

1st Workshop

Memory-Centric Computing: Research Challenges & Closing Remarks

Geraldo F. Oliveira

<https://geraldofojunior.github.io>

ASPLOS 2025

30 March 2025

Eliminating the Adoption Barriers

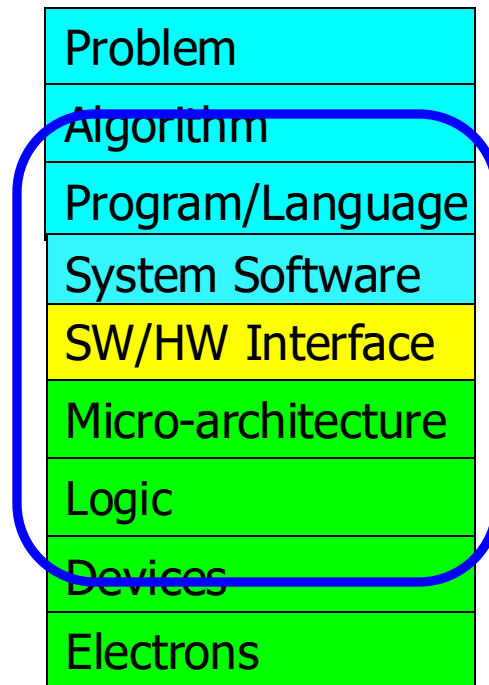
How to Enable Adoption of Processing in Memory

Potential Barriers to Adoption of PIM

1. **Applications & software** for PIM
2. Ease of **programming** (interfaces and compiler/HW support)
3. **System** and **security** support: coherence, synchronization, virtual memory, isolation, communication interfaces, ...
4. **Runtime** and **compilation** systems for adaptive scheduling, data mapping, access/sharing control, ...
5. **Infrastructures** to assess benefits and feasibility

All can be solved with change of mindset

We Need to Revisit the Entire Stack



We can get there step by step

Adoption: How to Keep It Simple?

- Junwhan Ahn, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi, **"PIM-Enabled Instructions: A Low-Overhead, Locality-Aware Processing-in-Memory Architecture"** *Proceedings of the 42nd International Symposium on Computer Architecture (ISCA)*, Portland, OR, June 2015. [[Slides \(pdf\)](#)] [[Lightning Session Slides \(pdf\)](#)]

PIM-Enabled Instructions: A Low-Overhead, Locality-Aware Processing-in-Memory Architecture

Junwhan Ahn Sungjoo Yoo Onur Mutlu[†] Kiyoung Choi

junwhan@snu.ac.kr, sungjoo.yoo@gmail.com, onur@cmu.edu, kchoi@snu.ac.kr

Seoul National University

[†]Carnegie Mellon University

Adoption: How to Ease Programmability? (I)

- Geraldo F. Oliveira, Alain Kohli, David Novo, Juan Gómez-Luna, Onur Mutlu,
“DaPPA: A Data-Parallel Framework for Processing-in-Memory Architectures,”
in *PACT SRC Student Competition*, Vienna, Austria, October 2023.

DaPPA: A Data-Parallel Framework for Processing-in-Memory Architectures

Geraldo F. Oliveira*

Alain Kohli*

David Novo‡

Juan Gómez-Luna*

Onur Mutlu*

**ETH Zürich*

‡*LIRMM, Univ. Montpellier, CNRS*

Adoption: How to Ease Programmability? (II)

- Jinfan Chen, Juan Gómez-Luna, Izzat El Hajj, YuXin Guo, and Onur Mutlu,
"SimplePIM: A Software Framework for Productive and Efficient Processing in Memory"
Proceedings of the 32nd International Conference on Parallel Architectures and Compilation Techniques (PACT), Vienna, Austria, October 2023.

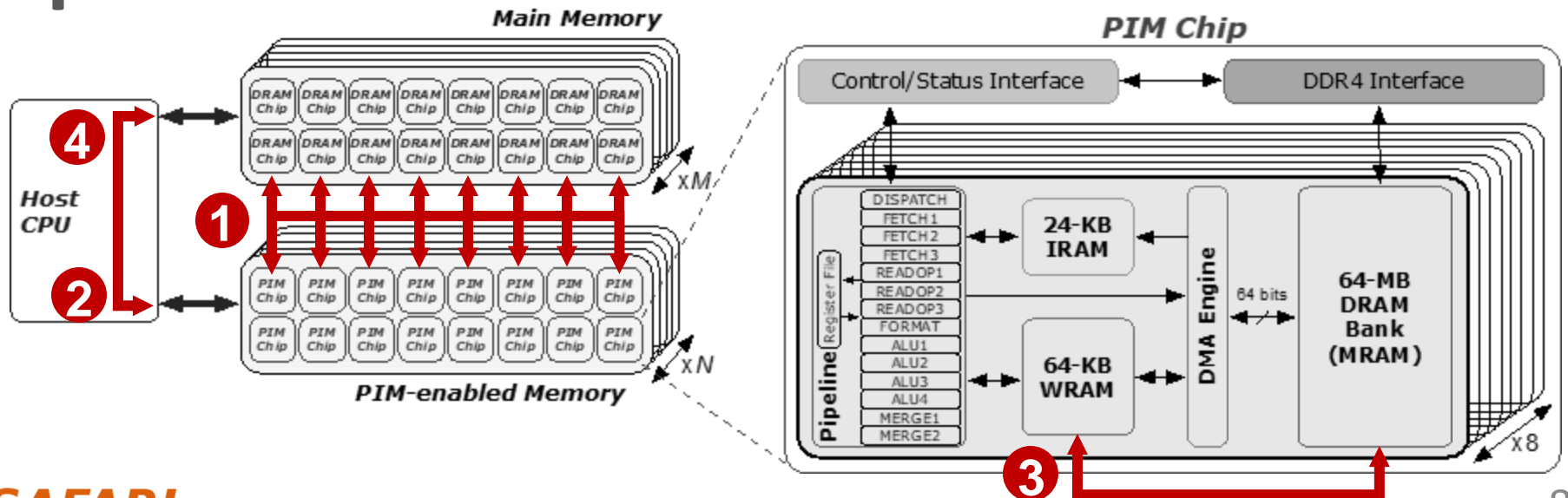
SimplePIM: A Software Framework for Productive and Efficient Processing-in-Memory

Jinfan Chen¹ Juan Gómez-Luna¹ Izzat El Hajj² Yuxin Guo¹ Onur Mutlu¹
¹ETH Zürich ²American University of Beirut

The Programmability Barrier: Overview

Programming the UPMEM-based system requires:

- 1 **Splitting** input data and computation across PIM chips
- 2 **Transferring input data** from main memory to PIM chips
- 3 **Manually handling caching** in PIM's scratchpad memory
- 4 **Transferring output data** from PIM chips to main memory



The Programmability Barrier:

Vector Addition Example

Programmer's Tasks:



The Programmability Barrier: Vector Addition Example

Programmer's Tasks:

Align
data

```
const unsigned int input_size = 1073741824; // Example value
const unsigned int input_size_8bytes =
    ((input_size * sizeof(T)) % 8) != 0
    ? roundup(input_size, 8) : input_size;

const unsigned int input_size_dpu = divceil(input_size, nr_of_dpus);
const unsigned int input_size_dpu_8bytes =
    ((input_size_dpu * sizeof(T)) % 8) != 0
    ? roundup(input_size_dpu, 8) : input_size_dpu;
```

The Programmability Barrier: Vector Addition Example

Programmer's Tasks:

Align
data

Collect
parameters

```
unsigned int kernel = 0;
dpu_arguments_t input_arguments[NR_DPUS];
for(i=0; i<nr_of_dpus-1; i++) {
    input_arguments[i].size = input_size_dpu_8bytes * sizeof(T);
    input_arguments[i].transfer_size = input_size_dpu_8bytes * sizeof(T);
    input_arguments[i].kernel = kernel;
}
input_arguments[nr_of_dpus-1].size =
    (input_size_8bytes - input_size_dpu_8bytes * (NR_DPUS-1)) * sizeof(T);
input_arguments[nr_of_dpus-1].transfer_size =
    input_size_dpu_8bytes * sizeof(T);
input_arguments[nr_of_dpus-1].kernel = kernel;
```

The Programmability Barrier: Vector Addition Example

Programmer's Tasks:

Align
data

Collect
parameters

Distribute
parameters

```
DPU_ASSERT(dpu_push_xfer(dpu_set, DPU_XFER_TO_DPU, "DPU_INPUT_ARGUMENTS",
                        0, sizeof(input_arguments[0]), DPU_XFER_DEFAULT));
DPU_FOREACH(dpu_set, dpu, i) {
    DPU_ASSERT(dpu_prepare_xfer(dpu, bufferA + input_size_dpu_8bytes * i));
}
DPU_ASSERT(dpu_push_xfer(dpu_set, DPU_XFER_TO_DPU, DPU_MRAM_HEAP_POINTER_NAME,
                        0, input_size_dpu_8bytes * sizeof(T), DPU_XFER_DEFAULT));
DPU_FOREACH(dpu_set, dpu, i) {
    DPU_ASSERT(dpu_prepare_xfer(dpu, bufferB + input_size_dpu_8bytes * i));
}
DPU_ASSERT(dpu_push_xfer(dpu_set, DPU_XFER_TO_DPU, DPU_MRAM_HEAP_POINTER_NAME,
                        input_size_dpu_8bytes * sizeof(T),
                        input_size_dpu_8bytes * sizeof(T), DPU_XFER_DEFAULT));
```


The Programmability Barrier: Vector Addition Example

Programmer's Tasks:

Align
data

Collect
parameters

Distribute
parameters

Launch
computation

```
DPU_ASSERT(dpu_launch(dpu_set, DPU_SYNCHRONOUS));
```

The Programmability Barrier: Vector Addition Example

Programmer's Tasks:

Align
data

Collect
parameters

Distribute
parameters

Launch
computation

Collect
results

```
i = 0;
DPU_FOREACH(dpu_set, dpu, i) {
    DPU_ASSERT(dpu_prepare_xfer(dpu,
        bufferC + input_size_dpu_8bytes * i));
}
DPU_ASSERT(dpu_push_xfer(dpu_set, DPU_XFER_FROM_DPU,
    DPU_MRAM_HEAP_POINTER_NAME,
    input_size_dpu_8bytes * sizeof(T),
    input_size_dpu_8bytes * sizeof(T),
    DPU_XFER_DEFAULT));
```

The Programmability Barrier: Vector Addition Example

Programmer's Tasks:

Align data	Collect parameters	Distribute parameters	Launch computation	Collect results	Manage scratchpad
---------------	-----------------------	--------------------------	-----------------------	--------------------	----------------------

```
barrier_wait(&my_barrier);
uint32_t input_size_dpu_bytes = DPU_INPUT_ARGUMENTS.size;
uint32_t input_size_dpu_bytes_transfer =
    DPU_INPUT_ARGUMENTS.transfer_size;
uint32_t base_tasklet = tasklet_id << BLOCK_SIZE_LOG2;
uint32_t mram_base_addr_A = (uint32_t)DPU_MRAM_HEAP_POINTER;
uint32_t mram_base_addr_B = (uint32_t)(DPU_MRAM_HEAP_POINTER
    + input_size_dpu_bytes_transfer);
T *cache_A = (T *) mem_alloc(BLOCK_SIZE);
T *cache_B = (T *) mem_alloc(BLOCK_SIZE);
```

The Programmability Barrier: Vector Addition Example

Programmer's Tasks:

Align
data

Collect
parameters

Distribute
parameters

Launch
computation

Collect
results

Manage
scratchpad

Orchestrate
computation

```
for (int byte_index = base_tasklet; byte_index < input_size_dpu_bytes;
     byte_index += BLOCK_SIZE * NR_TASKLETS){
uint32_t l_size_bytes = (byte_index + BLOCK_SIZE >=
                         input_size_dpu_bytes)
                         ? (input_size_dpu_bytes - byte_index) : BLOCK_SIZE;
mram_read((__mram_ptr void const*)(mram_base_addr_A + byte_index),
           cache_A, l_size_bytes);
mram_read((__mram_ptr void const*)(mram_base_addr_B + byte_index),
           cache_B, l_size_bytes);
vector_addition(cache_B, cache_A, l_size_bytes >> DIV);
mram_write(cache_B, (__mram_ptr void*)(mram_base_addr_B + byte_index)
            l_size_bytes);
}
```

The Programmability Barrier: Vector Addition Example

Programmer's Tasks:

Align
data

Collect
parameters

Distribute
parameters

Launch
computation

Collect
results

Manage
scratchpad

Orchestrate
computation

Goal:

Just write
my kernel

```
static void vector_addition(T *bufferB, T *bufferA, int l_size){  
    for (unsigned int i = 0; i < l_size; i++){  
        bufferB[i] += bufferA[i];  
    }  
}
```

The Programmability Barrier: Summary

Programmer's Tasks:

Align
data

Collect
parameters

Distribute
parameters

Launch
computation

Collect
results

Manage
scratchpad

Orchestrate
computation

Goal:

Just write
my kernel

Problem

Programming the UPMEM system
leads to non-trivial effort → requires
knowledge of the underlying hardware and
manual fine-grained data movement handling

Our Goal

Goal

To ease programmability for the UPMEM system,
allowing a programmer to write
efficient PIM-friendly code
without the need to
explicitly manage hardware resources

Outline

- 1 Introduction
- 2 The Programmability Barrier
- 3 SimplePIM Overview**
 - Management, Communication & Processing Interfaces
 - Evaluation Results
- 4 DaPPA Overview
 - DaPPA Main Components
 - Evaluation Results
- 5 Conclusion

SimplePIM:

A Software Framework for Productive and Efficient Processing in Memory

- Jinfan Chen, Juan Gómez-Luna, Izzat El Hajj, Yuxin Guo, and Onur Mutlu, ["SimplePIM: A Software Framework for Productive and Efficient Processing in Memory"](#)
*Proceedings of the [32nd International Conference on Parallel Architectures and Compilation Techniques \(PACT\)](#),
Vienna, Austria, October 2023.
[\[Slides \(pptx\) \(pdf\)\]](#)
[\[SimplePIM Source Code\]](#)*

SimplePIM: A Software Framework for Productive and Efficient Processing-in-Memory

Jinfan Chen¹ Juan Gómez-Luna¹ Izzat El Hajj² Yuxin Guo¹ Onur Mutlu¹
¹ETH Zürich ²American University of Beirut

SimplePIM Programming Framework: Overview

SimplePIM provides **standard abstractions** to **build** and **deploy** applications on PIM systems

1 **Management interface**
→ Metadata for PIM-resident arrays

2 **Communication interface**
→ Abstractions for host-PIM and PIM-PIM communication

3 **Processing interface**
→ Iterators (`map`, `reduce`, `zip`) to implement workloads

SimplePIM Programming Framework: Management Interface

- **Metadata for PIM-resident arrays**

- `array_meta_data_t` describes a PIM-resident array
- `simple_pim_management_t` for managing PIM-resident arrays

- **lookup:** Retrieves all relevant information of an array

```
array_meta_data_t* simple_pim_array_lookup(const char* id,  
simple_pim_management_t* management);
```

- **register:** Registers the metadata of an array

```
void simple_pim_array_register(array_meta_data_t* meta_data,  
simple_pim_management_t* management);
```

- **free:** Removes the metadata of an array

```
void simple_pim_array_free(const char* id, simple_pim_management_t* management);
```

SimplePIM Programming Framework: Communication Interface (I)

- **SimplePIM Host-to-CPU Broadcast**

- Transfers a host array to all PIM cores in the system

```
void simple_pim_array_broadcast(char* const id, void* arr, uint64_t len,  
uint32_t type_size, simple_pim_management_t* management);
```



SimplePIM Programming Framework: Communication Interface (II)

- **Host-to-PIM SimplePIM Scatter**

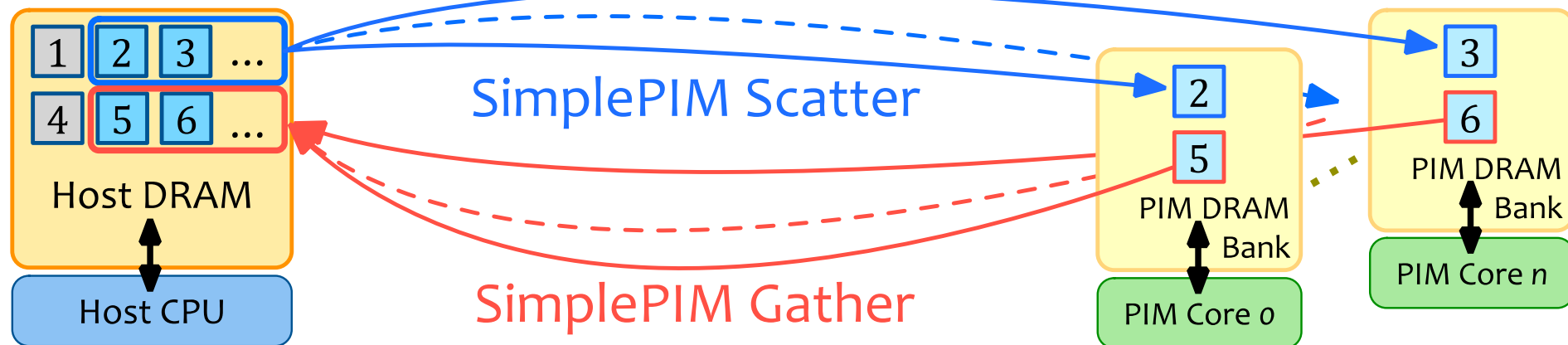
- Distributes an array to PIM DRAM banks

```
void simple_pim_array_scatter(char* const id, void* arr, uint64_t len,  
uint32_t type_size, simple_pim_management_t* management);
```

- **Host-to-PIM SimplePIM Gather**

- Collects portions of an array from PIM DRAM banks

```
void* simple_pim_array_gather(char* const id, simple_pim_management_t*  
management);
```

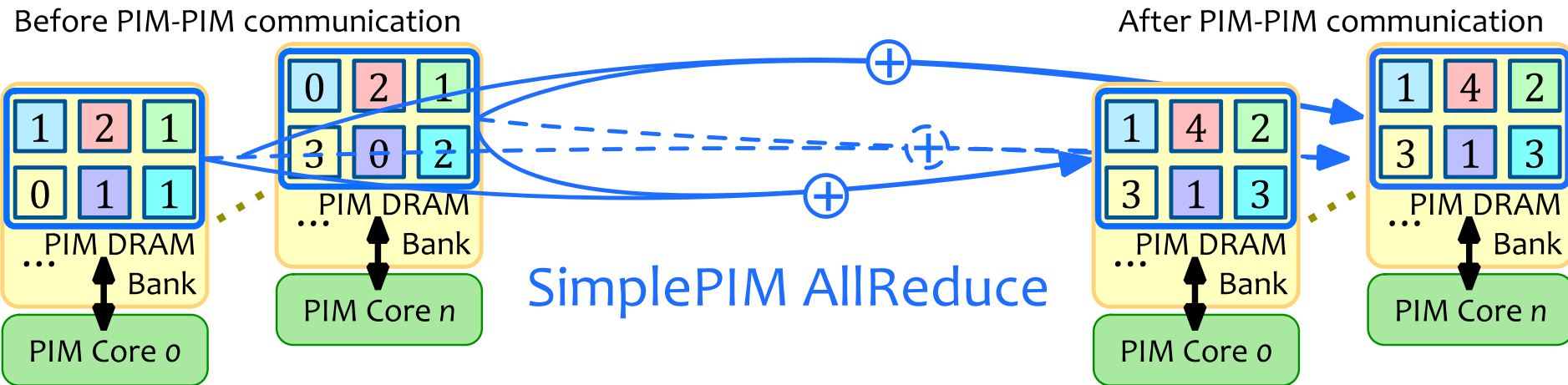


SimplePIM Programming Framework: Communication Interface (III)

• PIM-to-PIM Communication: AllReduce

- Used for algorithm synchronization
- The programmer specifies an accumulative function

```
void simple_pim_array_allreduce(char* const id, handle_t* handle,  
simple_pim_management_t* management);
```



SimplePIM Programming Framework: Communication Interface (IV)

• PIM-to-PIM Communication: AllGather

- Combines array pieces and distributes the complete array to all PIM cores

```
void simple_pim_array_allgather(char* const id, char* new_id,  
simple_pim_management_t* management);
```

Before PIM-PIM communication

After PIM-PIM communication

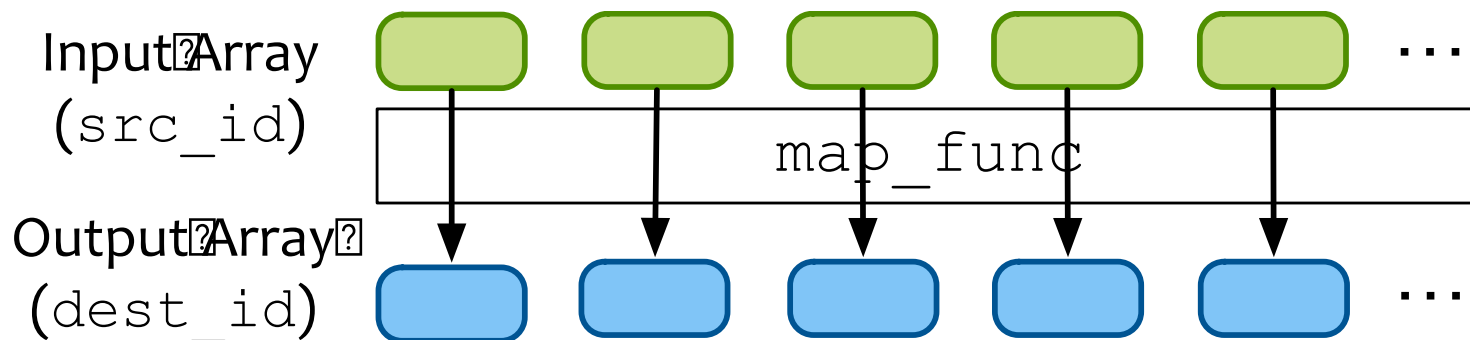


SimplePIM Programming Framework: Processing Interface (I)

- **Array Map**

- Applies `map_func` to every element of the data array

```
void simple_pim_array_map(const char* src_id, const char* dest_id,  
uint32_t output_type, handle_t* handle, simple_pim_management_t* management);
```

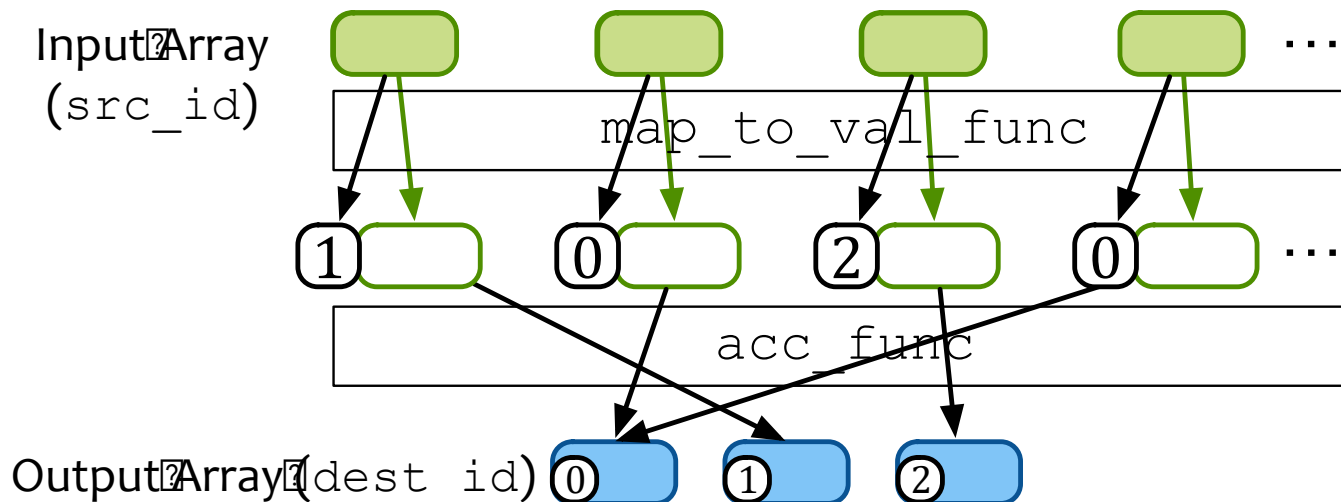


SimplePIM Programming Framework: Processing Interface (II)

• Array Reduction

- The `map_to_val_func` function transforms an input element to an output value and an output index
- The `acc_func` function accumulates the output values onto the output array

```
void simple_pim_array_red(const char* src_id, const char* dest_id,  
uint32_t output_type, uint32_t output_len, handle_t* handle,  
simple_pim_management_t* management);
```

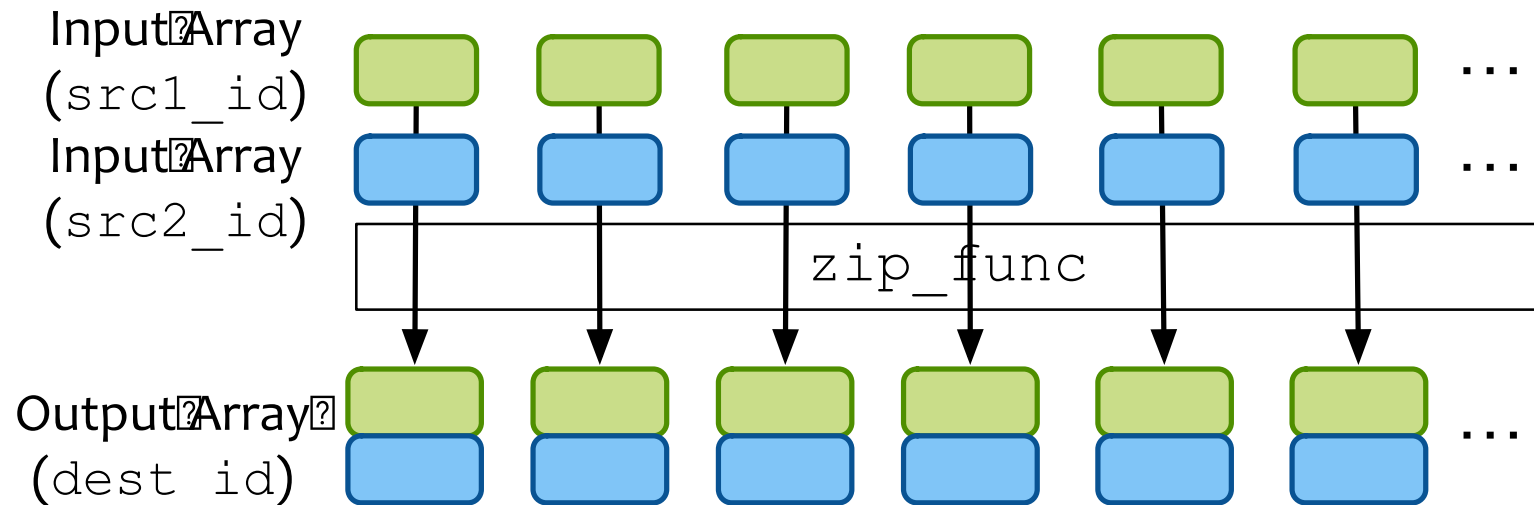


SimplePIM Programming Framework: Processing Interface (III)

- **Array Zip**

- Takes two input arrays and combines their elements into an output array

```
void simple_pim_array_zip(const char* src1_id, const char* src2_id,  
const char* dest_id, simple_pim_management_t* management);
```



SimplePIM Programming Framework: General Code Optimizations

- Strength reduction
- Loop unrolling
- Avoiding boundary checks
- Function inlining
- Adjustment of data transfer sizes

Evaluation Results:

Evaluation Methodology

- **Evaluated system**
 - UPMEM PIM system with 2,432 PIM cores with 159 GB of PIM DRAM
- **Real-world Benchmarks**
 - Vector addition
 - Reduction
 - Histogram
 - K-Means
 - Linear regression
 - Logistic regression
- Comparison to hand-optimized codes in terms of programming productivity and performance

Evaluation Results: Productive Improvement (I)

- Example: Hand-optimized histogram with UPMEM SDK

```
... // Initialize global variables and functions for histogram
int main_kernel() {
    if (tasklet_id == 0)
        mem_reset(); // Reset the heap
    ... // Initialize variables and the histogram
    T *input_buff_A = (T*)mem_alloc(2048); // Allocate buffer in scratchpad memory

    for (unsigned int byte_index = base_tasklet; byte_index < input_size; byte_index += stride) {
        // Boundary checking
        uint32_t l_size_bytes = (byte_index + 2048 >= input_size) ? (input_size - byte_index) : 2048;
        // Load scratchpad with a DRAM block
        mram_read((const __mram_ptr void*)(mram_base_addr_A + byte_index), input_buff_A, l_size_bytes);
        // Histogram calculation
        histogram(hist, bins, input_buff_A, l_size_bytes/sizeof(uint32_t));
    }
    ...
    barrier_wait(&my_barrier); // Barrier to synchronize PIM threads
    ... // Merging histograms from different tasklets into one histo_dpu
    // Write result from scratchpad to DRAM
    if (tasklet_id == 0)
        if (bins * sizeof(uint32_t) <= 2048)
            mram_write(histo_dpu, (__mram_ptr void*)mram_base_addr_histo, bins * sizeof(uint32_t));
        else
            for (unsigned int offset = 0; offset < ((bins * sizeof(uint32_t)) >> 11); offset++) {
                mram_write(histo_dpu + (offset << 9), (__mram_ptr void*)(mram_base_addr_histo +
                    (offset << 11)), 2048);
            }
    return 0;
}
```

Evaluation Results: Productive Improvement (II)

- Example: SimplePIM histogram

```
// Programmer-defined functions in the file "histo_filepath"
void init_func (uint32_t size, void* ptr) {
    char* casted_value_ptr = (char*) ptr;
    for (int i = 0; i < size; i++)
        casted_value_ptr[i] = 0;
}

void acc_func (void* dest, void* src) {
    *(uint32_t*)dest += *(uint32_t*)src;
}

void map_to_val_func (void* input, void* output, uint32_t* key) {
    uint32_t d = *(uint32_t*)input;
    *(uint32_t*)output = 1;
    *key = d * bins >> 12;
}

// Host side handle creation and iterator call
handle_t* handle = simple_pim_create_handle("histo_filepath", REDUCE, NULL, 0);

// Transfer (scatter) data to PIM, register as "t1"
simple_pim_array_scatter("t1", src, bins, sizeof(T), management);

// Run histogram on "t1" and produce "t2"
simple_pim_array_red("t1", "t2", sizeof(T), bins, handle, management);
```

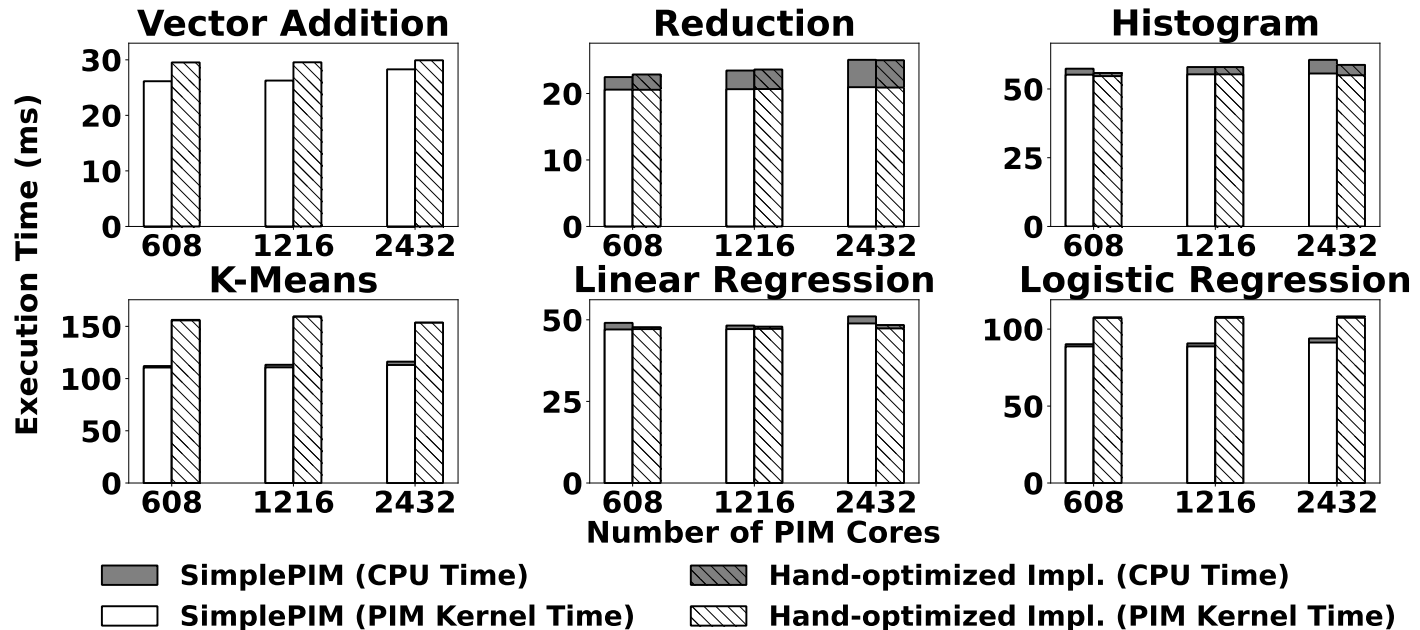
Evaluation Results: Productive Improvement (III)

- Lines of code (LoC) reduction

	SimplePIM	Hand-optimized	LoC Reduction
Reduction	14	83	5.93×
Vector Addition	14	82	5.86×
Histogram	21	114	5.43×
Linear Regression	48	157	3.27×
Logistic Regression	59	176	2.98×
K-Means	68	206	3.03×

SimplePIM **reduces** the number of lines of effective code by a factor of **2.98×** to **5.93×**

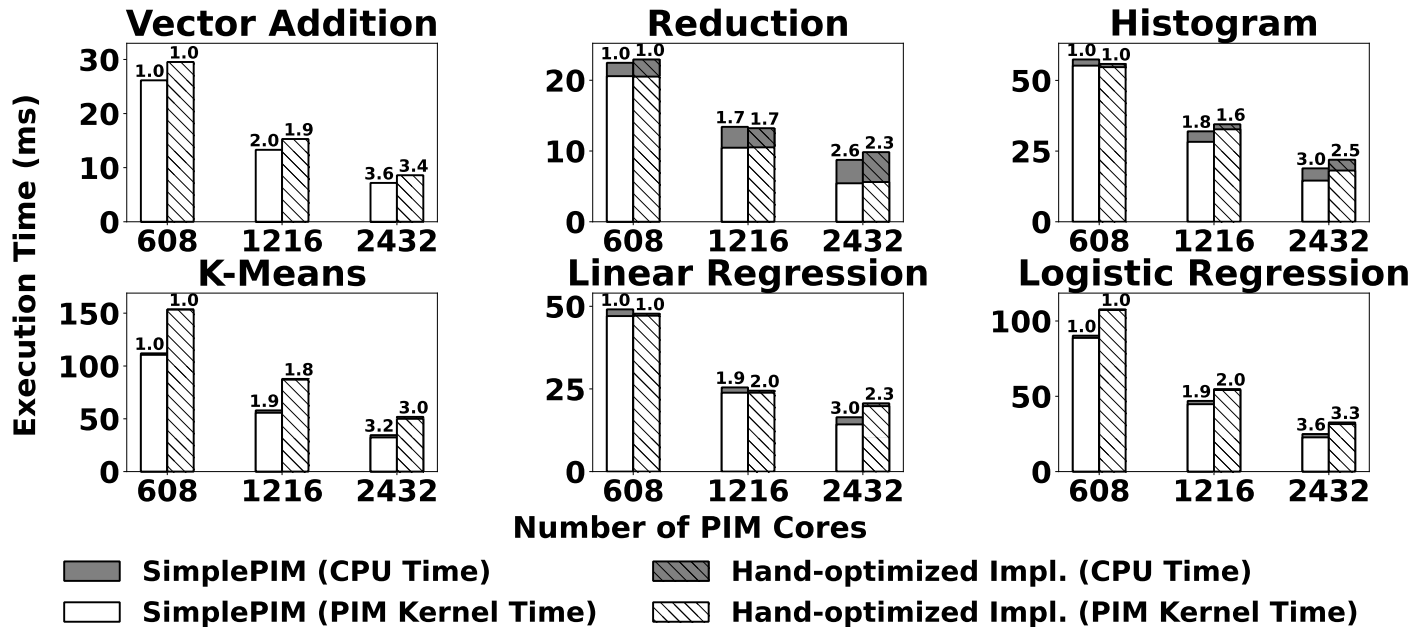
Evaluation Results: Weak Scaling Analysis



SimplePIM achieves **comparable performance** for reduction, histogram, and linear regression

SimplePIM **outperforms hand-optimized implementations** for vector addition, logistic regression, and k-means by **10%-37%**

Evaluation Results: Strong Scaling Analysis



SimplePIM scales better than hand-optimized implementations for reduction, histogram, and linear regression

SimplePIM outperforms hand-optimized implementations for vector addition, logistic regression, and k-means by 15%-43%

Source Code

- <https://github.com/CMU-SAFARI/SimplePIM>

CMU-SAFARI / SimplePIM Public

Notifications F

<> Code Issues Pull requests Actions Projects Security Insights

main Branches Tags Code

13 Commits

- benchmarks
- lib
- .gitignore
- LICENSE
- README.md

README MIT license

SimplePIM: A Software Framework for Productive and Efficient In-Memory Processing

This project implements SimplePIM, a software framework for easy and efficient in-memory-hardware programming. The code is implemented on UPMEM, an actual, commercially available PIM hardware that combines traditional DRAM memory with general-purpose in-order cores inside the same chip. SimplePIM processes arrays of arbitrary elements on a PIM device by calling iterator functions from the host and provides primitives for communication among PIM cores and between PIM and the host system.

We implement six applications with SimplePIM on UPMEM:

About

SimplePIM is the first high-level programming framework for real-world processing-in-memory (PIM) architectures. Described in the PACT 2023 paper by Chen et al. (<https://arxiv.org/pdf/2310.01893.pdf>).

- Readme
- MIT license
- Activity
- Custom properties
- 18 stars
- 6 watching
- 4 forks

Report repository

Releases

No releases published

Packages

No packages published

Contributors 3

SimplePIM:

A Software Framework for Productive and Efficient Processing in Memory

- Jinfan Chen, Juan Gómez-Luna, Izzat El Hajj, Yuxin Guo, and Onur Mutlu, ["SimplePIM: A Software Framework for Productive and Efficient Processing in Memory"](#)
*Proceedings of the [32nd International Conference on Parallel Architectures and Compilation Techniques \(PACT\)](#),
Vienna, Austria, October 2023.
[\[Slides \(pptx\) \(pdf\)\]](#)
[\[SimplePIM Source Code\]](#)*

SimplePIM: A Software Framework for Productive and Efficient Processing-in-Memory

Jinfan Chen¹ Juan Gómez-Luna¹ Izzat El Hajj² Yuxin Guo¹ Onur Mutlu¹
¹ETH Zürich ²American University of Beirut

DaPPA:

A Data-Parallel Framework for Processing-in-Memory Architectures

- Geraldo F. Oliveira, Alain Kohli, David Novo, Juan Gómez-Luna, Onur Mutlu

["DaPPA: A Data-Parallel Framework for Processing-in-Memory Architectures,"](#)

[arXiv:2310.10168 \[cs.AR\]](#)

2nd Place ACM Student Research Competition at [the 32nd International Conference on Parallel Architectures and Compilation Techniques \(PACT\)](#), Vienna, Austria, October 2023.

DaPPA: A Data-Parallel Framework for Processing-in-Memory Architectures

Geraldo F. Oliveira*

Alain Kohli*

David Novo[‡]

Juan Gómez-Luna*

Onur Mutlu*

**ETH Zürich*

[‡]*LIRMM, Univ. Montpellier, CNRS*

1. Motivation & Problem

The increasing prevalence and growing size of data in modern applications have led to high costs for computation in traditional *processor-centric computing* systems. To mitigate these costs, the *processing-in-memory* (PIM) [1–6] paradigm moves computation closer to where the data resides, reducing the need to move data between memory and the processor. Even though the concept of PIM has been first proposed in the 1960s [7, 8], real-world PIM systems have only recently

face [15, 16] that abstracts the hardware components of the UPMEM system. Using this key idea, DaPPA transforms a data-parallel pattern-based application code into the appropriate UPMEM-target code, including the required APIs for data management and code partition, which can then be compiled into a UPMEM-based binary *transparently* from the programmer. While generating UPMEM-target code, DaPPA implements several code optimizations to improve end-to-end performance.

DaPPA:

Key Idea & Overview

Key Idea

Leverage an intuitive
data-parallel pattern-based interface for
PIM programming



DaPPA , a Data-Parallel PIM Architecture that automatically distributes input and gathers output data, handles memory management, and parallelizes work across PIM cores

DaPPA is composed of three main components:

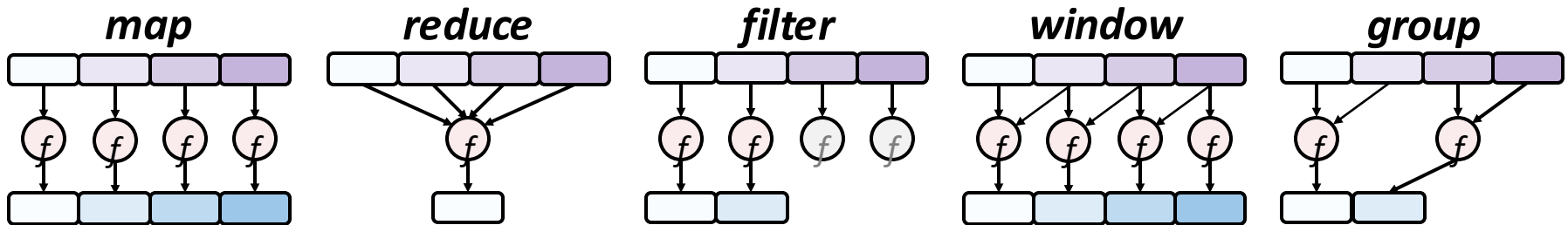
- 1** Data-Parallel Pattern APIs
- 2** Dataflow Programming Interface
- 3** Dynamic Template-Based Compilation

DaPPA:

Data-Parallel Pattern APIs

Pre-defined functions that implement high-level
data-parallel pattern primitives

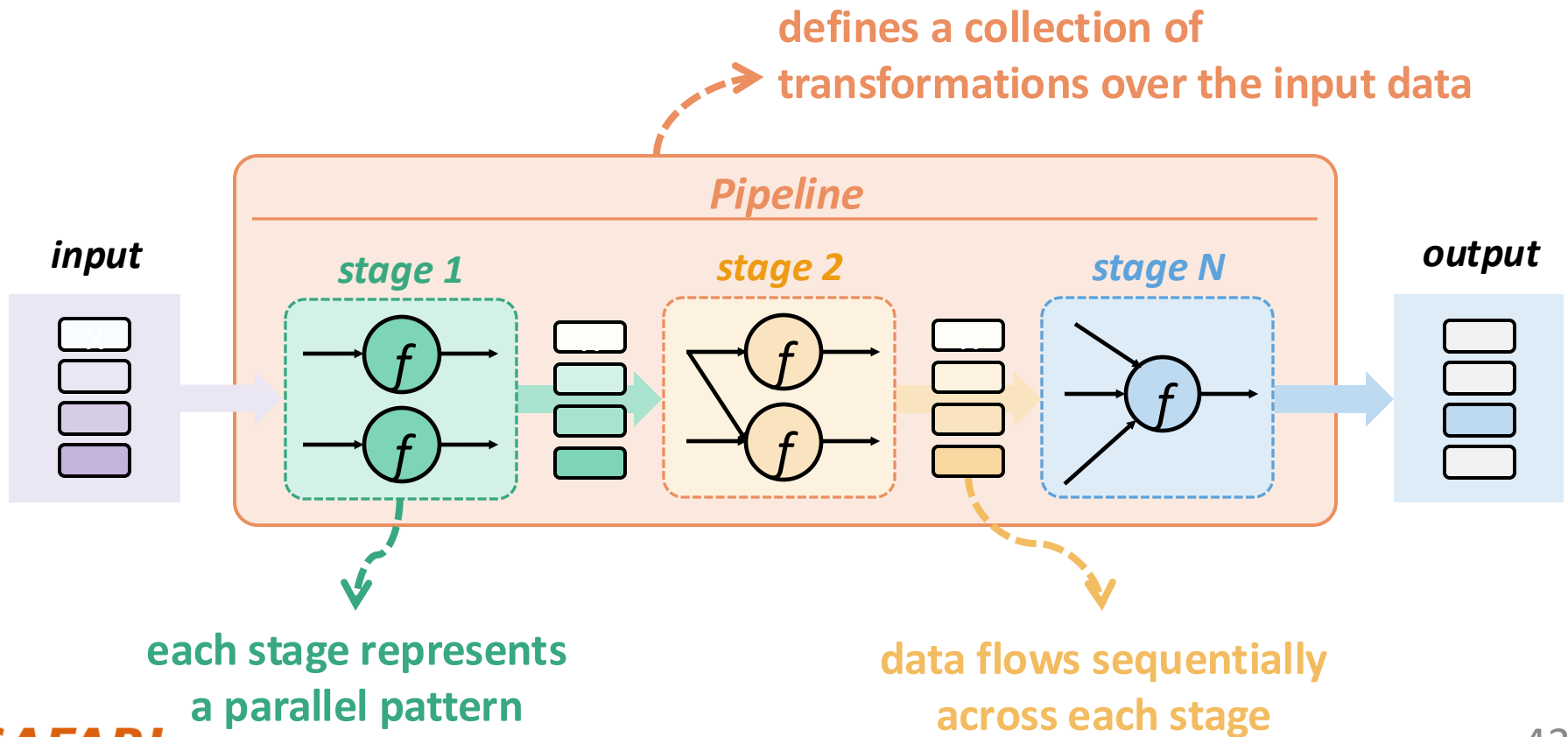
- Skeleton and pattern-based parallel programming are a **common abstraction for parallel architectures**
 - M. Cole, “*Bringing Skeletons Out of the Closet: A Pragmatic Manifesto for Skeletal Parallel Programming*,” *Parallel Computing*, 2004
- DaPPA supports **five primary** data-parallel patterns



The user can combine all five data-parallel primitives
to describe complex data transformations

DaPPA: Dataflow Programming Interface

DaPPA exposes to the user a
dataflow-based programming interface



DaPPA:

Dynamic Template-Based Compilation

DaPPA uses a dynamic template-based compilation to generate PIM code **in two main steps**

1 Templating: DaPPA creates a base UPMEM code based on a *basic skeleton* of a UPMEM application

- We use the Inja C++ templating engine

2 Optimizations: DaPPA uses a series of transformations to

- *extract* data required by the UPMEM code template
- calculate the *memory offsets* for MRAMs and WRAMs
- *divide computation* between CPU and PIM cores

DaPPA compiles and executes each stage in a Pipeline per time → allows for runtime optimizations

DaPPA: Putting All Together

Example of DaPPA's implementation of a
vector dot product operation

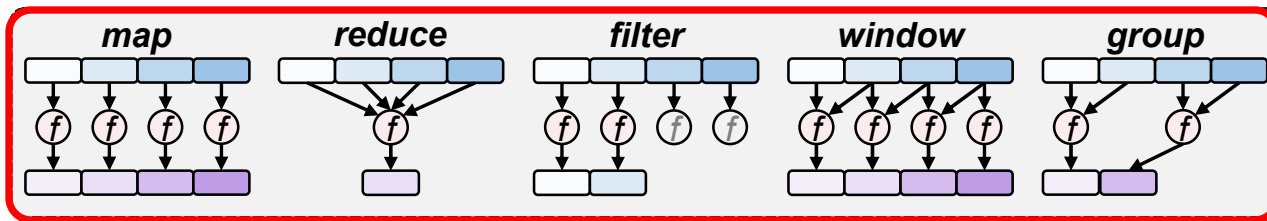
reduce
 $C = A_0B_0 +$
map $A_1B_1 +$
 $A_2B_2 +$
 A_3B_3

*target
computation*

DaPPA: Putting All Together

Example of DaPPA's implementation of a
vector dot product operation

① *data-parallel pattern APIs*



reduce

$$C = A_0B_0 +$$

map

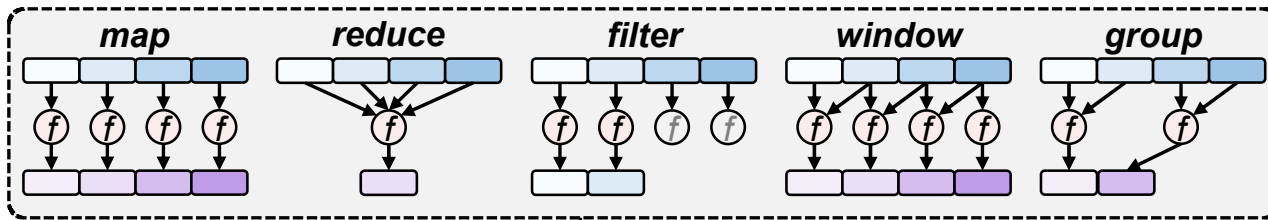
$$A_1B_1 +$$
$$A_2B_2 +$$
$$A_3B_3$$

*target
computation*

DaPPA: Putting All Together

Example of DaPPA's implementation of a
vector dot product operation

① data-parallel pattern APIs



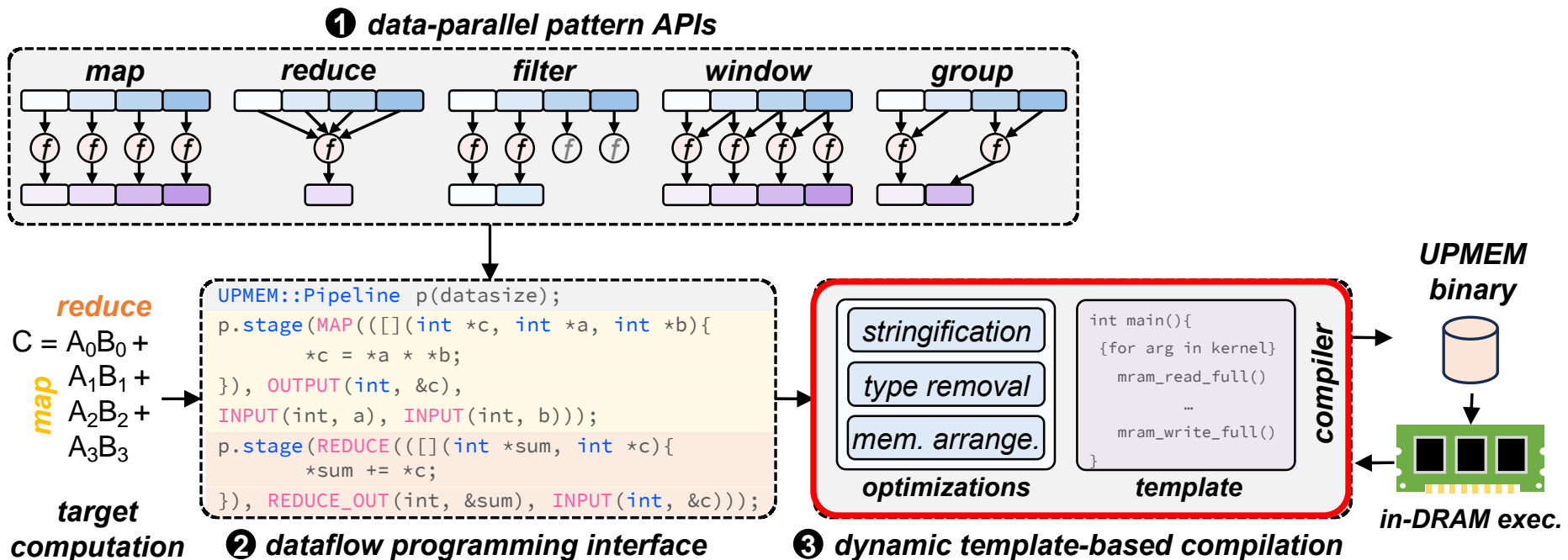
reduce
 $C = A_0B_0 +$
map $A_1B_1 +$
 $A_2B_2 +$
 A_3B_3
target
computation

```
UPMEM::Pipeline p(datasize);
p.stage(MAP(([](int *c, int *a, int *b){
    *c = *a * *b;
}), OUTPUT(int, &c),
INPUT(int, a), INPUT(int, b)));
p.stage(REDUCE(([](int *sum, int *c){
    *sum += *c;
}), REDUCE_OUT(int, &sum), INPUT(int, &c)));
```

② dataflow programming interface

DaPPA: Putting All Together

Example of DaPPA's implementation of a
vector dot product operation



Evaluation: Methodology Overview

- **Evaluation Setup**

- **Host CPU:** 2-socket Intel® Xeon Silver 4110 CPU
- **PIM Cores:** 20 UPMEM PIM DIMMs (160 GB PIM memory)
- **2560 DPUs** in total

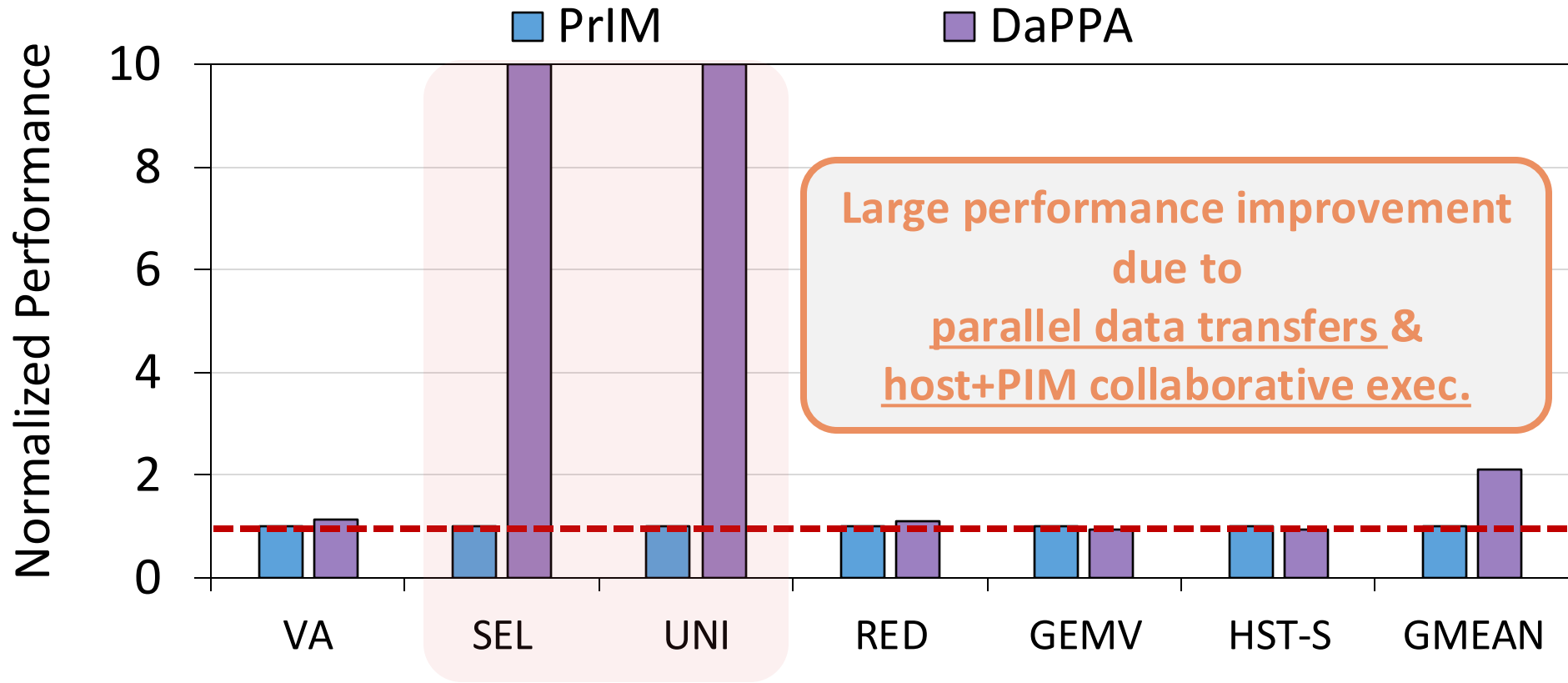
- **Workloads:** 6 workloads from the PRIM benchmark suite

- Vector addition (**VA**); Select (**SEL**); Unique (**UNI**); Reduce (**RED**); General matrix-vector multiply (**GEMV**); Histogram small (**HST-S**)

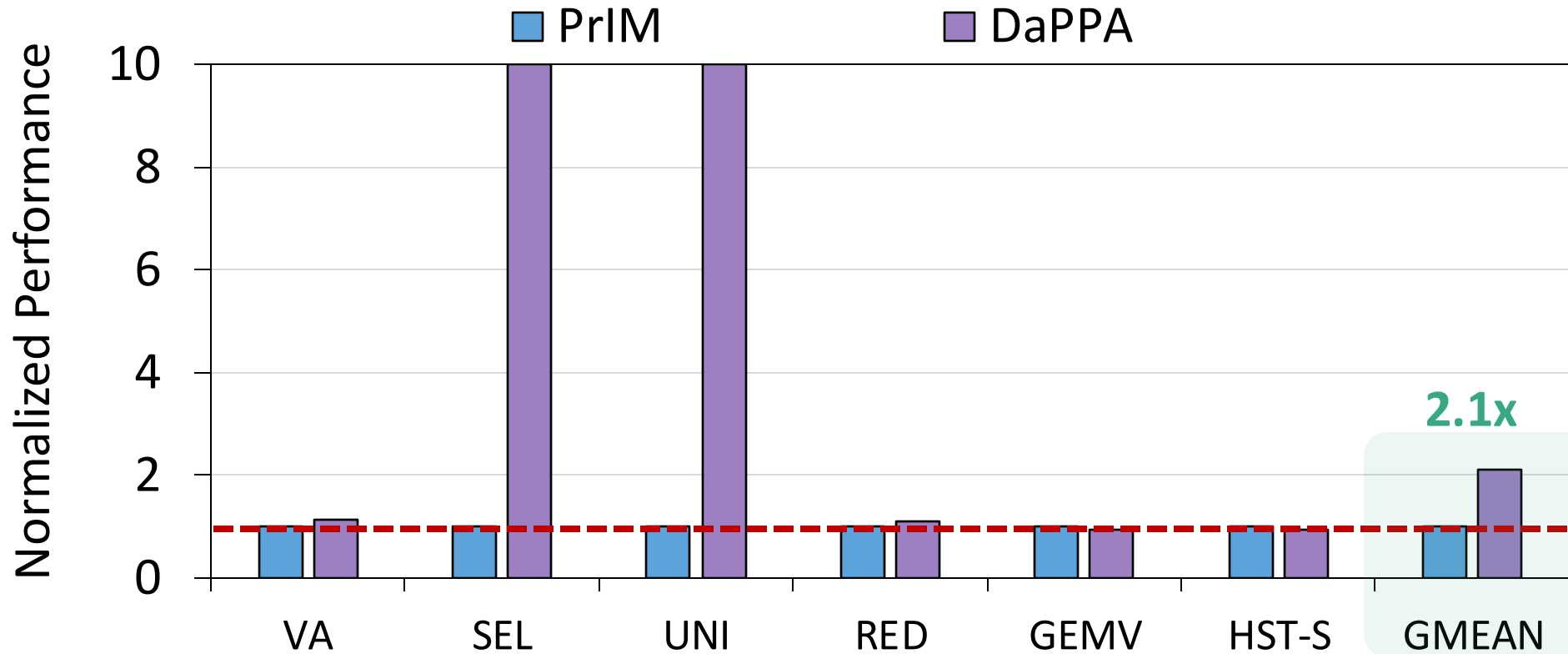
- **Metrics**

- **End-to-end execution time** (average of 10 runs)
- **Programming complexity** (in lines of code)

Evaluation: Performance Analysis

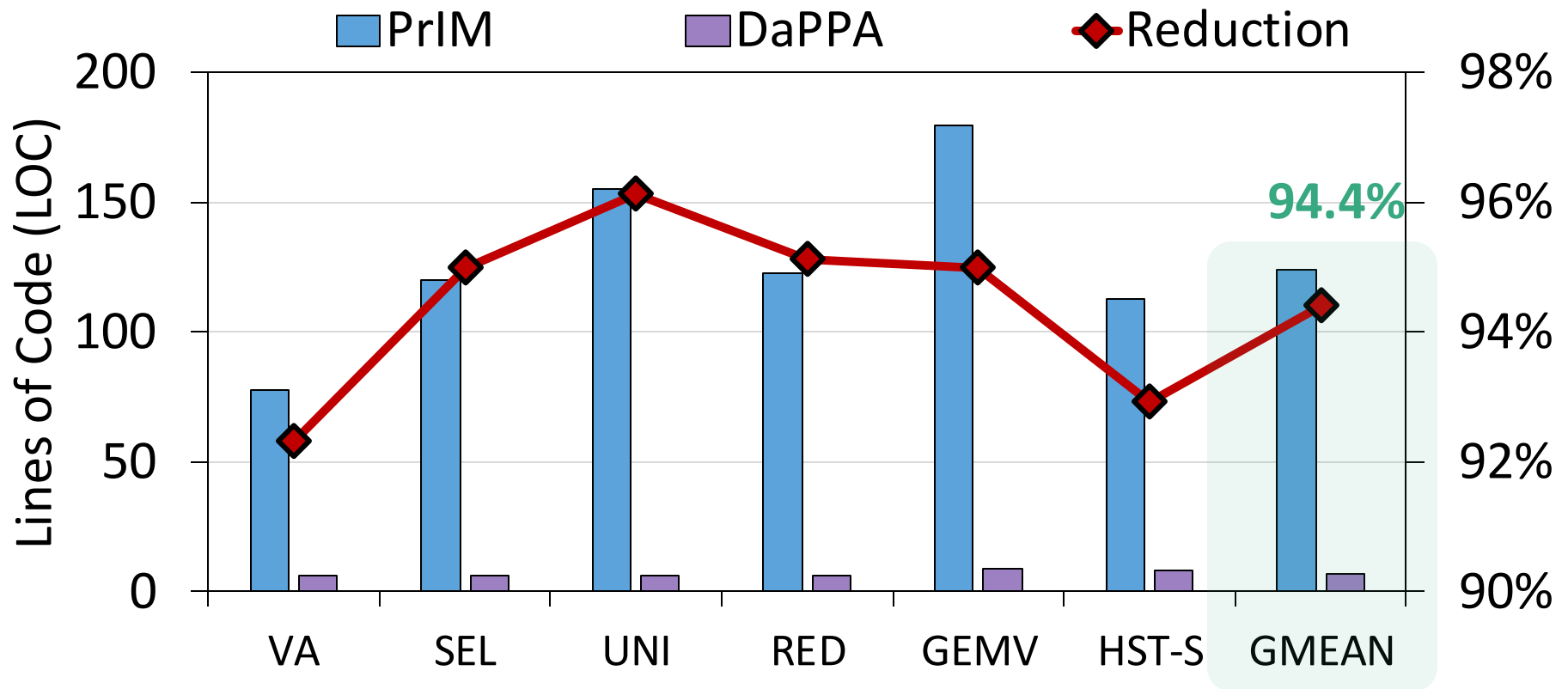


Evaluation: Performance Analysis



DaPPA significantly improves end-to-end performance compared to hand-tuned implementations

Evaluation: Programming Complexity Analysis



DaPPA significantly reduces programming complexity by abstracting hardware components

Evaluation:

Comparison to State-of-the-Art

SimplePIM [Chen+, PACT'23]: a framework that uses (1) iterator functions and (2) primitives for communication to aid PIM programmability

Compared to SimplePIM, DaPPA provides three key benefits

1. **Higher abstraction level** → The programmer does *not* need to manually specify communication patterns used during computation
2. **Support for more parallel patterns** → DaPPA supports two more parallel primitives (window and group), and allows the mixing of parallel patterns
3. **Further execution optimizations** → DaPPA allows using idle host resources for collaborative execution

DaPPA improves state-of-the-art frameworks for PIM programmability

DaPPA:

A Data-Parallel Framework for Processing-in-Memory Architectures

- Geraldo F. Oliveira, Alain Kohli, David Novo, Juan Gómez-Luna, Onur Mutlu

["DaPPA: A Data-Parallel Framework for Processing-in-Memory Architectures,"](#)
[arXiv:2310.10168 \[cs.AR\]](#)

2nd Place ACM Student Research Competition at [the 32nd International Conference on Parallel Architectures and Compilation Techniques \(PACT\)](#),
Vienna, Austria, October 2023.

DaPPA: A Data-Parallel Framework for Processing-in-Memory Architectures

Geraldo F. Oliveira*

Alain Kohli*

David Novo[‡]

Juan Gómez-Luna*

Onur Mutlu*

**ETH Zürich*

[‡]*LIRMM, Univ. Montpellier, CNRS*

1. Motivation & Problem

The increasing prevalence and growing size of data in modern applications have led to high costs for computation in traditional *processor-centric computing* systems. To mitigate these costs, the *processing-in-memory* (PIM) [1–6] paradigm moves computation closer to where the data resides, reducing the need to move data between memory and the processor. Even though the concept of PIM has been first proposed in the 1960s [7, 8], real-world PIM systems have only recently

face [15, 16] that abstracts the hardware components of the UPMEM system. Using this key idea, DaPPA transforms a data-parallel pattern-based application code into the appropriate UPMEM-target code, including the required APIs for data management and code partition, which can then be compiled into a UPMEM-based binary *transparently* from the programmer. While generating UPMEM-target code, DaPPA implements several code optimizations to improve end-to-end performance.

Adoption: How to Maintain Coherence? (I)

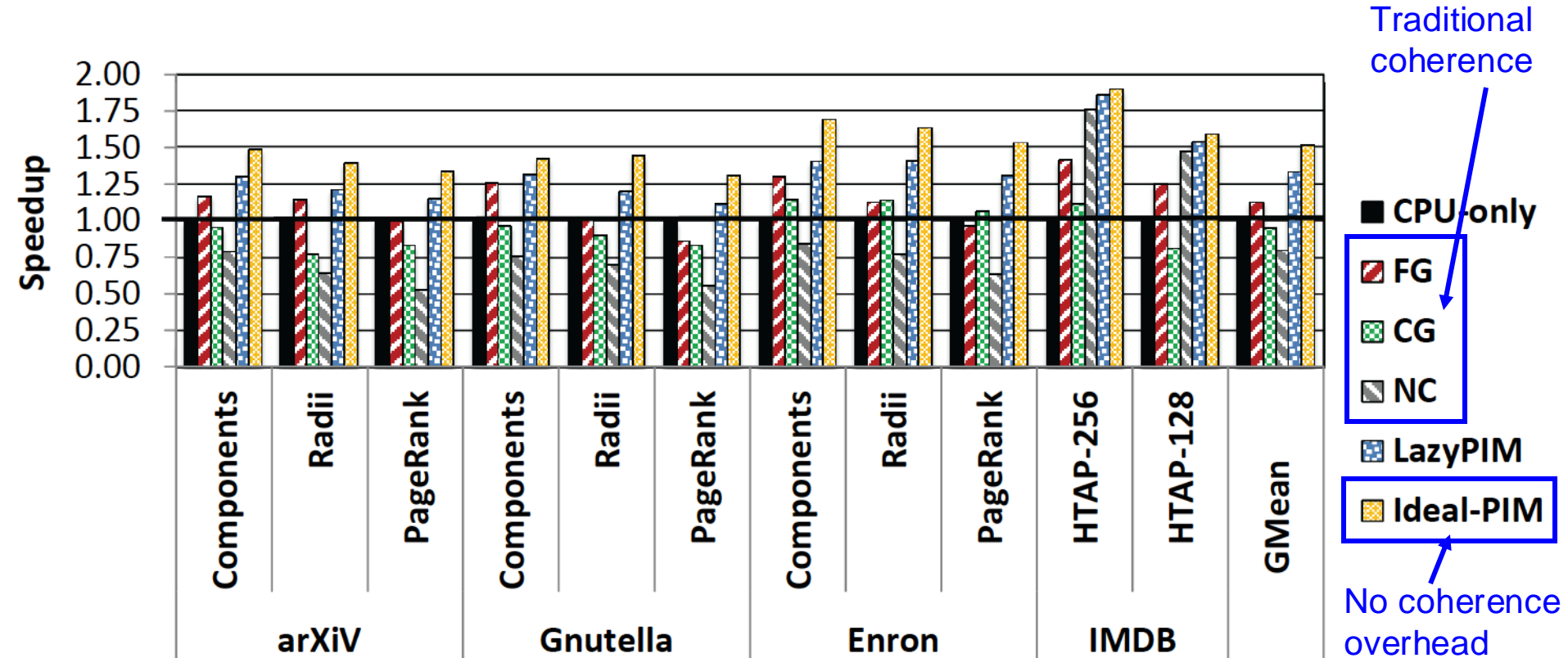
- Amirali Boroumand, Saugata Ghose, Minesh Patel, Hasan Hassan, Brandon Lucia, Kevin Hsieh, Krishna T. Malladi, Hongzhong Zheng, and Onur Mutlu,
"LazyPIM: An Efficient Cache Coherence Mechanism for Processing-in-Memory"
IEEE Computer Architecture Letters (CAL), June 2016.

LazyPIM: An Efficient Cache Coherence Mechanism for Processing-in-Memory

Amirali Boroumand[†], Saugata Ghose[†], Minesh Patel[†], Hasan Hassan^{†§}, Brandon Lucia[†],
Kevin Hsieh[†], Krishna T. Malladi^{*}, Hongzhong Zheng^{*}, and Onur Mutlu^{‡†}

[†] *Carnegie Mellon University* ^{*} *Samsung Semiconductor, Inc.* [§] *TOBB ETÜ* [‡] *ETH Zürich*

Challenge: Coherence for Hybrid CPU-PIM Apps



Adoption: How to Maintain Coherence? (II)

- Amirali Boroumand, Saugata Ghose, Minesh Patel, Hasan Hassan, Brandon Lucia, Kevin Hsieh, Krishna T. Malladi, Hongzhong Zheng, and Onur Mutlu,
"CoNDA: Efficient Cache Coherence Support for Near-Data Accelerators"
Proceedings of the 46th International Symposium on Computer Architecture (ISCA), Phoenix, AZ, USA, June 2019.

CoNDA: Efficient Cache Coherence Support for Near-Data Accelerators

Amirali Boroumand[†]

Saugata Ghose[†]

Minesh Patel^{*}

Hasan Hassan^{*}

Brandon Lucia[†]

Rachata Ausavarungnirun^{†‡}

Kevin Hsieh[†]

Nastaran Hajinazar^{◇†}

Krishna T. Malladi[§]

Hongzhong Zheng[§]

Onur Mutlu^{*†}

[†]Carnegie Mellon University

^{*}ETH Zürich

[‡]KMUTNB

[◇]Simon Fraser University

[§]Samsung Semiconductor, Inc.

Adoption: How to Support Synchronization?

- Christina Giannoula, Nandita Vijaykumar, Nikela Papadopoulou, Vasileios Karakostas, Ivan Fernandez, Juan Gómez-Luna, Lois Orosa, Nectarios Koziris, Georgios Goumas, Onur Mutlu, [**"SynCron: Efficient Synchronization Support for Near-Data-Processing Architectures"**](#)
Proceedings of the 27th International Symposium on High-Performance Computer Architecture (HPCA), Virtual, February-March 2021.
[[Slides \(pptx\)](#)] [[pdf](#)]
[[Short Talk Slides \(pptx\)](#)] [[pdf](#)]
[[Talk Video](#) (21 minutes)]
[[Short Talk Video](#) (7 minutes)]

***SynCron*: Efficient Synchronization Support for Near-Data-Processing Architectures**

Christina Giannoula^{†‡} Nandita Vijaykumar^{*‡} Nikela Papadopoulou[†] Vasileios Karakostas[†] Ivan Fernandez^{§‡}
Juan Gómez-Luna[‡] Lois Orosa[‡] Nectarios Koziris[†] Georgios Goumas[†] Onur Mutlu[‡]
[†]*National Technical University of Athens* [‡]*ETH Zürich* ^{*}*University of Toronto* [§]*University of Malaga*

Adoption: How to Support Virtual Memory?

- Kevin Hsieh, Samira Khan, Nandita Vijaykumar, Kevin K. Chang, Amirali Boroumand, Saugata Ghose, and Onur Mutlu,
["Accelerating Pointer Chasing in 3D-Stacked Memory: Challenges, Mechanisms, Evaluation"](#)
Proceedings of the 34th IEEE International Conference on Computer Design (ICCD), Phoenix, AZ, USA, October 2016.

Accelerating Pointer Chasing in 3D-Stacked Memory: Challenges, Mechanisms, Evaluation

Kevin Hsieh[†] Samira Khan[‡] Nandita Vijaykumar[†]

Kevin K. Chang[†] Amirali Boroumand[†] Saugata Ghose[†] Onur Mutlu^{§†}

[†]*Carnegie Mellon University* [‡]*University of Virginia* [§]*ETH Zürich*

Adoption: Code and Data Mapping

- Kevin Hsieh, Eiman Ebrahimi, Gwangsun Kim, Niladrish Chatterjee, Mike O'Connor, Nandita Vijaykumar, Onur Mutlu, and Stephen W. Keckler, **"Transparent Offloading and Mapping (TOM): Enabling Programmer-Transparent Near-Data Processing in GPU Systems"**

Proceedings of the 43rd International Symposium on Computer Architecture (ISCA), Seoul, South Korea, June 2016.

[[Slides \(pptx\)](#) ([pdf](#))]

[[Lightning Session Slides \(pptx\)](#) ([pdf](#))]

Transparent Offloading and Mapping (TOM): Enabling Programmer-Transparent Near-Data Processing in GPU Systems

Kevin Hsieh[‡] Eiman Ebrahimi[†] Gwangsun Kim^{*} Niladrish Chatterjee[†] Mike O'Connor[†]
Nandita Vijaykumar[‡] Onur Mutlu^{§‡} Stephen W. Keckler[†]

[‡]Carnegie Mellon University [†]NVIDIA ^{*}KAIST [§]ETH Zürich

DAMOV Analysis Methodology & Workloads

DAMOV: A New Methodology and Benchmark Suite for Evaluating Data Movement Bottlenecks

GERALDO F. OLIVEIRA, ETH Zürich, Switzerland

JUAN GÓMEZ-LUNA, ETH Zürich, Switzerland

LOIS OROSA, ETH Zürich, Switzerland

SAUGATA GHOSE, University of Illinois at Urbana-Champaign, USA

NANDITA VIJAYKUMAR, University of Toronto, Canada

IVAN FERNANDEZ, University of Malaga, Spain & ETH Zürich, Switzerland

MOHAMMAD SADROSADATI, Institute for Research in Fundamental Sciences (IPM), Iran & ETH Zürich, Switzerland

ONUR MUTLU, ETH Zürich, Switzerland

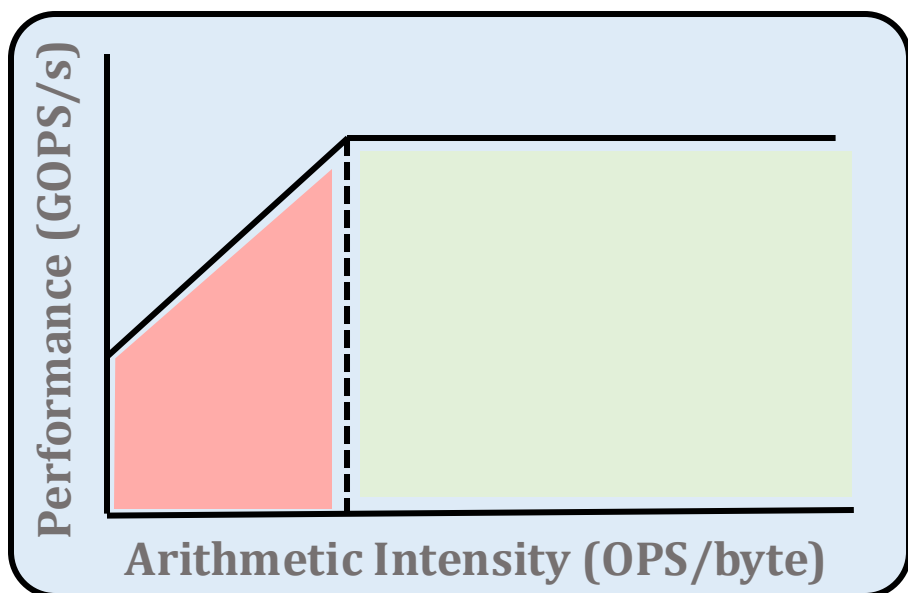
Data movement between the CPU and main memory is a first-order obstacle against improving performance, scalability, and energy efficiency in modern systems. Computer systems employ a range of techniques to reduce overheads tied to data movement, spanning from traditional mechanisms (e.g., deep multi-level cache hierarchies, aggressive hardware prefetchers) to emerging techniques such as Near-Data Processing (NDP), where some computation is moved close to memory. Prior NDP works investigate the root causes of data movement bottlenecks using different profiling methodologies and tools. However, there is still a lack of understanding about the key metrics that can identify different data movement bottlenecks and their relation to traditional and emerging data movement mitigation mechanisms. Our goal is to methodically identify potential sources of data movement over a broad set of applications and to comprehensively compare traditional compute-centric data movement mitigation techniques (e.g., caching and prefetching) to more memory-centric techniques (e.g., NDP), thereby developing a rigorous understanding of the best techniques to mitigate each source of data movement.

With this goal in mind, we perform the first large-scale characterization of a wide variety of applications, across a wide range of application domains, to identify fundamental program properties that lead to data movement to/from main memory. We develop the first systematic methodology to classify applications based on the sources contributing to data movement bottlenecks. From our large-scale characterization of 77K functions across 345 applications, we select 144 functions to form the first open-source benchmark suite (DAMOV) for main memory data movement studies. We select a diverse range of functions that (1) represent different types of data movement bottlenecks, and (2) come from a wide range of application domains. Using NDP as a case study, we identify new insights about the different data movement bottlenecks and use these insights to determine the most suitable data movement mitigation mechanism for a particular application. We open-source DAMOV and the complete source code for our new characterization methodology at <https://github.com/CMU-SAFARI/DAMOV>.

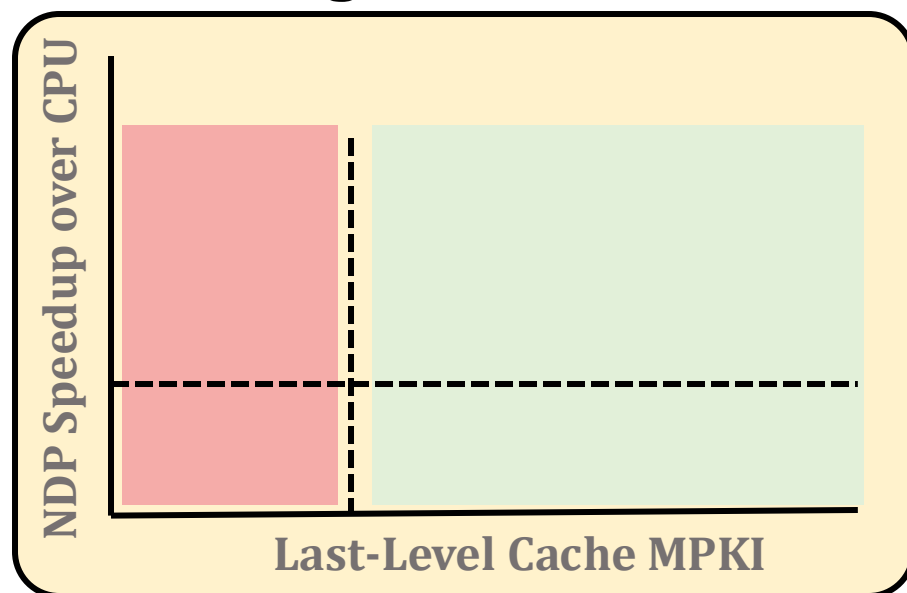
Identifying Memory Bottlenecks

- **Multiple approaches** to **identify** applications that:
 - suffer from data movement bottlenecks
 - take advantage of NDP
- Existing approaches are **not comprehensive enough**

Roofline model

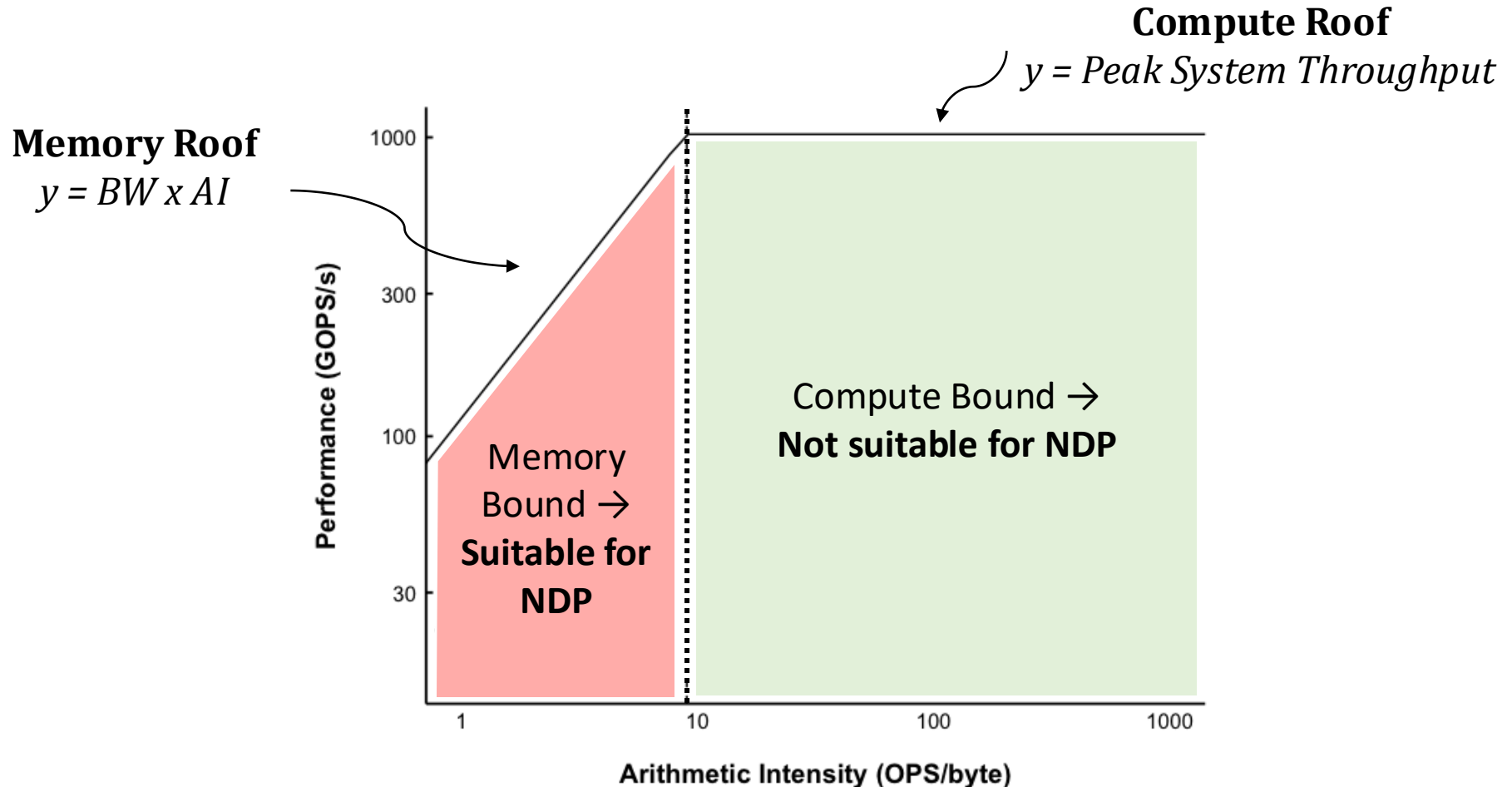


High LLC MPKI



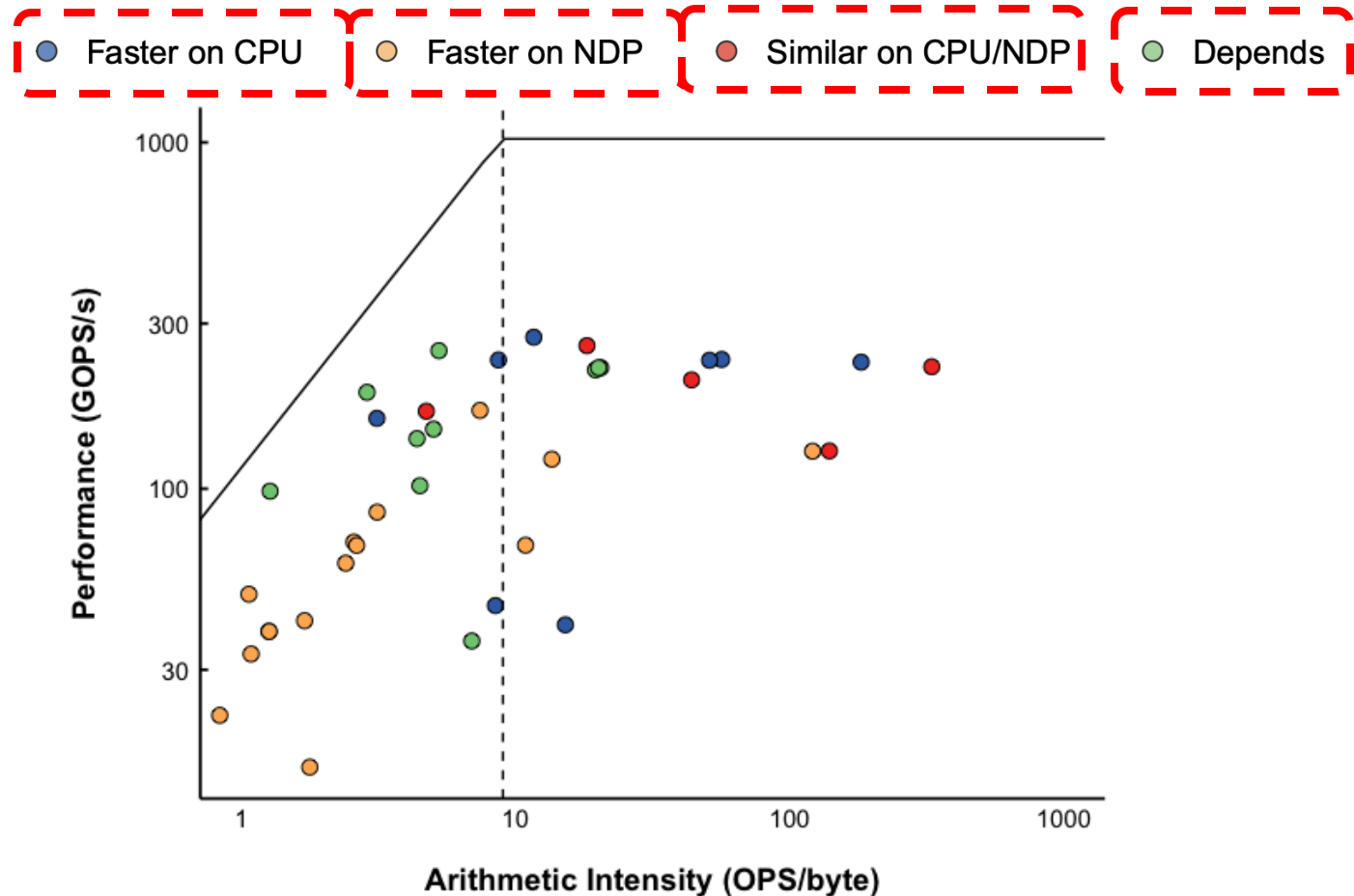
Limitations of Prior Approaches (1/2)

- **Roofline model** → identifies when an application is *bounded* by **compute** or **memory** units



Limitations of Prior Approaches (1/2)

- **Roofline model** → identifies when an application is *bounded* by **compute** or **memory** units

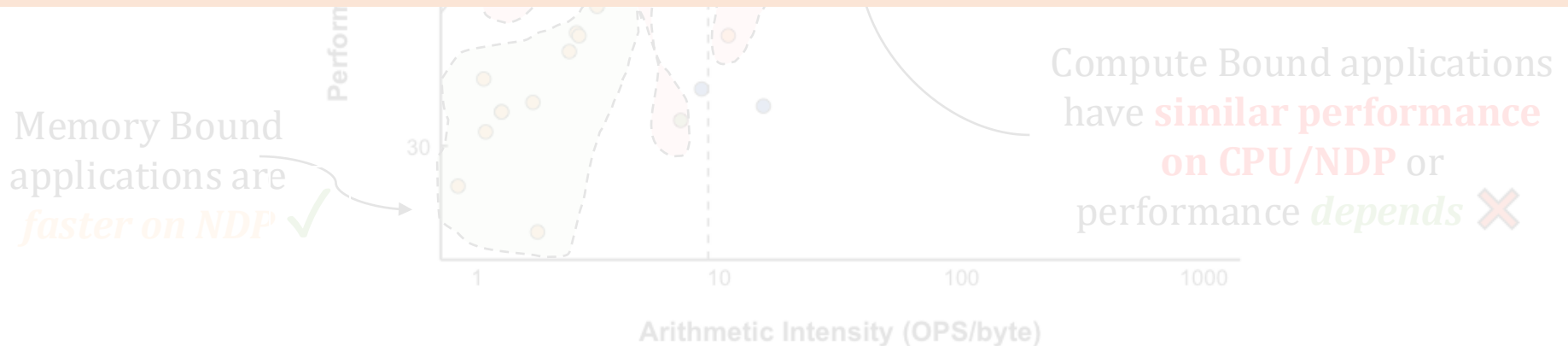


Limitations of Prior Approaches (1/2)

- **Roofline model** → identifies when an application is *bounded* by **compute** or **memory** units

● Faster on CPU ● Faster on NDP ● Similar on CPU/NDP ● Depends

Roofline model **does not accurately account** for the **NDP suitability** of memory-bound applications



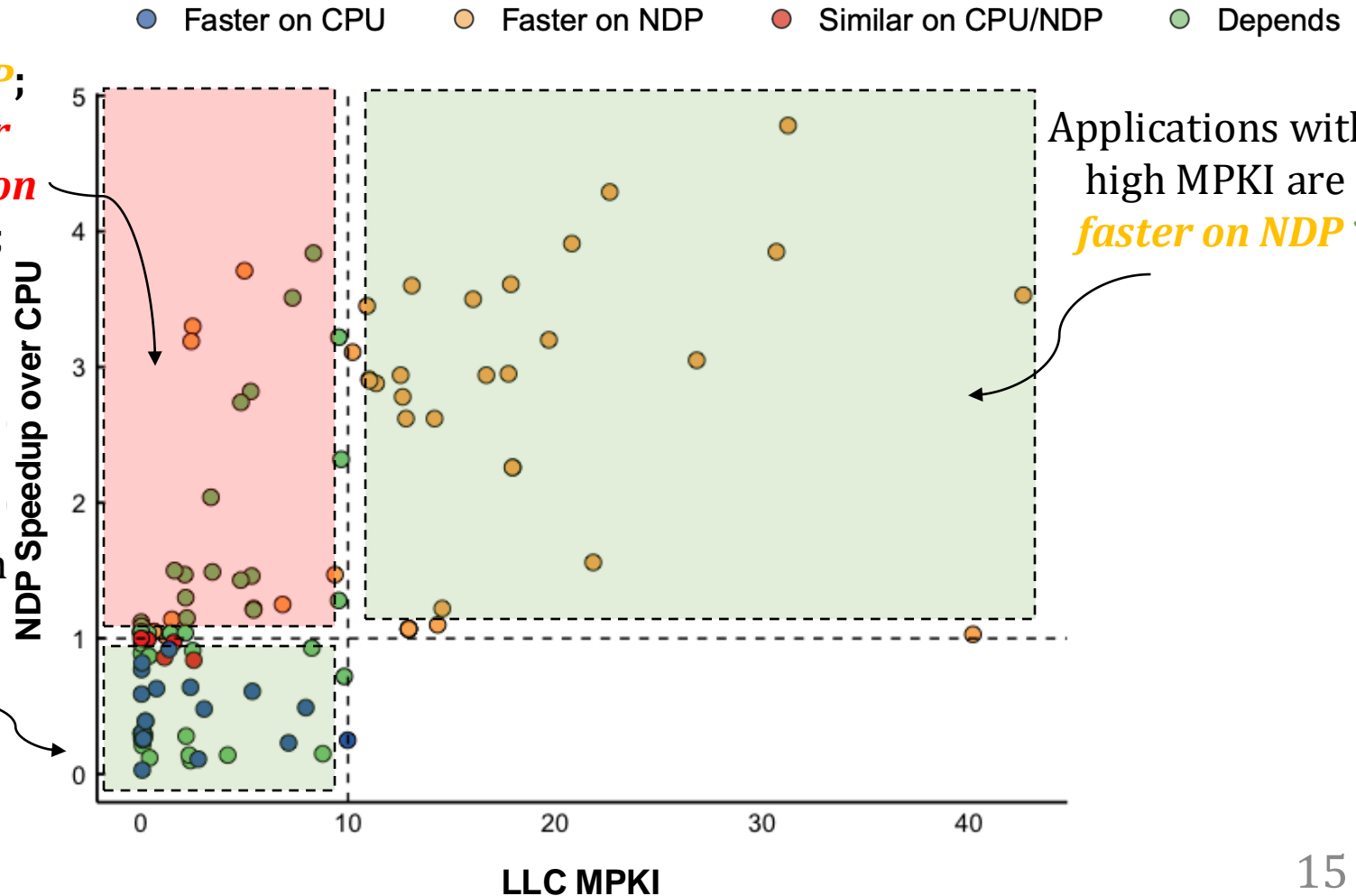
Limitations of Prior Approaches (2/2)

- Application with a last-level cache **MPKI > 10**
→ **memory intensive** and **benefits from NDP**

Applications with low MPKI can be

faster on NDP;
have *similar performance on CPU/NDP* or;
performance can *depend*
X

Applications with low MPKI are *faster on CPU*
✓



Applications with high MPKI are *faster on NDP* ✓

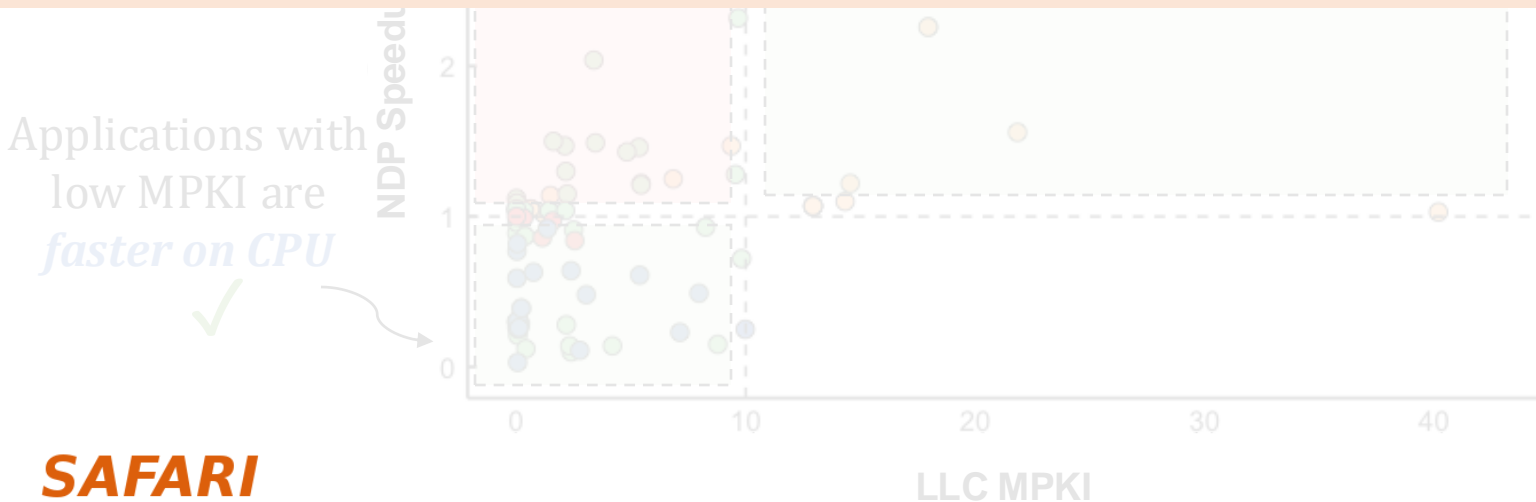
Limitations of Prior Approaches (2/2)

- Application with a last-level cache MPKI > 10
→ **memory intensive** and **benefits from NDP**

Applications with low MPKI can be *faster on NDP*;

● Faster on CPU ● Faster on NDP ● Similar on CPU/NDP ● Depends

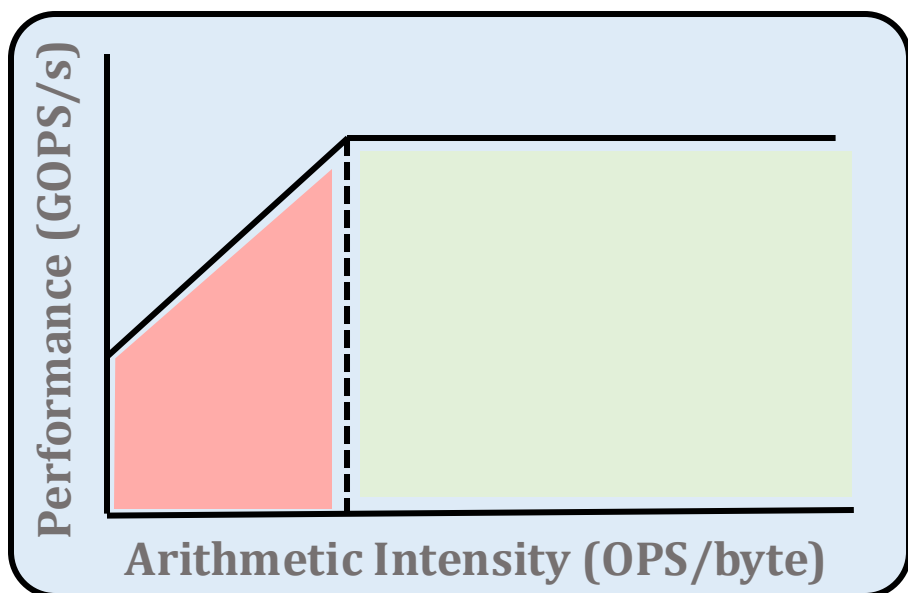
LLC MPKI does not accurately account for the NDP suitability of memory-bound applications



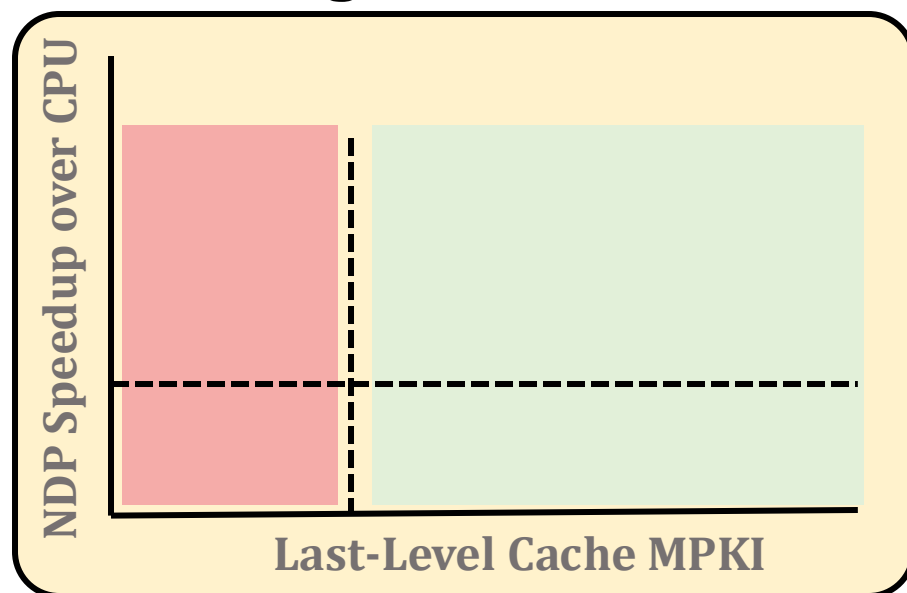
Identifying Memory Bottlenecks

- **Multiple approaches** to **identify** applications that:
 - suffer from data movement bottlenecks
 - take advantage of NDP
- Existing approaches are **not comprehensive enough**

Roofline model



High LLC MPKI

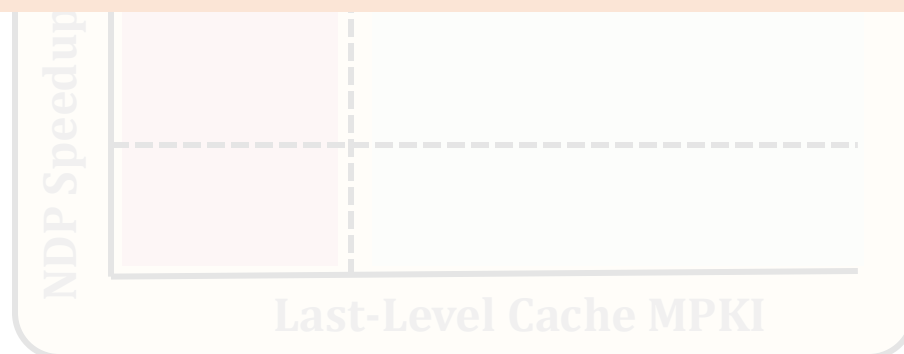


The Problem

- Multiple approaches to identify applications that:
 - suffer from data movement bottlenecks
 - take advantage of NDP

No available methodology can comprehensively:

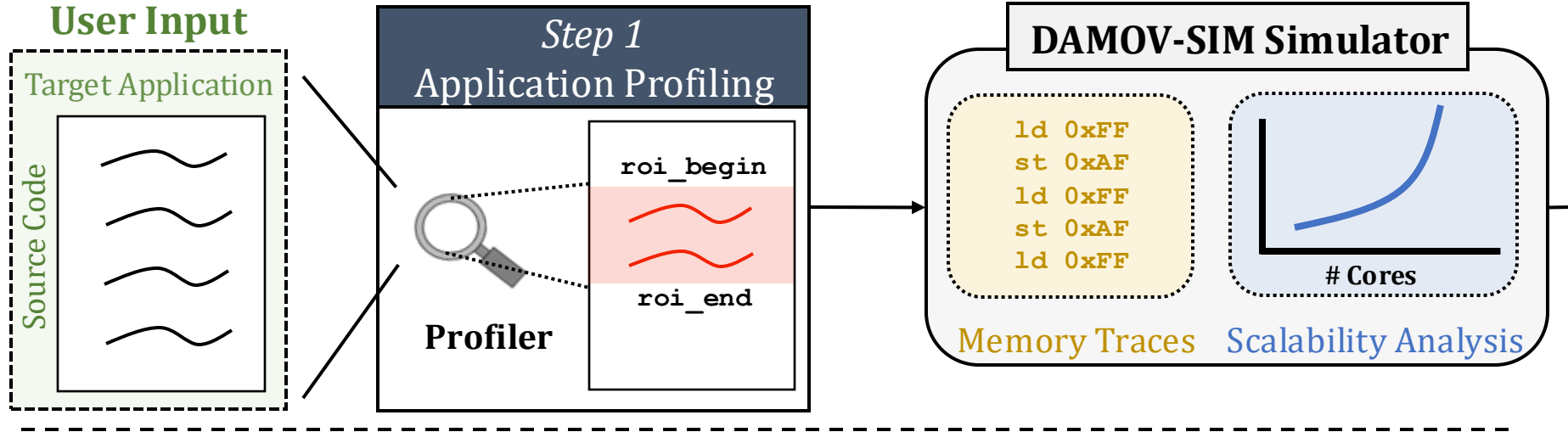
- **identify** data movement bottlenecks
- **correlate** them with the **most suitable** data movement mitigation mechanism



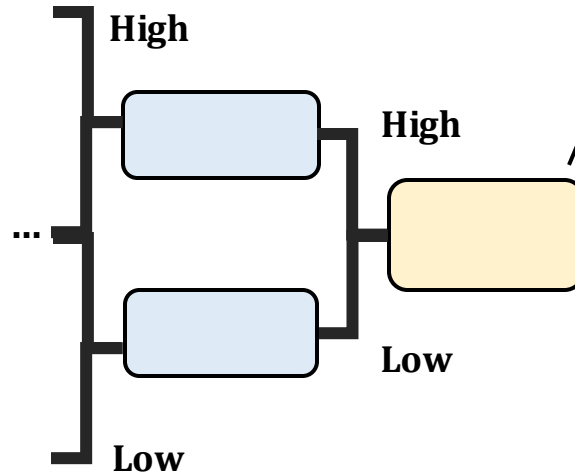
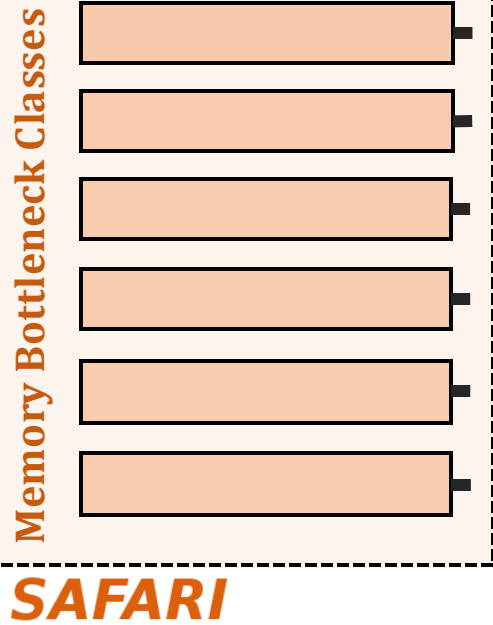
Our Goal

- **Our Goal:** develop a methodology to:
 - **methodically identify** sources of data movement bottlenecks
 - **comprehensively compare** compute- and memory-centric data movement mitigation techniques

Methodology Overview



Methodology Output



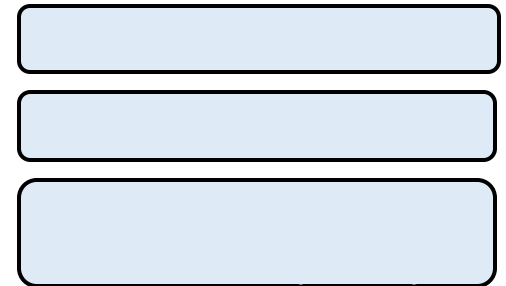
Step 2

Locality-based Clustering

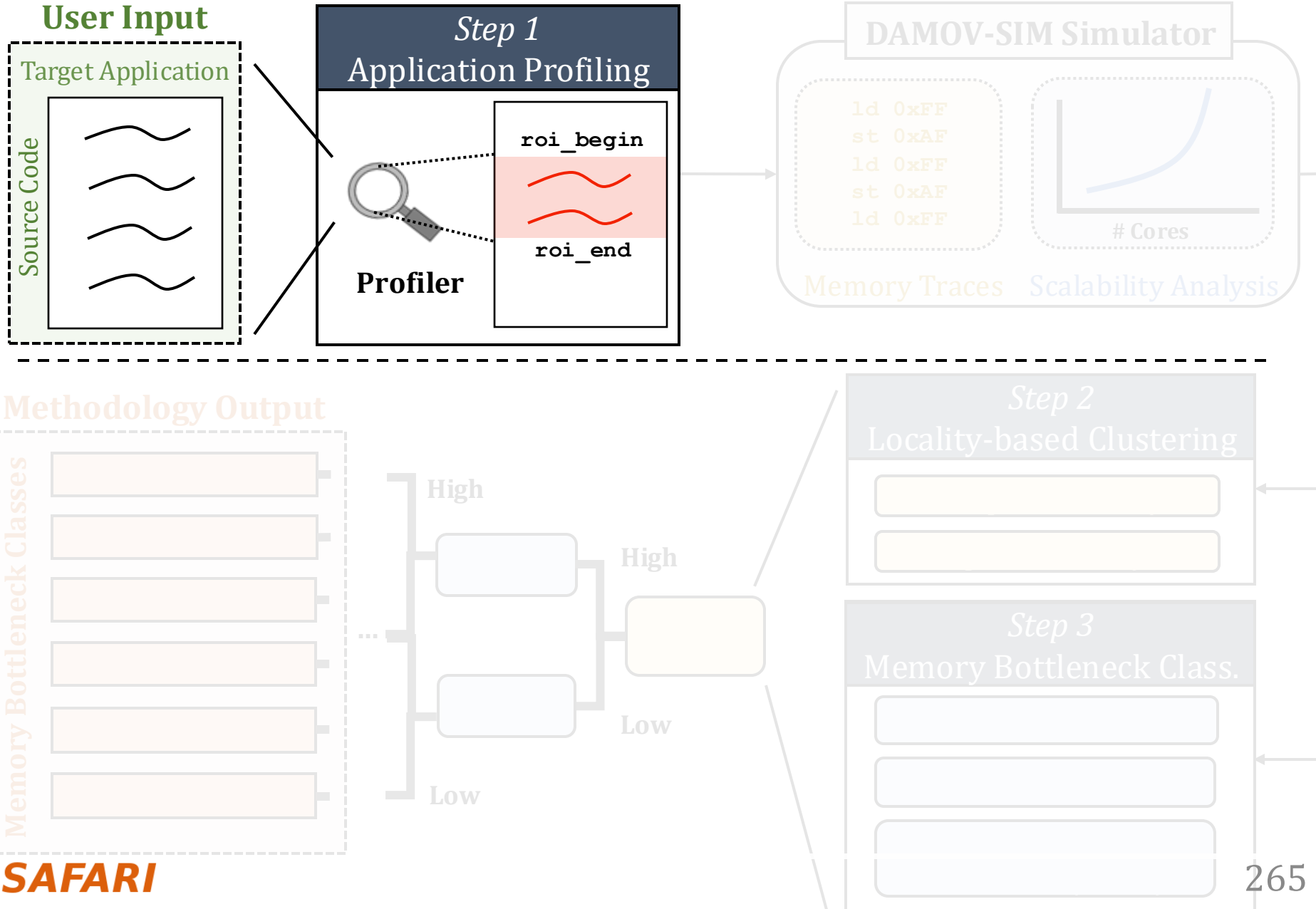


Step 3

Memory Bottleneck Class.

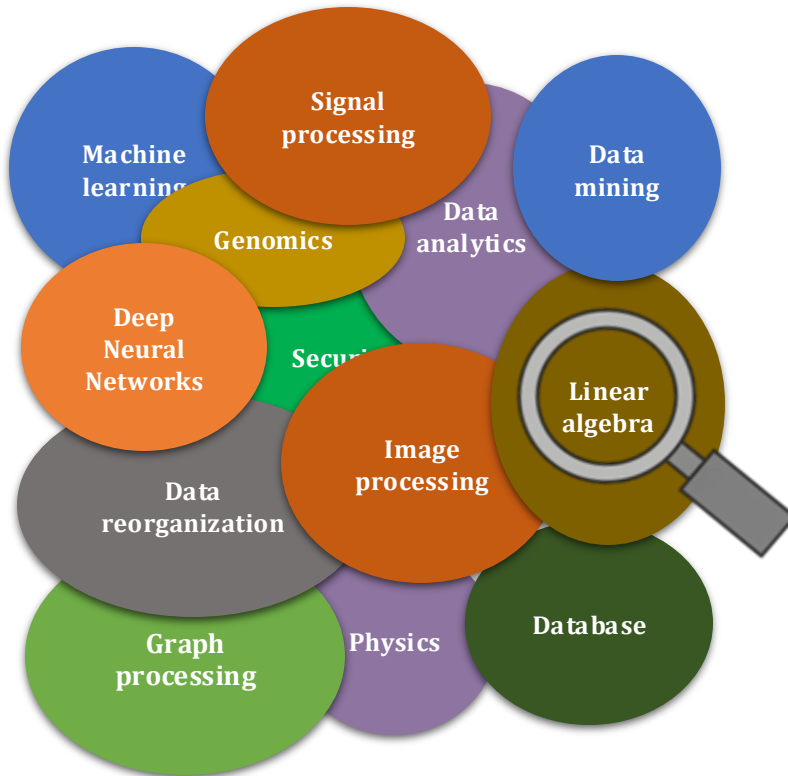


Methodology Overview

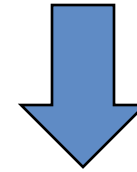


Step 1: Application Profiling

Goal: Identify **application functions** that suffer from **data movement bottlenecks**

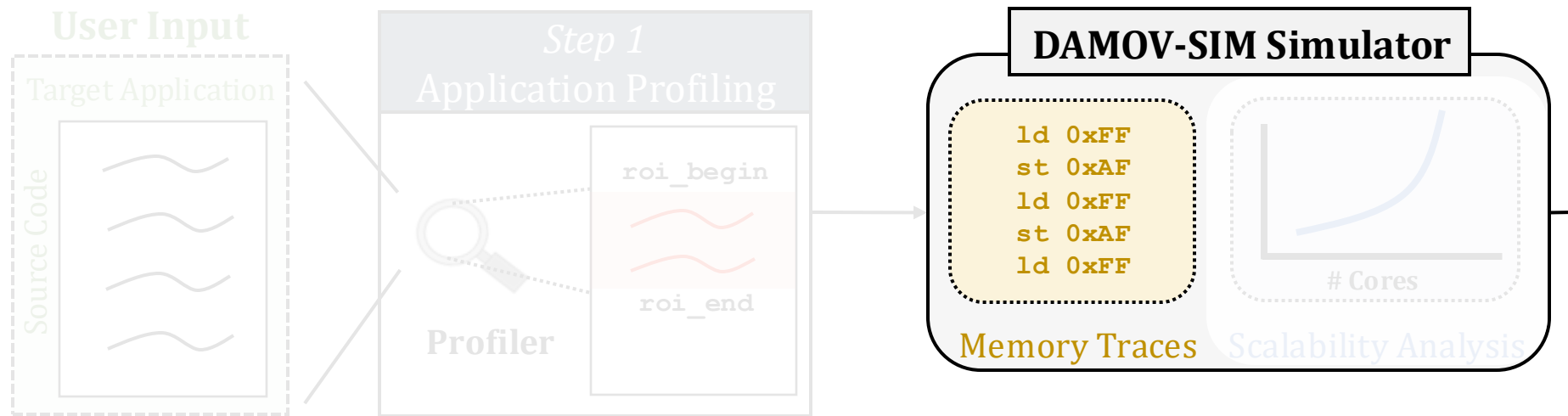


Hardware Profiling Tool:
Intel VTune

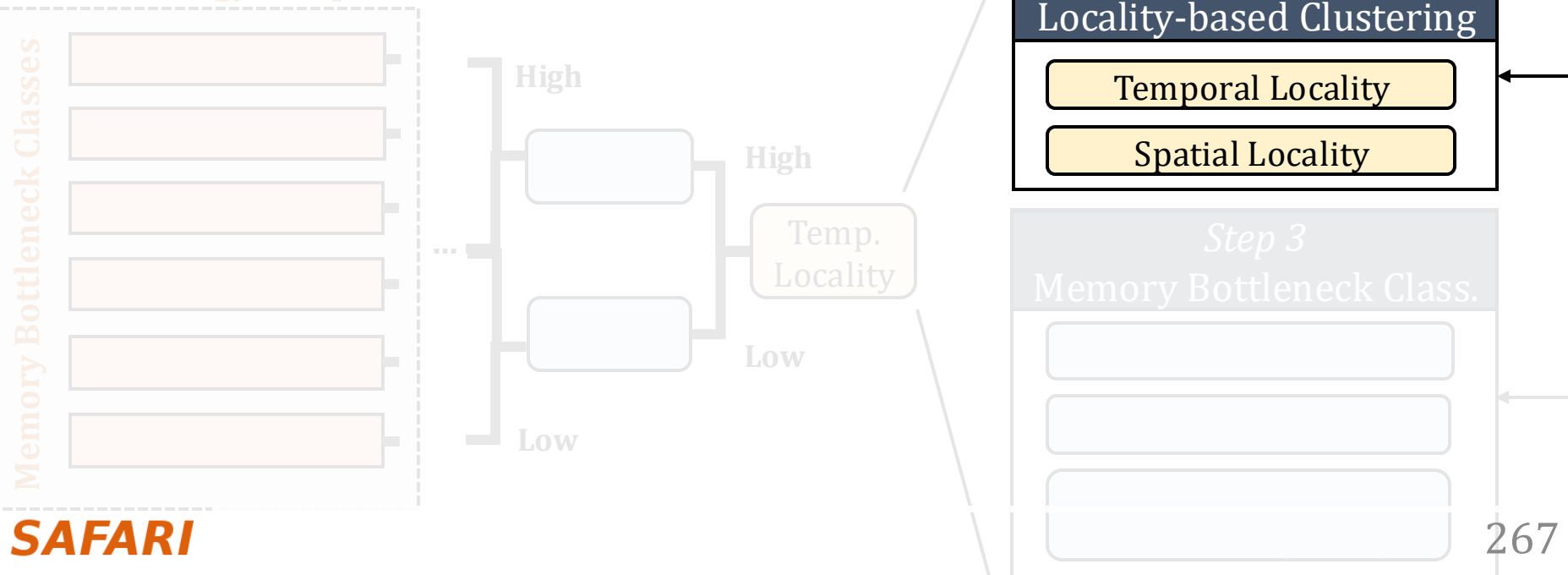


MemoryBound:
CPU is stalled due to load/store

Methodology Overview



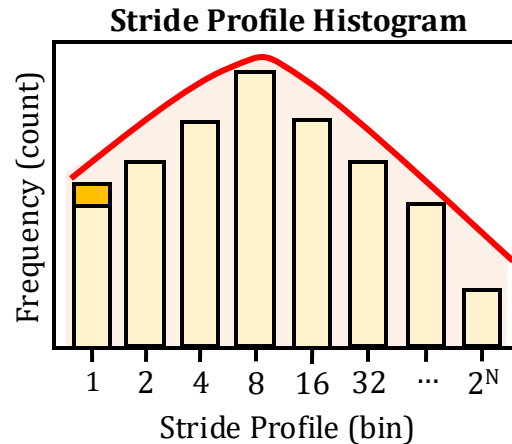
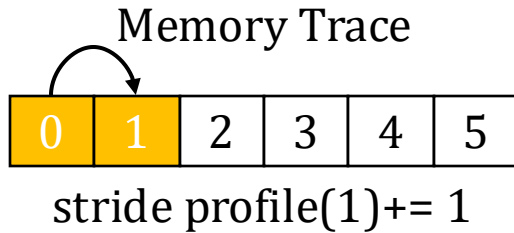
Methodology Output



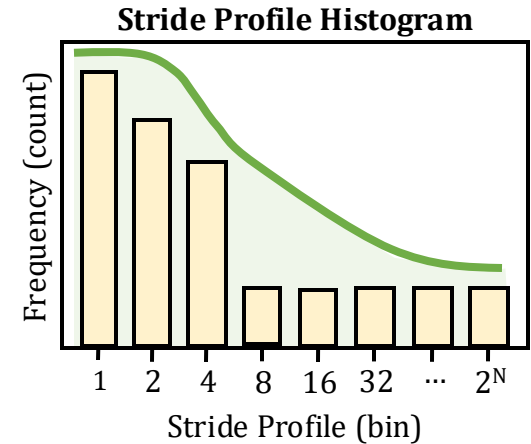
Step 2: Locality-Based Clustering

- **Goal:** analyze application's memory characteristics

Spatial Locality⁷



Low spatial locality

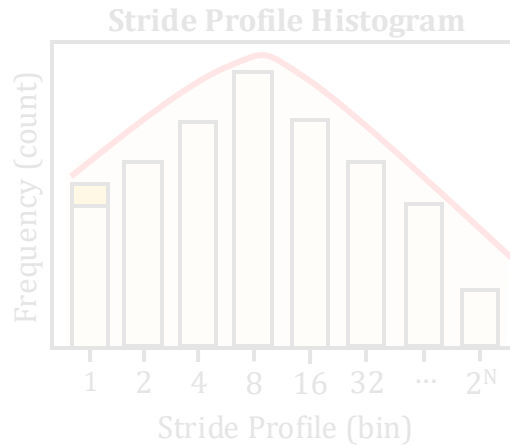
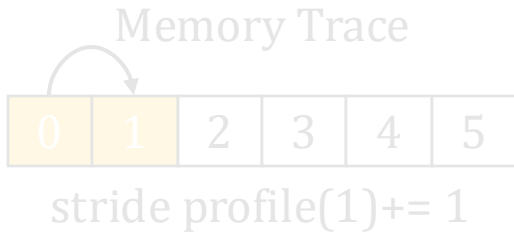


High spatial locality

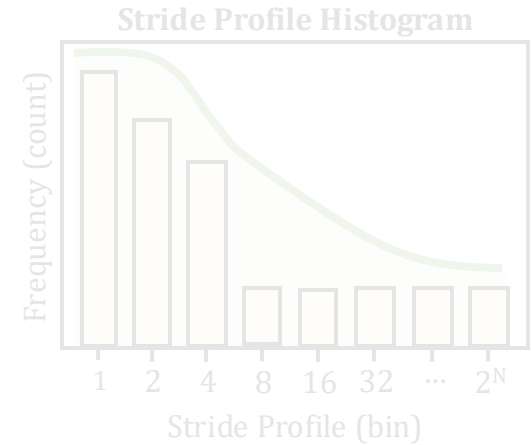
Step 2: Locality-Based Clustering

- **Goal:** analyze application's memory characteristics

Spatial Locality⁷

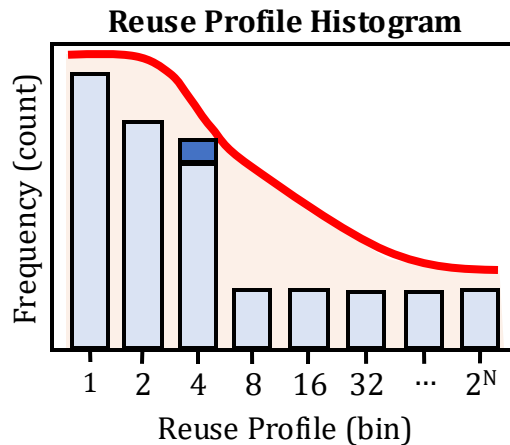
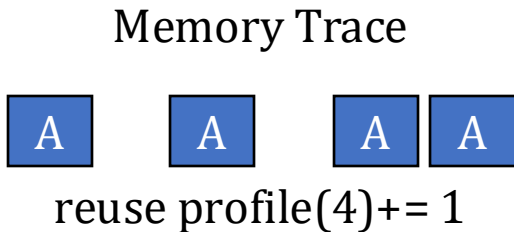


Low spatial locality

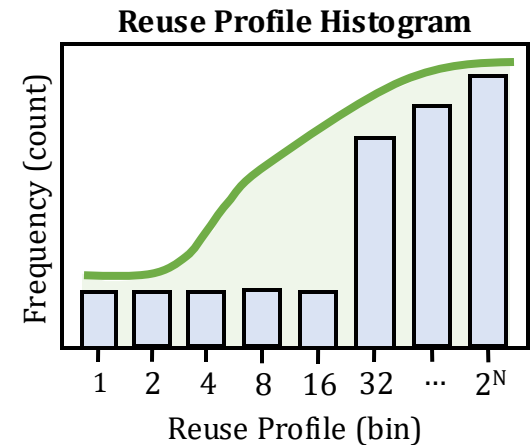


High spatial locality

Temporal Locality⁷

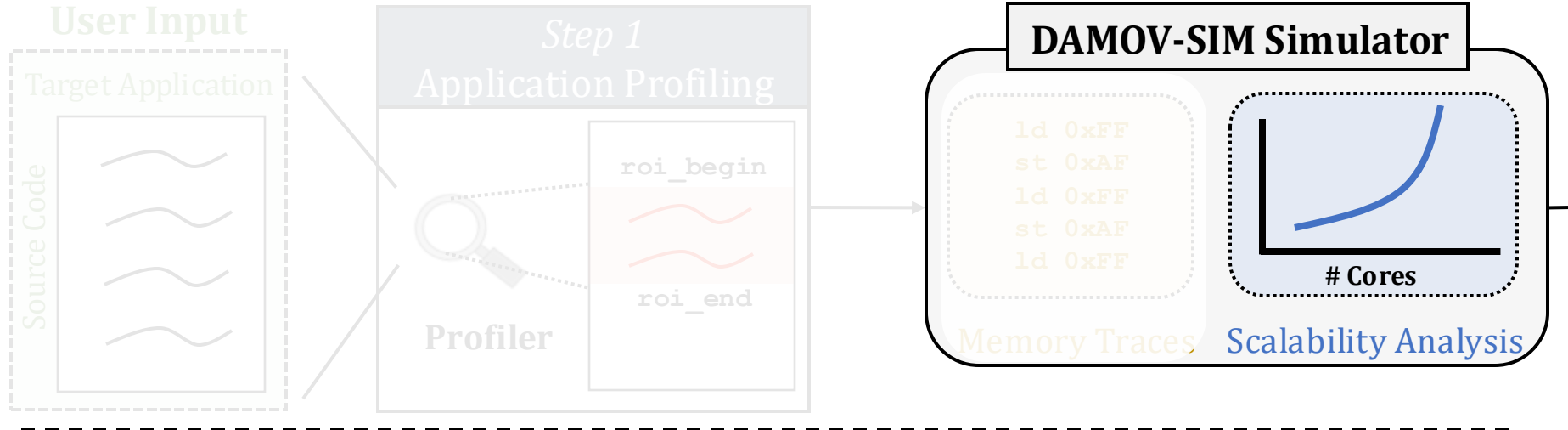


Low temporal locality



High temporal locality

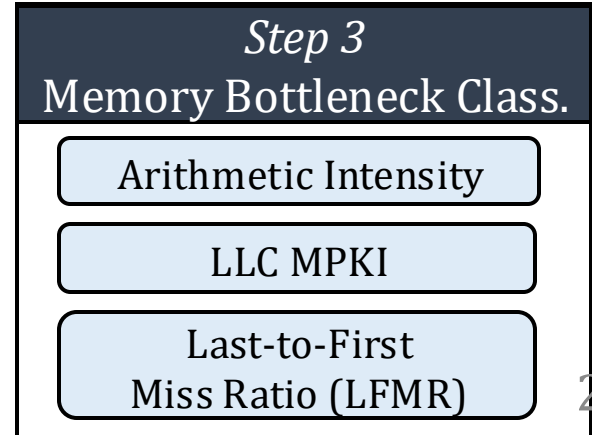
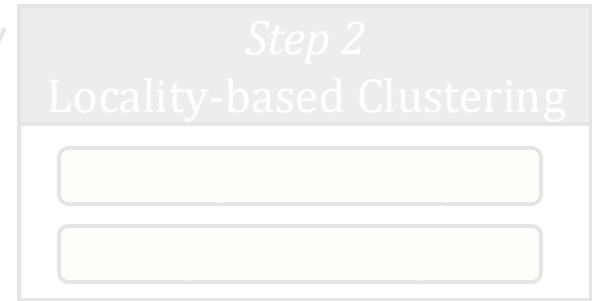
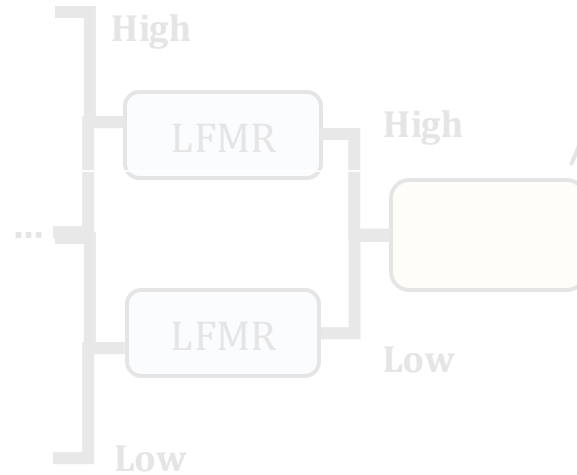
Methodology Overview



Methodology Output



SAFARI



Step 3: Memory Bottleneck Classification (1/2)

Arithmetic Intensity (AI)

- floating-point/arithmetic operations per L1 cache lines accessed
→ shows **computational intensity** per memory request

LLC Misses-per-Kilo-Instructions (MPKI)

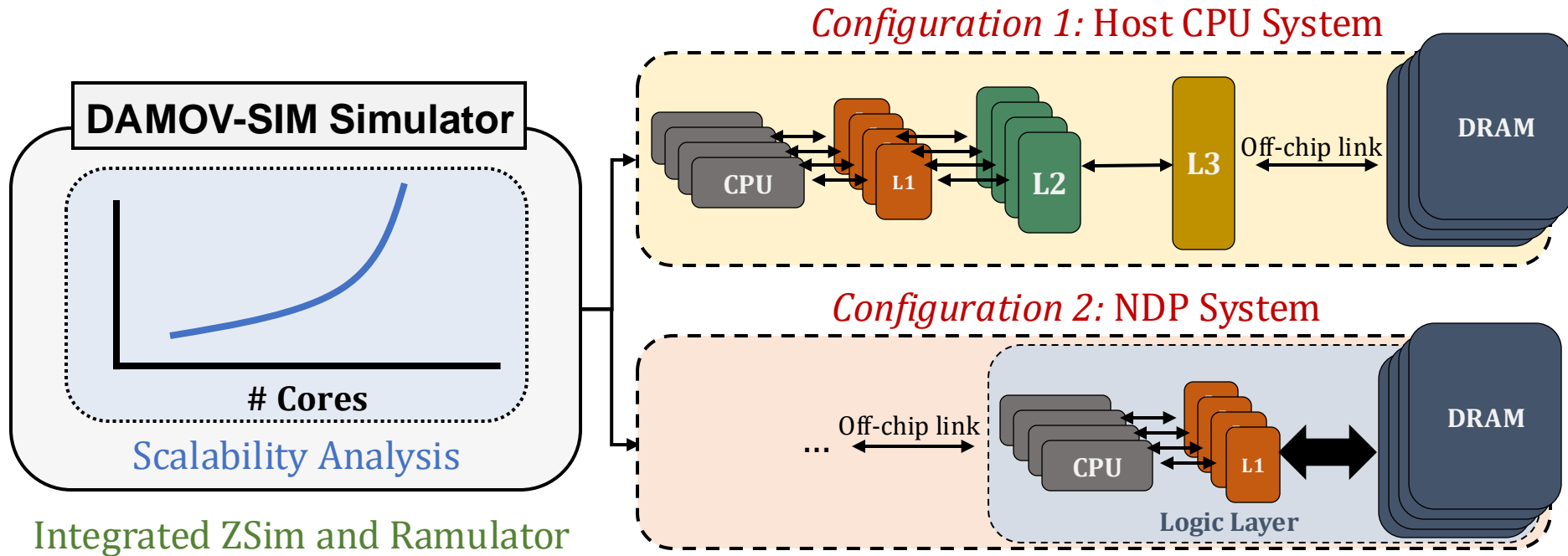
- LLC misses per one thousand instructions
→ shows **memory intensity**

Last-to-First Miss Ratio (LFMR)

- LLC misses per L1 misses
→ shows if an application **benefits from L2/L3 caches**

Step 3: Memory Bottleneck Classification (2/2)

- **Goal:** identify the specific sources of data movement bottlenecks



- **Scalability Analysis:**

- 1, 4, 16, 64, and 256 out-of-order/in-order host and NDP CPU cores
- 3D-stacked memory as main memory

Step 3: Memory Bottleneck Analysis

Six classes of data movement bottlenecks:

each class \leftrightarrow data movement mitigation mechanism

Memory Bottleneck Class

1a: *DRAM Bandwidth*

1b: *DRAM Latency*

1c: *L1/L2 Cache Capacity*

2a: *L3 Cache Contention*

2b: *L1 Cache Capacity*

2c: *Compute-Bound*

DAMOV is Open Source

- We open-source our **benchmark suite** and our **toolchain**

CMU-SAFARI / DAMOV

<> Code Issues Pull requests Actions Projects Security Insights Settings

main 1 branch 0 tags

Go to file

Add file

Code

About

DAMOV is a benchmark suite and a methodical framework targeting the study of data movement bottlenecks in modern applications. It is intended to study new architectures, such as near-data processing. Described by Oliveira et al. (preliminary version at <https://arxiv.org/pdf/2105.03725.pdf>)

omutlu Update README.md

ce1b4ea 17 days ago 5 commits

simulator	Cleaning	19 days ago
README.md	Update README.md	17 days ago
get_workloads.sh	DAMOV -- first commit	19 days ago

README.md

DAMOV: A New Methodology and Benchmark Suite for Evaluating Data Movement Bottlenecks

DAMOV is a benchmark suite and a methodical framework targeting the study of data movement bottlenecks in modern applications. It is intended to study new architectures, such as near-data processing.

The DAMOV benchmark suite is the first open-source benchmark suite for main memory data movement-related studies, based on our systematic characterization methodology. This suite consists of 144 functions representing different sources of data movement bottlenecks and can be used as a baseline benchmark set for future data-movement mitigation research. The applications in the DAMOV benchmark suite belong to popular benchmark suites, including [BWA](#), [Chai](#), [Darknet](#), [GASE](#), [Hardware Effects](#), [Hashjoin](#), [HPCC](#), [HPCG](#), [Ligra](#), [PARSEC](#), [Parboil](#), [PolyBench](#), [Phoenix](#), [Rodinia](#), [SPLASH-2](#), [STREAM](#).

Readme

Releases

No releases published
[Create a new release](#)

Packages

No packages published
[Publish your first package](#)

Languages



DAMOV-SIM

DAMOV
Benchmarks

DAMOV is Open Source

- We open-source our [benchmark suite](#) and our [toolchain](#)

CMU-SAFARI / DAMOV

<> Code Issues Pull requests Actions Projects Security Insights Settings

main 1 branch 0 tags

Go to file

Add file

Code

About

DAMOV is a benchmark suite and a

Get DAMOV at:

<https://github.com/CMU-SAFARI/DAMOV>

README.md

DAMOV: A New Methodology and Benchmark Suite for Evaluating Data Movement Bottlenecks

DAMOV is a benchmark suite and a methodical framework targeting the study of data movement bottlenecks in modern applications. It is intended to study new architectures, such as near-data processing.

The DAMOV benchmark suite is the first open-source benchmark suite for main memory data movement-related studies, based on our systematic characterization methodology. This suite consists of 144 functions representing different sources of data movement bottlenecks and can be used as a baseline benchmark set for future data-movement mitigation research. The applications in the DAMOV benchmark suite belong to popular benchmark suites, including [BWA](#), [Chai](#), [Darknet](#), [GASE](#), [Hardware Effects](#), [Hashjoin](#), [HPC](#), [HPCG](#), [Ligra](#), [PARSEC](#), [Parboil](#), [PolyBench](#), [Phoenix](#), [Rodinia](#), [SPLASH-2](#), [STREAM](#).

Readme

Releases

No releases published
[Create a new release](#)

Packages

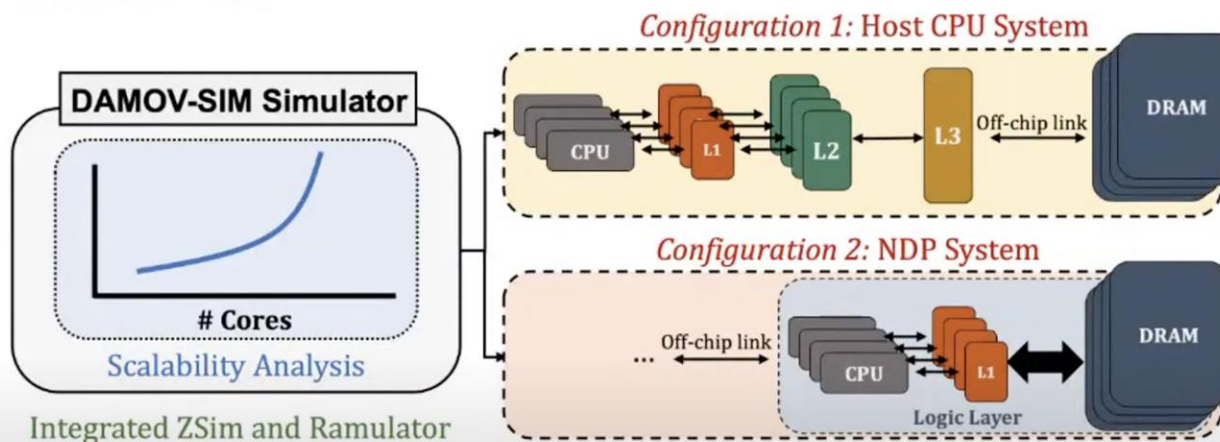
No packages published
[Publish your first package](#)

Languages

More on DAMOV Analysis Methodology & Workloads

Step 3: Memory Bottleneck Classification (2/)

- **Goal:** identify the specific sources of data movement bottlenecks



- **Scalability Analysis:**
 - 1, 4, 16, 64, and 256 out-of-order/in-order host and NDP CPU cores
 - 3D-stacked memory as main memory

SAFARI DAMOV-SIM: <https://github.com/CMU-SAFARI/DAMOV> 30

SAFARI Live Seminar: DAMOV: A New Methodology & Benchmark Suite for Data Movement Bottlenecks

352 views • Streamed live on Jul 22, 2021

18 0 SHARE SAVE ...



Onur Mutlu Lectures
17.7K subscribers

ANALYTICS

EDIT VIDEO

More on DAMOV Methods & Benchmarks

- Geraldo F. Oliveira, Juan Gomez-Luna, Lois Orosa, Saugata Ghose, Nandita Vijaykumar, Ivan fernandez, Mohammad Sadrosadati, and Onur Mutlu, **"DAMOV: A New Methodology and Benchmark Suite for Evaluating Data Movement Bottlenecks"**
IEEE Access, 8 September 2021.
Preprint in arXiv, 8 May 2021.
[[arXiv preprint](#)]
[[IEEE Access version](#)]
[[DAMOV Suite and Simulator Source Code](#)]
[[SAFARI Live Seminar Video](#) (2 hrs 40 mins)]
[[Short Talk Video](#) (21 minutes)]

DAMOV: A New Methodology and Benchmark Suite for Evaluating Data Movement Bottlenecks

GERALDO F. OLIVEIRA, ETH Zürich, Switzerland

JUAN GÓMEZ-LUNA, ETH Zürich, Switzerland

LOIS OROSA, ETH Zürich, Switzerland

SAUGATA GHOSE, University of Illinois at Urbana–Champaign, USA

NANDITA VIJAYKUMAR, University of Toronto, Canada

IVAN FERNANDEZ, University of Malaga, Spain & ETH Zürich, Switzerland

MOHAMMAD SADROSADATI, ETH Zürich, Switzerland

ONUR MUTLU, ETH Zürich, Switzerland

Fundamentally
Energy-Efficient
(Data-Centric)

Computing Architectures

Fundamentally High-Performance **(Data-Centric)** Computing Architectures

Computing Architectures with Minimal Data Movement

Concluding Remarks

- We must design systems to be **balanced, high-performance, energy-efficient** (all at the same time) → intelligent systems
 - **Data-centric, data-driven, data-aware**
- Enable computation capability inside and close to memory
- This can
 - Lead to **orders-of-magnitude** improvements
 - **Enable new applications & computing platforms**
 - **Enable better understanding of nature**
 - ...
- Future of **truly memory-centric computing** is bright
 - We need to do research & design across the computing stack

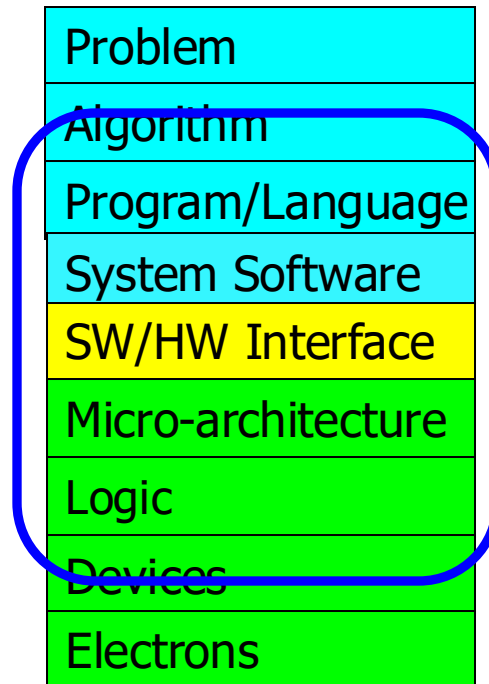
Fundamentally Better Architectures

Data-centric

Data-driven

Data-aware

We Need to Revisit the Entire Stack



We can get there step by step

We Need to Exploit Good Principles

- Data-centric system design
- All components intelligent
- Better (cross-layer) communication, better interfaces
- Better-than-worst-case design
- Heterogeneity
- Flexibility, adaptability

Open minds

PIM Review and Open Problems

A Modern Primer on Processing in Memory

Onur Mutlu^{a,b}, Saugata Ghose^{b,c}, Juan Gómez-Luna^a, Rachata Ausavarungnirun^d

SAFARI Research Group

^a*ETH Zürich*

^b*Carnegie Mellon University*

^c*University of Illinois at Urbana-Champaign*

^d*King Mongkut's University of Technology North Bangkok*

Onur Mutlu, Saugata Ghose, Juan Gomez-Luna, and Rachata Ausavarungnirun,

"A Modern Primer on Processing in Memory"

*Invited Book Chapter in **Emerging Computing: From Devices to Systems - Looking Beyond Moore and Von Neumann**, Springer, to be published in 2021.*

Special Research Sessions & Courses (I)

- Special Session at ISVLSI 2022: 9 cutting-edge talks



The image shows a YouTube video player interface. The video title is "In-Memory Processing ISVLSI 2022 Special Session". The subtitle is "IEEE Computer Society Annual Symposium on VLSI". The video content area displays the event details: "ISVLSI 2022" logo, "Adonis room", "Ailathon resort, Paphos, Cyprus", and "July 4th, 2022". The video player controls show a progress bar at 0:04 / 3:36:35. The video is by "Onur Mutlu Lectures" with 26.9K subscribers. The video has 1,286 views and premiered on Aug 9, 2022. The video player includes standard YouTube controls like play, volume, and full screen, as well as engagement buttons like like (61), dislike, share, download, clip, save, analytics, and edit video.

In-Memory Processing
ISVLSI 2022 Special Session

IEEE Computer Society Annual Symposium on VLSI

ISVLSI 2022

Adonis room
Ailathon resort, Paphos, Cyprus
July 4th, 2022

0:04 / 3:36:35 • Dr. Juan Gómez-Luna, "Introduction to the ISVLSI 2022 Special Session on Processing-in-Memory" >

ISVLSI 2022 Special Session on Processing-in-Memory

1,286 views • Premiered Aug 9, 2022

61 DISLIKE SHARE DOWNLOAD CLIP SAVE

Onur Mutlu Lectures
26.9K subscribers




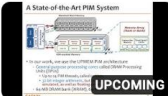
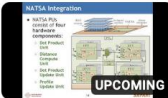

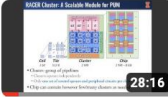


ANALYTICS EDIT VIDEO

Special Research Sessions & Courses (II)

Special Session at ISVLSI 2022: 9 cutting-edge talks

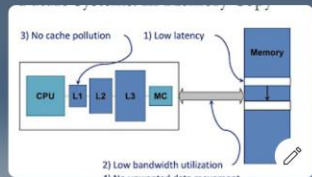
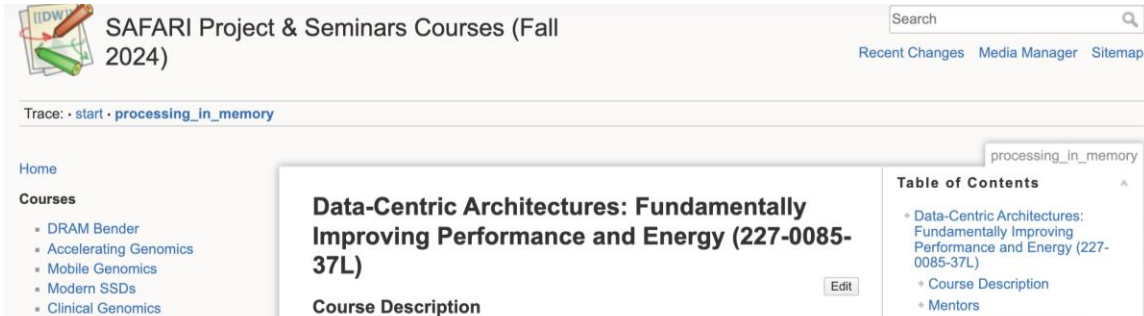
Livestream - P&S Data-Centric Architectures: Fundamentally...
Onur Mutlu Lectures
27 videos 1,034 views Last updated on Feb 25, 2023

Play all Shuffle

- 19  **GenStore: In-Storage Filtering for High-Performance and Energy-Efficient Genome Analysis**
Onur Mutlu Lectures • Premieres 3/12/23, 7:00 PM
- 20  **Introduction to the ISVLSI 2022 Special Session on Processing-in-Memory**
Onur Mutlu Lectures • 286 views • 2 days ago
- 21  **Heterogeneous Data-Centric Architectures for Data-Intensive Applications: Case Studies in ML and DB**
Onur Mutlu Lectures • 2 waiting • Premieres 3/10/23, 7:00 PM
- 22  **Machine Learning Training on a Real Processing-In-Memory System**
Onur Mutlu Lectures • Premieres 3/14/23, 7:00 PM
- 23  **Exploiting Near-Data Processing to Accelerate Time Series Analysis**
Onur Mutlu Lectures • Premieres 3/11/23, 7:00 PM
- 24  **PIDRAM: An FPGA-Based Framework for End-To-End Evaluation of Processing-In-DRAM Techniques**
Onur Mutlu Lectures • Premieres 3/9/23, 7:00 PM
- 25  **The Road to Widely Deploying Processing-In-Memory: Challenges and Opportunities**
Onur Mutlu Lectures • 399 views • 1 day ago
- 26  **SparseP: Efficient Sparse Matrix Vector Multiplication on Real Processing-In-Memory Architectures**
Onur Mutlu Lectures • 1 waiting • Premieres 3/13/23, 7:00 PM
- 27  **HPCA 2023 Tutorial: Real-World Processing-in-Memory Architectures**
Onur Mutlu Lectures • 1.6K views • Streamed 10 days ago

Processing-in-Memory Course (Fall 2024)

- Short weekly lectures
- Hands-on projects

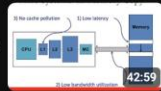


Livestream - Data-Centric Architectures: ...

Onur Mutlu Lectures
Playlist • Public • 4 videos • 179 views

Play all

Sort



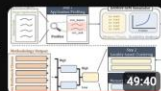
PIM Course: Lecture 4: Processing-Using-Memory for Data Manipulation (Fall 2024)

Onur Mutlu Lectures • 338 views • Streamed 2 days ago



PIM Course: Lecture 3: Processing-Near-Memory (Fall 2024)

Onur Mutlu Lectures • 324 views • Streamed 9 days ago



PIM Course: Lecture 2: How to Evaluate Data Movement Bottlenecks (Fall 2024)

Onur Mutlu Lectures • 434 views • Streamed 2 weeks ago



PIM Course: Lecture 1: Data-Centric Architectures: Improving Performance & Energy (Fall 2024)

Onur Mutlu Lectures • 337 views • Streamed 3 weeks ago

compute units of energy bottleneck. When costs dominate energy consumption. and the energy in consumer is a huge burden of modern computing systems. This phenomenon is and the processor, which leads to the data movement

learning, computational biology, graph processing, suffer greatly from the data movement bottleneck. processes, relatively low data reuse, low cache line (s per accessed byte), and large datasets that greatly e workloads cannot usually compensate for the data nt bottleneck, we need a paradigm shift from the on takes place in the compute units, to a more data- bser to or inside where the data resides. This paradigm

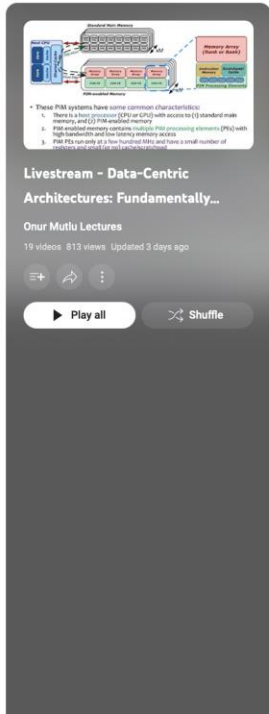
https://safari.ethz.ch/projects_and_seminars/fall2024/doku.php?id=processing_in_memory

<https://www.youtube.com/playlist?list=PL5Q2soXY2Zi9DSQg7OAlSNO8dOQKx>

F9sl

Processing-in-Memory Course (Spring 2023)

- Short weekly lectures
- Hands-on projects




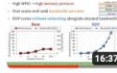

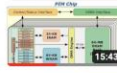
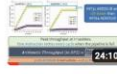



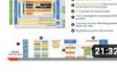
Livestream - Data-Centric Architectures: Fundamentally...

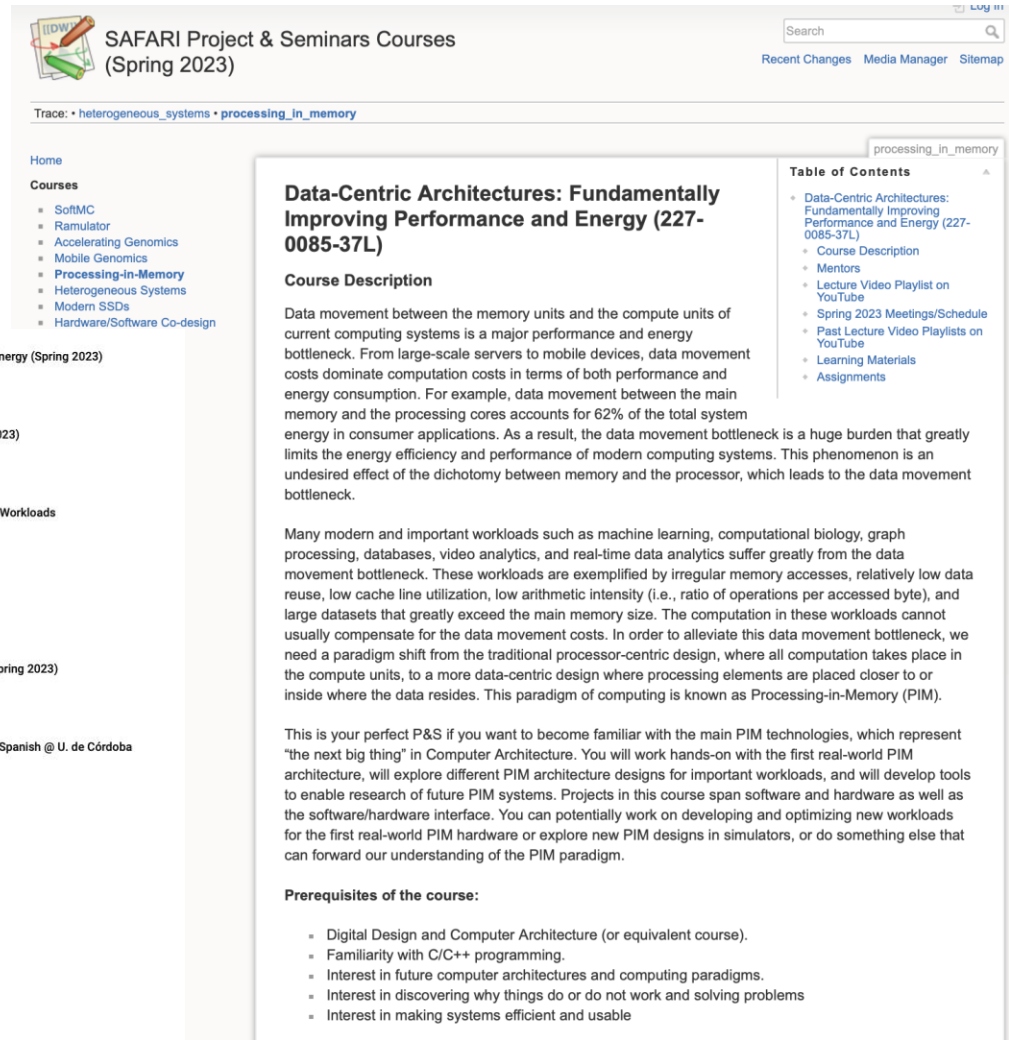
Onur Mutlu Lectures

19 videos · 813 views · Updated 3 days ago

▶ Play all

⌵ Shuffle

-  **PIM Course: Lecture 1: Data-Centric Architectures: Improving Performance & Energy (Spring 2023)**
Onur Mutlu Lectures • 1.1K views • Streamed 3 months ago
-  **PIM Course: Lecture 2: How to Evaluate Data Movement Bottlenecks (Spring 2023)**
Onur Mutlu Lectures • 332 views • 2 months ago
-  **ASPLoS 2023 Tutorial: Real-world Processing-in-Memory Systems for Modern Workloads**
Onur Mutlu Lectures • 1.5K views • Streamed 2 months ago
-  **PIM Course: Lecture 3: Real-world PIM: UPMEM PIM (Spring 2023)**
Onur Mutlu Lectures • 411 views • 2 months ago
-  **PIM Course: Lecture 4: Real-world PIM: Microbenchmarking of UPMEM PIM (Spring 2023)**
Onur Mutlu Lectures • 188 views • 2 months ago
-  **Análisis Experimental de una Arquitectura PIM - Juan Gómez Luna - Lecture in Spanish @ U. de Córdoba**
Onur Mutlu Lectures • 169 views • 2 months ago
-  **PIM Course: Lecture 5: Real-world PIM: Samsung HBM-PIM (Spring 2023)**
Onur Mutlu Lectures • 483 views • 2 months ago
-  **PIM Course: Lecture 6: Real-world PIM: SK Hynix AIM (Spring 2023)**
Onur Mutlu Lectures • 573 views • 1 month ago
-  **PIM Course: Lecture 7: Real-world PIM: Samsung AxDIMM (Spring 2023)**
Onur Mutlu Lectures • 325 views • 1 month ago



SAFARI Project & Seminars Courses (Spring 2023)

Trace: heterogeneous_systems • processing_in_memory

Home

Courses

- SoftMC
- Ramulator
- Accelerating Genomics
- Mobile Genomics
- Processing-in-Memory**
- Heterogeneous Systems
- Modern SSDs
- Hardware/Software Co-design

processing_in_memory

Data-Centric Architectures: Fundamentally Improving Performance and Energy (227-0085-37L)

Course Description

Data movement between the memory units and the compute units of current computing systems is a major performance and energy bottleneck. From large-scale servers to mobile devices, data movement costs dominate computation costs in terms of both performance and energy consumption. For example, data movement between the main memory and the processing cores accounts for 62% of the total system energy in consumer applications. As a result, the data movement bottleneck is a huge burden that greatly limits the energy efficiency and performance of modern computing systems. This phenomenon is an undesired effect of the dichotomy between memory and the processor, which leads to the data movement bottleneck.

Many modern and important workloads such as machine learning, computational biology, graph processing, databases, video analytics, and real-time data analytics suffer greatly from the data movement bottleneck. These workloads are exemplified by irregular memory accesses, relatively low data reuse, low cache line utilization, low arithmetic intensity (i.e., ratio of operations per accessed byte), and large datasets that greatly exceed the main memory size. The computation in these workloads cannot usually compensate for the data movement costs. In order to alleviate this data movement bottleneck, we need a paradigm shift from the traditional processor-centric design, where all computation takes place in the compute units, to a more data-centric design where processing elements are placed closer to or inside where the data resides. This paradigm of computing is known as Processing-in-Memory (PIM).

This is your perfect P&S if you want to become familiar with the main PIM technologies, which represent "the next big thing" in Computer Architecture. You will work hands-on with the first real-world PIM architecture, will explore different PIM architecture designs for important workloads, and will develop tools to enable research of future PIM systems. Projects in this course span software and hardware as well as the software/hardware interface. You can potentially work on developing and optimizing new workloads for the first real-world PIM hardware or explore new PIM designs in simulators, or do something else that can forward our understanding of the PIM paradigm.

Prerequisites of the course:

- Digital Design and Computer Architecture (or equivalent course).
- Familiarity with C/C++ programming.
- Interest in future computer architectures and computing paradigms.
- Interest in discovering why things do or do not work and solving problems
- Interest in making systems efficient and usable

https://www.youtube.com/playlist?list=PL5Q2soXY2zi_F0buoAZVSq_o6UySWQHvz

https://safari.ethz.ch/projects_and_seminars/spring2023/doku.php?id=processing_in_memory

PIM Course (Fall 2022)

■ Fall 2022 Edition:

- https://safari.ethz.ch/projects_and_seminars/fall2022/doku.php?id=processing_in_memory

■ Spring 2022 Edition:

- https://safari.ethz.ch/projects_and_seminars/spring2022/doku.php?id=processing_in_memory

■ Youtube Livestream (Fall 2022):

- <https://www.youtube.com/watch?v=QLL0wQ9I4Dw&list=PL5Q2soXY2Zi8KzG2CQYRNQOVD0GOBBrnKy>

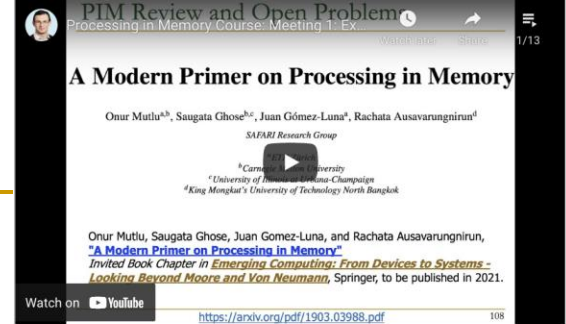
■ Youtube Livestream (Spring 2022):

- <https://www.youtube.com/watch?v=9e4Chnwdovo&list=PL5Q2soXY2Zi-841fUYYUK9EsXKhQKRPyX>

■ Project course

- Taken by Bachelor's/Master's students
- Processing-in-Memory lectures
- Hands-on research exploration
- Many research readings

<https://www.youtube.com/onurmutlulectures>




Spring 2022 Meetings/Schedule

Week	Date	Livestream	Meeting	Learning Materials	Assignments
W1	10.03 Thu.	YouTube Live	M1: P&S PIM Course Presentation (PDF) (PPT)	Required Materials Recommended Materials	HW 0 Out
W2	15.03 Tue.		Hands-on Project Proposals		
	17.03 Thu.	YouTube Premiere	M2: Real-world PIM: UPMEM PIM (PDF) (PPT)		
W3	24.03 Thu.	YouTube Live	M3: Real-world PIM: Microbenchmarking of UPMEM PIM (PDF) (PPT)		
W4	31.03 Thu.	YouTube Live	M4: Real-world PIM: Samsung HBM-PIM (PDF) (PPT)		
W5	07.04 Thu.	YouTube Live	M5: How to Evaluate Data Movement Bottlenecks (PDF) (PPT)		
W6	14.04 Thu.	YouTube Live	M6: Real-world PIM: SK Hynix AIM (PDF) (PPT)		
W7	21.04 Thu.	YouTube Premiere	M7: Programming PIM Architectures (PDF) (PPT)		
W8	28.04 Thu.	YouTube Premiere	M8: Benchmarking and Workload Suitability on PIM (PDF) (PPT)		
W9	05.05 Thu.	YouTube Premiere	M9: Real-world PIM: Samsung AxDIMM (PDF) (PPT)		
W10	12.05 Thu.	YouTube Premiere	M10: Real-world PIM: Alibaba HB-PNM (PDF) (PPT)		
W11	19.05 Thu.	YouTube Live	M11: SpMV on a Real PIM Architecture (PDF) (PPT)		
W12	26.05 Thu.	YouTube Live	M12: End-to-End Framework for Processing-using-Memory (PDF) (PPT)		
W13	02.06 Thu.	YouTube Live	M13: Bit-Serial SIMD Processing using DRAM (PDF) (PPT)		
W14	09.06 Thu.	YouTube Live	M14: Analyzing and Mitigating ML Inference Bottlenecks (PDF) (PPT)		
W15	15.06 Thu.	YouTube Live	M15: In-Memory HTAP Databases with HW/SW Co-design (PDF) (PPT)		
W16	23.06 Thu.	YouTube Live	M16: In-Storage Processing for Genome Analysis (PDF) (PPT)		
W17	18.07 Mon.	YouTube Premiere	M17: How to Enable the Adoption of PIM? (PDF) (PPT)		
W18	09.08 Tue.	YouTube Premiere	SS1: ISVLSI 2022 Special Session on PIM (PDF & PPT)		

Real PIM Tutorials [ISCA'23, ASPLOS'23, HPCA'23]

- June, March, Feb : Lectures + Hands-on labs + Invited talks



ISCA 2023 Real-World PIM Tutorial

Search

[Recent Changes](#) [Media Manager](#) [Sitemap](#)

Trace: • [start](#)

Real-world Processing-in-Memory Systems for Modern Workloads

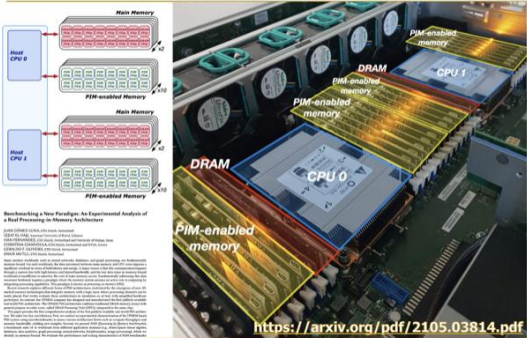
Tutorial Description

Processing-in-Memory (PIM) is a computing paradigm that aims at overcoming the data movement bottleneck (i.e., the waste of execution cycles and energy resulting from the back-and-forth data movement between memory units and compute units) by making memory compute-capable.

Explored over several decades since the 1960s, PIM systems are becoming a reality with the advent of the first commercial products and prototypes.

A number of startups (e.g., UPMEM, Neuroblade) are already commercializing real PIM hardware, each with its own design approach and target applications. Several major vendors (e.g., Samsung, SK Hynix, Alibaba) have presented real PIM chip prototypes in the last two years. Most of these architectures have in common that they place compute units near the memory arrays. This type of PIM is called processing near memory (PNM).

2,560-DPU Processing-in-Memory System



<https://arxiv.org/pdf/2105.03814.pdf>

PIM can provide large improvements in both performance and energy consumption for many modern applications, thereby enabling a commercially viable way of dealing with huge amounts of data that is bottlenecking our computing systems. Yet, it is critical to (1) study and understand the characteristics that make a workload suitable for a PIM architecture, (2) propose optimization strategies for PIM kernels, and (3) develop programming frameworks and tools that can lower the learning curve and ease the adoption of PIM.

This tutorial focuses on the latest advances in PIM technology, workload characterization for PIM, and programming and optimizing PIM kernels. We will (1) provide an introduction to PIM and taxonomy of PIM systems, (2) give an overview and a rigorous analysis of existing real-world PIM hardware, (3) conduct hand-on labs about important workloads (machine learning, sparse linear algebra, bioinformatics, etc.) using real PIM systems, and (4) shed light on how to improve future PIM systems for such workloads.

Table of Contents

- Real-world Processing-in-Memory Systems for Modern Workloads
- Tutorial Description
- Organizers
- Agenda (June 18, 2023)
- Lectures (tentative)
- Hands-on Labs (tentative)
- Learning Materials

Real PIM Tutorial [ISCA 2023]

■ June 18: Lectures + Hands-on labs + Invited talks

ISCA 2023 Real-World PIM Tutorial
 Sunday, June 18, Orlando, Florida

Organizers: Juan Gómez Luna, Onur Mutlu, Ataberk Olgun
 Program: <https://events.safari.ethz.ch/isca-pim-tutorial/>




Overview PIM | PNM | UPMEM PIM | PNM for neural networks | PNM for recommender systems | PNM for ML workloads | How to enable PIM? | PUM prototypes
Hands-on Labs: Benchmarking | Accelerating real-world workloads

International Symposium on Computer Architecture (ISCA)

Real-world Processing-in-Memory Systems for Modern Workloads

<https://www.youtube.com/live/GIb5EgSrWko?feature=share>

Room: Magnolia 16
 Marriott World Center Orlando
 Orlando, FL, USA
 July 18th, 2023

SAFARI zoom

ISCA 2023 Tutorial: Real-world Processing-in-Memory Systems for Modern Workloads

Onur Mutlu Lectures 33.9K subscribers

1,687 views Streamed live on Jun 18, 2023 Livestream - Data-Centric Architectures: Fundamentally Improving Performance and Energy (Spring 2023)

<https://www.youtube.com/live/GIb5EgSrWk0>

<https://events.safari.ethz.ch/isca-pim-tutorial/>

Tutorial Materials

Time	Speaker	Title	Materials
8:55am-9:00am	Dr. Juan Gómez Luna	Welcome & Agenda	(PDF) (PPT)
9:00am-10:20am	Prof. Onur Mutlu	Memory-Centric Computing	(PDF) (PPT)
10:20am-11:00am	Dr. Juan Gómez Luna	Processing-Near-Memory: Real PNM Architectures / Programming General-purpose PIM	(PDF) (PPT)
11:20am-11:50am	Prof. Izzat El Hajj	High-throughput Sequence Alignment using Real Processing-in-Memory Systems	(PDF) (PPT)
11:50am-12:30pm	Dr. Christina Giannoula	SparseP: Towards Efficient Sparse Matrix Vector Multiplication for Real Processing-In-Memory Systems	(PDF) (PPT)
2:00pm-2:45pm	Dr. Sukhan Lee	Introducing Real-world HBM-PIM Powered System for Memory-bound Applications	(PDF) (PPT)
2:45pm-3:30pm	Dr. Juan Gómez Luna / Ataberk Olgun	Processing-Using-Memory: Exploiting the Analog Operational Properties of Memory Components / PUM Prototypes: PiDRAM	(PDF) (PPT)
4:00pm-4:40pm	Dr. Juan Gómez Luna	Accelerating Modern Workloads on a General-purpose PIM System	(PDF) (PPT)
4:40pm-5:20pm	Dr. Juan Gómez Luna	Adoption Issues: How to Enable PIM?	(PDF) (PPT)
5:20pm-5:30pm	Dr. Juan Gómez Luna	Hands-on Lab: Programming and Understanding a Real Processing-in-Memory Architecture	(Handout) (PDF) (PPT)

Real PIM Tutorial [ASPLOS 2023]

■ March 26: Lectures + Hands-on labs + Invited talks

Tutorial Materials

Time	Speaker	Title	Materials
9:00am-10:20am	Prof. Onur Mutlu	Memory-Centric Computing	PDF PPT
10:40am-12:00pm	Dr. Juan Gómez Luna	Processing-Near-Memory: Real PNM Architectures Programming General-purpose PIM	PDF PPT
1:40pm-2:20pm	Prof. Alexandra (Sasha) Fedorova (UBC)	Processing in Memory in the Wild	PDF PPT
2:20pm-3:20pm	Dr. Juan Gómez Luna & Ataberk Olgun	Processing-Using-Memory: Exploiting the Analog Operational Properties of Memory Components	PDF PPT PDF PPT
3:40pm-4:10pm	Dr. Juan Gómez Luna	Adoption issues: How to enable PIM? Accelerating Modern Workloads on a General-purpose PIM System	PDF PPT PDF PPT
4:10pm-4:50pm	Dr. Yongkee Kwon & Eddy (Chanwook) Park (SK Hynix)	System Architecture and Software Stack for GDDR6-AiM	PDF PPT
4:50pm-5:00pm	Dr. Juan Gómez Luna	Hands-on Lab: Programming and Understanding a Real Processing-in-Memory Architecture	Handout PDF PPT

ASPLOS 2023 Tutorial: Real-world Processing-in-Memory Systems for Modern Workloads

Onur Mutlu Lectures
32.1K subscribers

33 likes

Streamed 7 days ago Livestream - Data-Centric Architectures: Fundamentally Improving Performance and Energy (Spring 2023)
LOS 2023 Tutorial: Real-world Processing-in-Memory Systems for Modern Workloads
<https://events.safari.ethz.ch/asplos-...>

<https://www.youtube.com/watch?v=oYCaLcT0Kmo>

<https://events.safari.ethz.ch/asplos-pim-tutorial/>

Real PIM Tutorial [HPCA 2023]

February 26: Lectures + Hands-on labs + Invited Talks

Time	Speaker	Title	Materials
8:00am-8:40am	Prof. Onur Mutlu	Memory-Centric Computing	PDF (PDF) PPT (PPT)
8:40am-10:00am	Dr. Juan Gómez Luna	Processing-Near-Memory: Real PNM Architectures Programming General-purpose PIM	PDF (PDF) PPT (PPT)
10:20am-11:00am	Dr. Dimin Niu	A 3D Logic-to-DRAM Hybrid Bonding Process-Near-Memory Chip for Recommendation System	
11:00am-11:40am	Dr. Christina Giannoula	SparseP: Towards Efficient Sparse Matrix Vector Multiplication on Real Processing-In-Memory Architectures	PDF (PDF) PPT (PPT)
1:30pm-2:10pm	Dr. Juan Gómez Luna	Processing-Using-Memory: Exploiting the Analog Operational Properties of Memory Components	PDF (PDF) PPT (PPT)
2:10pm-2:50pm	Dr. Manuel Le Gallo	Deep Learning Inference Using Computational Phase-Change Memory	
2:50pm-3:30pm	Dr. Juan Gómez Luna	PIM Adoption Issues: How to Enable PIM Adoption?	PDF (PDF) PPT (PPT)
3:40pm-5:40pm	Dr. Juan Gómez Luna	Hands-on Lab: Programming and Understanding a Real Processing-in-Memory Architecture	Handout (Handout) PDF (PDF) PPT (PPT)

<https://www.youtube.com/watch?v=f5-nT1tbz5w>

<https://events.safari.ethz.ch/real-pim-tutorial/>

Real PIM Tutorial [MICRO 2023]

■ October 29: Lectures + Hands-on labs + Invited talks

Real-world Processing-in-Memory Systems for Modern Workloads

Tutorial Description

Processing-in-Memory (PIM) is a computing paradigm that aims at overcoming the data movement bottleneck (i.e., the waste of execution cycles and energy resulting from the back-and-forth data movement between memory units and compute units) by making memory compute-capable.

Explored over several decades since the 1960s, PIM systems are becoming a reality with the advent of the first commercial products and prototypes.

A number of startups (e.g., UPMEM, Neuroblade) are already commercializing real PIM hardware, each with its own design approach and target applications. Several major vendors (e.g., Samsung, SK Hynix, Alibaba) have presented real PIM chip prototypes in the last two years. Most of these architectures have in common that they place compute units near the memory arrays. This type of PIM is called processing near memory (PNM).

2,560-DPU Processing-in-Memory System

PIM can provide large improvements in both performance and energy consumption for many modern applications, thereby enabling a commercially viable way of dealing with huge amounts of data that is bottlenecking our computing systems. Yet, it is critical to (1) study and understand the characteristics that make a workload suitable for a PIM architecture, (2) propose optimization strategies for PIM kernels, and (3) develop programming frameworks and tools that can lower the learning curve and ease the adoption of PIM.

This tutorial focuses on the latest advances in PIM technology, workload characterization for PIM, and programming and optimizing PIM kernels. We will (1) provide an introduction to PIM and taxonomy of PIM systems, (2) give an overview and a rigorous analysis of existing real-world PIM hardware, (3) conduct hand-on labs about important workloads (machine learning, sparse linear algebra, bioinformatics, etc.) using real PIM systems, and (4) shed light on how to improve future PIM systems for such workloads.

2,560-DPU Processing-in-Memory System

<https://arxiv.org/pdf/2105.03814.pdf>

MICRO 2023 Tutorial: Real-world Processing-in-Memory Systems for Modern Workloads

Onur Mutlu Lectures
34.6K subscribers

<https://www.youtube.com/live/ohUooNSIxOI>

<https://events.safari.ethz.ch/micro-pim-tutorial>

PIM Tutorial at HEART 2024

HEART 2024 Memory-Centric Computing Systems Tutorial

Friday, June 21, Porto, Portugal

Organizers: Geraldo F. Oliveira, Dr. Mohammad Sadrosadati, Ataberk Olgun, Professor Onur Mutlu

Program: <https://events.safari.ethz.ch/heart24-memorycentric-tutorial/>

Overview of PIM | PIM taxonomy
PIM in memory & storage
Real-world PNM systems
PUM for bulk bitwise operations
Programming techniques & tools
Infrastructures for PIM Research
Research challenges & opportunities



PIM Tutorial at ISCA 2024

ISCA 2024 Memory-Centric Computing Systems Tutorial

Saturday, June 29, Buenos Aires, Argentina

Organizers: Geraldo F. Oliveira, Dr. Mohammad Sadrosadati, Ataberk Olgun, Professor Onur Mutlu

Program: <https://events.safari.ethz.ch/isca24-memorycentric-tutorial/>

Overview of PIM | PIM taxonomy
PIM in memory & storage
Real-world PNM systems
PUM for bulk bitwise operations
Programming techniques & tools
Infrastructures for PIM Research
Research challenges & opportunities



Tutorial at MICRO 2024

MICRO 2024 - Tutorial on Memory-Centric Computing Systems

Saturday, November 2nd, Austin, Texas, USA

Organizers: Geraldo F. Oliveira, Dr. Mohammad Sadrosadati, Ataberk Olgun, Professor Onur Mutlu

Program: <https://events.safari.ethz.ch/micro24-memorycentric-tutorial/>

Overview of PIM | PIM taxonomy
PIM in memory & storage
Real-world PNM systems
PUM for bulk bitwise operations
Programming techniques & tools
Infrastructures for PIM Research
Research challenges & opportunities



PIM Tutorial at PPOPP 2025

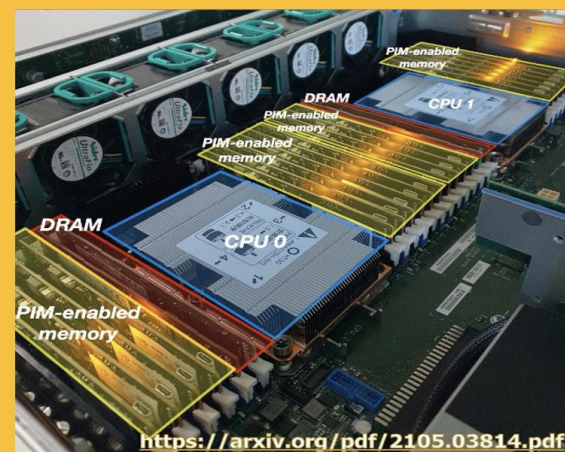
PPoPP 2025 - Tutorial on Memory-Centric Computing Systems

March 1st – March 5th, Las Vegas, Nevada, USA

Organizers: Geraldo F. Oliveira, Dr. Mohammad Sadrosadati, Ataberk Olgun, Professor Onur Mutlu

Program: <https://events.safari.ethz.ch/ppopp25-memorycentric-tutorial/>

Overview of PIM | PIM taxonomy
PIM in memory & storage
Real-world PNM systems
PUM for bulk bitwise operations
Programming techniques & tools
Infrastructures for PIM Research
Research challenges & opportunities



Referenced Papers, Talks, Artifacts

- All are available at

<https://people.inf.ethz.ch/omutlu/projects.htm>

<https://www.youtube.com/onurmutlulectures>

<https://github.com/CMU-SAFARI/>

Open Source Tools: SAFARI GitHub



SAFARI Research Group at ETH Zurich and Carnegie Mellon University


Site for source code and tools distribution from SAFARI Research Group at ETH Zurich and Carnegie Mellon University.

👤 241 followers 📍 ETH Zurich and Carnegie Mellon U... 🔗 <https://safari.ethz.ch/> ✉ omutlu@gmail.com

🏠 Overview 📁 Repositories 80 📁 Projects 📁 Packages 👤 People 13


Pinned

Customize pins

 **ramulator** Public ⋮

A Fast and Extensible DRAM Simulator, with built-in support for modeling many different DRAM technologies including DDRx, LPDDRx, GDDRx, WIOx, HBMx, and various academic proposals. Described in the...

● C++ ☆ 442 🍷 195

 **prim-benchmarks** Public ⋮


PrIM (Processing-In-Memory benchmarks) is the first benchmark suite for a real-world processing-in-memory (PIM) architecture. PrIM is developed to evaluate, analyze, and characterize the first publ...

● C ☆ 100 🍷 38

 **MQSim** Public ⋮


MQSim is a fast and accurate simulator modeling the performance of modern multi-queue (MQ) SSDs as well as traditional SATA based SSDs. MQSim faithfully models new high-bandwidth protocol implement...

● C++ ☆ 213 🍷 120

 **rowhammer** Public ⋮

Source code for testing the Row Hammer error mechanism in DRAM devices. Described in the ISCA 2014 paper by Kim et al. at http://users.ece.cmu.edu/~omutlu/pub/dram-row-hammer_isca14.pdf.

● C ☆ 208 🍷 41

 **SoftMC** Public ⋮

SoftMC is an experimental FPGA-based memory controller design that can be used to develop tests for DDR3 SODIMMs using a C++ based API. The design, the interface, and its capabilities and limitatio...

● Verilog ☆ 104 🍷 26

 **Pythia** Public ⋮

A customizable hardware prefetching framework using online reinforcement learning as described in the MICRO 2021 paper by Bera et al. (<https://arxiv.org/pdf/2109.12021.pdf>).

● C++ ☆ 85 🍷 25

1st Workshop

Memory-Centric Computing: Research Challenges & Closing Remarks

Geraldo F. Oliveira

<https://geraldofojunior.github.io>

ASPLOS 2025

30 March 2025