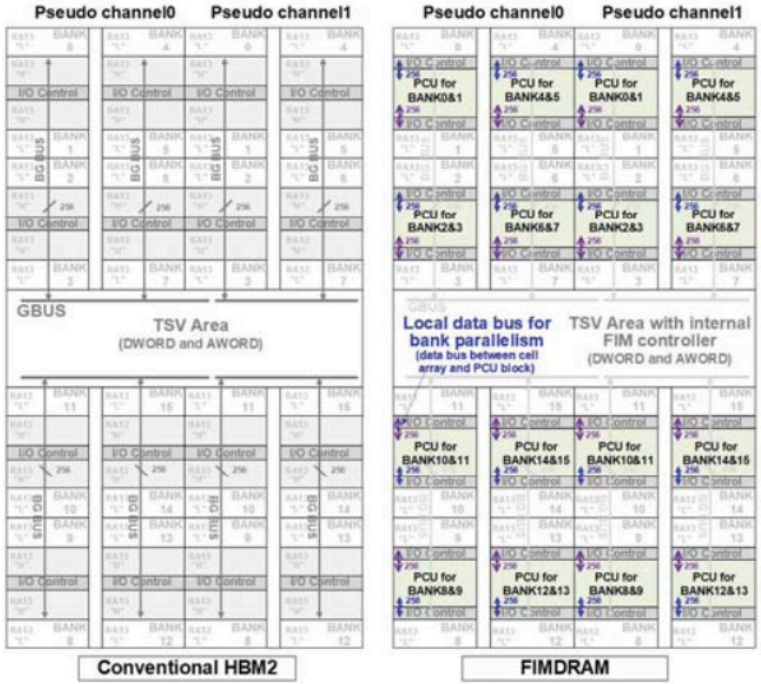
Lee et al., Hardware Architecture and Software Stack for PIM Based on Commercial DRAM Technology, ISCA 2021

Samsung's FIMDRAM architecture



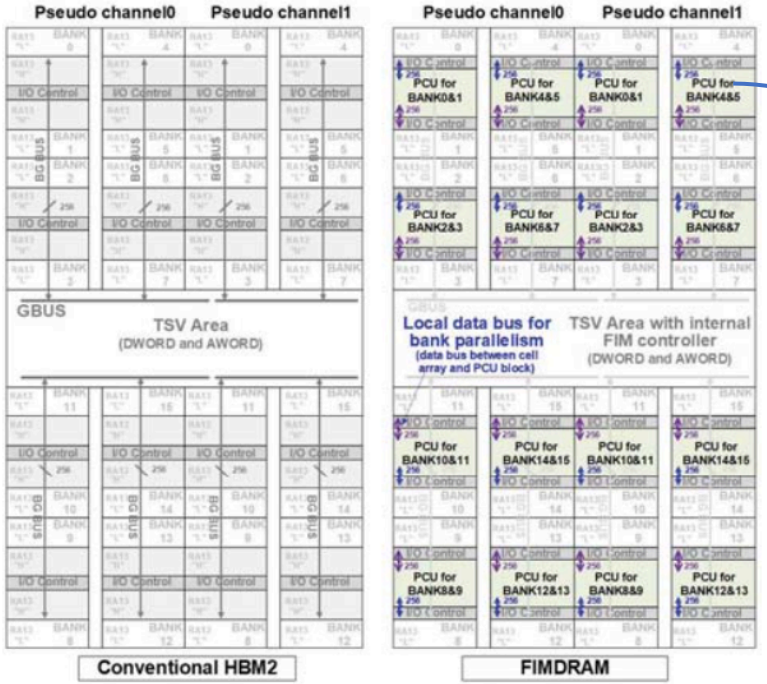
Lee et al., Hardware Architecture and Software Stack for PIM Based on Commercial DRAM Technology, ISCA 2021

Samsung's FIMDRAM architecture



Kwon et al., ISSCC 2021

Samsung's FIMDRAM architecture



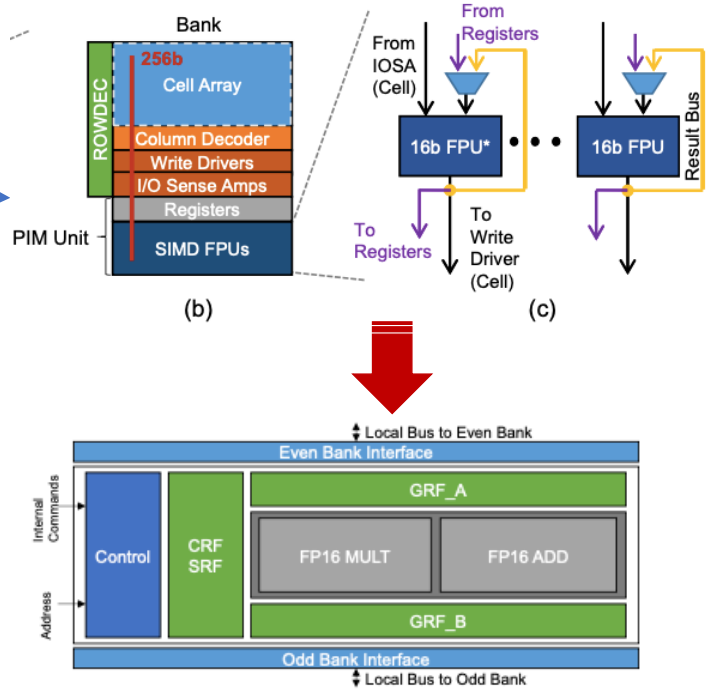
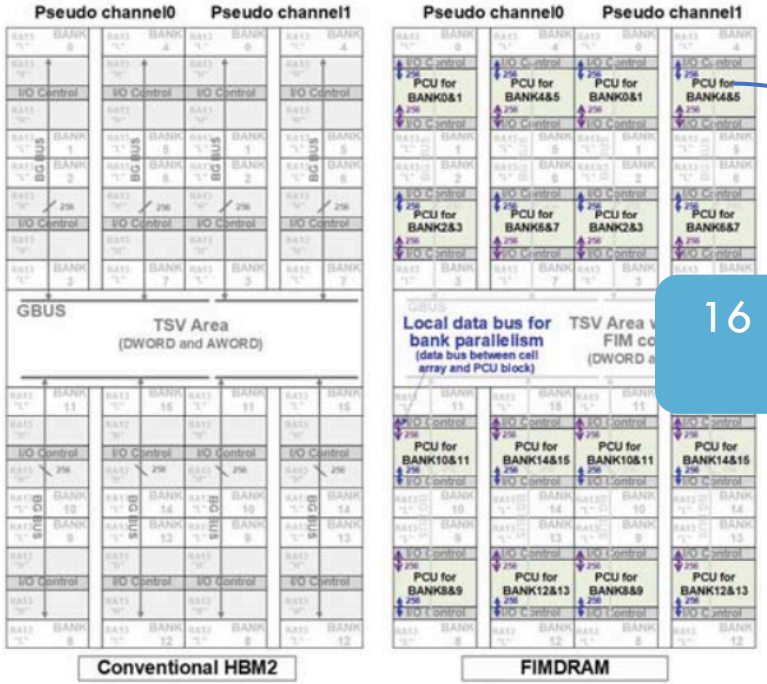


Fig. 4: The microarchitecture of PIM execution unit (1) an instruction sequence manager (blue color), (2) register files (green color), and (3) a 16-wide SIMD FPU (gray color).

Kwon et al., ISSCC 2021

Samsung's FIMDRAM architecture



16 SIMD FPUs per PIM unit

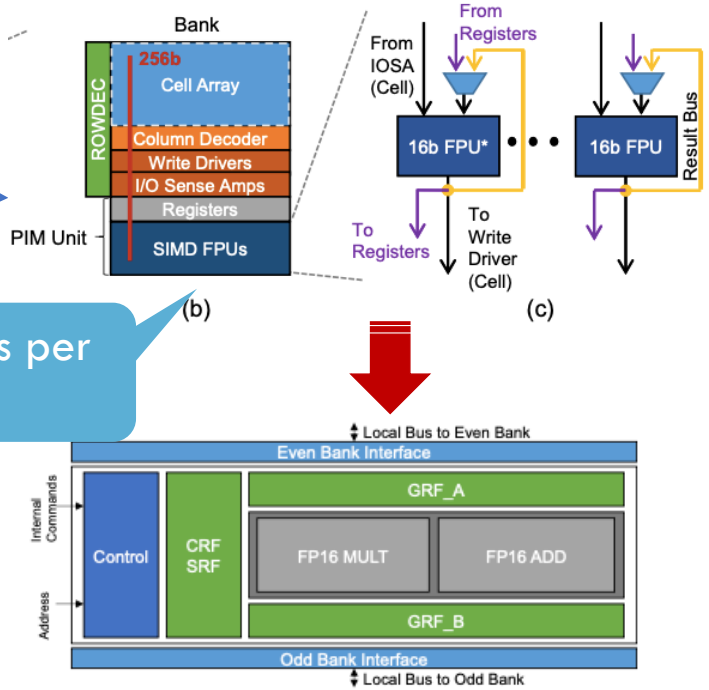


Fig. 4: The microarchitecture of PIM execution unit (1) an instruction sequence manager (blue color), (2) register files (green color), and (3) a 16-wide SIMD FPU (gray color).

Kwon et al., ISSCC 2021

Samsung's FIMDRAM architecture

- Responds to standard DRAM commands

Samsung's FIMDRAM architecture

- Responds to standard DRAM commands
- Pipeline having 5 stages

Samsung's FIMDRAM architecture

- Responds to standard DRAM commands
- Pipeline having 5 stages
- Bank-level parallelism (AB, SB)

Samsung's FIMDRAM architecture

- Responds to standard DRAM commands
- Pipeline having 5 stages
- Bank-level parallelism (AB, SB)
- No changes to the memory controller on the CPU side

Samsung's FIMDRAM architecture

- Responds to standard DRAM commands
- Pipeline having 5 stages
- Bank-level parallelism (AB, SB)
- No changes to the memory controller on the CPU side

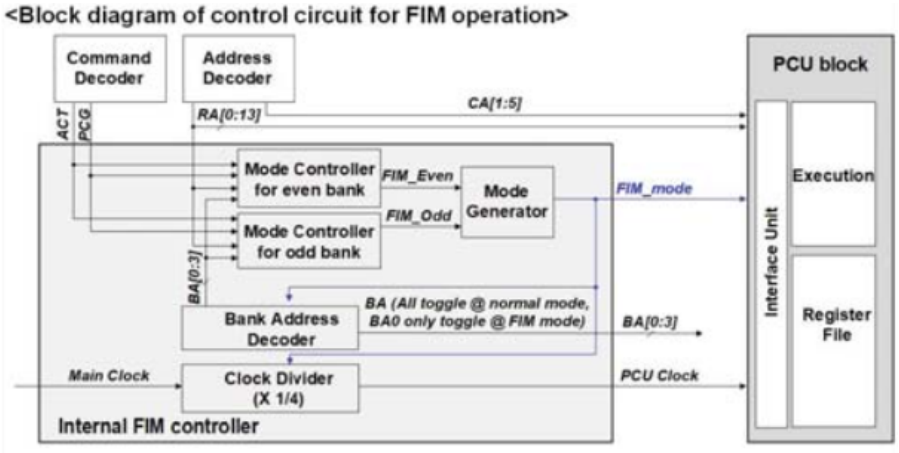
The UPMEM architectures

<Key Feature Summary>

	[3]	[7]	[8]	FIMDRAM (this work)
Type of DRAM	HBM2	LPDDR4	DDR4	HBM2
Process	20 nm	20 nm	2x nm	20 nm
Memory density	8GB/cube (Buffer-die + 8H 8Gb core-die)	8GB/chip (8H 8Gb mono die)	8GB/DIMM	6GB/cube (Buffer-die + 4H 4Gb core-die with PCU + 4H 8Gb core-die)
Data rate	2.4Gbps	3.2Gbps	2.4Gbps	2.4Gbps
Bandwidth	307GB/s per cube	25.6GB/s per chip	19.2GB/s per DIMM	307GB/s per cube
# of CH	8 per cube	1 per chip	16 per DIMM	8 per cube
# of processing unit	No	2048 per chip (256 per die)	128 per DIMM (8 per chip)	128 per cube (32 per core-die)
Processing operation speed	-	250Mhz @simulation	500MHz @ Measurement	300MHz @ Measurement
Peak throughput	-	0.5 TOPS per chip (250MHz x 256 x 8byte)	0.5 TOPS per DIMM (500MHz x 128 x 8byte)	1.2 TFLOPS per cube (300MHz x 128 x 32byte)
Operation Precision	-	INT8	INT8	FP16

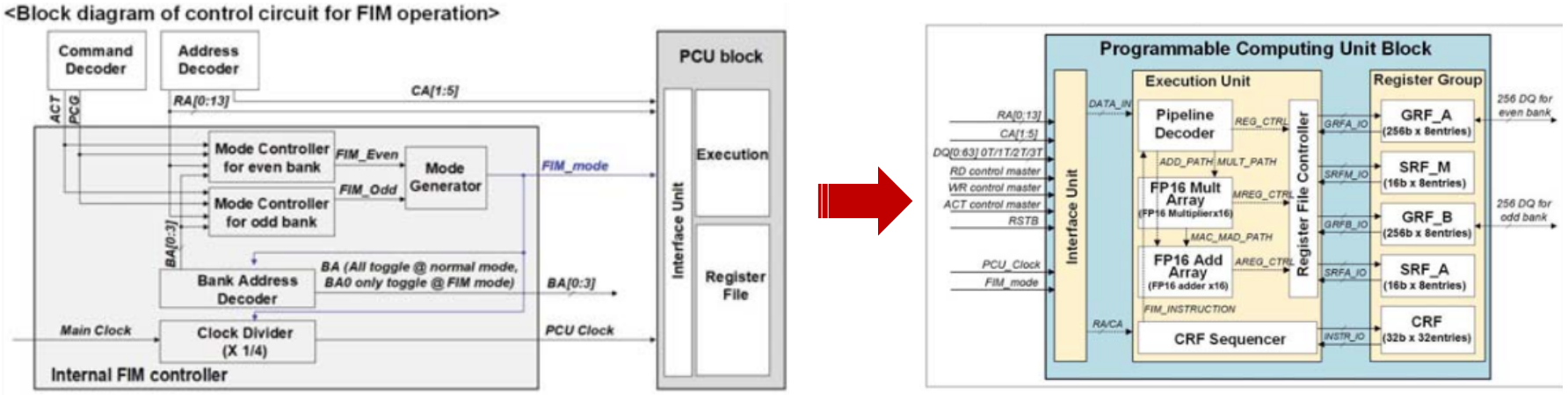
Kwon et al., ISSCC 2021

Samsung's FIMDRAM architecture



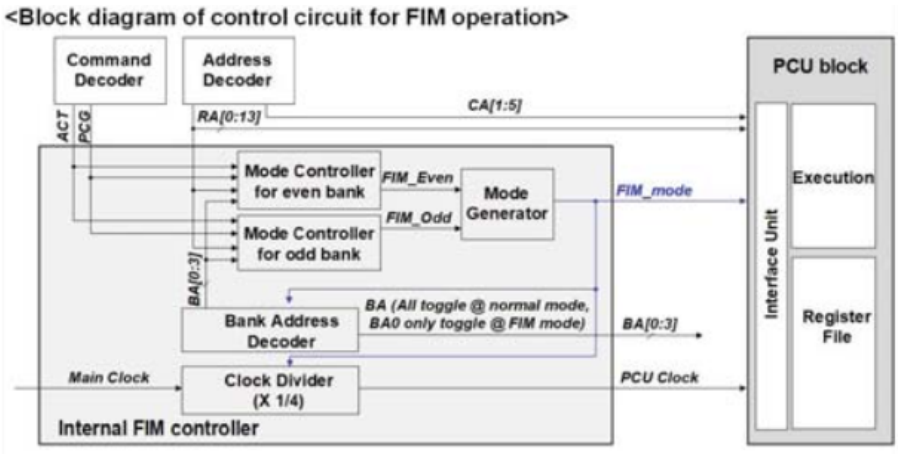
Kwon et al., ISSCC 2021

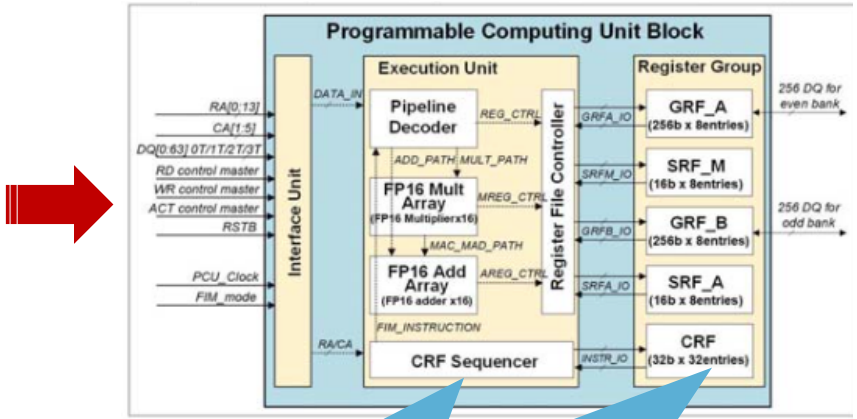
Samsung's FIMDRAM architecture



Kwon et al., ISSCC 2021

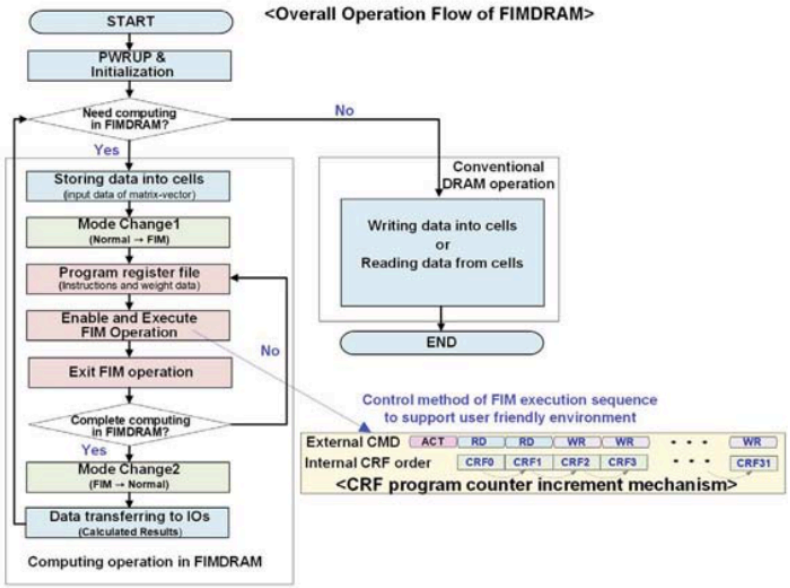
Samsung's FIMDRAM architecture





Kwon et al., ISSCC 2021

Samsung's FIMDRAM architecture



<Available instruction list for FIM operation>

Type	CMD	Description
Floating Point	ADD	FP16 addition
	MUL	FP16 multiplication
	MAC	FP16 multiply-accumulate
	MAD	FP16 multiply and add
Data Path	MOVE	Load or store data
	FILL	Copy data from bank to GRFs
Control Path	NOP	Do nothing
	JUMP	Jump instruction
	EXIT	Exit instruction

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Control	OPCODE				U									IMM0								IMM1										
Data	OPCODE				DST				SRC0					U						R	U	DST #			U	SRC0 #			U	SRC1 #		
ALU	OPCODE				DST				SRC0			SRC1		SRC2			A		U		U	DST #			U	SRC0 #			U	SRC1 #		

Figure 25.4.5: Operation flow for data computing of FIMDRAM.

Kwon et al., ISSCC 2021

Samsung's FIMDRAM architecture

On system boot, dedicate PIM space

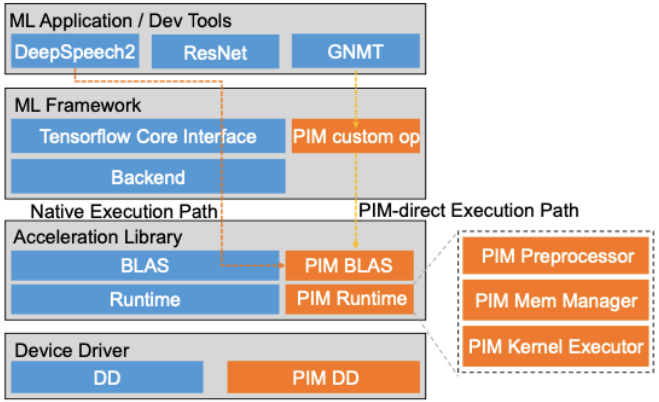


Fig. 6: PIM software stack. The native execution path does not require any modification of application source code (orange arrow). The PIM-direct execution path needs to explicitly use PIM TF custom ops (yellow arrow). 'DD' denotes device driver.

Lee et al., ISCA 2021

Samsung's FIMDRAM architecture

No changes to application sources

On system boot, dedicate PIM space

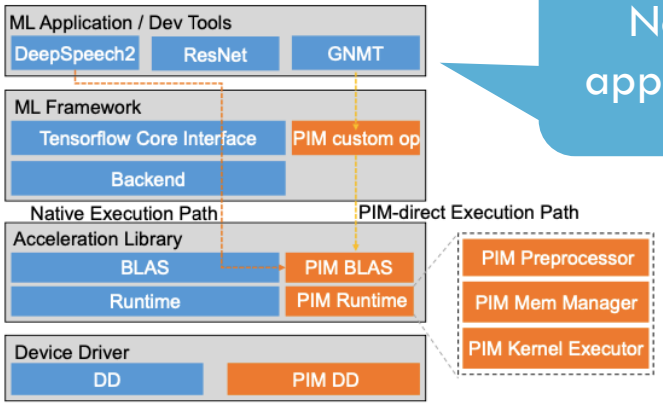
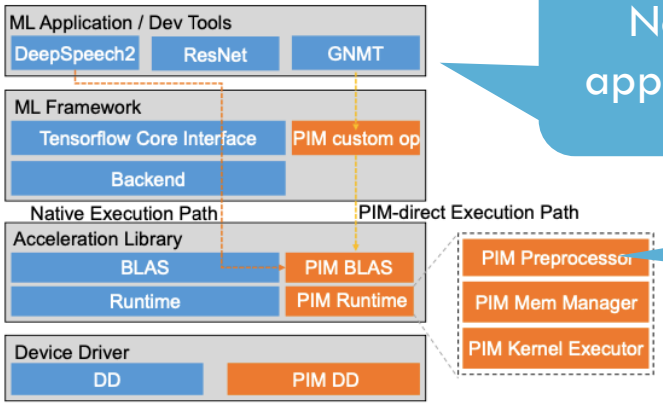


Fig. 6: PIM software stack. The native execution path does not require any modification of application source code (orange arrow). The PIM-direct execution path needs to explicitly use PIM TF custom ops (yellow arrow). 'DD' denotes device driver.

Lee et al., ISCA 2021

Samsung's FIMDRAM architecture

No changes to application sources



Identify PIM friendly OPs

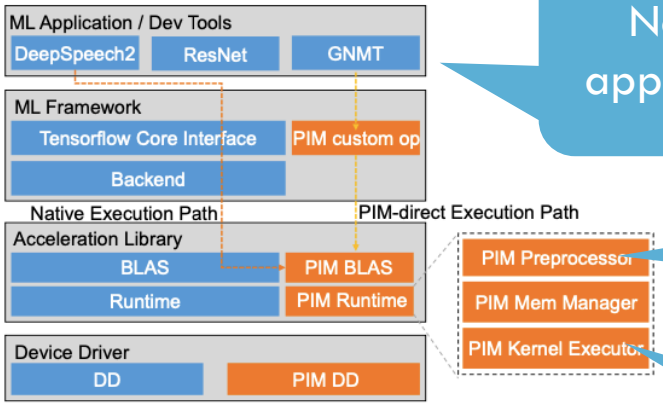
On system boot, dedicate PIM space

Fig. 6: PIM software stack. The native execution path does not require any modification of application source code (orange arrow). The PIM-direct execution path needs to explicitly use PIM TF custom ops (yellow arrow). 'DD' denotes device driver.

Lee et al., ISCA 2021

Samsung's FIMDRAM architecture

No changes to application sources



On system boot, dedicate PIM space

Identify PIM friendly OPs

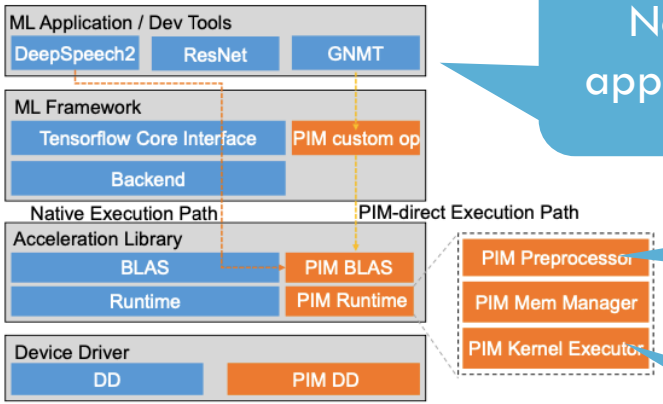
Trigger PIM units exec

Fig. 6: PIM software stack. The native execution path does not require any modification of application source code (orange arrow). The PIM-direct execution path needs to explicitly use PIM TF custom ops (yellow arrow). 'DD' denotes device driver.

Lee et al., ISCA 2021

Samsung's FIMDRAM architecture

No changes to application sources



Identify PIM friendly OPs

On system boot, dedicate PIM space

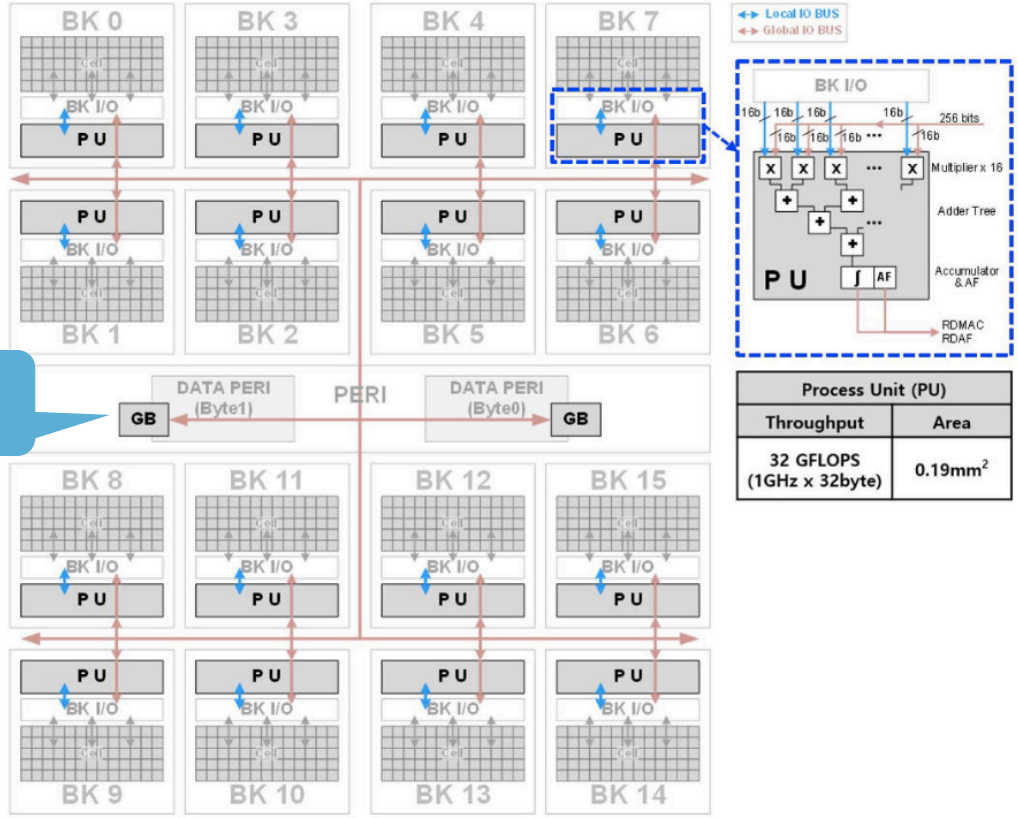
Trigger PIM units exec

Fig. 6: PIM software stack. The native execution path does not require any modification of application source code (orange arrow). The PIM-direct execution path needs to explicitly use PIM TF custom ops (yellow arrow). 'DD' denotes device driver.

Summary: Compared to UPMEM, it has its own strengths and limits

Lee et al., ISCA 2021

AiM: Accelerator in memory (SK Hynix)

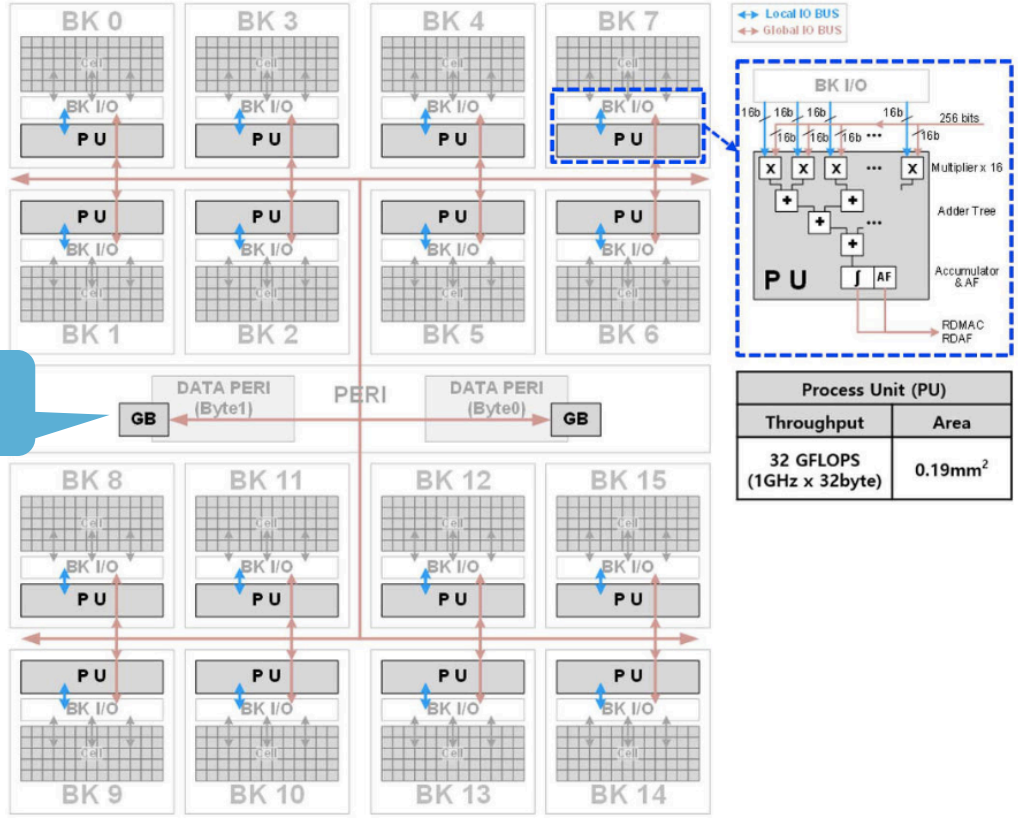


2KB SRAM

Process Unit (PU)	
Throughput	Area
32 GFLOPS (1GHz x 32byte)	0.19mm^2

Lee et al., ISSCC 2022

AiM: Accelerator in memory (SK Hynix)



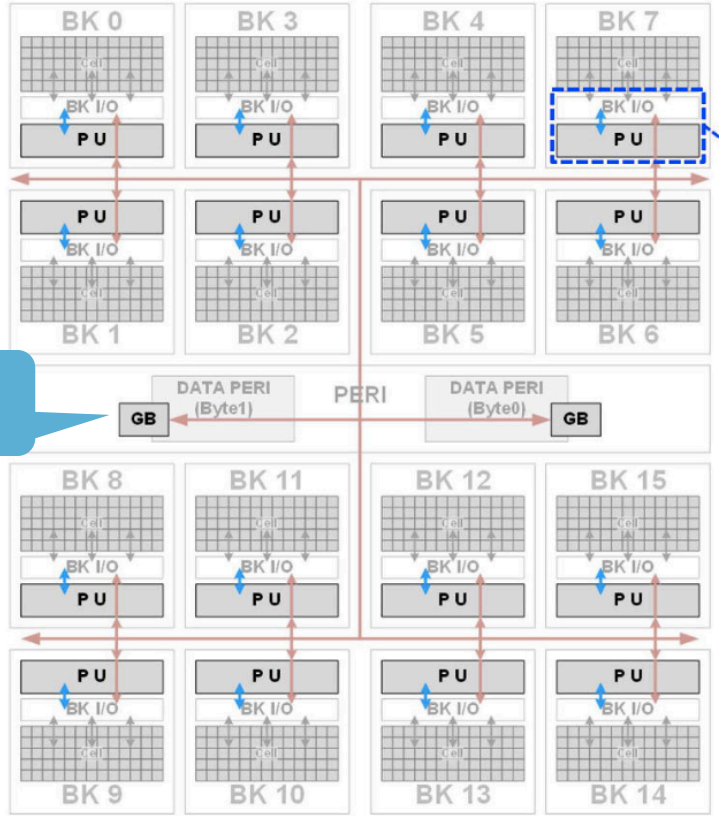
Process Unit (PU)	
Throughput	Area
32 GFLOPS (1GHz x 32byte)	0.19mm^2

2KB SRAM

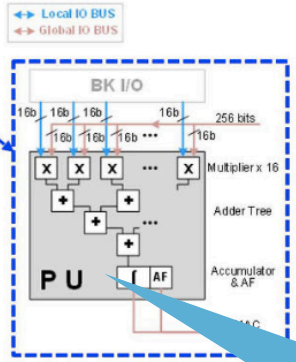
Mul/Add tree

Lee et al., ISSCC 2022

AiM: Accelerator in memory (SK Hynix)



2KB SRAM



Process Unit (PU)	
Throughput	Area
32 GFLOPS (1GHz x 32byte)	0.19mm^2

Mul/Add tree

Adder tree can be disabled (e.g., for vec dot product)

Lee et al., ISSCC 2022

AiM: Accelerator in memory (SK Hynix)

Type	CMD	Description
Bank Activation	ACT4, ACT16	Activate four/sixteen banks in parallel
	ACTAF4, ACTAF16	Activate rows storing activation function LUTs in four/sixteen banks in parallel
Compute	MACSB, MAC4B, MACAB	Perform MAC in one/four/sixteen banks in parallel
	AF	Compute activation function in all banks
	EWMUL	Perform element-wise multiplication
Data Movement	RDCP	Copy data from a bank to Global Buffer
	WRCP	Copy data from Global Buffer to a bank
	WRGB	Write to Global Buffer
	RDMAC	Read from MAC result register
	RDAF	Read from activation function result register
	WRBIAS	Write bias to MAC result register
	WRBK	Write to all activated banks in parallel

Interesting ops such as row clone can be performed

Lee et al., ISSCC 2022

AiM: Accelerator in memory (SK Hynix)

Comparison

	[1]	[2]	[3],[7]	This work
DRAM Type	LPDDR4	DDR4	HBM2	GDDR6
Process	20 nm	2x nm	20 nm	1y nm
Memory Density	8GB/chip (8H 8Gb mono die)	8GB/DIMM	6GB/cube (Buffer die + 4H 4Gb core-die with PCU + 4H 8Gb core-die)	8Gb/chip (4Gb DDP)
Data Rate	3.2Gbps	2.4Gbps	2.4Gbps	16Gbps
Bandwidth	25.6GB/s per chip	19.2GB/s per DIMM	307GB/s per cube	64GB/s per chip
# of Channel	1 per chip	16 per DIMM	8 per cube	2 per chip
# of Processing Unit (PU)	2048 per chip (256 per die)	128 per DIMM (8 per chip)	128 per cube (32 per core-die)	32 per chip (16 per die)
Processing Operation Speed	250MHz	500MHz	300MHz	1GHz
1 PU Throughput	2 GOPS (250MHz x 8byte)	4 GOPS (500MHz x 8byte)	9.6 GFLOPS (300MHz x 32byte)	32 GFLOPS (1GHz x 32byte)
Total Throughput (1 PU Throughput x # of PU)	0.5 TOPS per chip (2 GOPS x 256)	0.5 TOPS per DIMM (4 GOPS x 128)	1.2 TFLOPS per cube (9.6 GFLOPS x 128)	1 TFLOPS per chip (32 GFLOPS x 32)
Operation precision	INT8	INT8	FP16	BF16
Supported Activation Functions	-	-	ReLU	Sigmoid, Tanh, GELU, ReLU, Leaky ReLU, and Arbitrary AF

Probably not fair though

Lee et al., ISSCC 2022

AiM: Accelerator in memory (SK Hynix)

Comparison

	[1]	[2]	[3],[7]	This work
DRAM Type	LPDDR4	DDR4	HBM2	GDDR6
Process	20 nm	2x nm	20 nm	1y nm
Memory Density	8GB/chip (8H 8Gb mono die)	8GB/DIMM	6GB/cube (Buffer die + 4H 4Gb core-die with PCU + 4H 8Gb core-die)	8Gb/chip (4Gb DDP)
Data Rate	3.2Gbps	2.4Gbps	2.4Gbps	16Gbps
Processing Operation Speed	250MHz	500MHz	300MHz	1GHz
1 PU Throughput	2 GOPS (250MHz x 8byte)	4 GOPS (500MHz x 8byte)	9.6 GFLOPS (300MHz x 32byte)	32 GFLOPS (1GHz x 32byte)
Total Throughput (1 PU Throughput x # of PU)	0.5 TOPS per chip (2 GOPS x 256)	0.5 TOPS per DIMM (4 GOPS x 128)	1.2 TFLOPS per cube (9.6 GFLOPS x 128)	1 TFLOPS per chip (32 GFLOPS x 32)
Operation precision	INT8	INT8	FP16	BF16
Supported Activation Functions	-	-	ReLU	Sigmoid, Tanh, GELU, ReLU, Leaky ReLU, and Arbitrary AF

Probably not fair though

No winner. Some are more general, others are more performant. Slowly and steadily, we will get there.

Lee et al., ISSCC 2022

Thank you!

asif.ali@uetpeshawar.edu.pk