ARIES: An Agile MLIR-Based Compilation Flow for Reconfigurable Devices with AI Engines

Jinming Zhuang*, Shaojie Xiang*†, Hongzheng Chen†, Niansong Zhang†, ZhuopingYang,
Tony Mao†, Zhiru Zhang† and Peipei Zhou

FPGA'25
Brown University; Cornell University†
Equal Contribution*

jinming_zhuang@brown.edu
peipei_zhou@brown.edu

https://peipeizhou-eecs.github.io/
https://github.com/arc-research-lab/Aries

BROWN

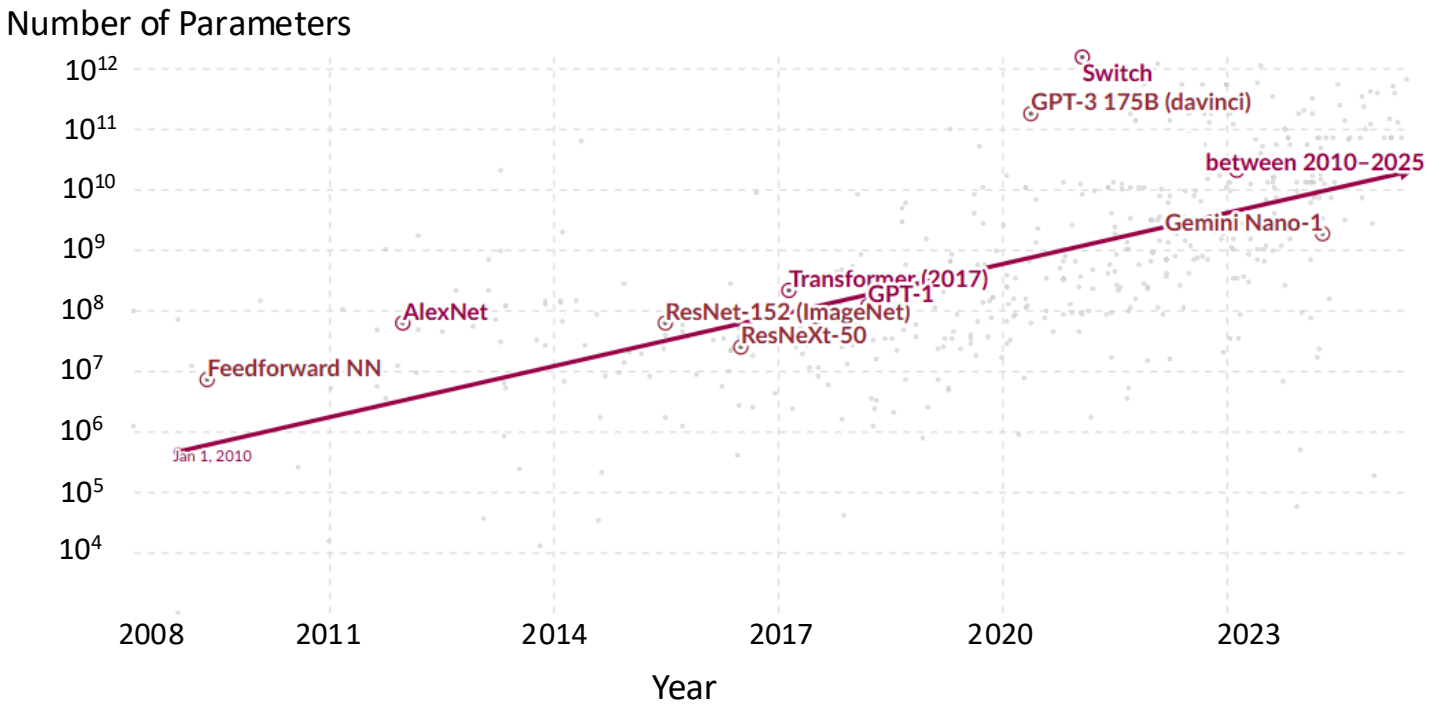
Cornell University

Heterogenous Architectures for Continuous Compute Scaling

Epoch (2024) – with major processing by Our World in Data. “Exponential growth of parameters in notable AI systems” [dataset]. Epoch, “Parameter, Compute and Data Trends in Machine Learning” [original data]. Retrieved February 19, 2025 from https://ourworldindata.org/grapher/exponential-growth-of-parameters-in-notable-ai-systems

Heterogenous Architectures for Continuous Compute Scaling

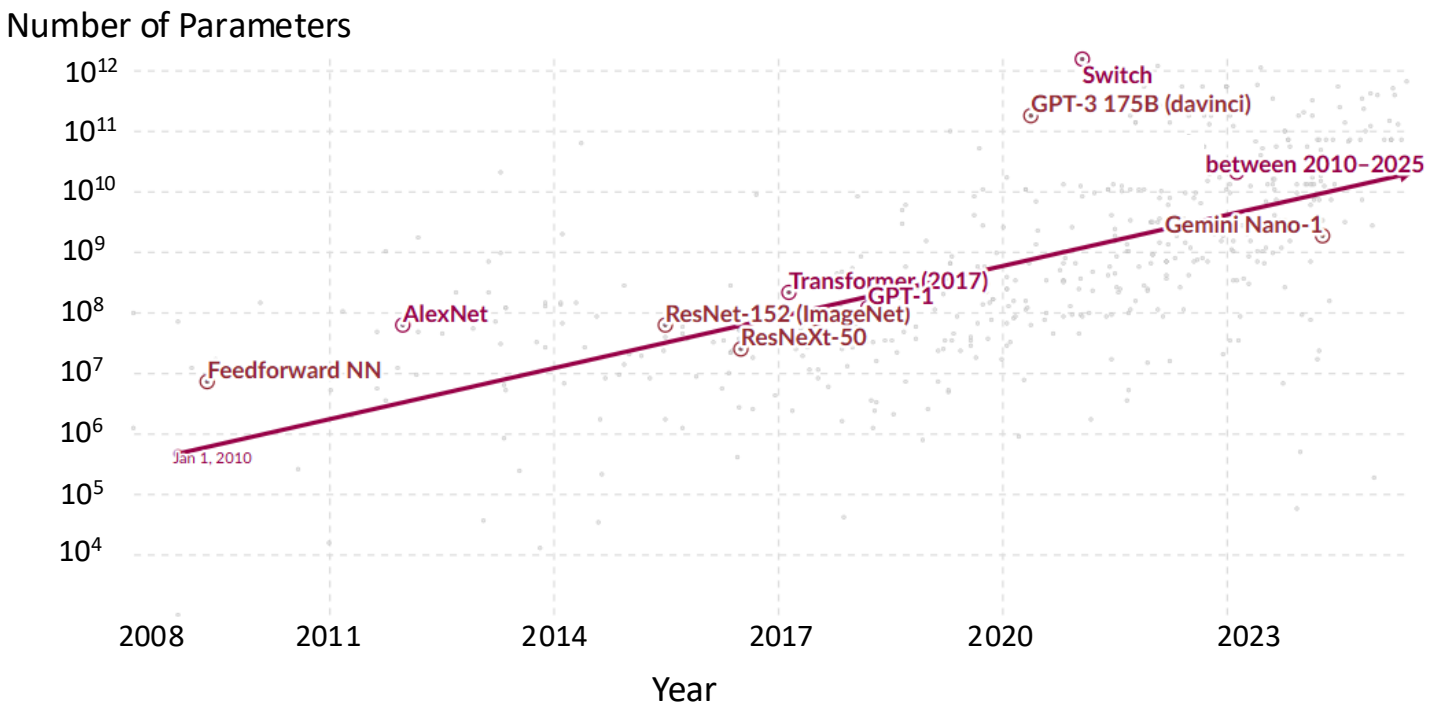
Parameter Size of AI Models Increases Exponentially



Epoch (2024) – with major processing by Our World in Data. “Exponential growth of parameters in notable AI systems” [dataset]. Epoch, “Parameter, Compute and Data Trends in Machine Learning” [original data]. Retrieved February 19, 2025 from https://ourworldindata.org/grapher/exponential-growth-of-parameters-in-notable-ai-systems

Heterogenous Architectures for Continuous Compute Scaling

Parameter Size of AI Models Increases Exponentially



Devices with AIE-V1

VCK5000

VCK190

Devices with AIE-ML

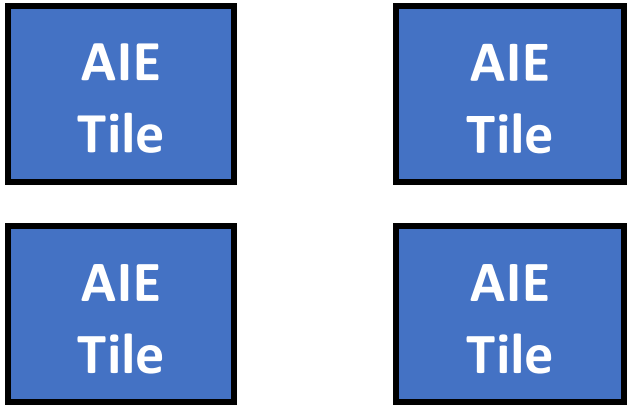
Ryzen AI CPUs

VEK280

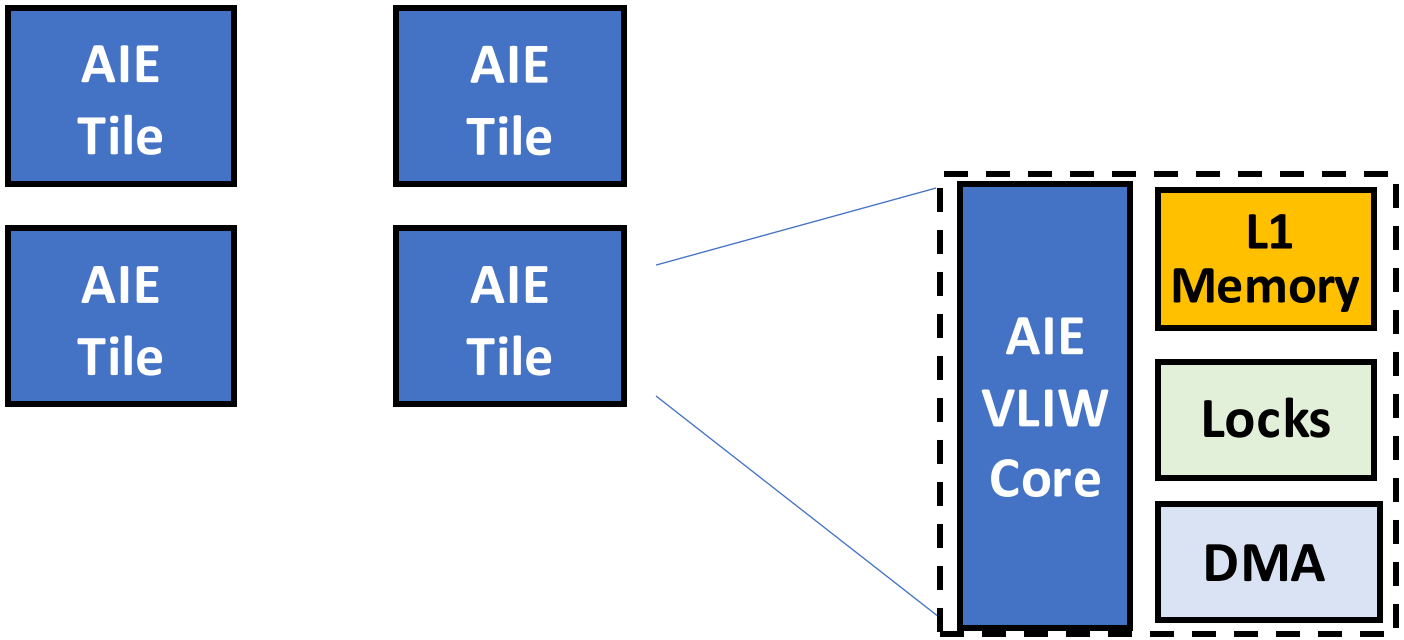
Epoch (2024) – with major processing by Our World in Data. "Exponential growth of parameters in notable AI systems" [dataset]. Epoch, "Parameter, Compute and Data Trends in Machine Learning" [original data]. Retrieved February 19, 2025 from https://ourworldindata.org/grapher/exponential-growth-of-parameters-in-notable-ai-systems

Reconfigurable Devices with AI Engines

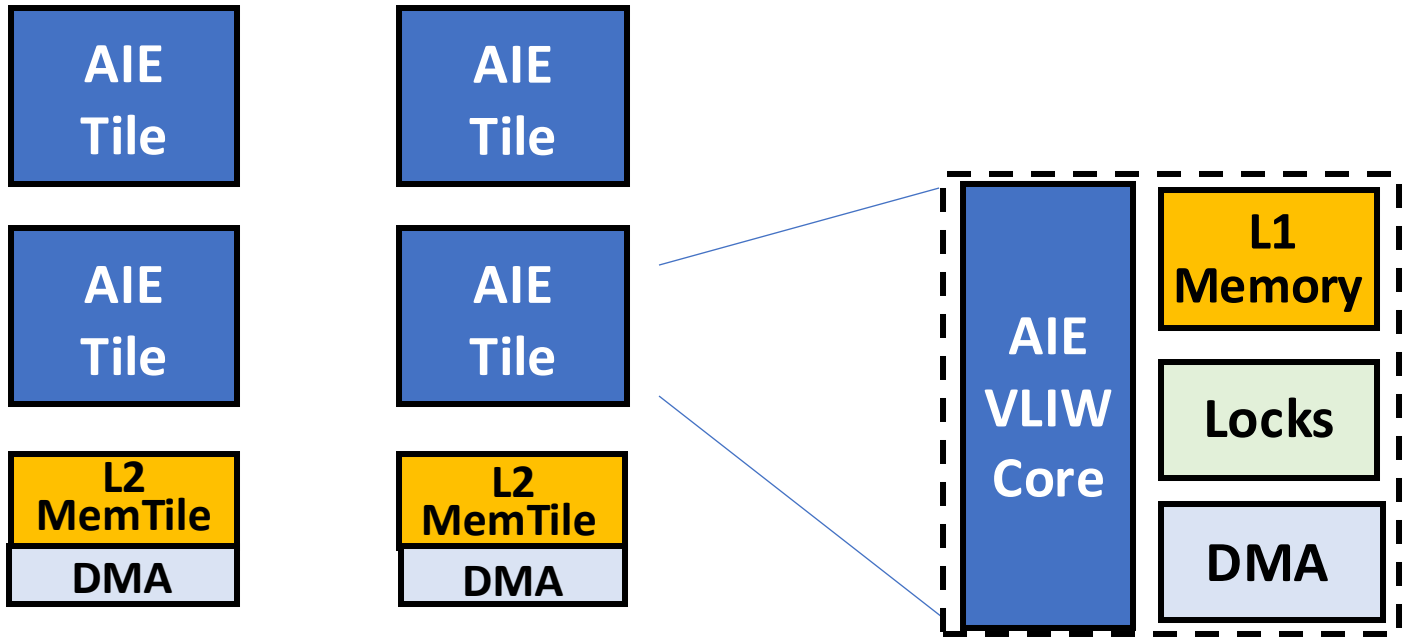
Reconfigurable Devices with AI Engines

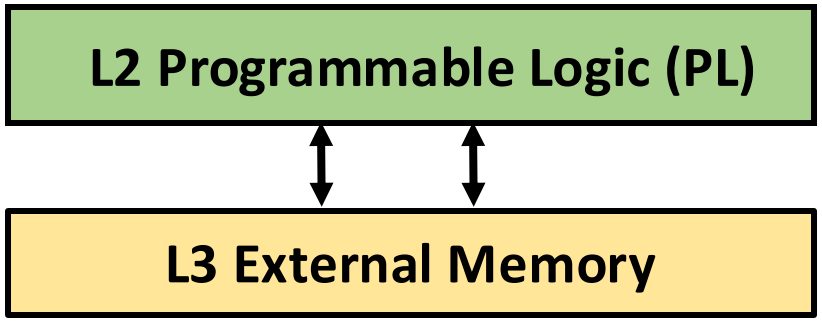


Reconfigurable Devices with AI Engines

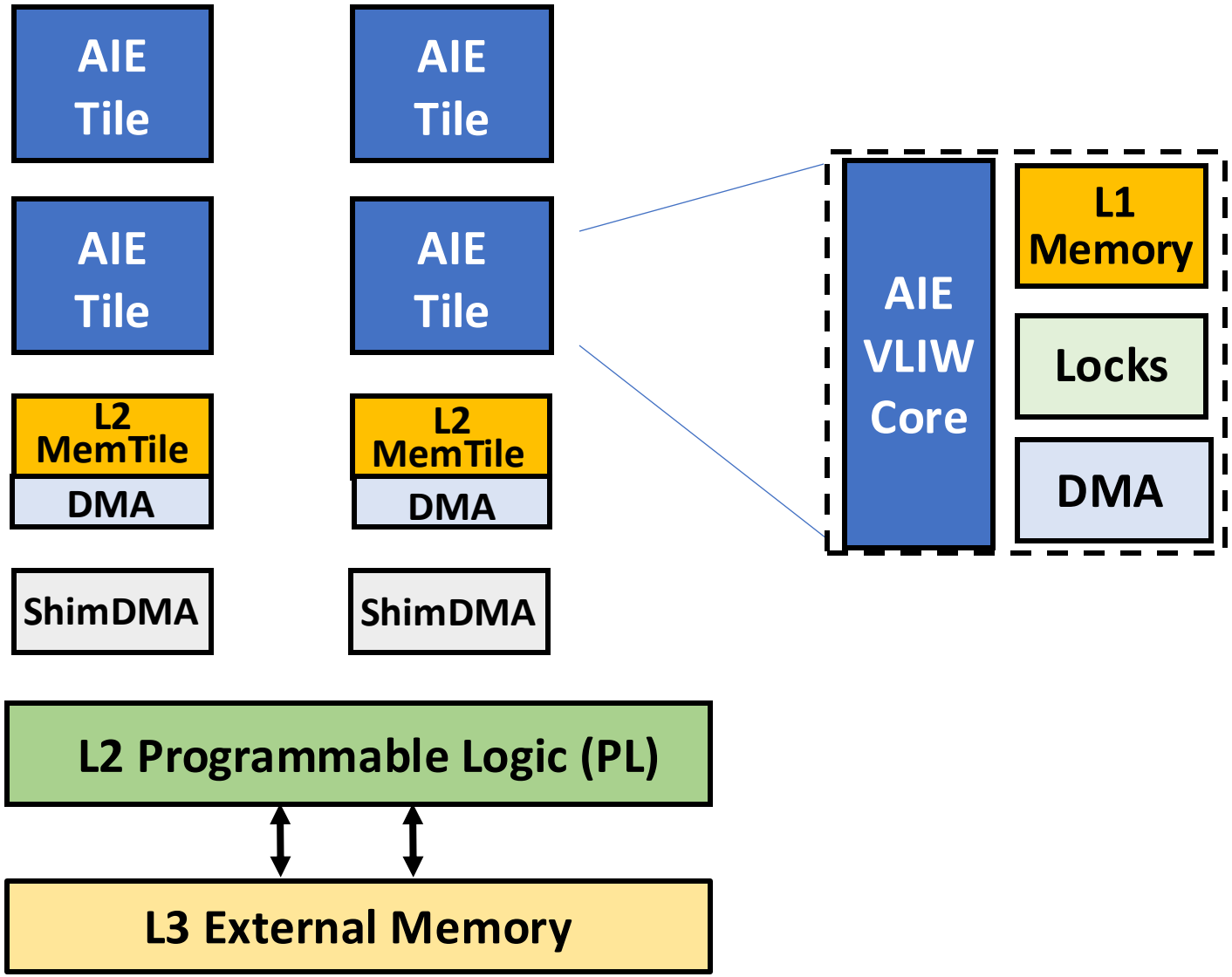


Reconfigurable Devices with AI Engines

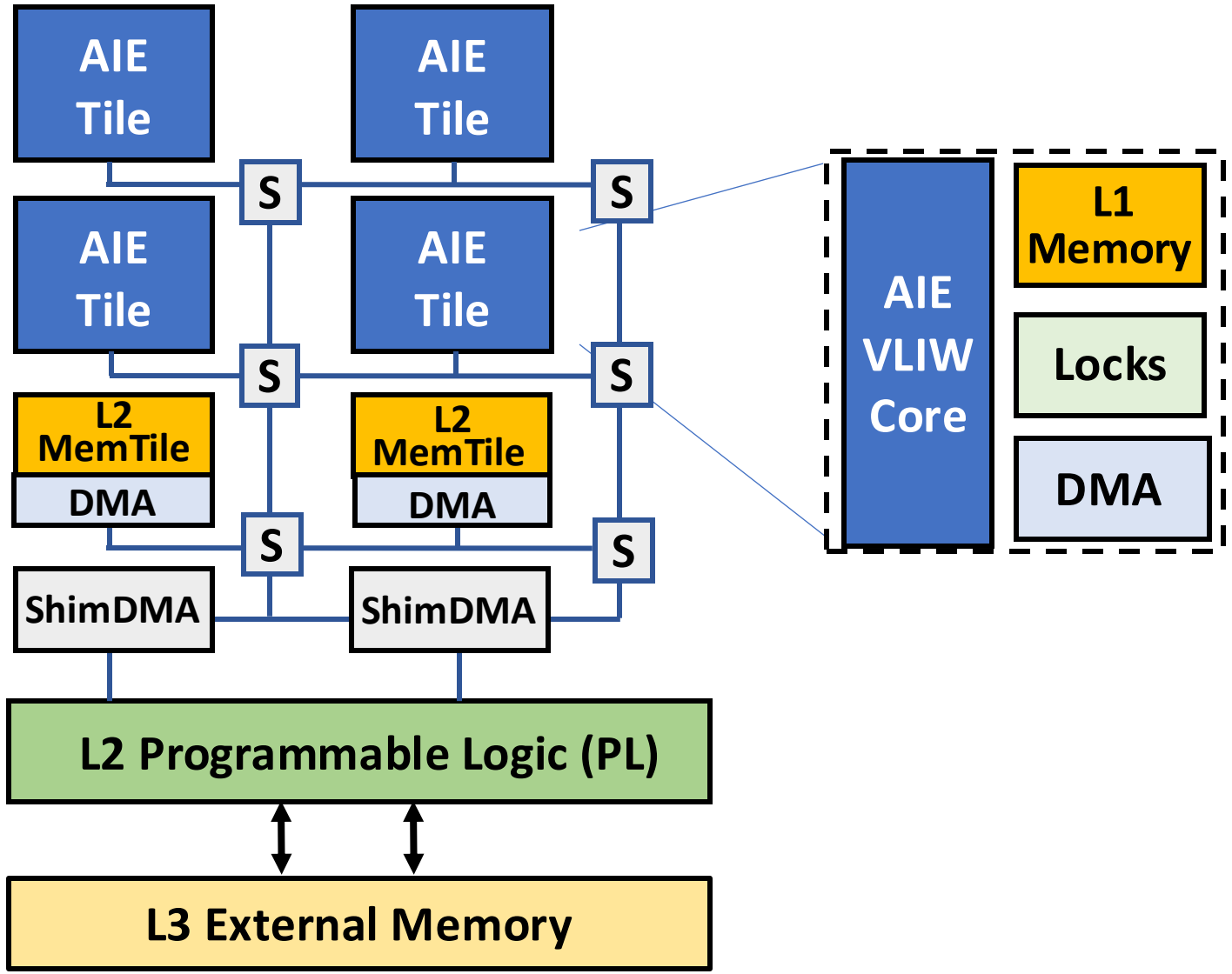




Reconfigurable Devices with AI Engines

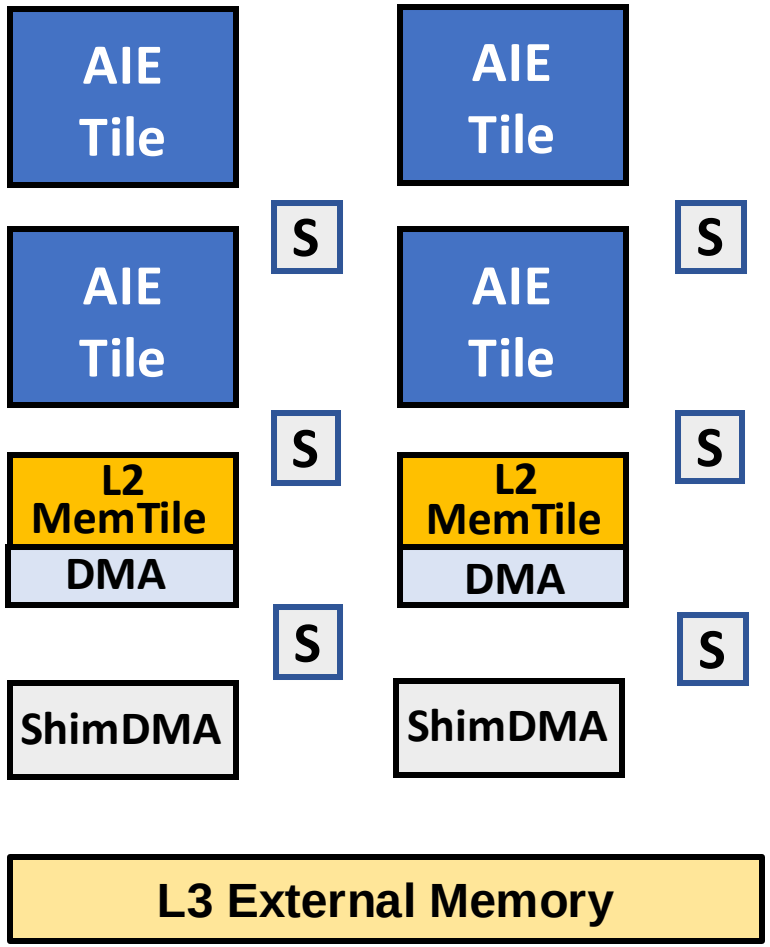


Reconfigurable Devices with AI Engines



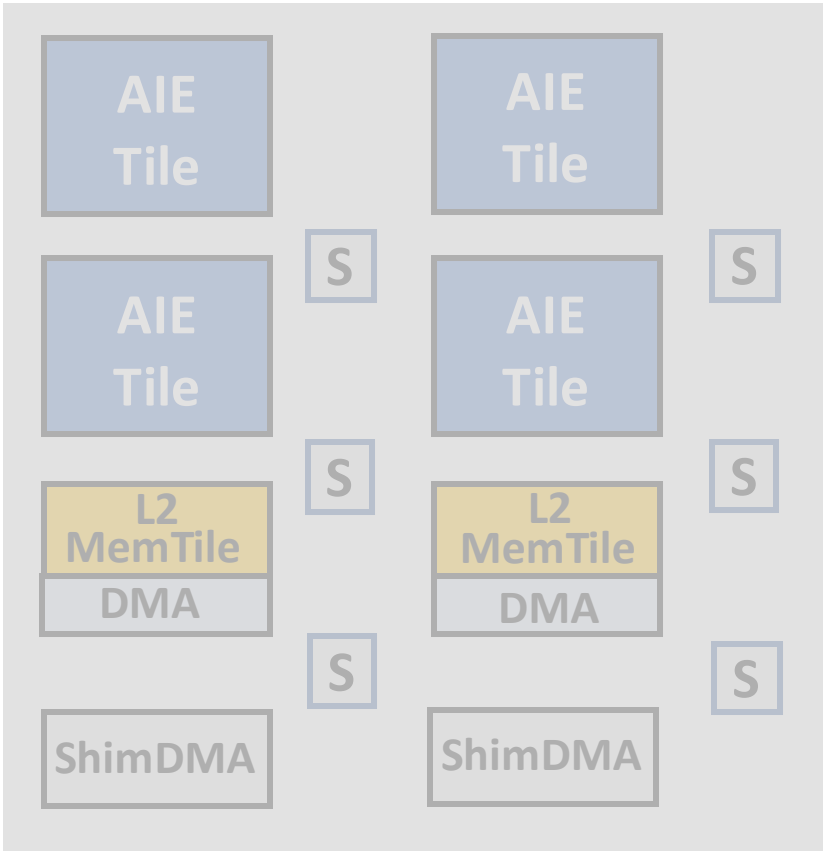
Challenge 1

- Fragmented abstraction for parallelism & data movement



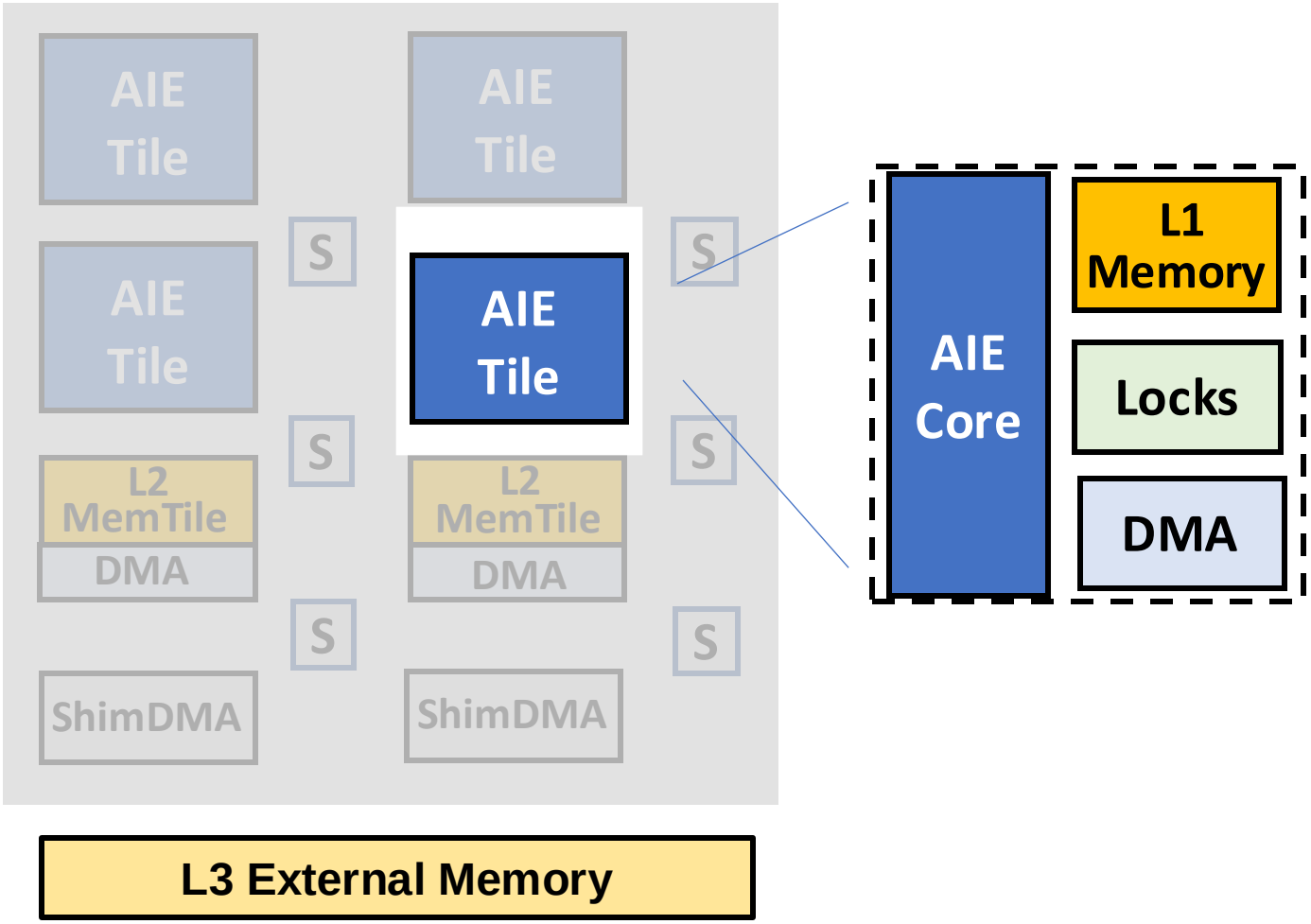
Challenge 1

- Fragmented abstraction for parallelism & data movement



Challenge 1

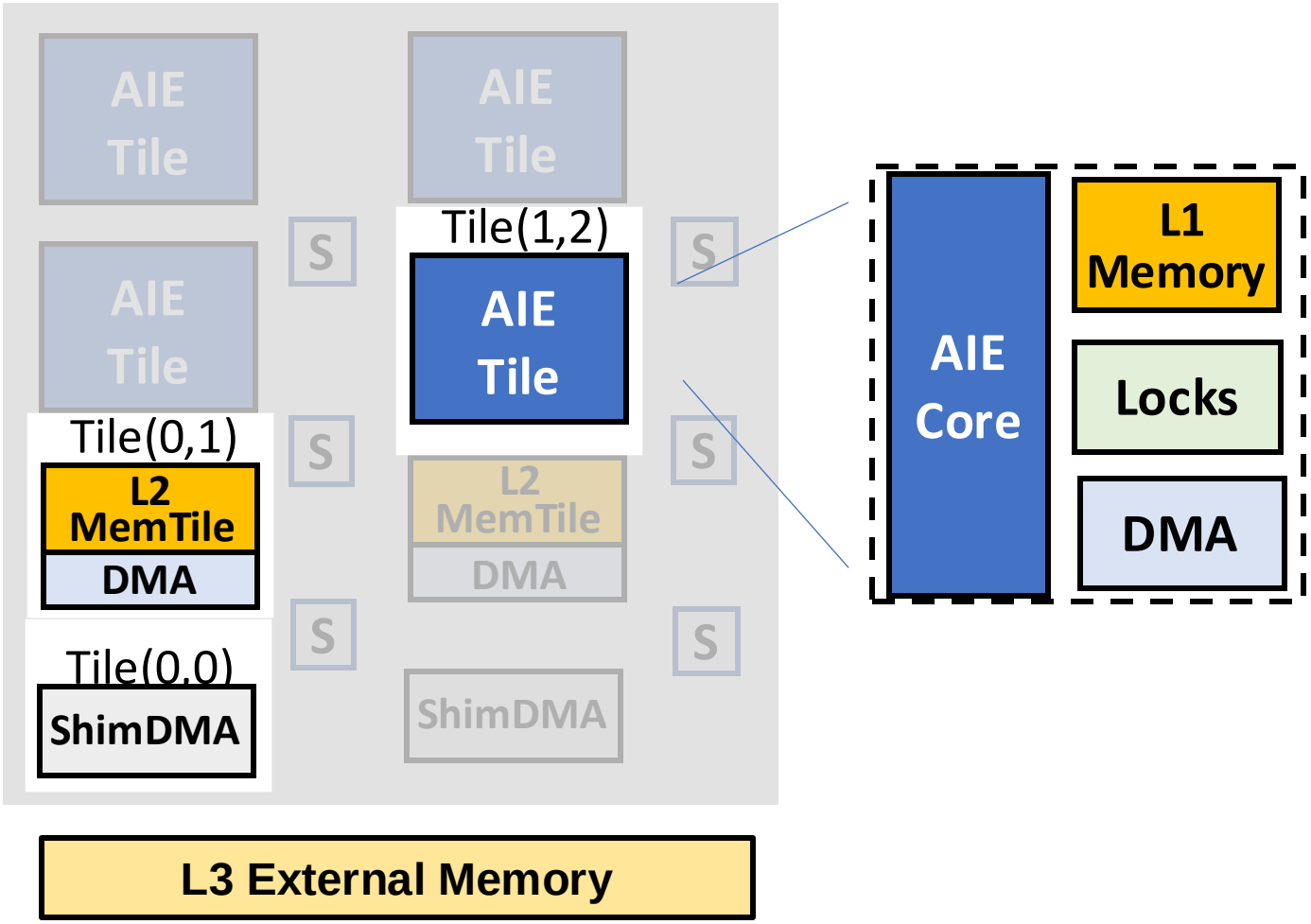
- Fragmented abstraction for parallelism & data movement



- Assign the workload to an AIE

Challenge 1

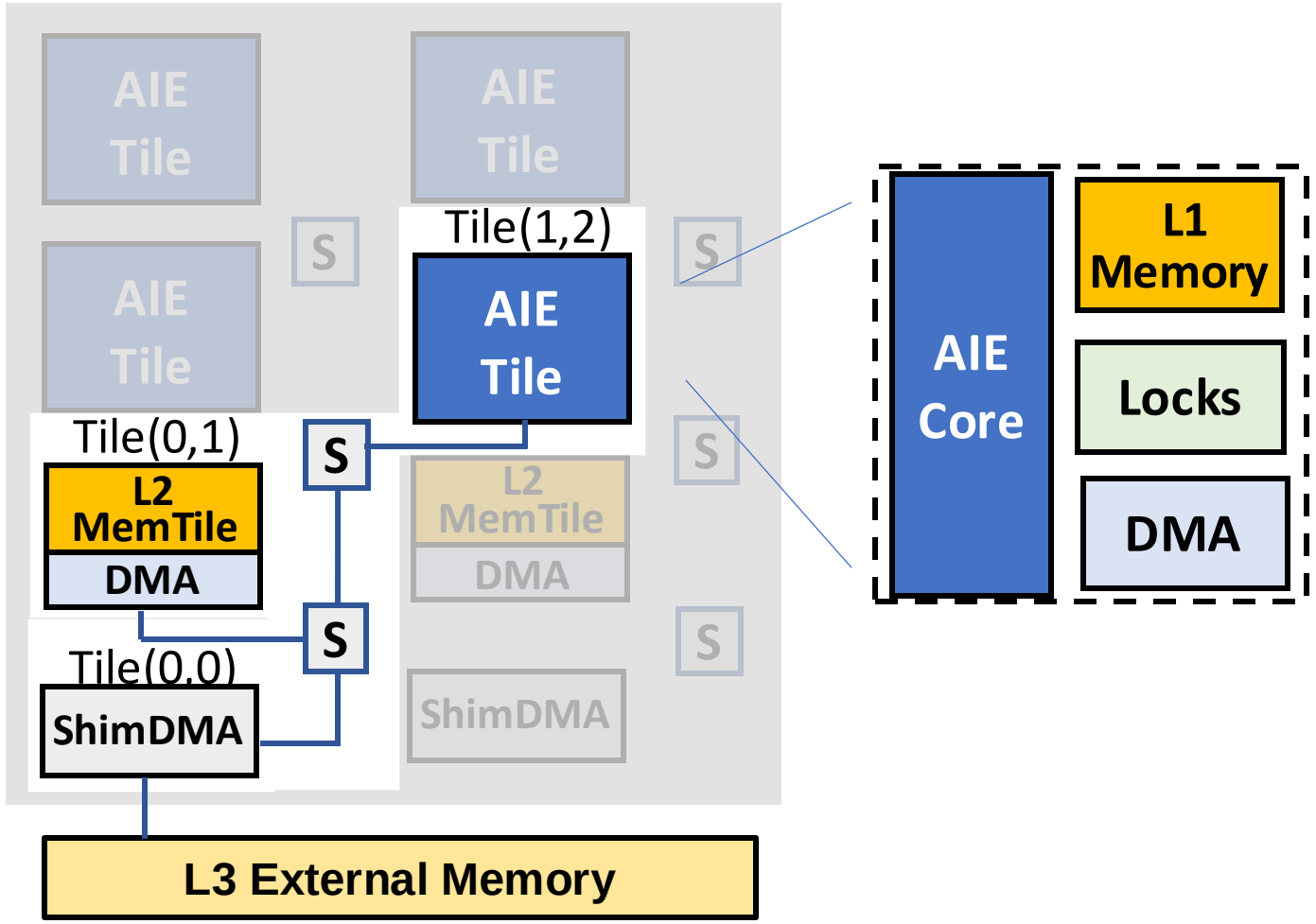
- Fragmented abstraction for parallelism & data movement



- Assign the workload to an AIE
- Specify location of each tile

Challenge 1

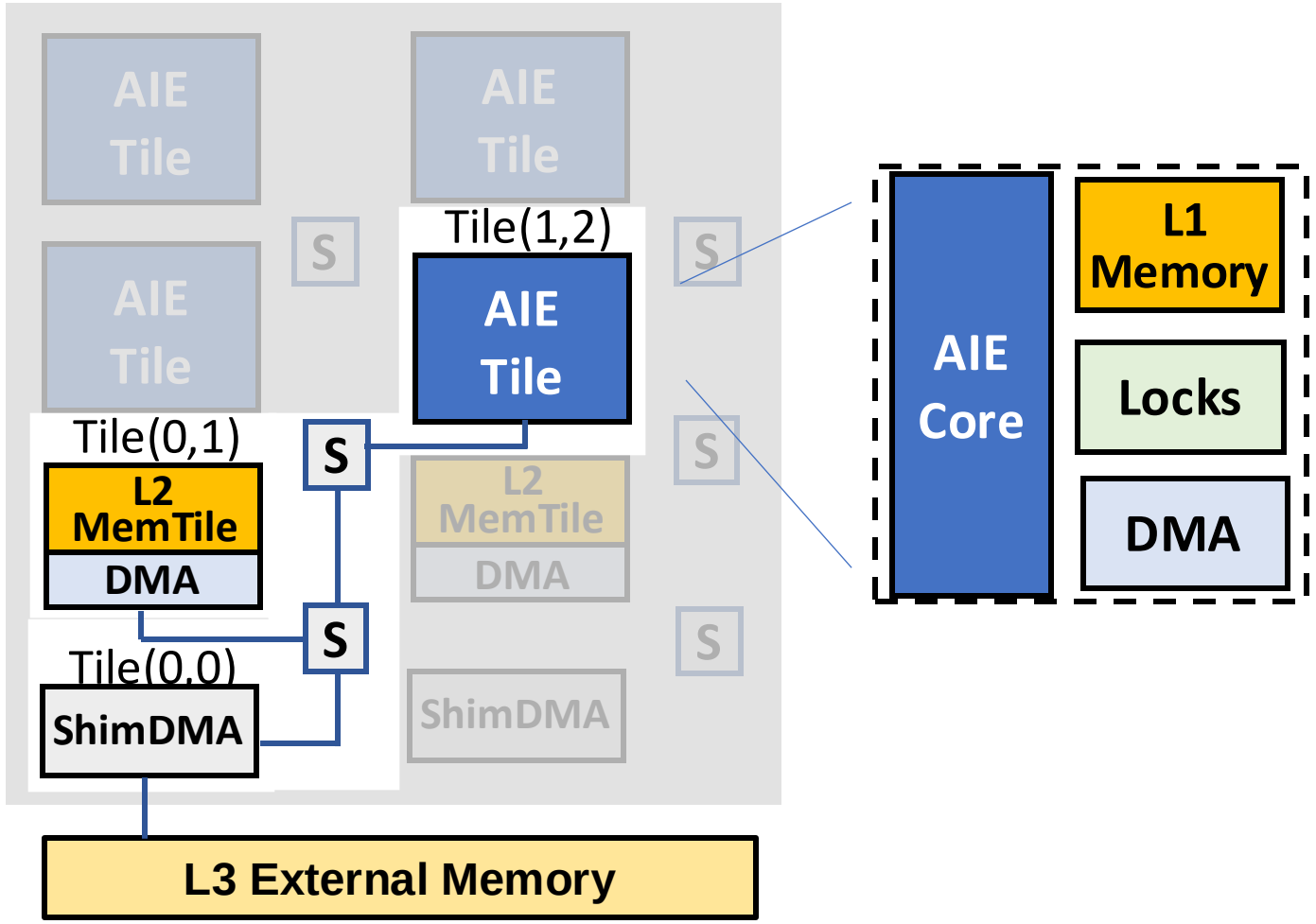
- Fragmented abstraction for parallelism & data movement



- Assign the workload to an AIE
- Specify location of each tile
- Define connections of the tiles

Challenge 1

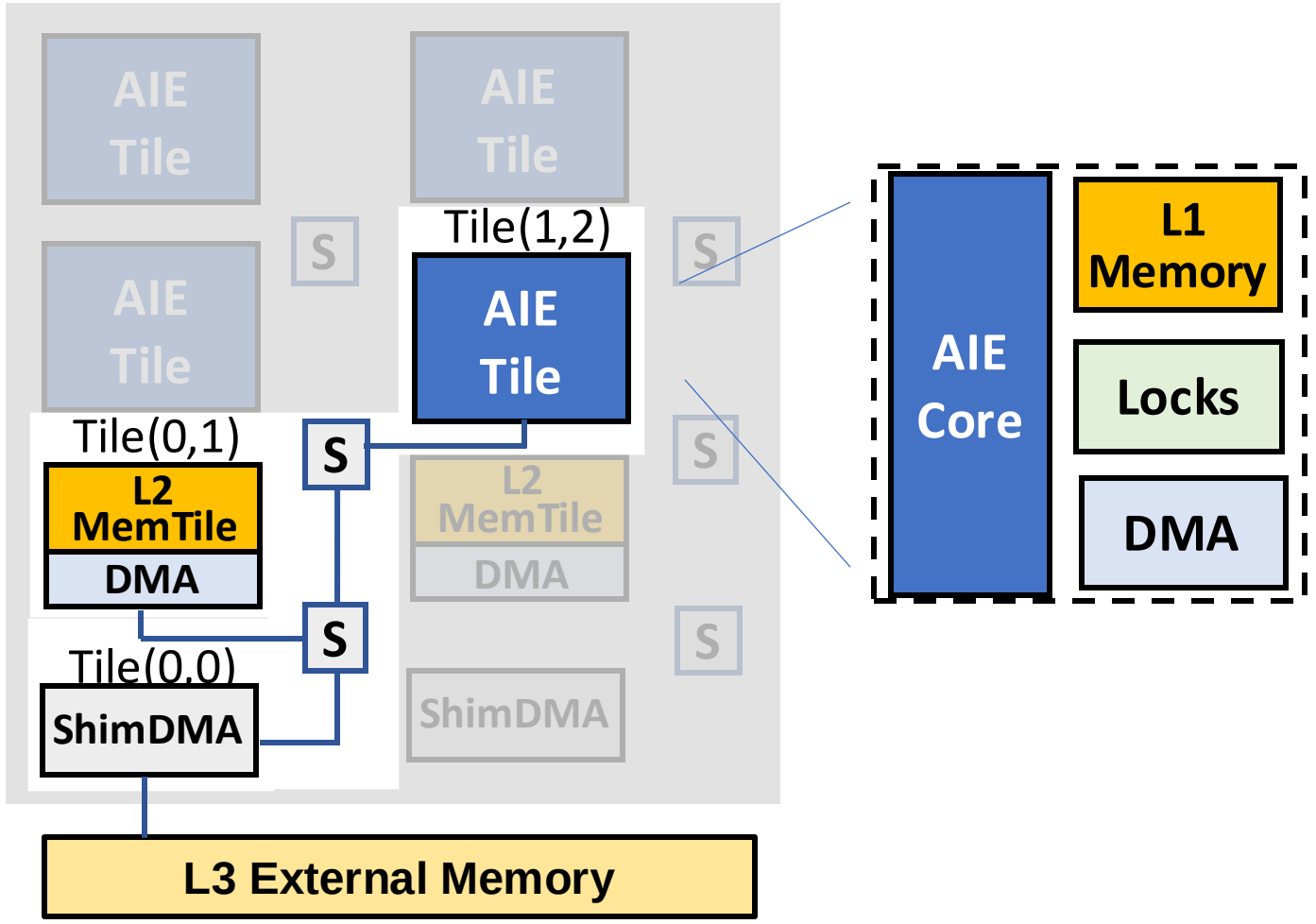
- Fragmented abstraction for parallelism & data movement



- Assign the workload to an AIE
- Specify location of each tile
- Define connections of the tiles
- L2 → L1 DMA instructions
- L3 → L2 DMA instructions

Challenge 1

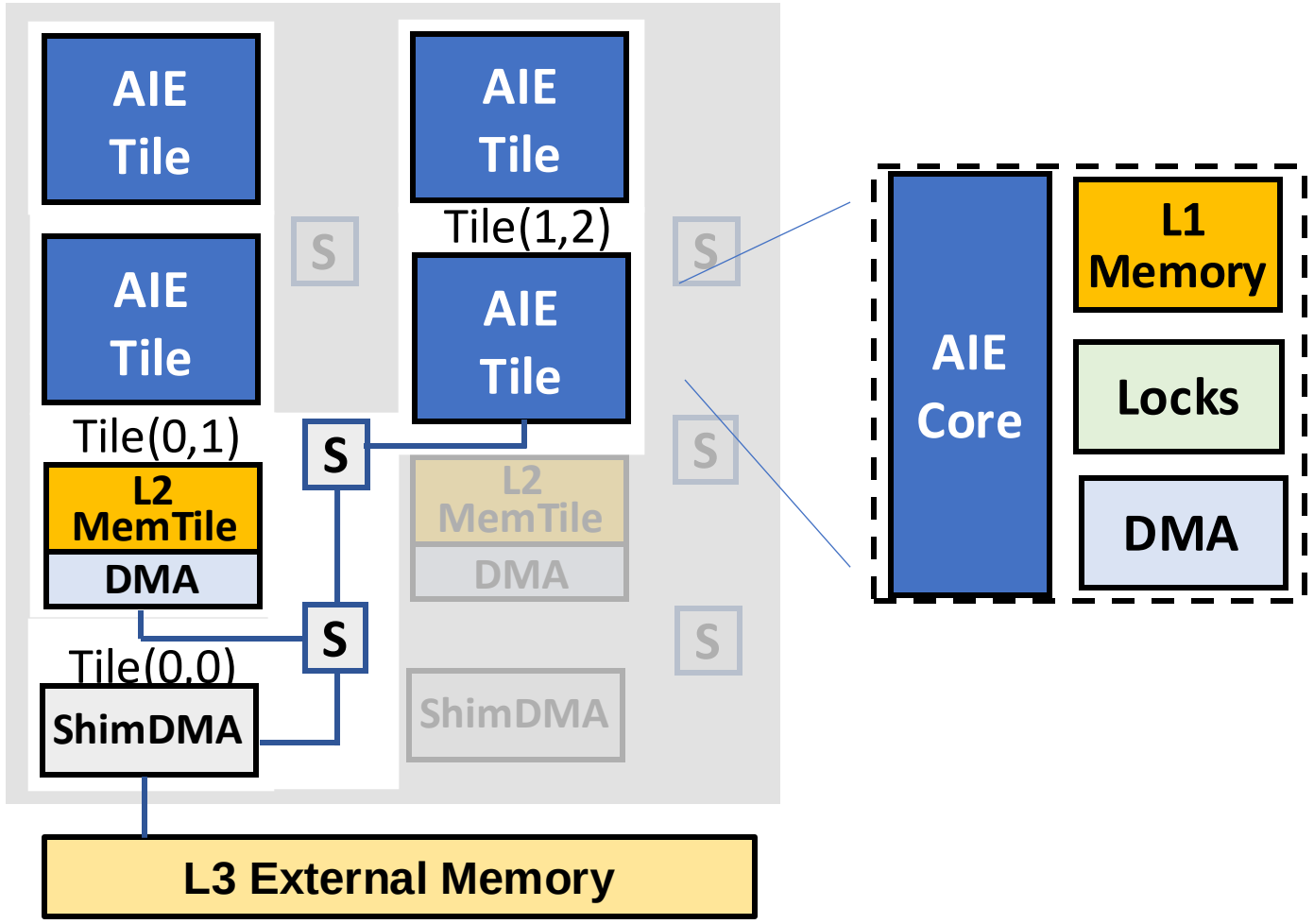
- Fragmented abstraction for parallelism & data movement



- Assign the workload to an AIE
- Specify location of each tile
- Define connections of the tiles
- L2 → L1 DMA instructions
- L3 → L2 DMA instructions
- L1 memory lock acquire/release

Challenge 1

- Fragmented abstraction for parallelism & data movement

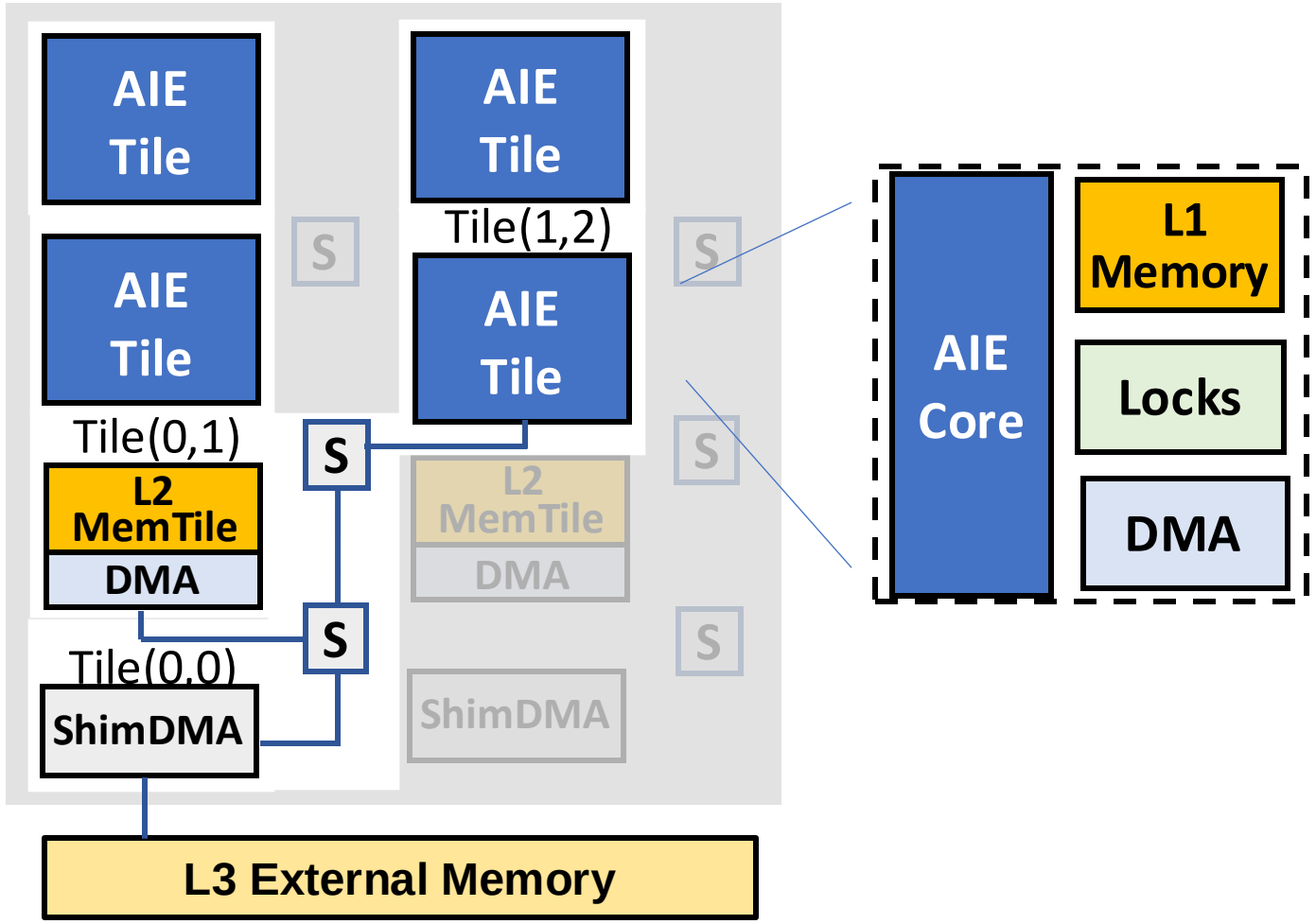


- Assign the workload to an AIE
- Specify location of each tile
- Define connections of the tiles
- L2 → L1 DMA instructions
- L3 → L2 DMA instructions
- L1 memory lock acquire/release
- Scale out to multi AIEs

Challenge 1

- Fragmented abstraction for parallelism & data movement

A simplified abstraction is highly needed to improve productivity



- Assign the workload to an AIE
- Specify location of each tile
- Define connections of the tiles
- L2 → L1 DMA instructions
- L3 → L2 DMA instructions
- L1 memory lock acquire/release
- Scale out to multi AIEs

Challenge 2

- Lack of a unified representation for the entire heterogenous systems

[1] AMD. MLIR-AIE: An MLIR-based AI Engine toolchain. https://xilinx.github.io/mlir-aie/

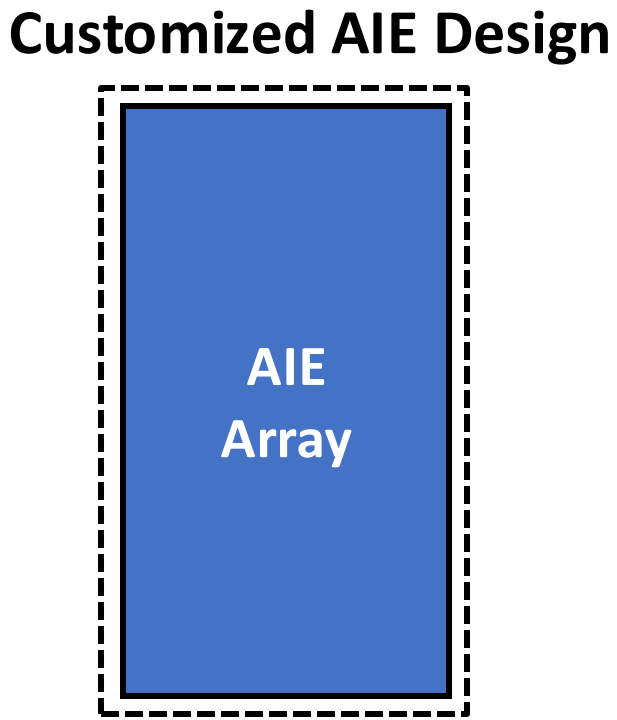
Challenge 2

- **Lack of a unified representation for the entire heterogenous systems**
  - MLIR-AIE[1] Flow

[1] AMD. MLIR-AIE: An MLIR-based AI Engine toolchain. https://xilinx.github.io/mlir-aie/

Challenge 2

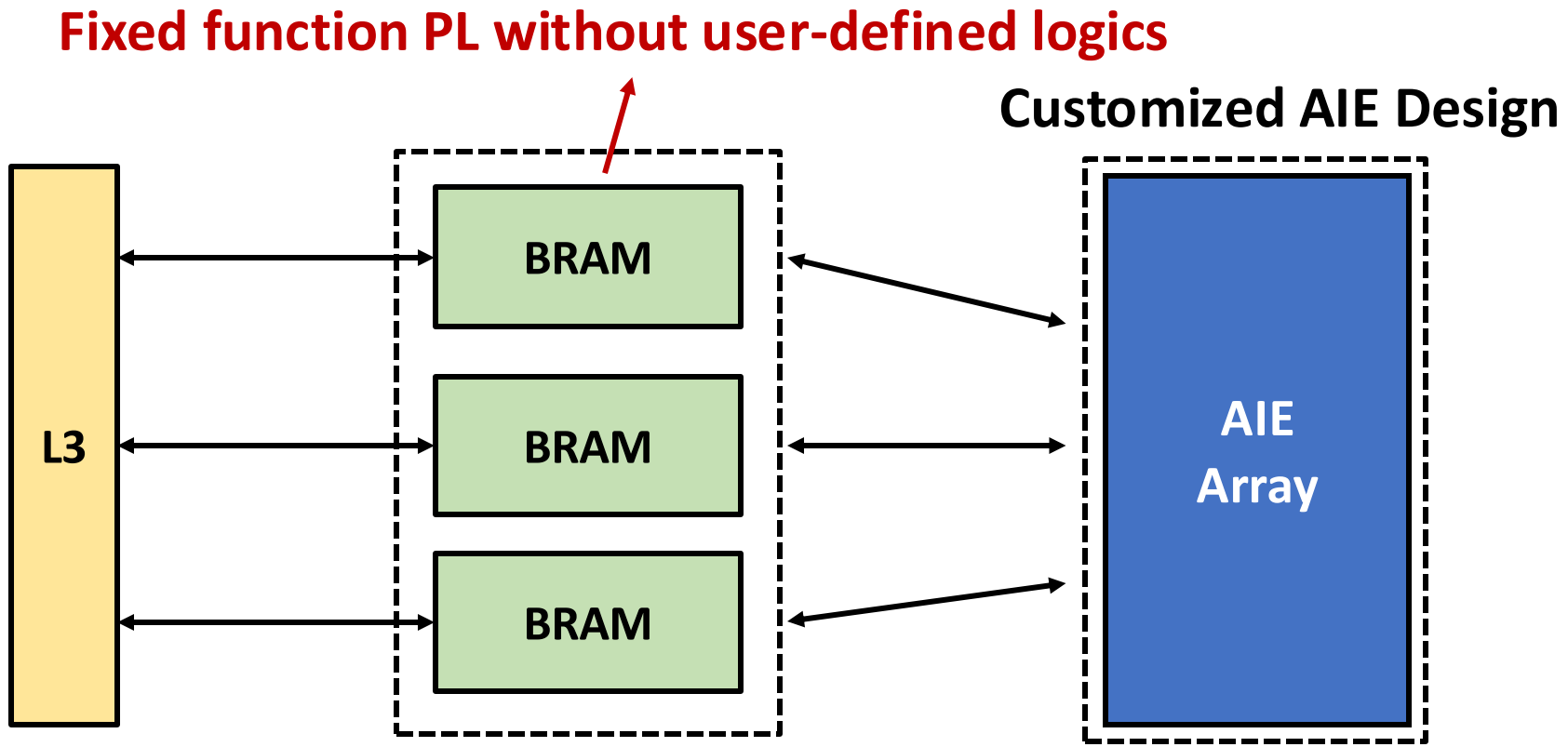
- **Lack of a unified representation for the entire heterogenous systems**
  - MLIR-AIE[1] Flow



[1] AMD. MLIR-AIE: An MLIR-based AI Engine toolchain. https://xilinx.github.io/mlir-aie/

Challenge 2

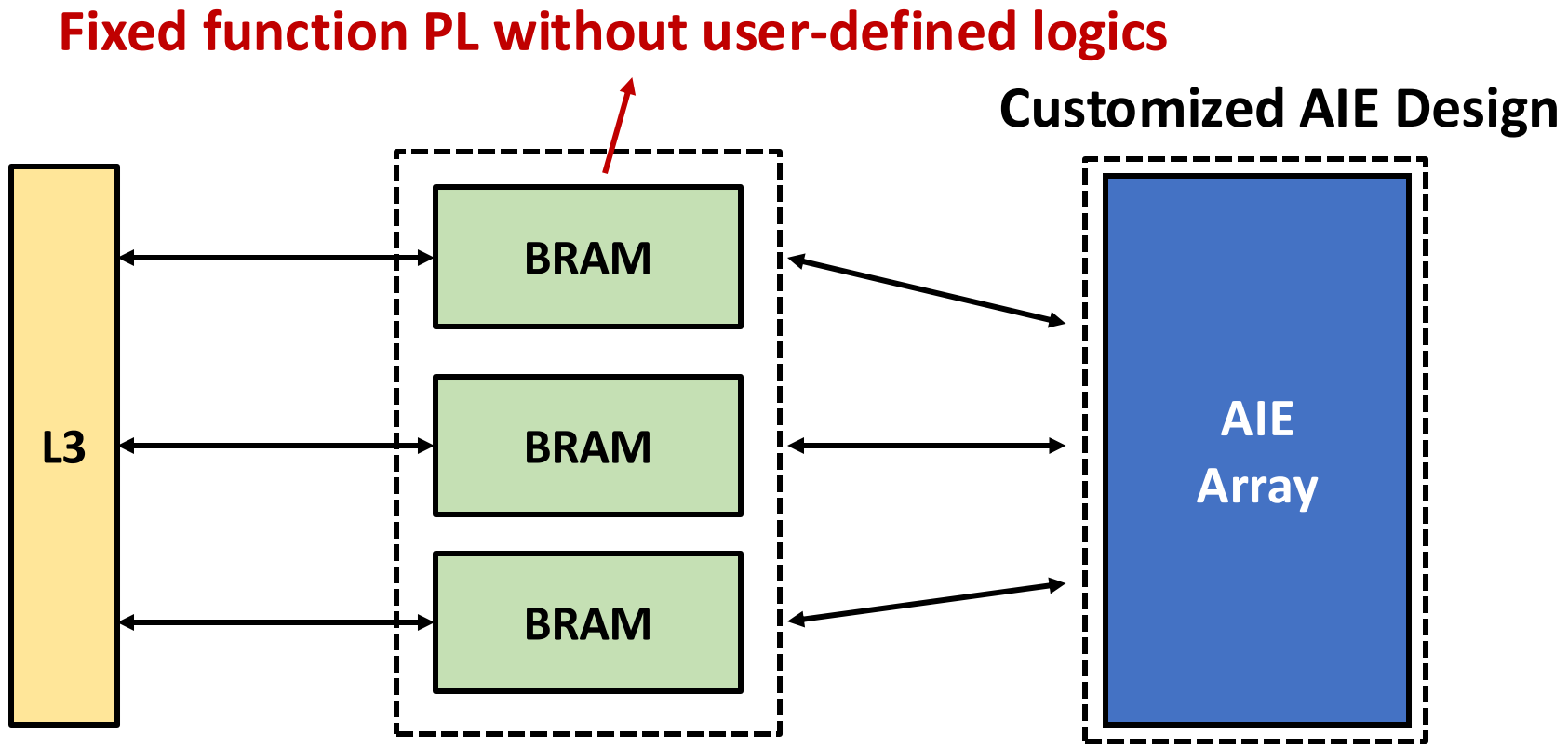
- Lack of a unified representation for the entire heterogenous systems
  - MLIR-AIE[1] Flow



[1] AMD. MLIR-AIE: An MLIR-based AI Engine toolchain. https://xilinx.github.io/mlir-aie/

Challenge 2

- **Lack of a unified representation for the entire heterogenous systems**
  - MLIR-AIE[1] Flow **Customizations on FPGA cannot be explored**



[1] AMD. MLIR-AIE: An MLIR-based AI Engine toolchain. https://xilinx.github.io/mlir-aie/

Challenge 2

- Lack of a unified representation for the entire heterogenous systems

Challenge 2

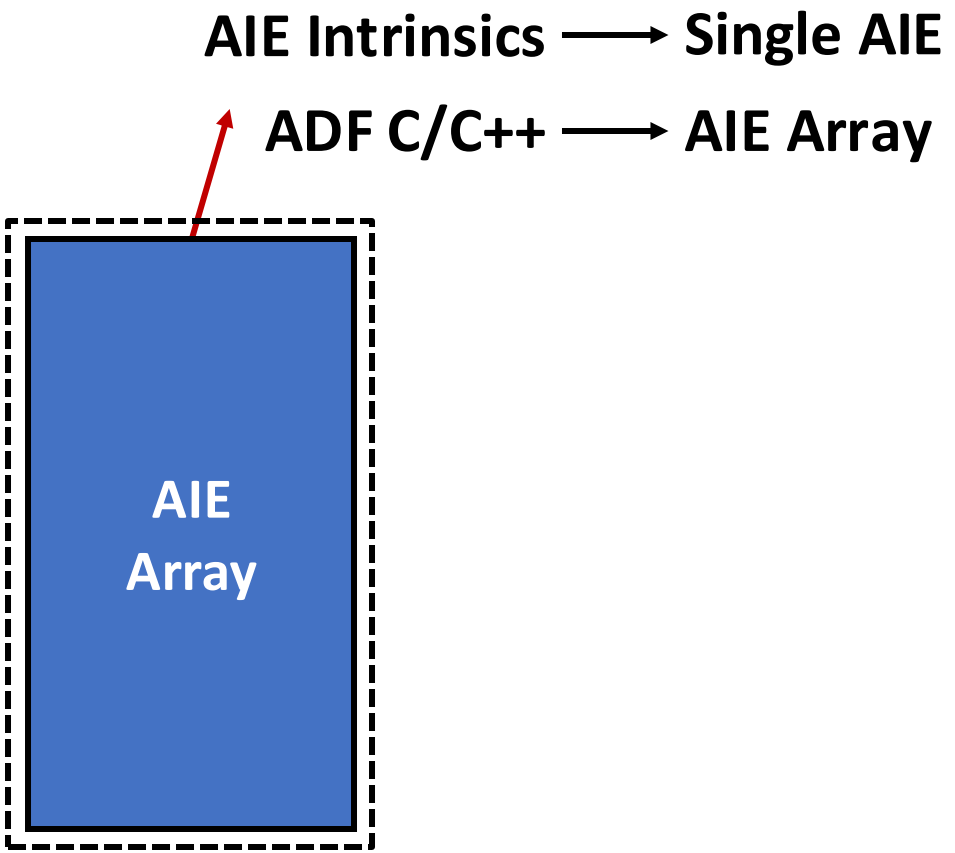
- **Lack of a unified representation for the entire heterogenous systems**
  - Vitis Flow

Challenge 2

- **Lack of a unified representation for the entire heterogenous systems**
  - Vitis Flow

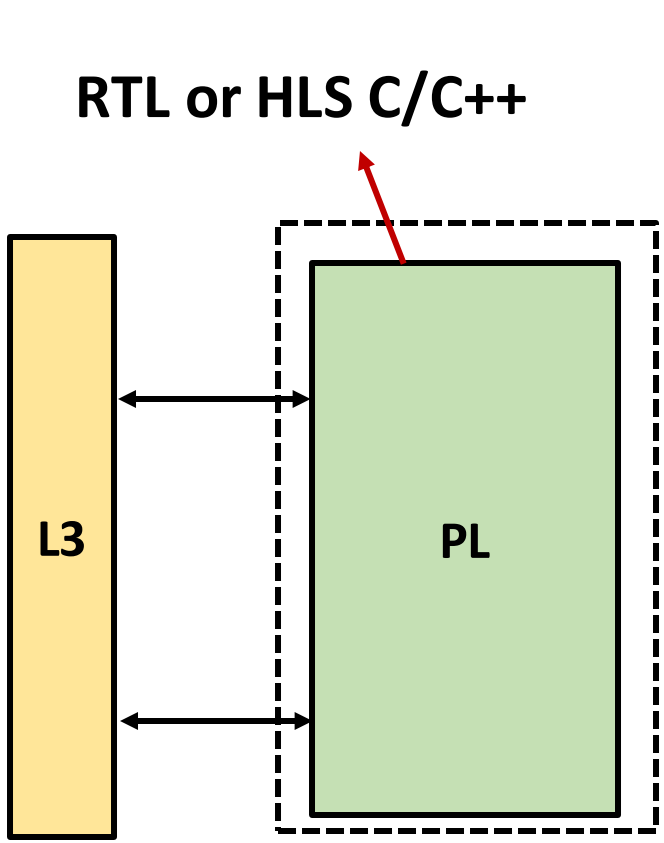
Challenge 2

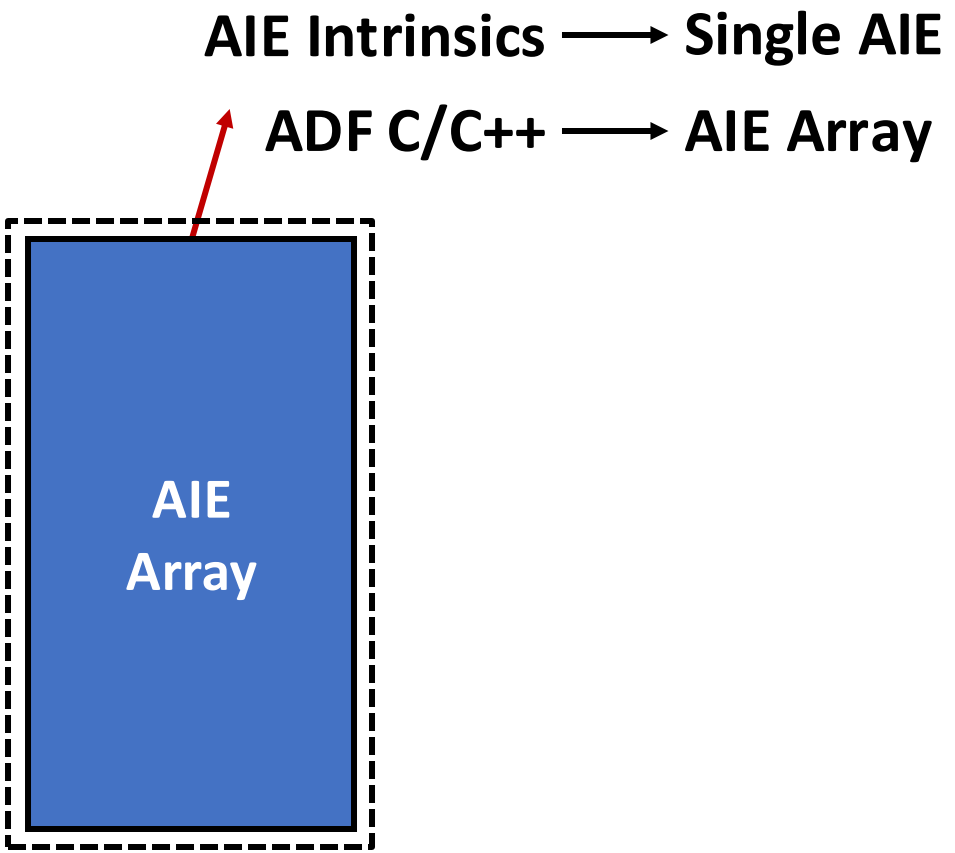
- **Lack of a unified representation for the entire heterogenous systems**
  - Vitis Flow



Challenge 2

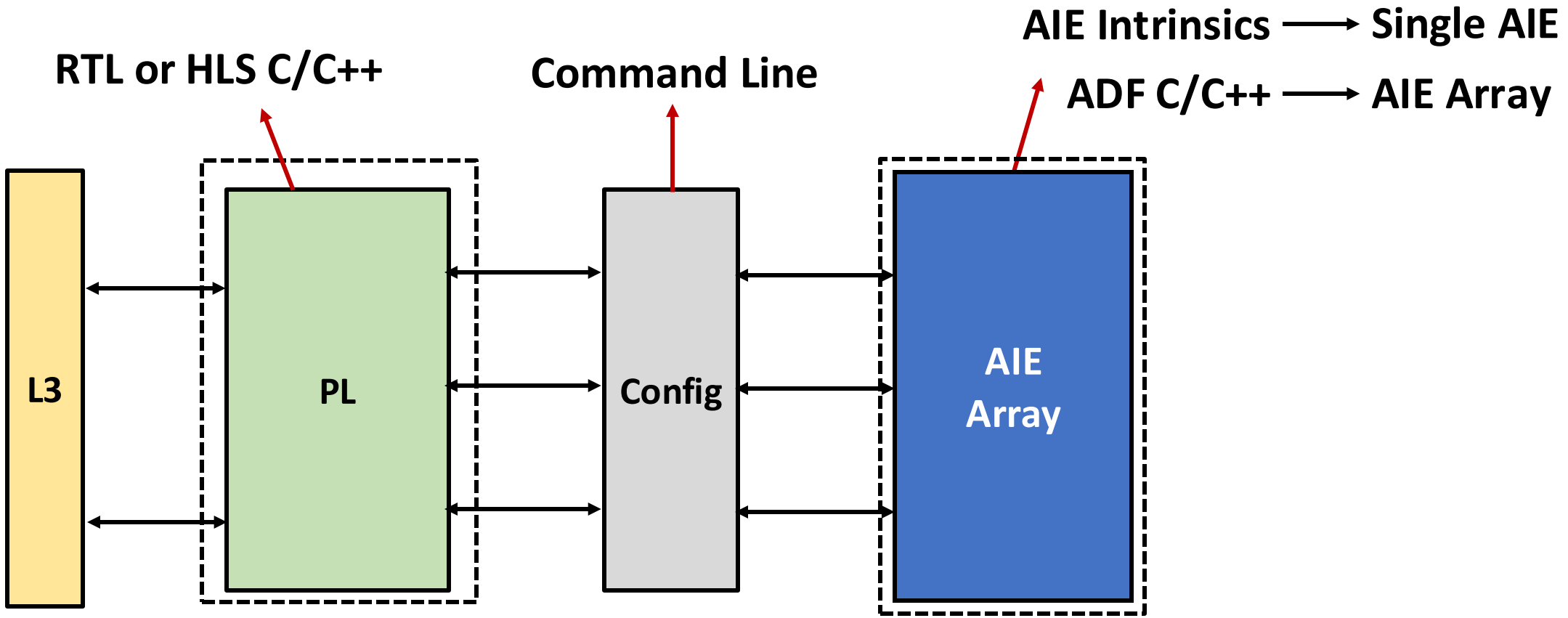
- **Lack of a unified representation for the entire heterogenous systems**
  - Vitis Flow





Challenge 2

- **Lack of a unified representation for the entire heterogenous systems**
  - Vitis Flow

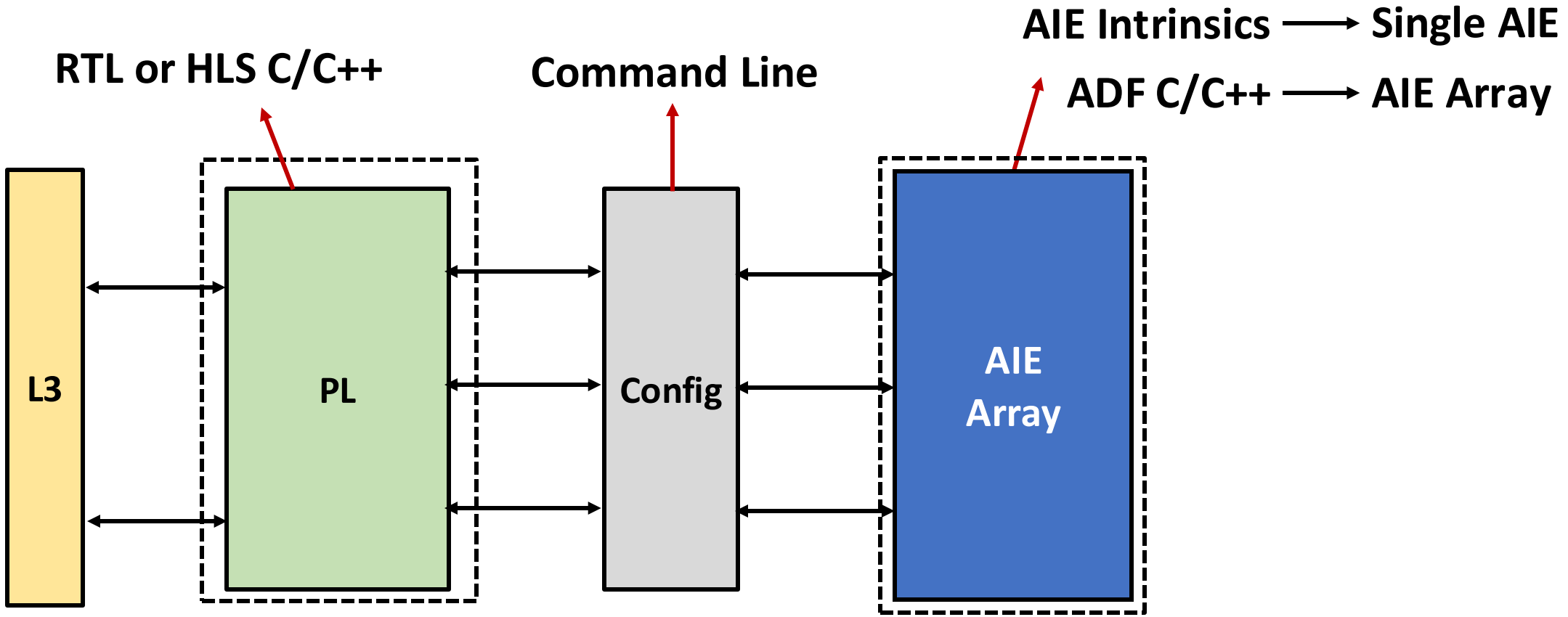


Challenge 2

- Lack of a unified representation for the entire heterogenous systems
  - Vitis Flow

How to improve the productivity?

How to optimize the design holistically?



“ARIES: An Agile MLIR-Based Compilation Flow for Reconfigurable Devices with AI Engines”

Challenges

Solutions

“ARIES: An Agile MLIR-Based Compilation Flow for Reconfigurable Devices with AI Engines”

Challenges

- Fragmented abstraction for parallelism and data movement

Solutions

- ARIES Python-Based programming interface that provides higher level abstraction for parallelism and data movement of AIE

“ARIES: An Agile MLIR-Based Compilation Flow for Reconfigurable Devices with AI Engines”

Challenges

- Fragmented abstraction for parallelism and data movement
- Lack of a unified representation for the entire heterogenous systems

Solutions

- ARIES Python-Based programming interface that provides higher level abstraction for parallelism and data movement of AIE
- ARIES Multi-Level Intermediate Representation (MLIR)-Based middle end with IRs and optimizations for different heterogenous component

“ARIES: An Agile MLIR-Based Compilation Flow for Reconfigurable Devices with AI Engines”

Challenges

- Fragmented abstraction for parallelism and data movement
- Lack of a unified representation for the entire heterogenous systems
- Extensibility and portability for different backends and future AIE architectures

Solutions

- ARIES Python-Based programming interface that provides higher level abstraction for parallelism and data movement of AIE
- ARIES Multi-Level Intermediate Representation (MLIR)-Based middle end with IRs and optimizations for different heterogenous component
- White-box open-sourced framework

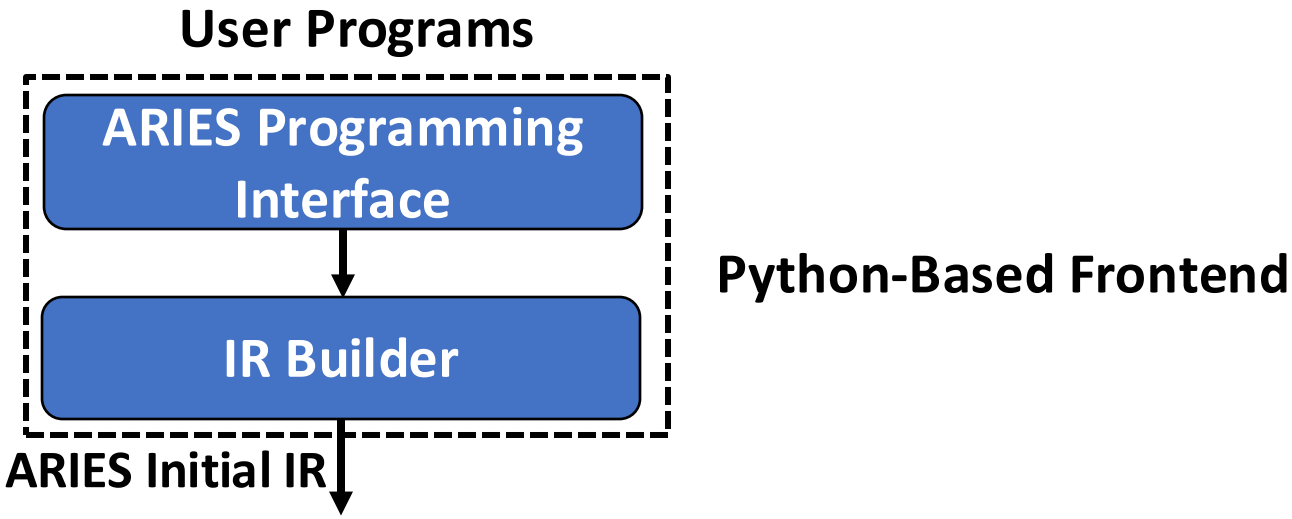
  https://github.com/arc-research-lab/Aries

ARIES Compilation Flow Overview

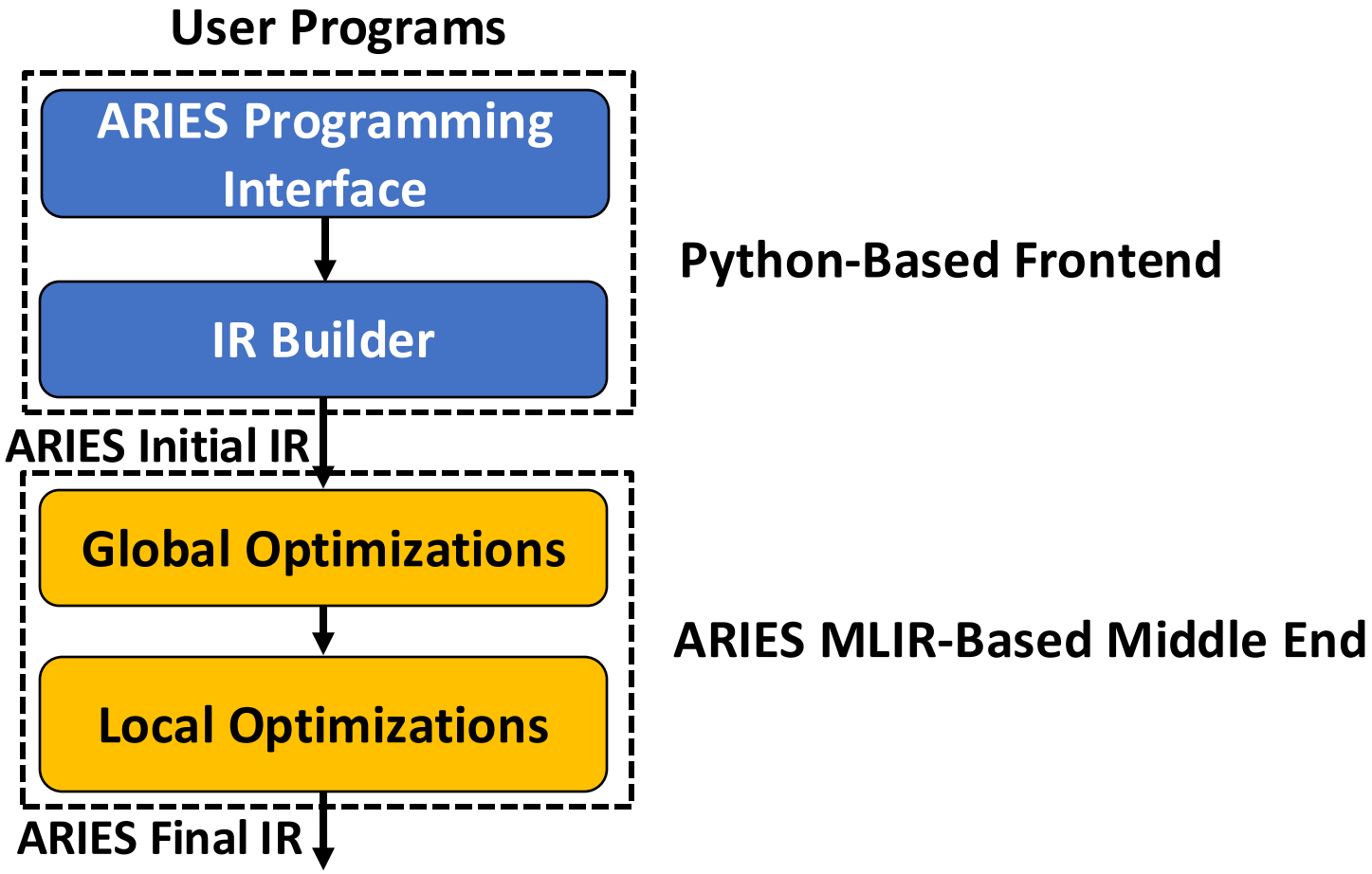
ARIES Compilation Flow Overview

User Programs

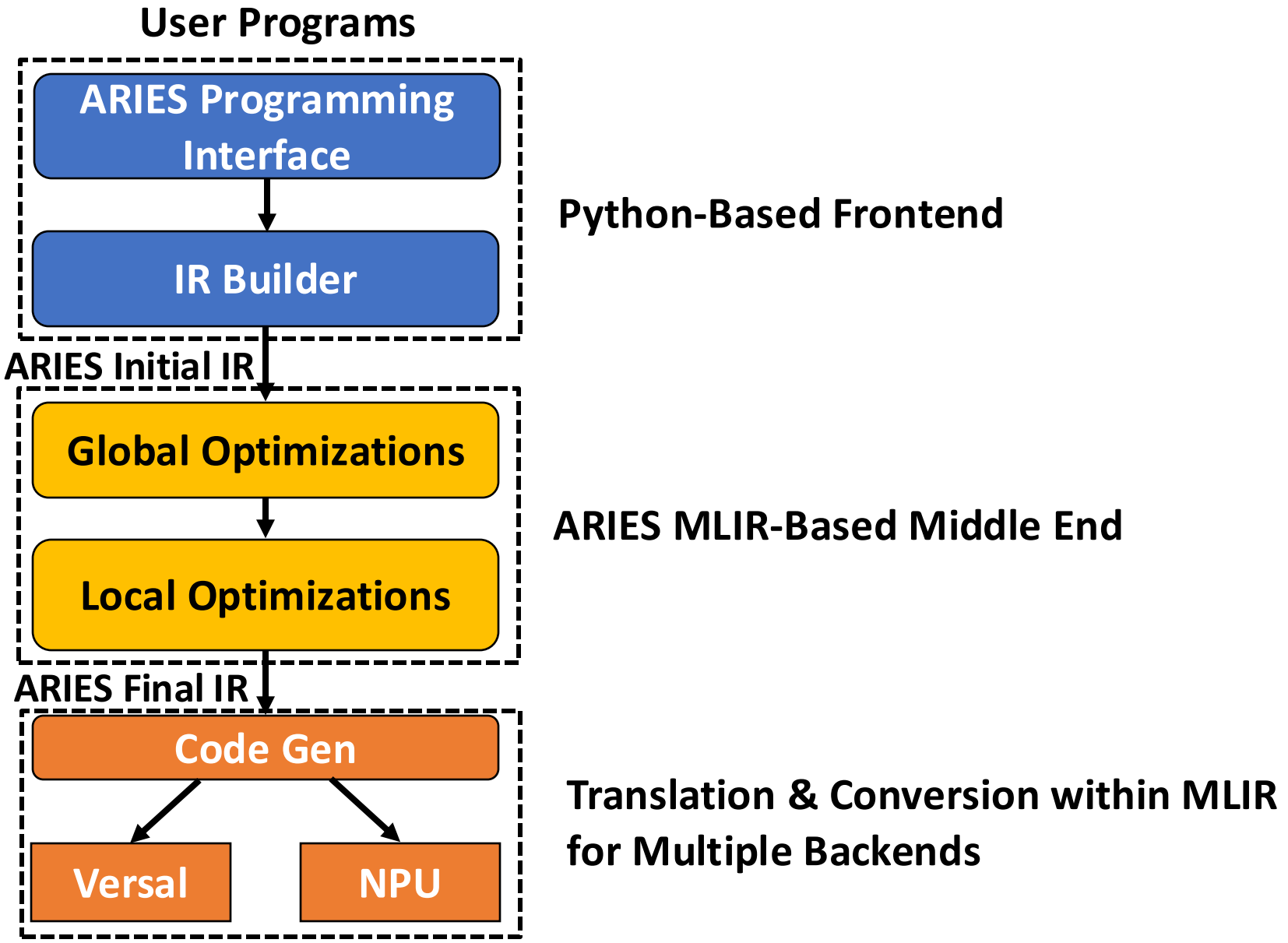
ARIES Compilation Flow Overview

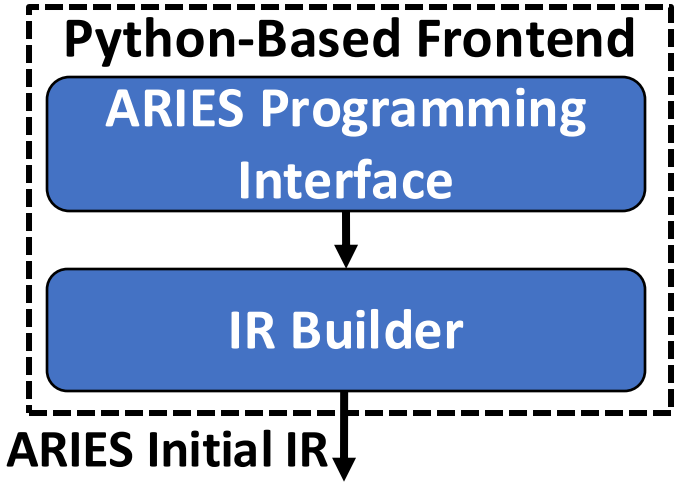


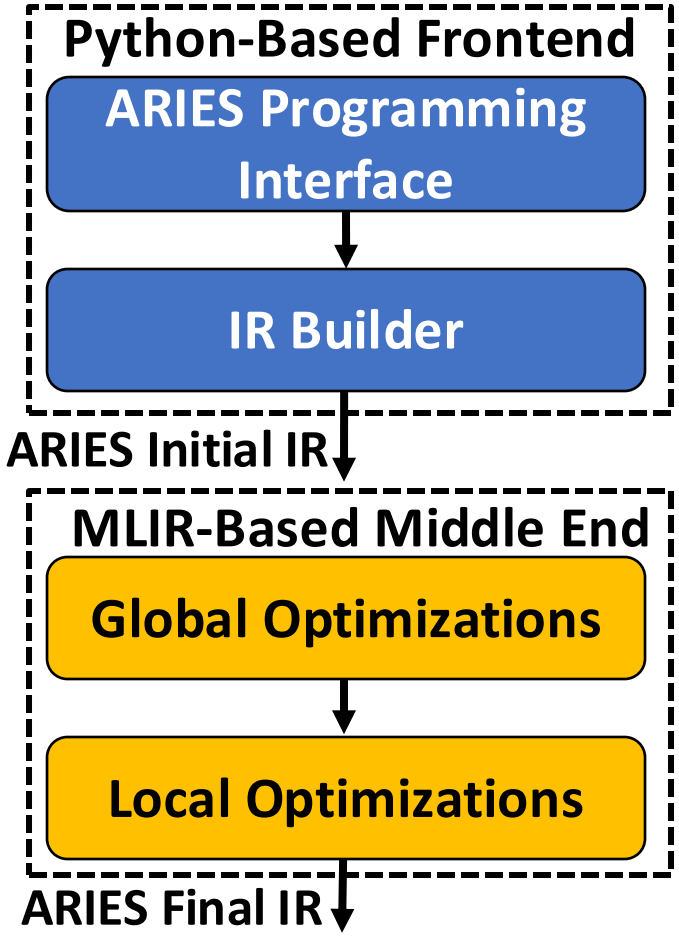
ARIES Compilation Flow Overview

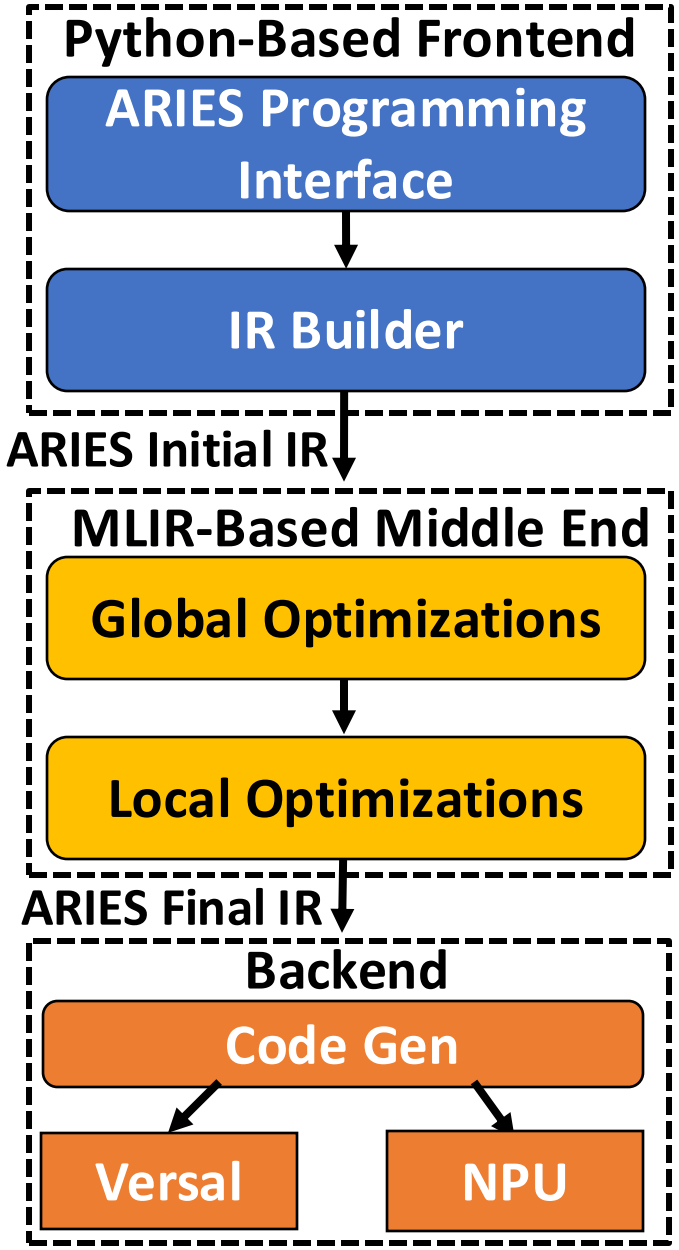


ARIES Compilation Flow Overview

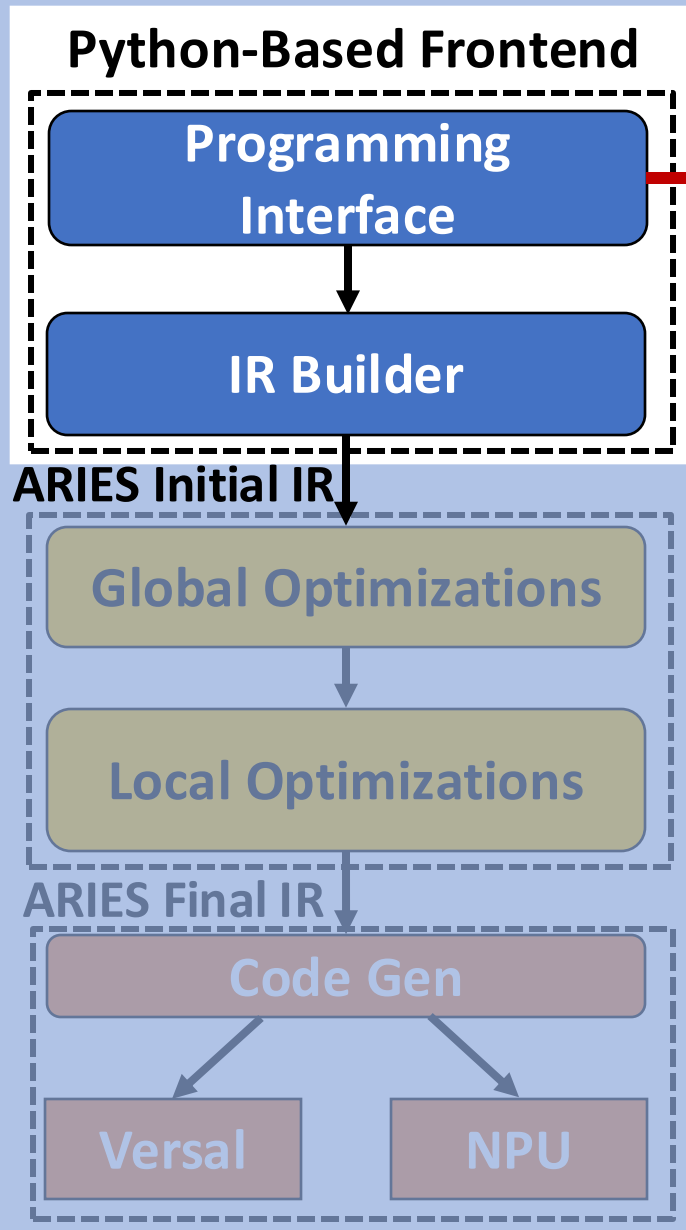



Python-Based Frontend
ARIES Programming Interface
IR Builder
ARIES Initial IR


Python-Based Frontend
ARIES Programming Interface
IR Builder
ARIES Initial IR
MLIR-Based Middle End
Global Optimizations
Local Optimizations
ARIES Final IR


Python-Based Frontend
ARIES Programming Interface
IR Builder
ARIES Initial IR
MLIR-Based Middle End
Global Optimizations
Local Optimizations
ARIES Final IR
Backend
Code Gen
Versal
NPU

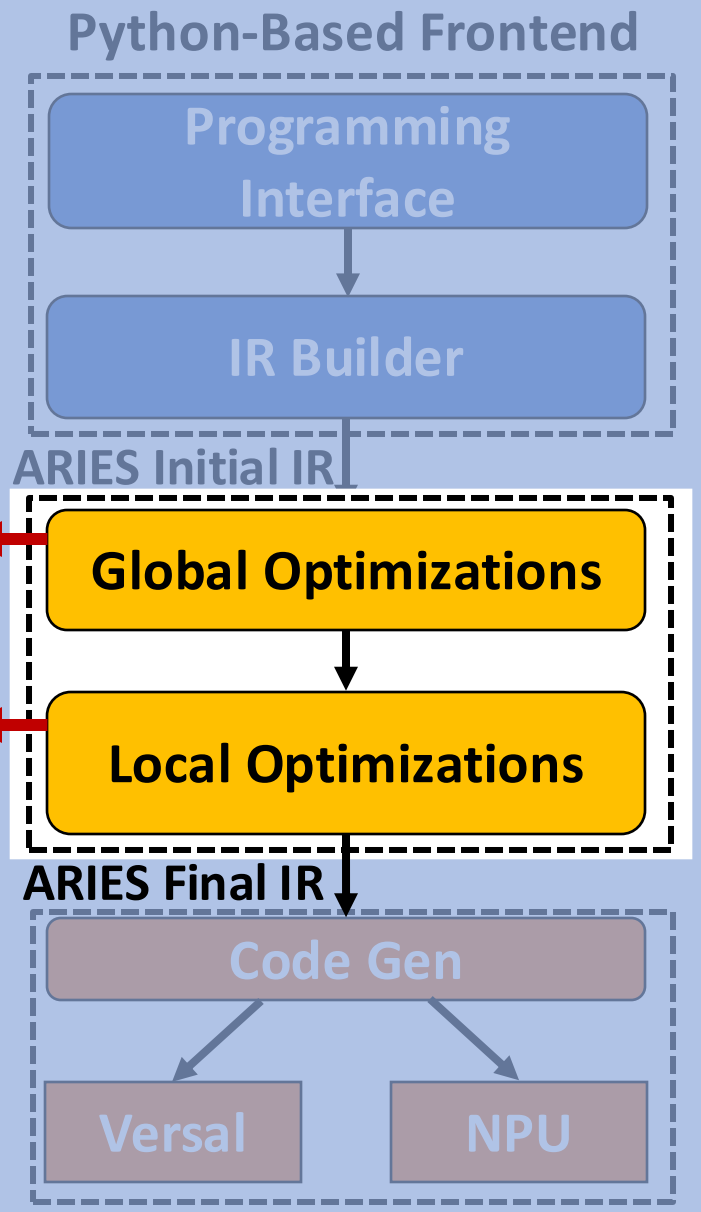
ARIES Compilation Flow Overview



Higher-level Abstraction:

- Explicit intra/inter AIE parallelism
- Simplified data movement
- Free of hardware details(locks, tiles)

ARIES Compilation Flow Overview



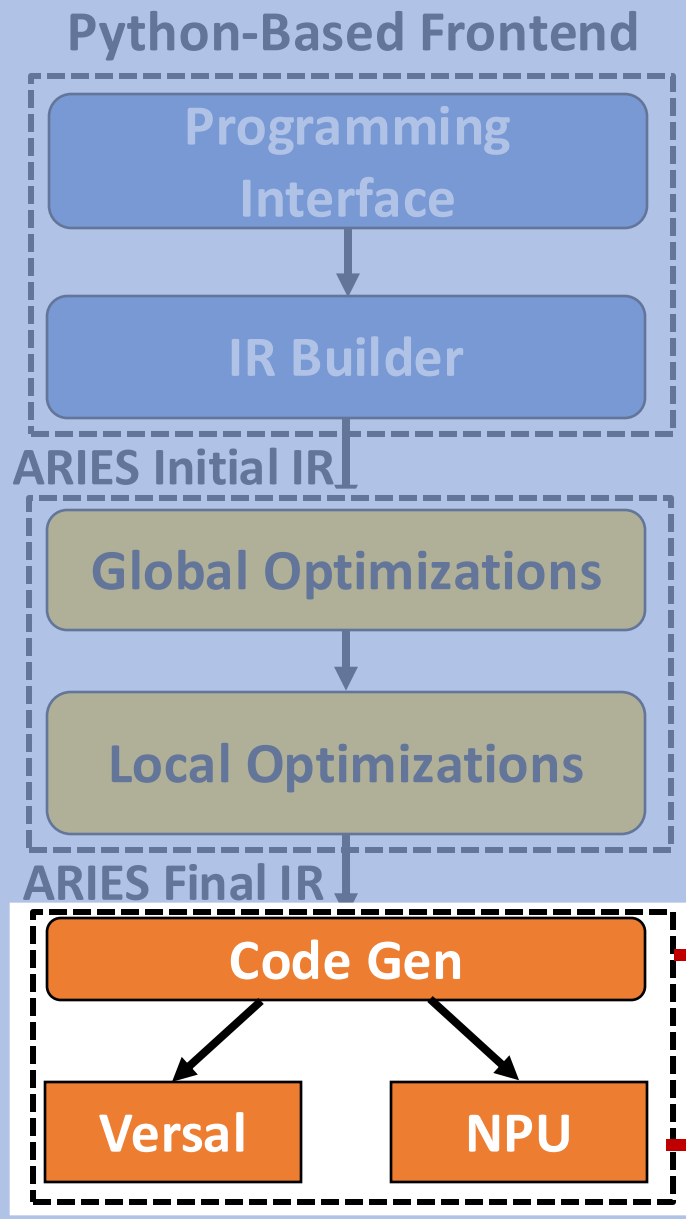
Hardware-agnostic Opts:

- Dataflow graph optimizations

Hardware dependent Opts under a unified representation:

- Single AIE → AIEVec Dialect
- AIE Array → ARIES ADF Dialect
- PL → MLIR Builtin Dialects

ARIES Compilation Flow Overview



Translation for Versal:

- AIEVec Dialect → AIE Intrinsics
- ADF Dialect → Vitis ADF Graph APIs
- MLIR Builtin Dialects→ HLS C/C++

Conversion for NPU:

- Final IR → AIE Objectfifo + AIEX Dialect

GEMM Example: ARIES Python-Based Frontend

- **ARIES Tile Programming Model**

① Defining whole workload: `@task_top()`

GEMM Example: ARIES Python-Based Frontend

- **ARIES Tile Programming Model**

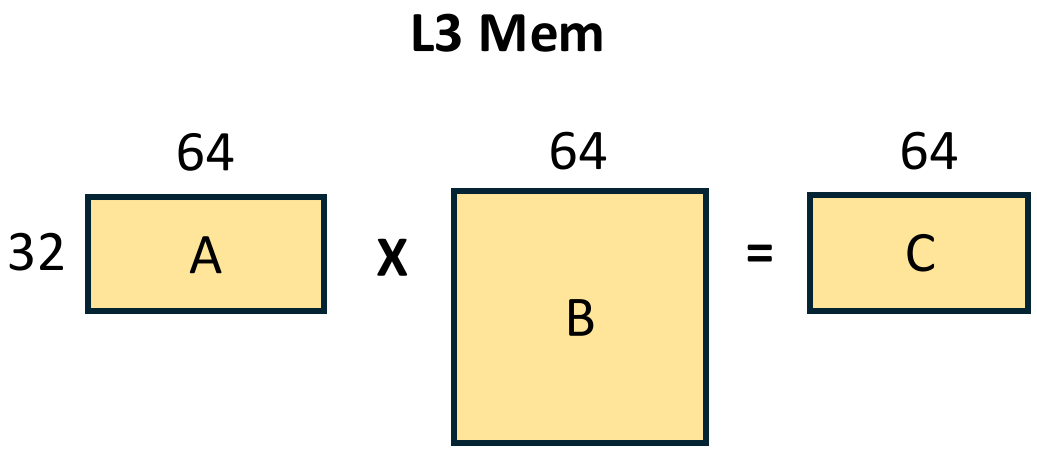
① Defining whole workload: `@task_top()`

```
@task_top()
def top(A: float32[32, 64], B: float32[64, 64],
        C: float32[32, 64]):
    grid = (1, 2, 2)
    tile_size = (32, 32, 32)
    gemm_task = gemm[grid, tile_size](A, B, C)
    return gemm_task
```

GEMM Example: ARIES Python-Based Frontend

- **ARIES Tile Programming Model**

① Defining whole workload: `@task_top()`



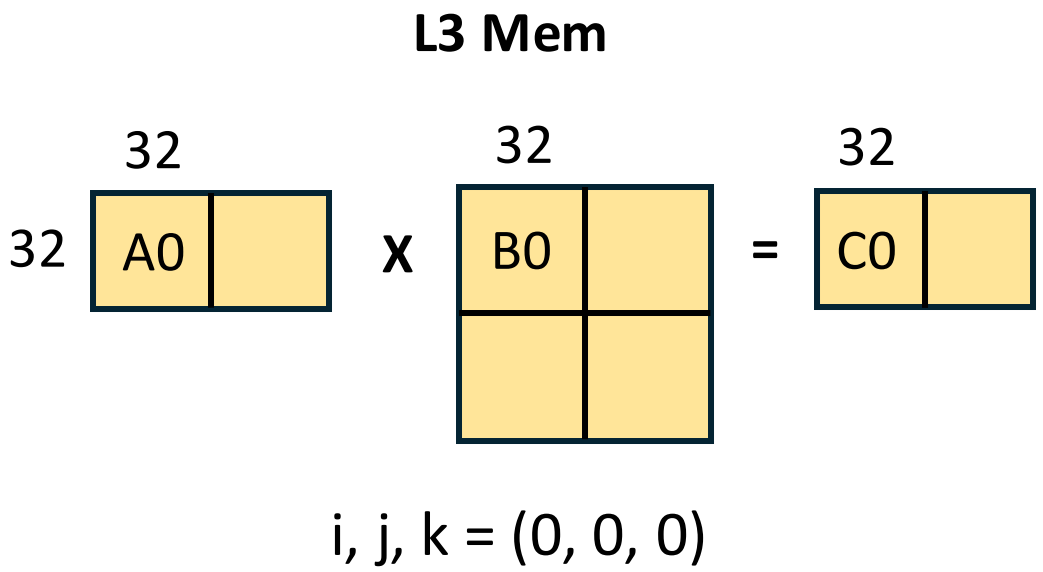

```
@task_top()
def top(A: float32[32, 64], B: float32[64, 64],
        C: float32[32, 64]):
    grid = (1, 2, 2)
    tile_size = (32, 32, 32)
    gemm_task = gemm[grid, tile_size](A, B, C)
    return gemm_task
```

GEMM Example: ARIES Python-Based Frontend

- **ARIES Tile Programming Model**

① Defining whole workload: `@task_top()`

```
@task_top()
def top(A: float32[32, 64], B: float32[64, 64],
        C: float32[32, 64]):
    grid = (1, 2, 2)
    tile_size = (32, 32, 32)
    gemm_task = gemm[grid, tile_size](A, B, C)
    return gemm_task
```

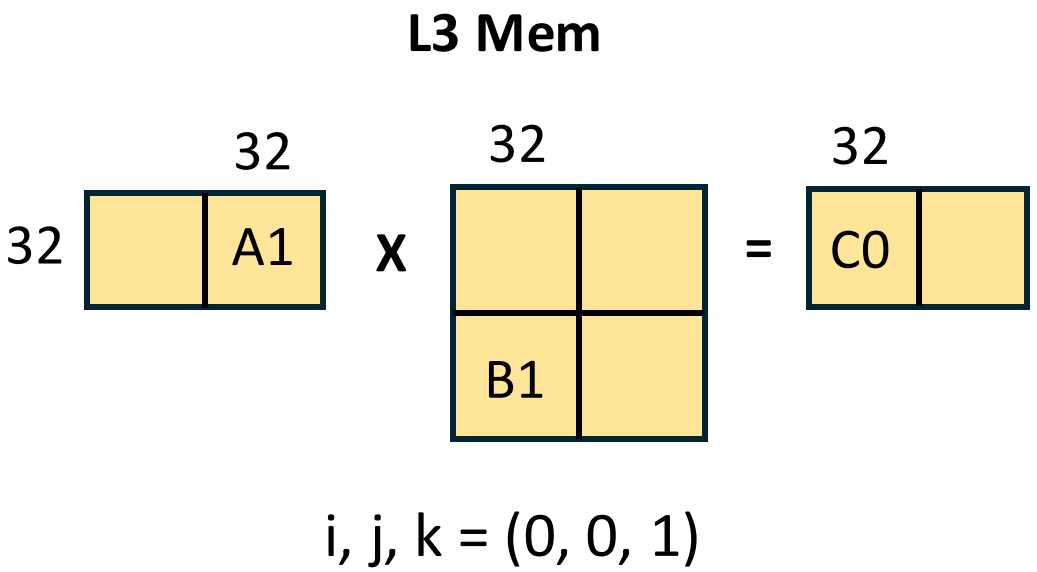


GEMM Example: ARIES Python-Based Frontend

- **ARIES Tile Programming Model**

① Defining whole workload: `@task_top()`

```
@task_top()
def top(A: float32[32, 64], B: float32[64, 64],
        C: float32[32, 64]):
    grid = (1, 2, 2)
    tile_size = (32, 32, 32)
    gemm_task = gemm[grid, tile_size](A, B, C)
    return gemm_task
```

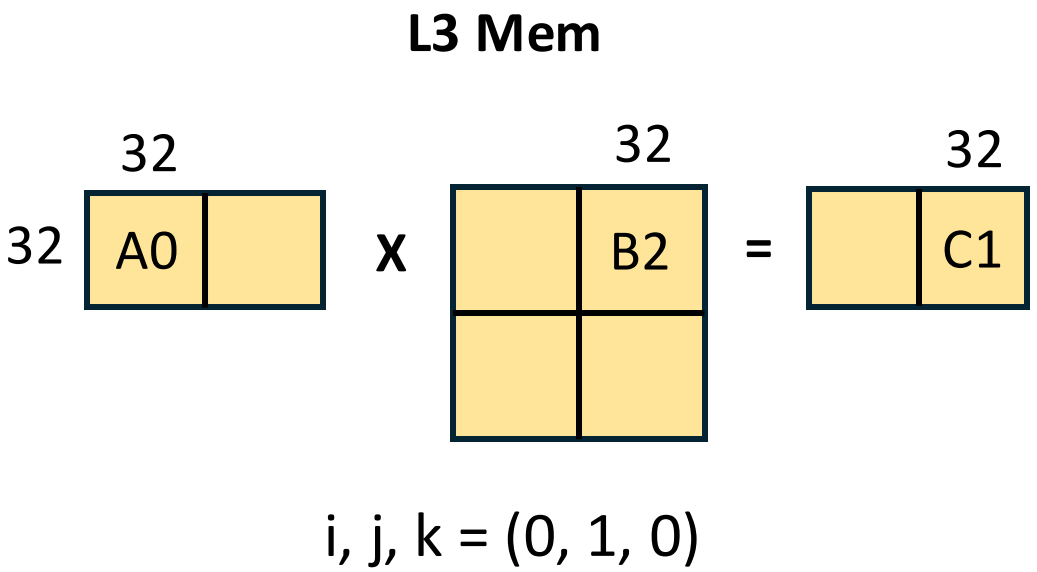


GEMM Example: ARIES Python-Based Frontend

- **ARIES Tile Programming Model**

① Defining whole workload: `@task_top()`

```
@task_top()
def top(A: float32[32, 64], B: float32[64, 64],
        C: float32[32, 64]):
    grid = (1, 2, 2)
    tile_size = (32, 32, 32)
    gemm_task = gemm[grid, tile_size](A, B, C)
    return gemm_task
```

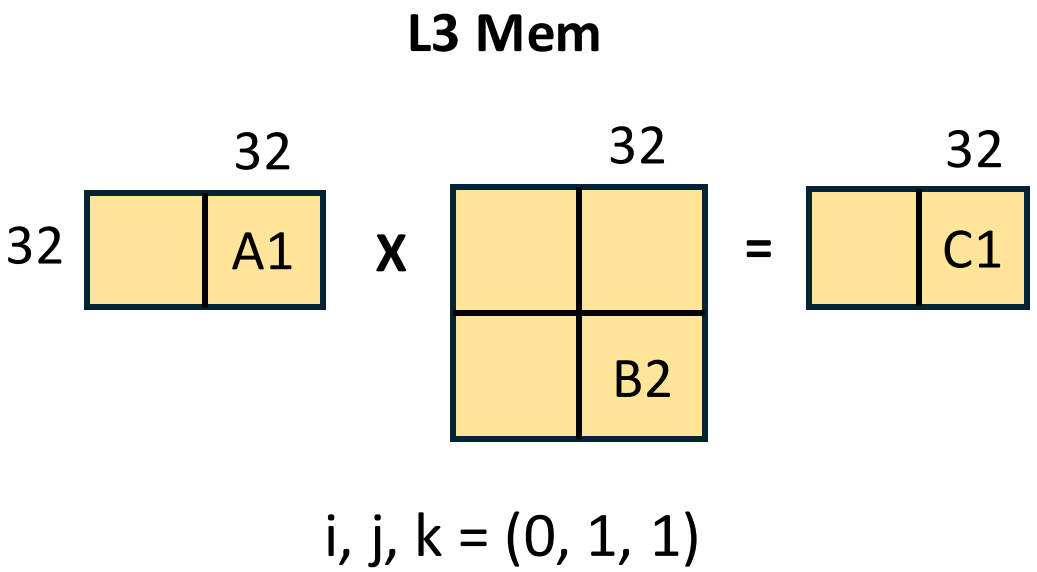


GEMM Example: ARIES Python-Based Frontend

- **ARIES Tile Programming Model**

① Defining whole workload: `@task_top()`

```
@task_top()
def top(A: float32[32, 64], B: float32[64, 64],
        C: float32[32, 64]):
    grid = (1, 2, 2)
    tile_size = (32, 32, 32)
    gemm_task = gemm[grid, tile_size](A, B, C)
    return gemm_task
```

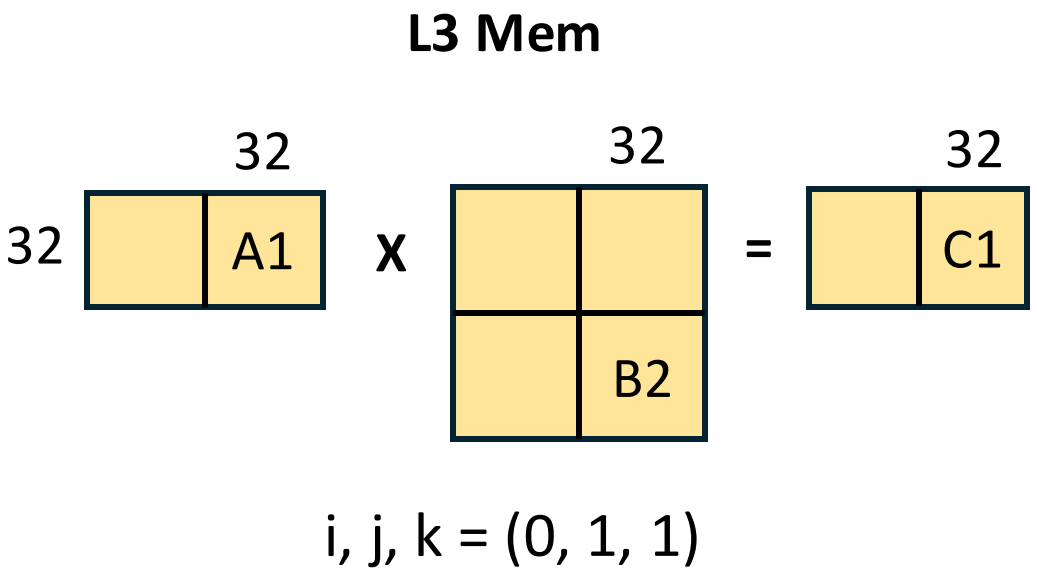


GEMM Example: ARIES Python-Based Frontend

- **ARIES Tile Programming Model**

① Defining whole workload: `@task_top()`

```
@task_top()
def top(A: float32[32, 64], B: float32[64, 64],
        C: float32[32, 64]):
    grid = (1, 2, 2)
    tile_size = (32, 32, 32)
    gemm_task = gemm[grid, tile_size](A, B, C)
    return gemm_task
```



GEMM Example: ARIES Python-Based Frontend

- **ARIES Tile Programming Model**

GEMM Example: ARIES Python-Based Frontend

- **ARIES Tile Programming Model**

② Single tile workload definition: `@task_kernel()`

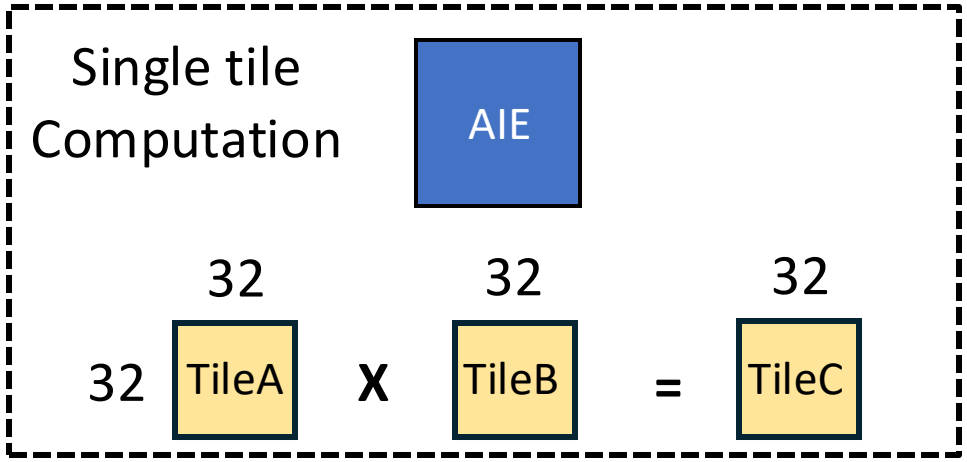
```
@task_kernel()
def compute(TileA: float32[32, 32],
            TileB: float32[32, 32],
            TileC: float32[32, 32]):
  for i0 in range(0, 32):
    for j0 in range(0, 32):
      for k0 in range(0, 32):
        TileC[i0, j0] += TileA[i0, k0] * TileB[k0, j0]
```

GEMM Example: ARIES Python-Based Frontend

- **ARIES Tile Programming Model**

② Single tile workload definition: `@task_kernel()`

```
@task_kernel()
def compute(TileA: float32[32, 32],
            TileB: float32[32, 32],
            TileC: float32[32, 32]):
  for i0 in range(0, 32):
    for j0 in range(0, 32):
      for k0 in range(0, 32):
        TileC[i0, j0] += TileA[i0, k0] * TileB[k0, j0]
```



GEMM Example: ARIES Python-Based Frontend

- **ARIES Tile Programming Model**

③ ARIES APIs for abstracted data transfer: `@task_tile()`

GEMM Example: ARIES Python-Based Frontend

- **ARIES Tile Programming Model**

③ ARIES APIs for abstracted data transfer: `@task_tile()`

```
@task_tile()
def gemm(A: float32[32, 64], B: float32[64, 64],
         C: float32[32, 64]):
    i, j, k = aries.tile_ranks() # Get grid ids
    TI, TJ, TK = aries.tile_sizes()
    L1_A = aries.buffer((TI, TK), "float32")
    L1_B = aries.buffer((TK, TJ), "float32")
    L1_C = aries.buffer((TI, TJ), "float32")
    ti = aries.arange(i*TI, (i+1)*TI)
    tj = aries.arange(j*TJ, (j+1)*TJ)
    tk = aries.arange(k*TK, (k+1)*TK)
    L1_A = aries.load(A, (ti, tk))
    L1_B = aries.load(B, (tk, tj))
    L1_C = aries.load(C, (ti, tj))
    compute(L1_A, L1_B, L1_C)
    aries.store(L1_C, C, (ti, tj))
```

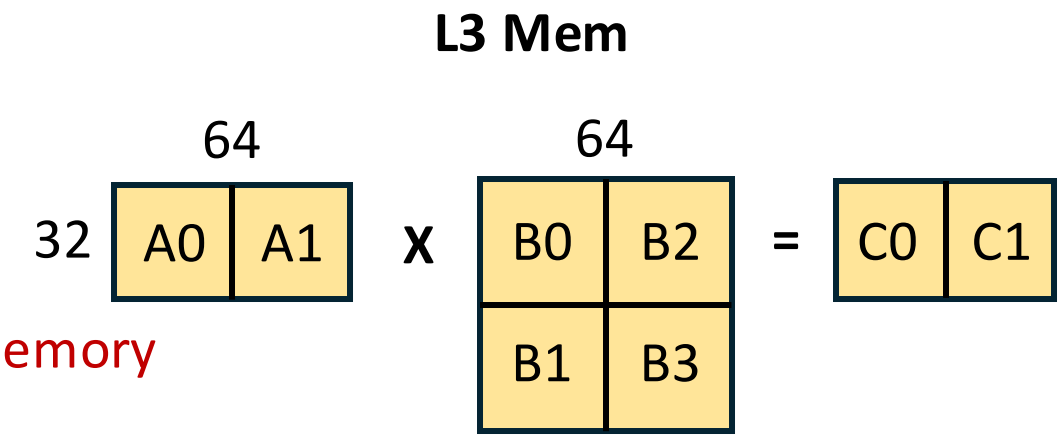
GEMM Example: ARIES Python-Based Frontend

- **ARIES Tile Programming Model**

③ ARIES APIs for abstracted data transfer: `@task_tile()`

```
@task_tile()
def gemm(A: float32[32, 64], B: float32[64, 64],
         C: float32[32, 64]):
    i, j, k = aries.tile_ranks() # Get grid ids
    TI, TJ, TK = aries.tile_sizes()
    L1_A = aries.buffer((TI, TK), "float32")
    L1_B = aries.buffer((TK, TJ), "float32")
    L1_C = aries.buffer((TI, TJ), "float32")
    ti = aries.arange(i*TI, (i+1)*TI)
    tj = aries.arange(j*TJ, (j+1)*TJ)
    tk = aries.arange(k*TK, (k+1)*TK)
    L1_A = aries.load(A, (ti, tk))
    L1_B = aries.load(B, (tk, tj))
    L1_C = aries.load(C, (ti, tj))
    compute(L1_A, L1_B, L1_C)
    aries.store(L1_C, C, (ti, tj))
```

L3 memory



GEMM Example: ARIES Python-Based Frontend

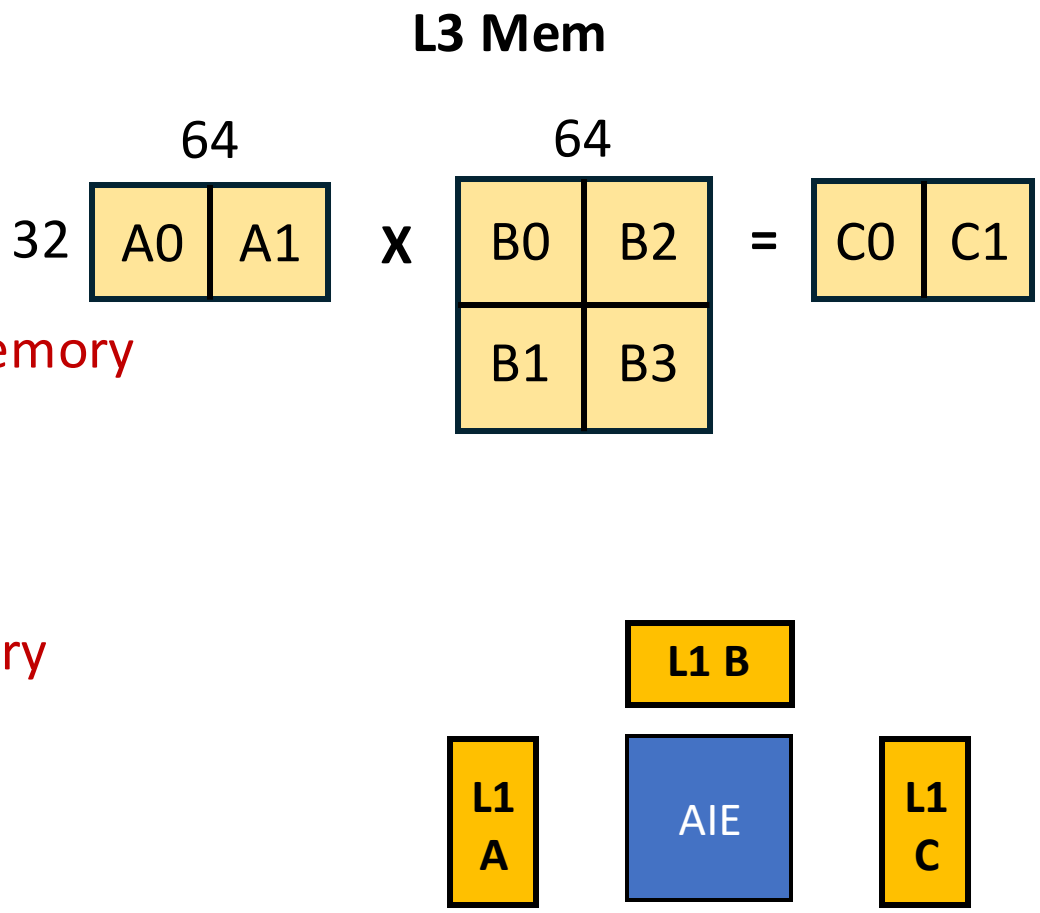
- **ARIES Tile Programming Model**

③ ARIES APIs for abstracted data transfer: `@task_tile()`

```
@task_tile()
def gemm(A: float32[32, 64], B: float32[64, 64],
         C: float32[32, 64]):
    i, j, k = aries.tile_ranks() # Get grid ids
    TI, TJ, TK = aries.tile_sizes()
    L1_A = aries.buffer((TI, TK), "float32")
    L1_B = aries.buffer((TK, TJ), "float32")
    L1_C = aries.buffer((TI, TJ), "float32")
    ti = aries.arange(i*TI, (i+1)*TI)
    tj = aries.arange(j*TJ, (j+1)*TJ)
    tk = aries.arange(k*TK, (k+1)*TK)
    L1_A = aries.load(A, (ti, tk))
    L1_B = aries.load(B, (tk, tj))
    L1_C = aries.load(C, (ti, tj))
    compute(L1_A, L1_B, L1_C)
    aries.store(L1_C, C, (ti, tj))
```

L3 memory

L1 memory



GEMM Example: ARIES Python-Based Frontend

- **ARIES Tile Programming Model**

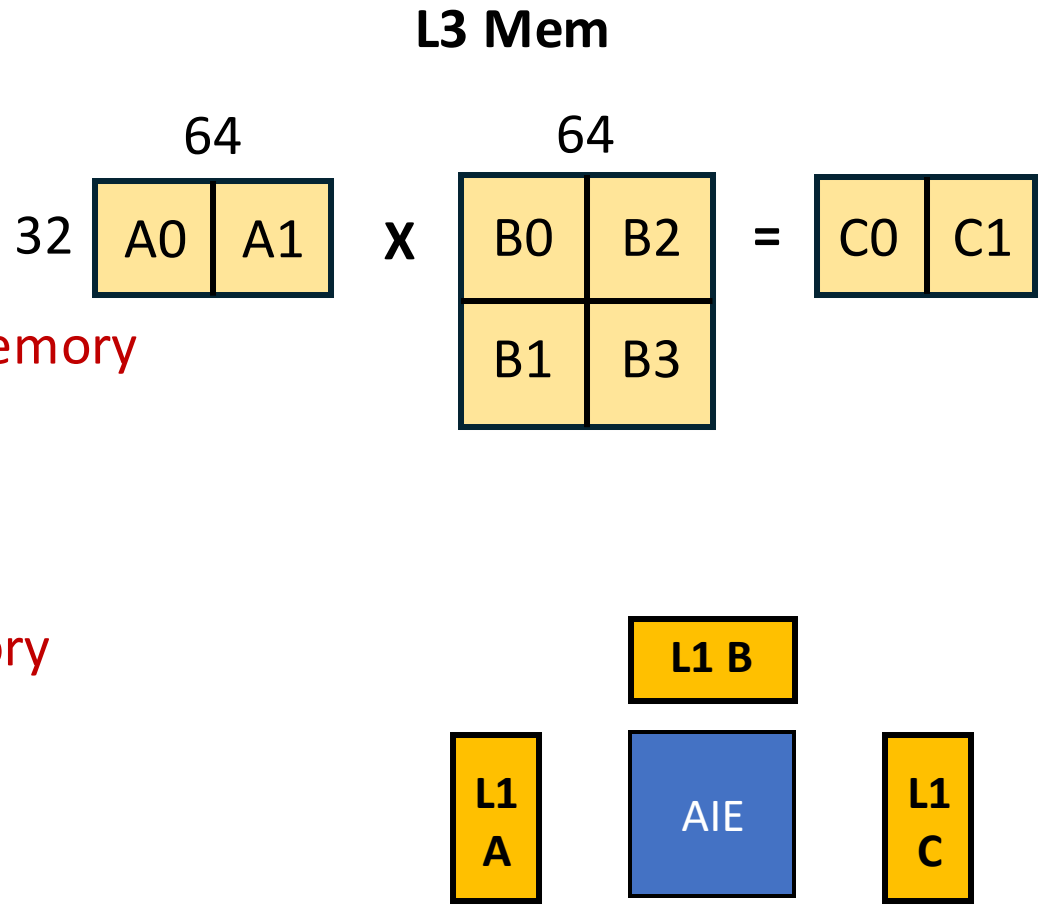
③ ARIES APIs for abstracted data transfer: `@task_tile()`

```
@task_tile()
def gemm(A: float32[32, 64], B: float32[64, 64],
         C: float32[32, 64]):
    i, j, k = aries.tile_ranks() # Get grid ids
    TI, TJ, TK = aries.tile_sizes()
    L1_A = aries.buffer((TI, TK), "float32")
    L1_B = aries.buffer((TK, TJ), "float32")
    L1_C = aries.buffer((TI, TJ), "float32")
    ti = aries.arange(i*TI, (i+1)*TI)
    tj = aries.arange(j*TJ, (j+1)*TJ)
    tk = aries.arange(k*TK, (k+1)*TK)
    L1_A = aries.load(A, (ti, tk))
    L1_B = aries.load(B, (tk, tj))
    L1_C = aries.load(C, (ti, tj))
    compute(L1_A, L1_B, L1_C)
    aries.store(L1_C, C, (ti, tj))
```

L3 memory

L1 memory

- L2 mem and ShimDMA will be inferred
- Tiles, locks, buffer descriptor IDs … are handled automatically

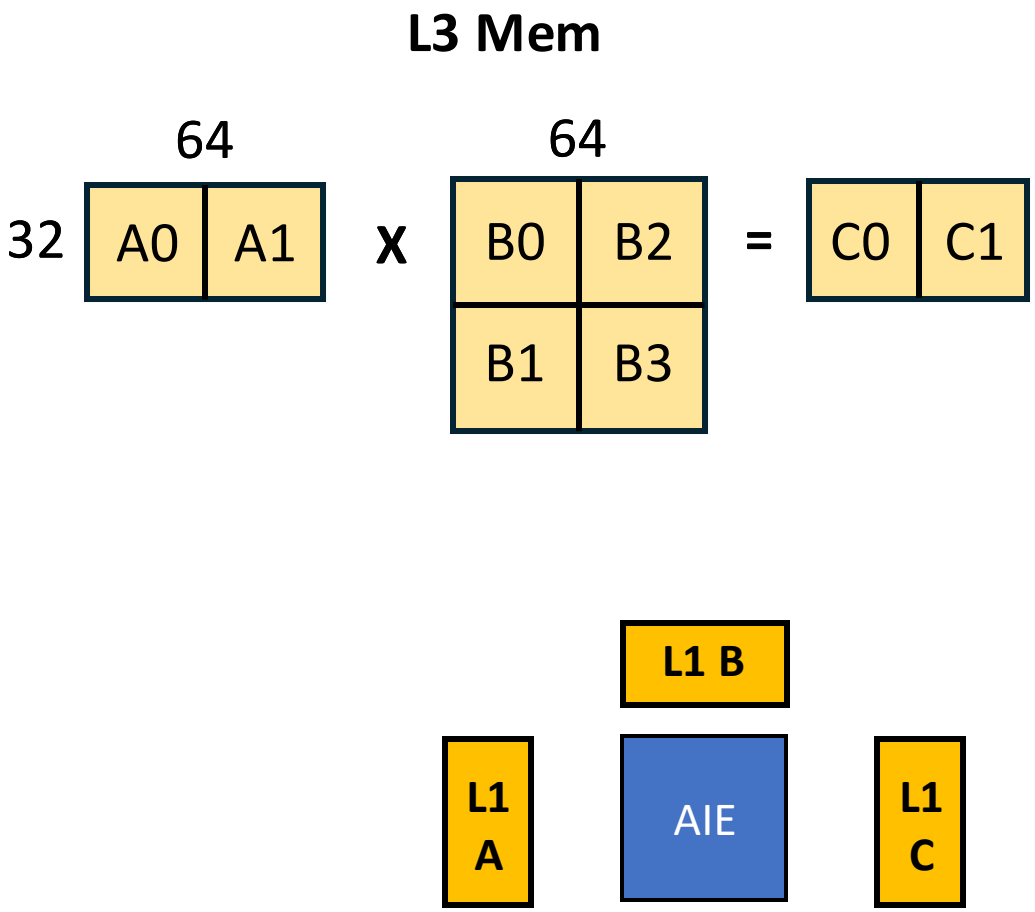


GEMM Example: ARIES Python-Based Frontend

- **ARIES Tile Programming Model**

③ ARIES APIs for abstracted data transfer: `@task_tile()`

```
@task_tile()
def gemm(A: float32[32, 64], B: float32[64, 64],
         C: float32[32, 64]):
    i, j, k = aries.tile_ranks() # Get grid ids
    TI, TJ, TK = aries.tile_sizes()
    L1_A = aries.buffer((TI, TK), "float32")
    L1_B = aries.buffer((TK, TJ), "float32")
    L1_C = aries.buffer((TI, TJ), "float32")
```



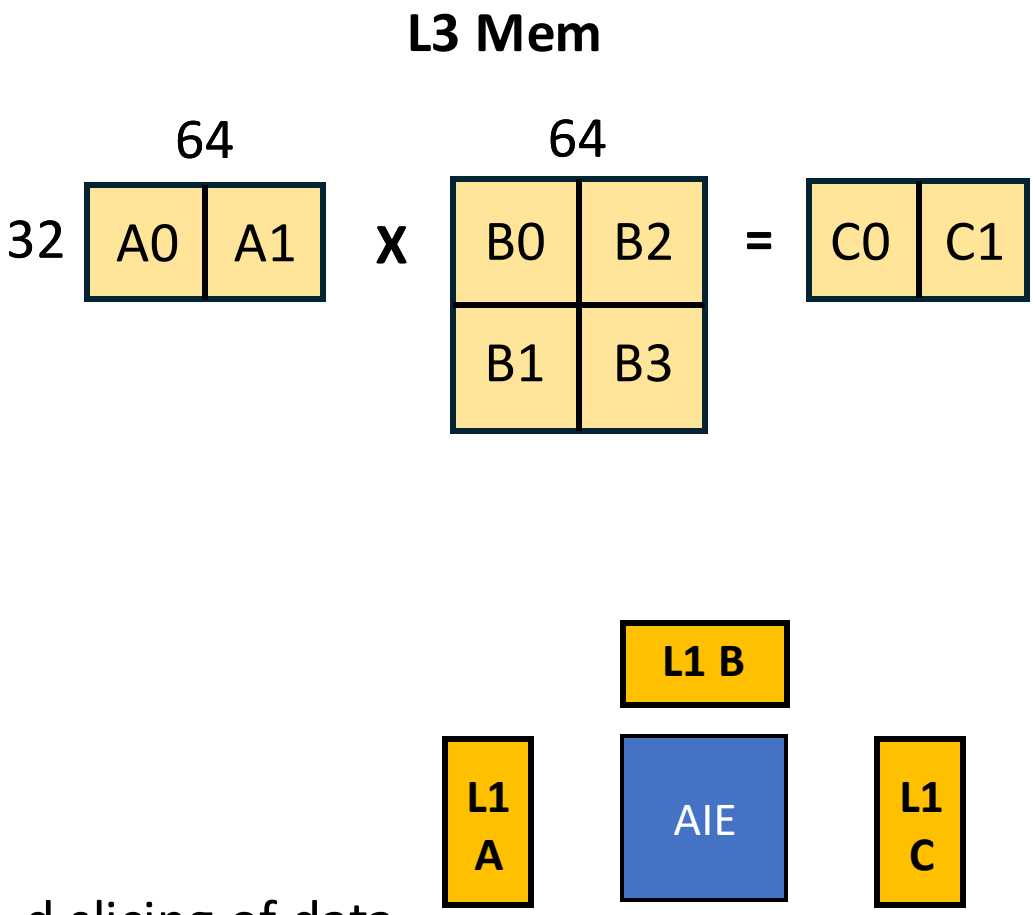
GEMM Example: ARIES Python-Based Frontend

- **ARIES Tile Programming Model**

③ ARIES APIs for abstracted data transfer: `@task_tile()`

```
@task_tile()
def gemm(A: float32[32, 64], B: float32[64, 64],
         C: float32[32, 64]):
    i, j, k = aries.tile_ranks() # Get grid ids
    TI, TJ, TK = aries.tile_sizes()
    L1_A = aries.buffer((TI, TK), "float32")
    L1_B = aries.buffer((TK, TJ), "float32")
    L1_C = aries.buffer((TI, TJ), "float32")
    ti = aries.arange(i*TI, (i+1)*TI)
    tj = aries.arange(j*TJ, (j+1)*TJ)
    tk = aries.arange(k*TK, (k+1)*TK)
    L1_A = aries.load(A, (ti, tk))
    L1_B = aries.load(B, (tk, tj))
    L1_C = aries.load(C, (ti, tj))
    compute(L1_A, L1_B, L1_C)
    aries.store(L1_C, C, (ti, tj))
```

Transfer n-d slicing of data between L3 & L1



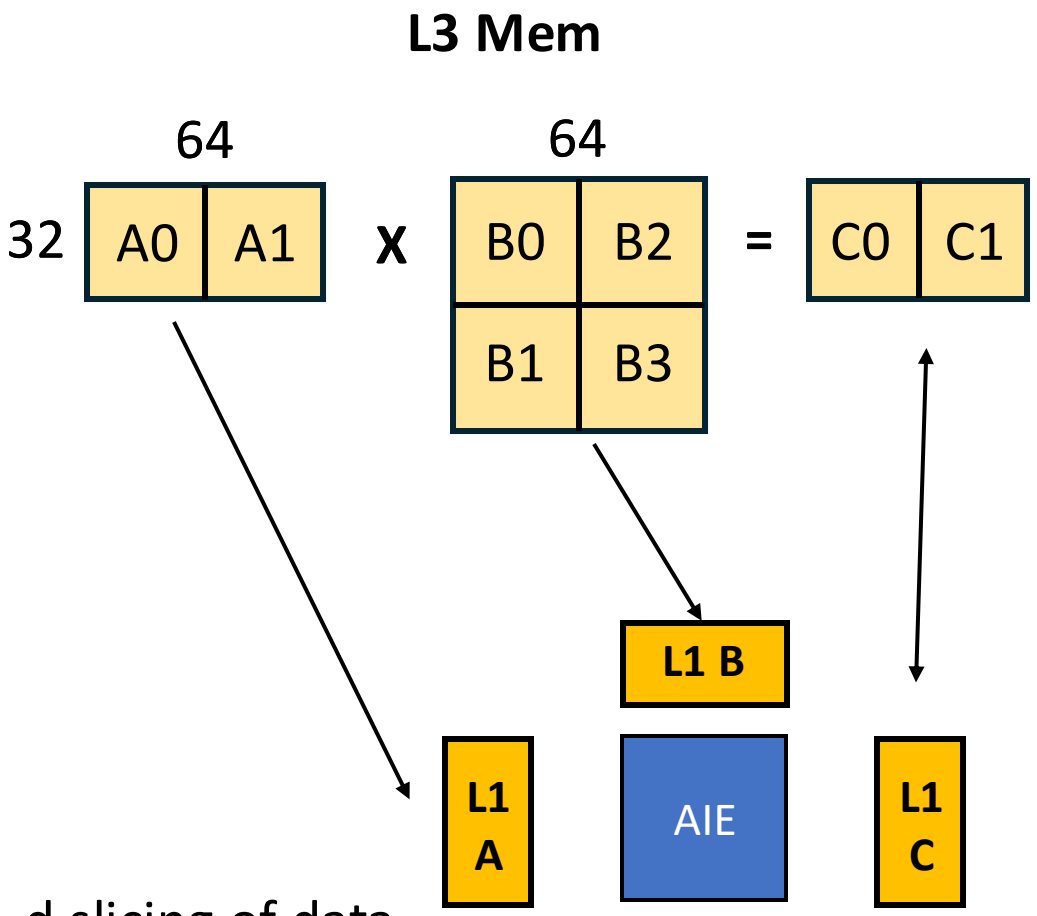
GEMM Example: ARIES Python-Based Frontend

- **ARIES Tile Programming Model**

③ ARIES APIs for abstracted data transfer: `@task_tile()`

```
@task_tile()
def gemm(A: float32[32, 64], B: float32[64, 64],
         C: float32[32, 64]):
    i, j, k = aries.tile_ranks() # Get grid ids
    TI, TJ, TK = aries.tile_sizes()
    L1_A = aries.buffer((TI, TK), "float32")
    L1_B = aries.buffer((TK, TJ), "float32")
    L1_C = aries.buffer((TI, TJ), "float32")
    ti = aries.arange(i*TI, (i+1)*TI)
    tj = aries.arange(j*TJ, (j+1)*TJ)
    tk = aries.arange(k*TK, (k+1)*TK)
    L1_A = aries.load(A, (ti, tk))
    L1_B = aries.load(B, (tk, tj))
    L1_C = aries.load(C, (ti, tj))
    compute(L1_A, L1_B, L1_C)
    aries.store(L1_C, C, (ti, tj))
```

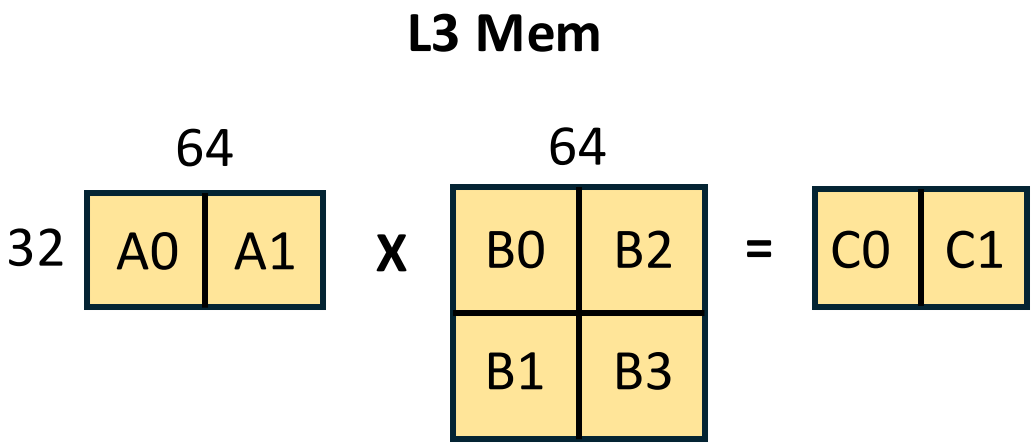
Transfer n-d slicing of data between L3 & L1

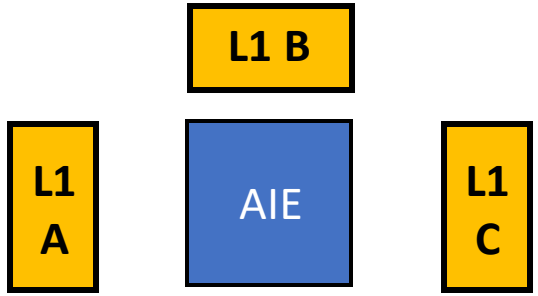


GEMM Example: ARIES Python-Based Frontend

- **ARIES Tile Programming Model**

④ Specify primitives for optimization





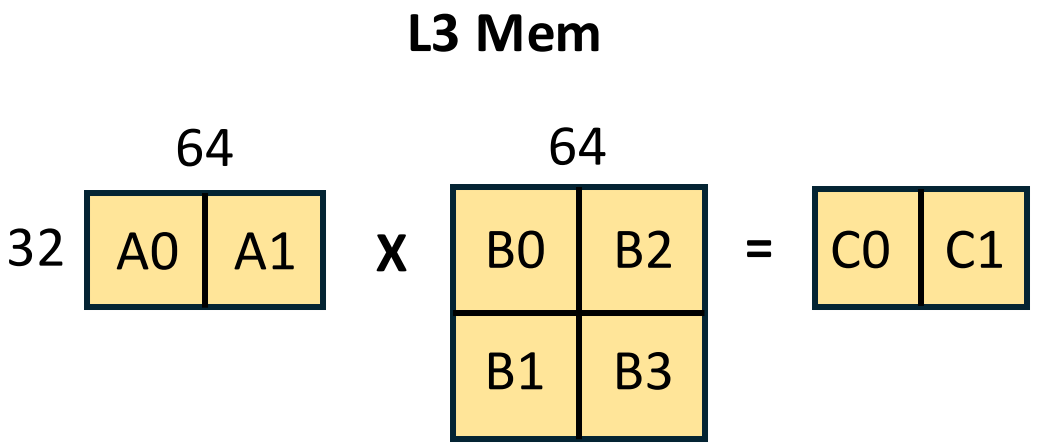
GEMM Example: ARIES Python-Based Frontend

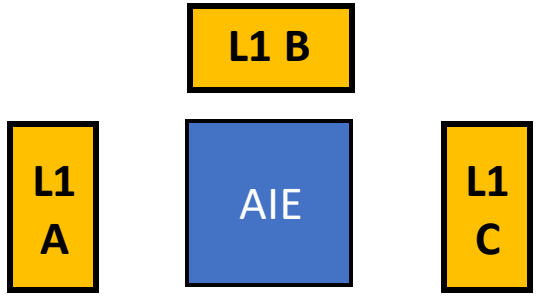
- **ARIES Tile Programming Model**

④ Specify primitives for optimization

```
gemm_task = top(A, B, C)

sch = Schedule(gemm_task)
sch.parallel(gemm_task, [i=1, j=2, k=2])
sch.vectorize(gemm_task, axis=0, factor=[8])
sch.to("VCK190")
sch.build()
```





GEMM Example: ARIES Python-Based Frontend

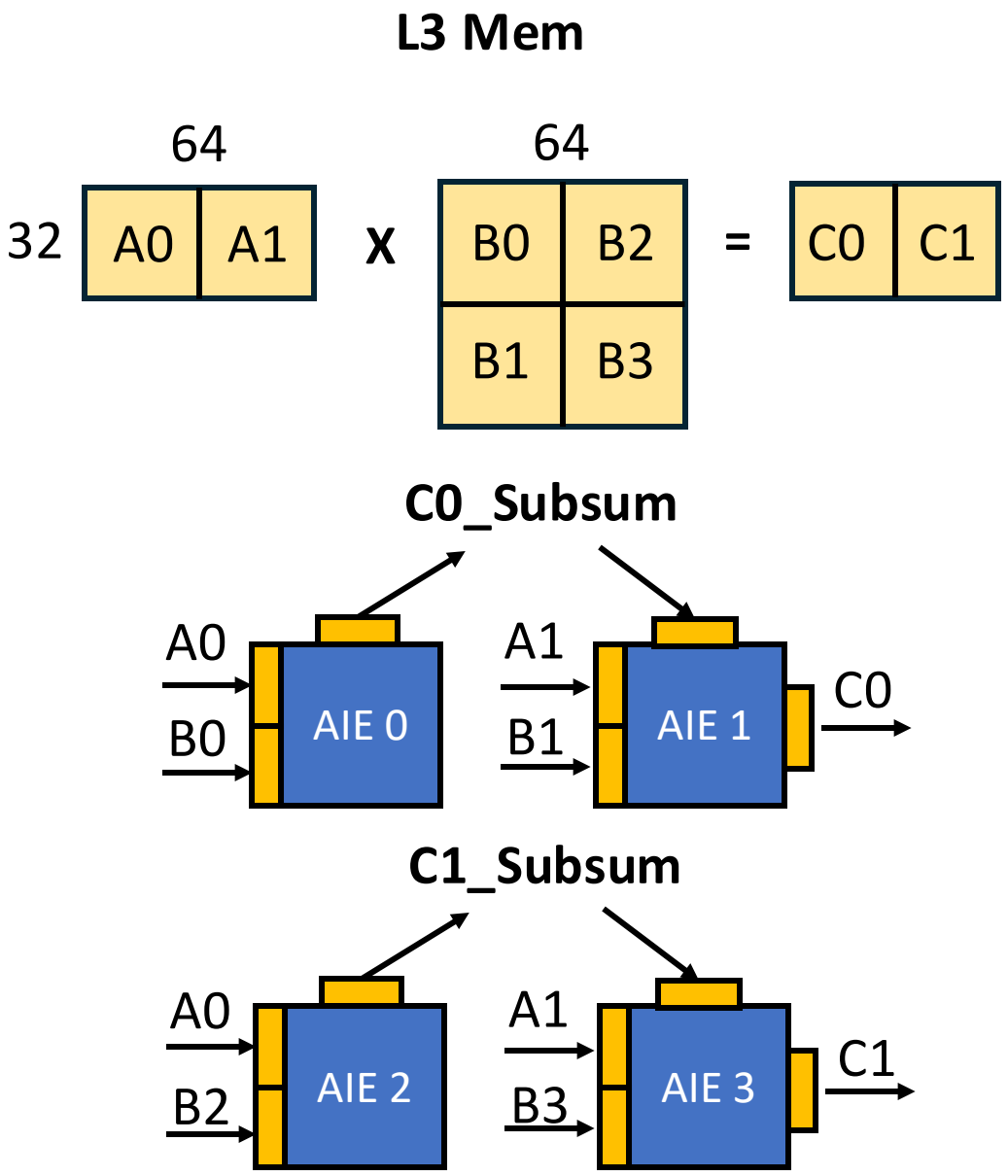
- **ARIES Tile Programming Model**

④ Specify primitives for optimization

```
gemm_task = top(A, B, C)
```

Specify the schedules

```
sch = Schedule(gemm_task)
sch.parallel(gemm_task, [i=1, j=2, k=2])
sch.vectorize(gemm_task, axis=0, factor=[8])
sch.to("VCK190")
sch.build()
```



GEMM Example: ARIES Python-Based Frontend

- **ARIES Tile Programming Model**

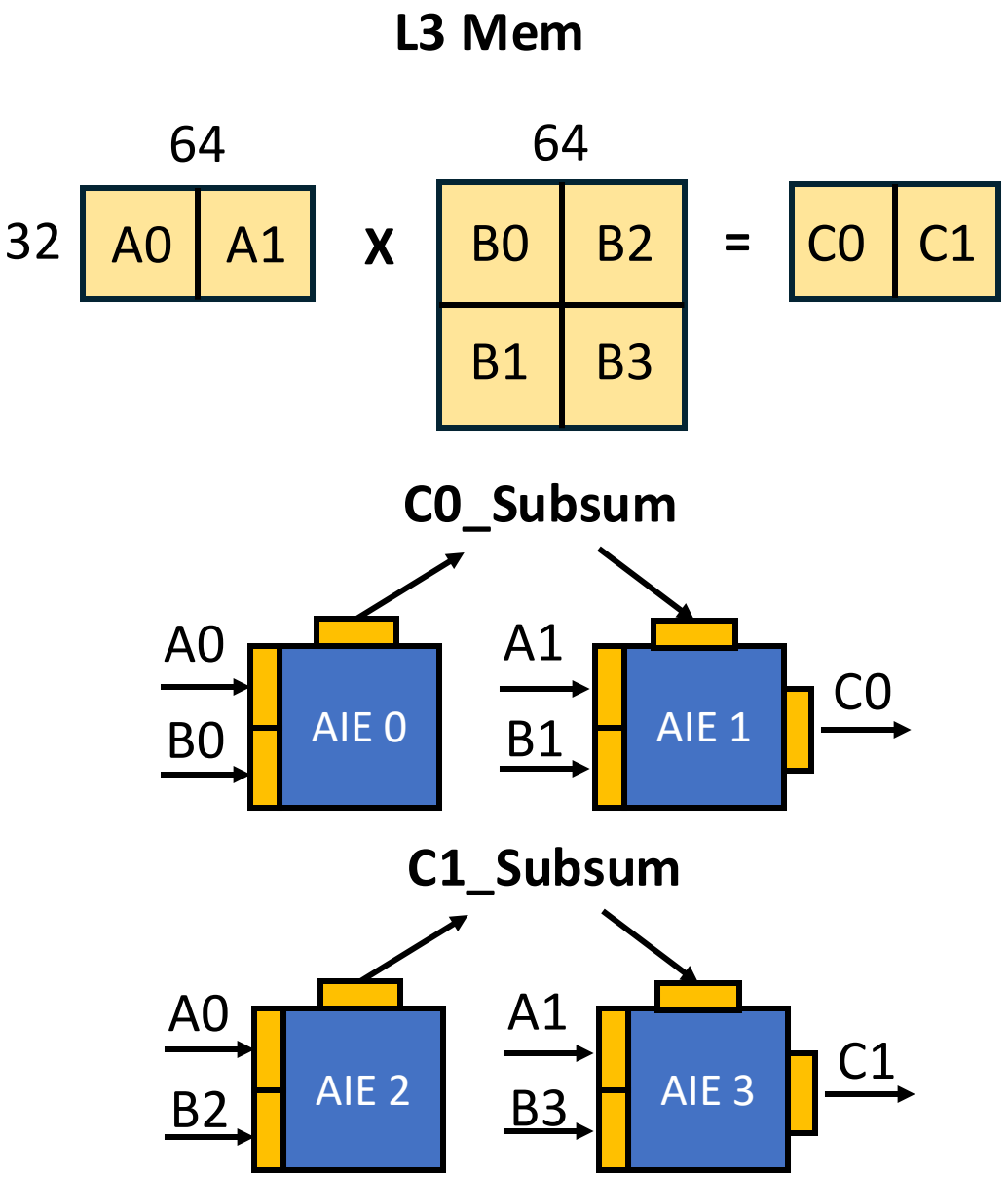
④ Specify primitives for optimization

```
gemm_task = top(A, B, C)
```

Specify the schedules

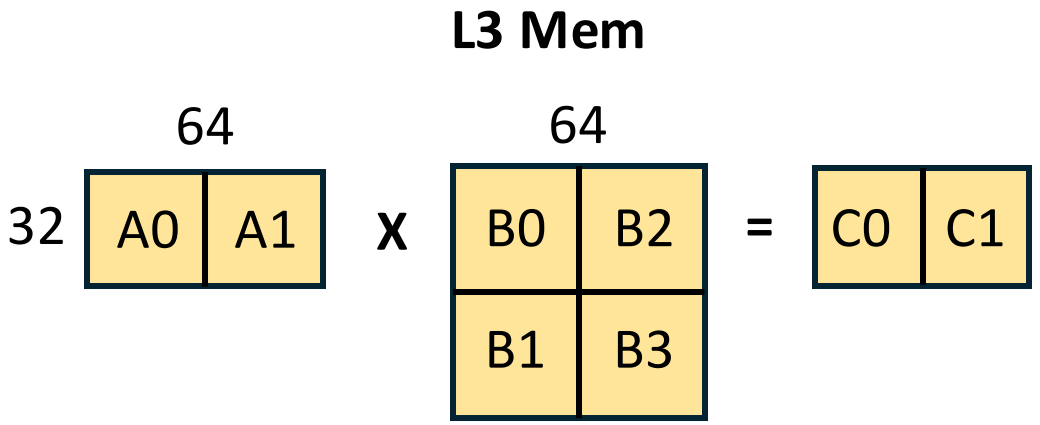
```
sch = Schedule(gemm_task)
sch.parallel(gemm_task, [i=1, j=2, k=2])
sch.vectorize(gemm_task, axis=0, factor=[8])
sch.to("VCK190")
sch.build()
```

→ Generate initial IR & ARIES commands



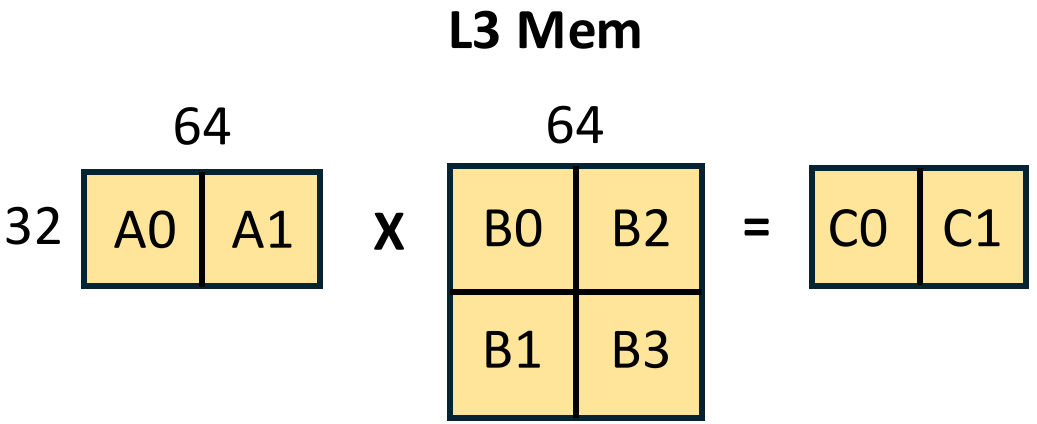
GEMM Example: ARIES MLIR-Based Middle End

- **Global optimizations**



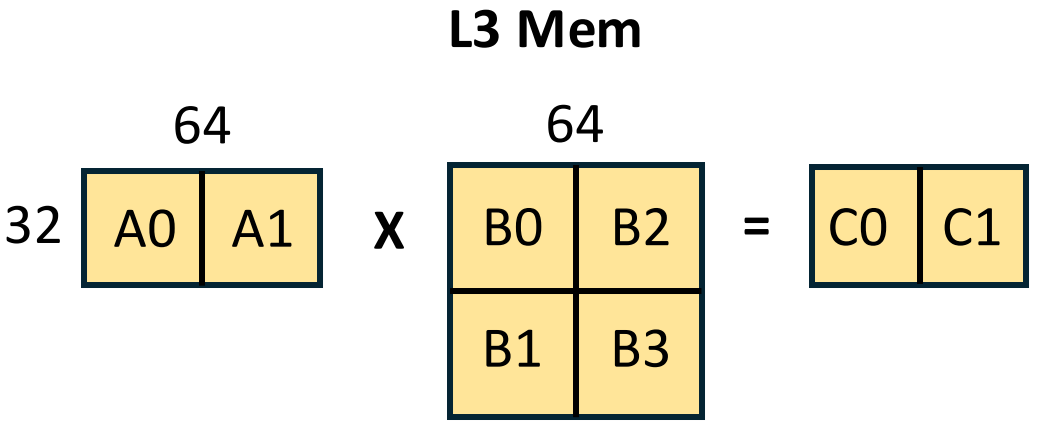
GEMM Example: ARIES MLIR-Based Middle End

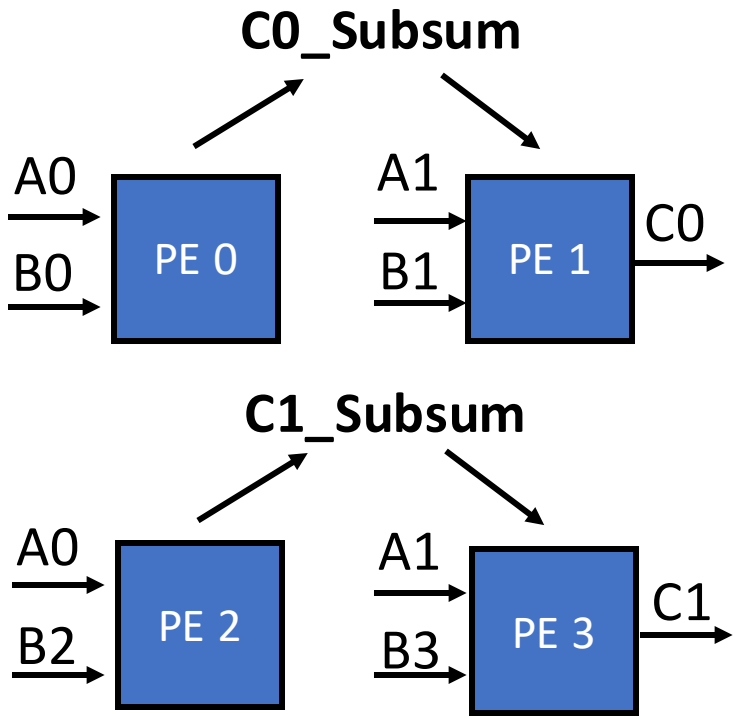
- **Global optimizations**
  - Hardware-agnostic data reuse pattern detections



GEMM Example: ARIES MLIR-Based Middle End

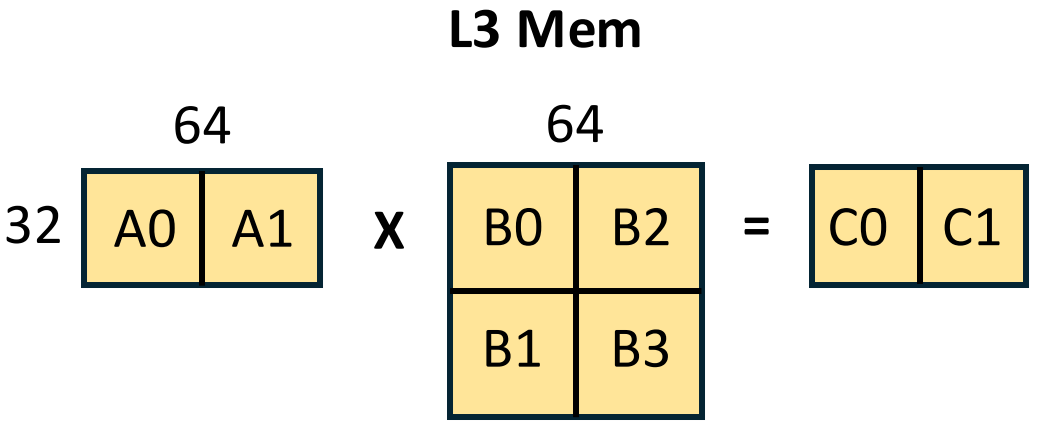
- **Global optimizations**
  - Hardware-agnostic data reuse pattern detections

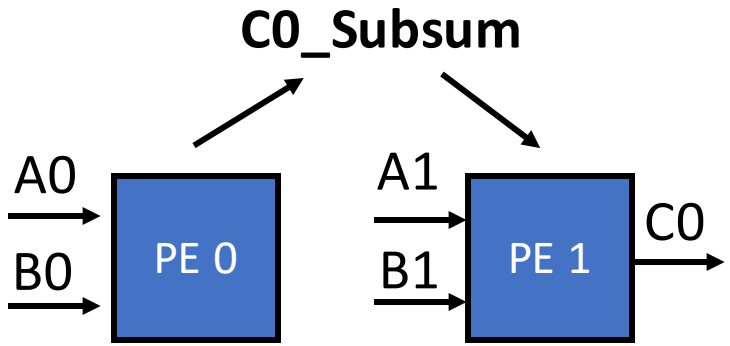


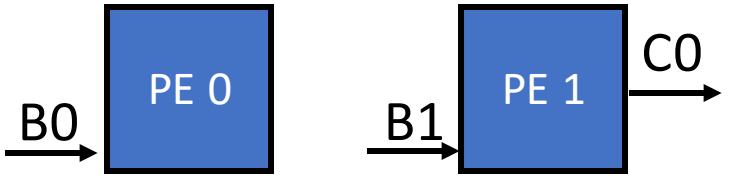


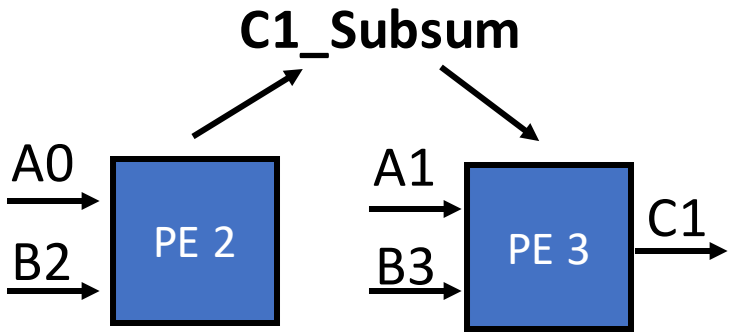
GEMM Example: ARIES MLIR-Based Middle End

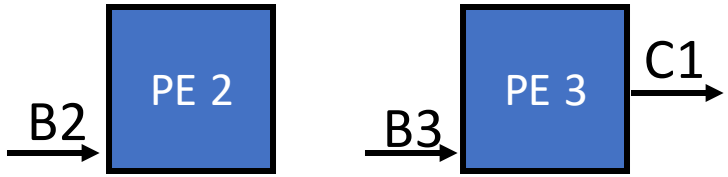
- **Global optimizations**
  - Hardware-agnostic data reuse pattern detections





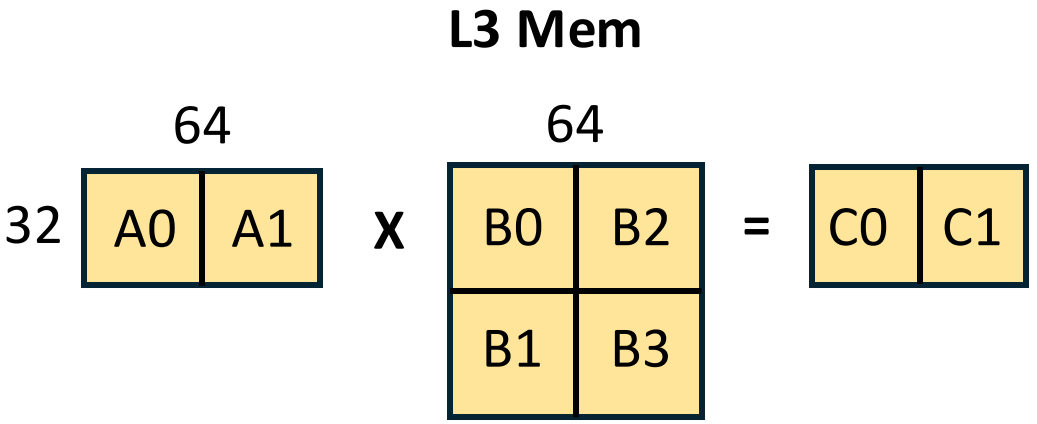


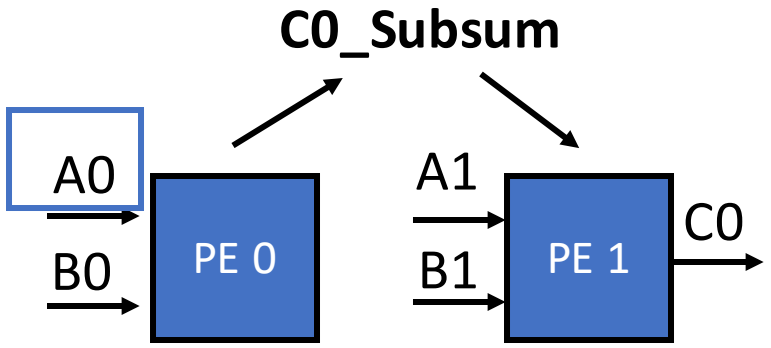


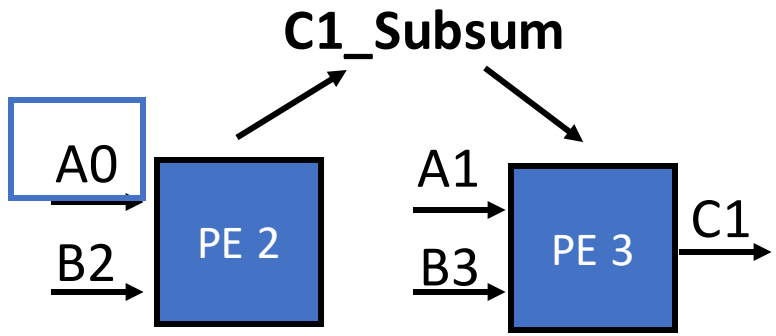


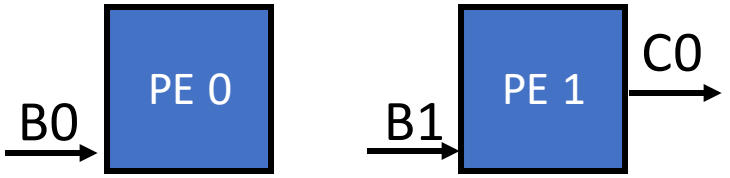
GEMM Example: ARIES MLIR-Based Middle End

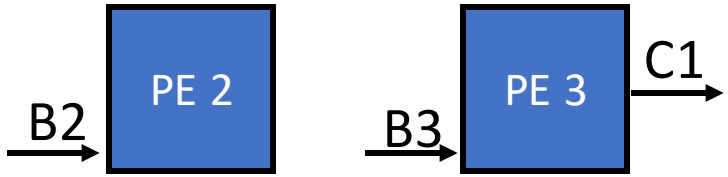
- **Global optimizations**
  - Hardware-agnostic data reuse pattern detections





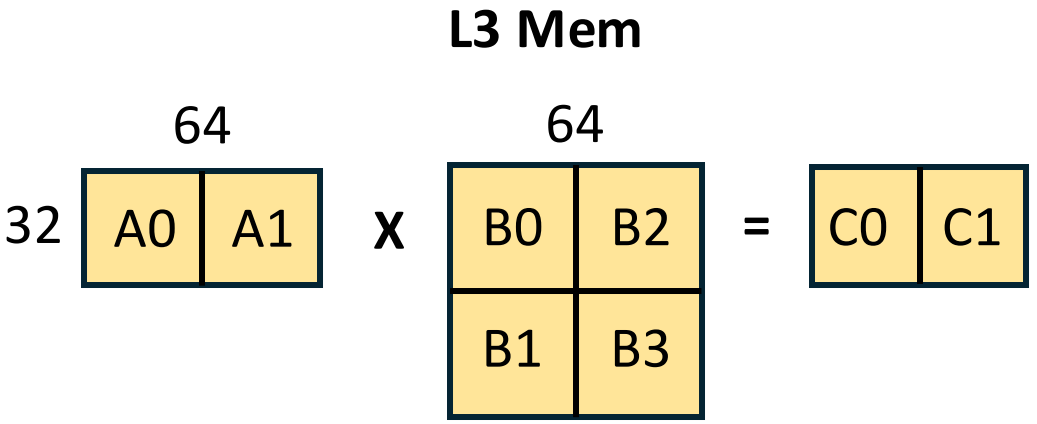


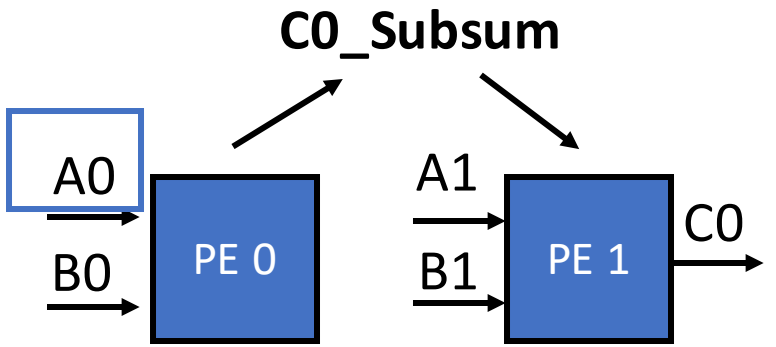


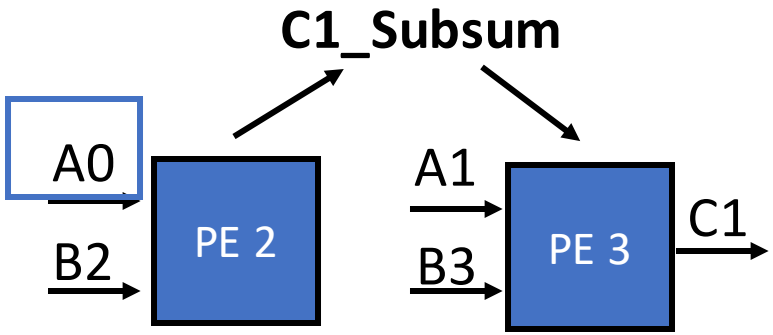


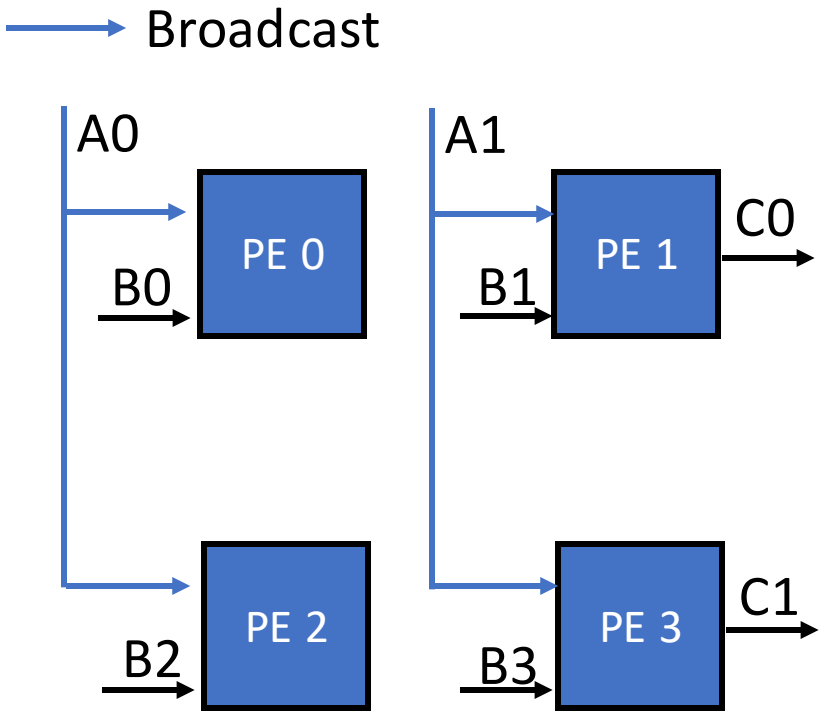
GEMM Example: ARIES MLIR-Based Middle End

- **Global optimizations**
  - Hardware-agnostic data reuse pattern detections



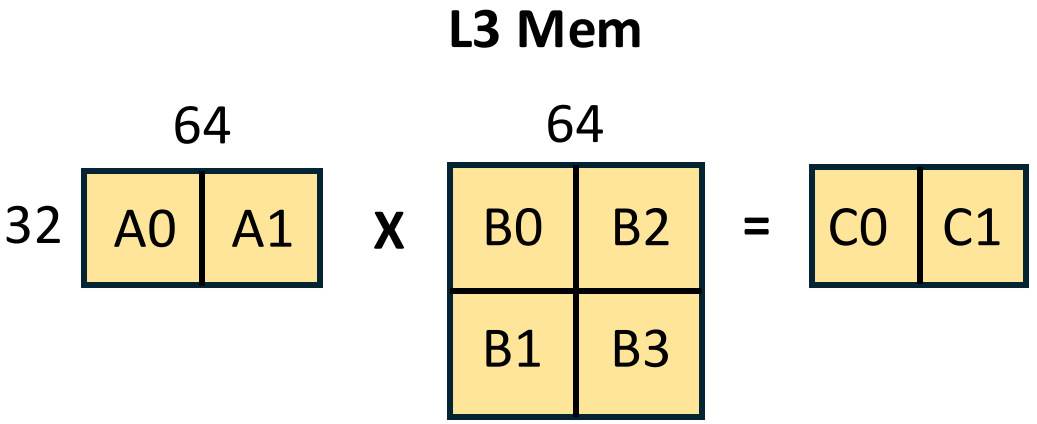


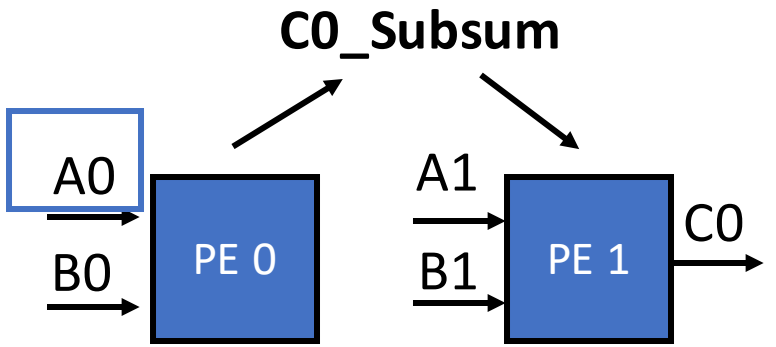


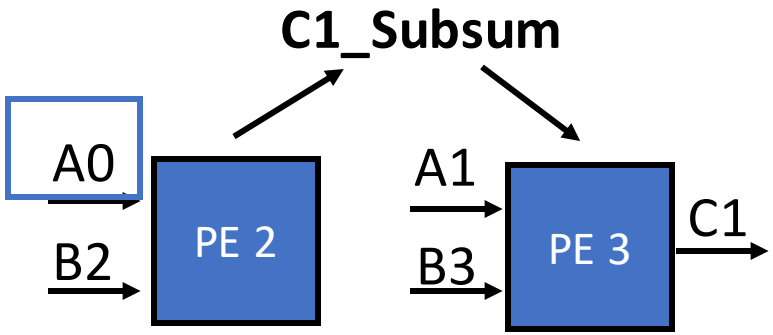


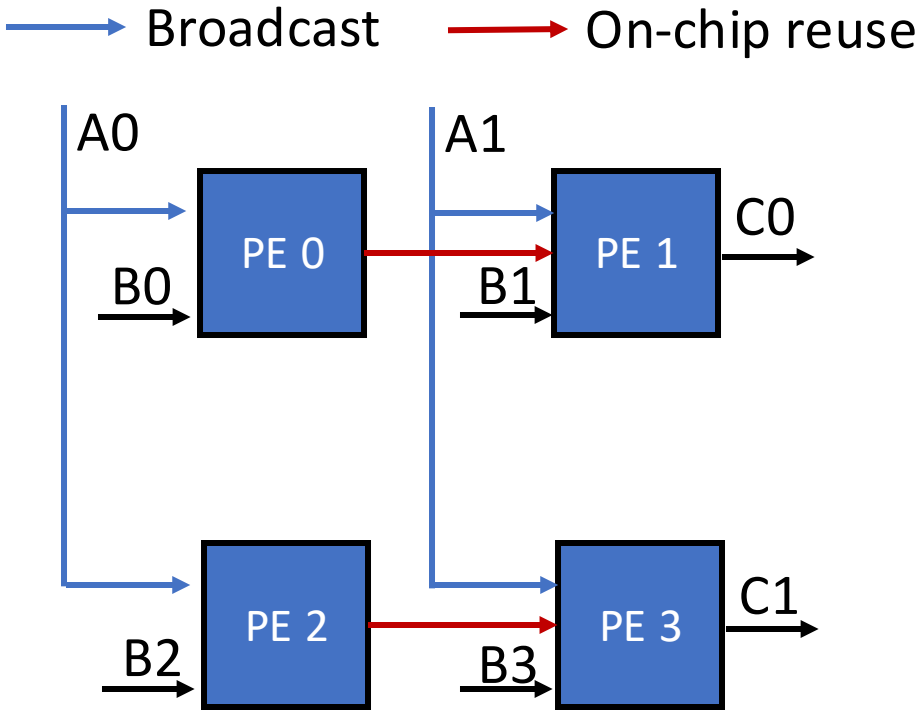
GEMM Example: ARIES MLIR-Based Middle End

- **Global optimizations**
  - Hardware-agnostic data reuse pattern detections









GEMM Example: ARIES MLIR-Based Middle End

GEMM Example: ARIES MLIR-Based Middle End

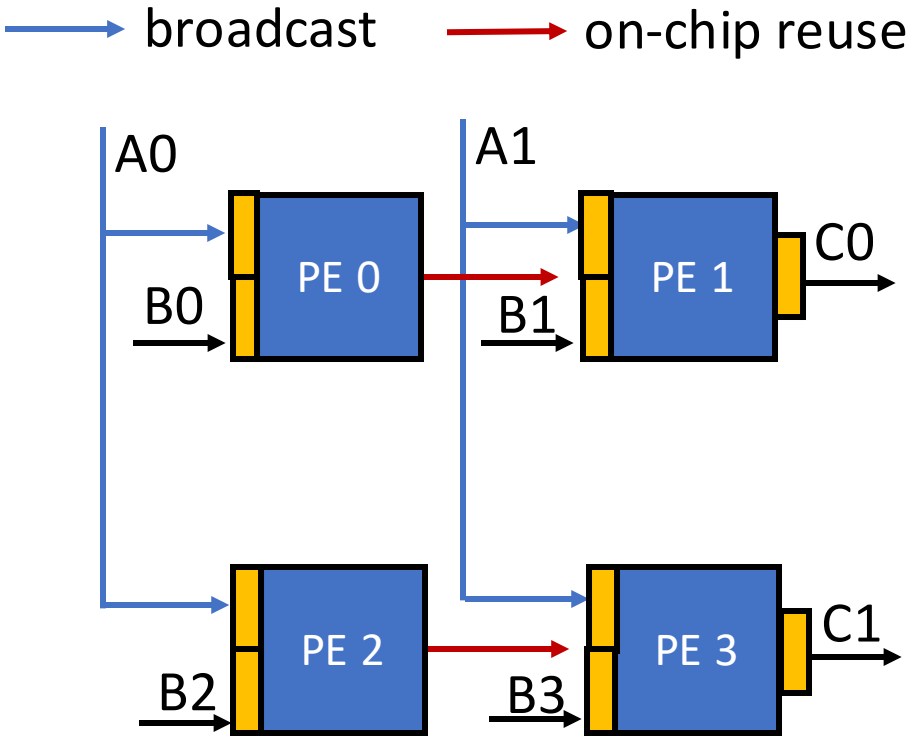
- **Local optimizations & automation**

  Conceptual graph to hardware features

GEMM Example: ARIES MLIR-Based Middle End

- **Local optimizations & automation**

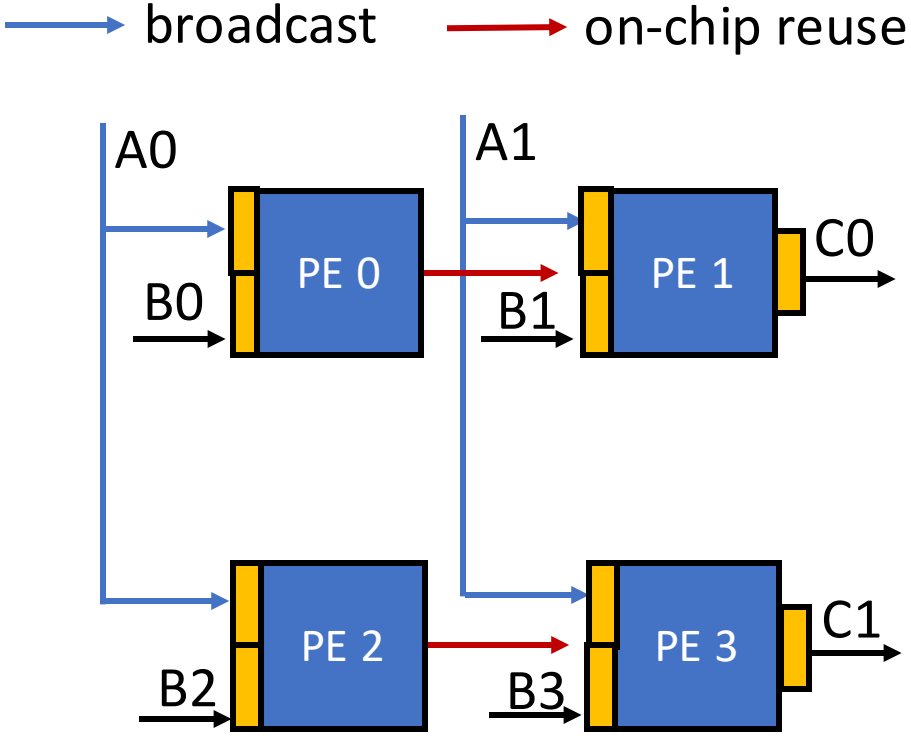
  Conceptual graph to hardware features

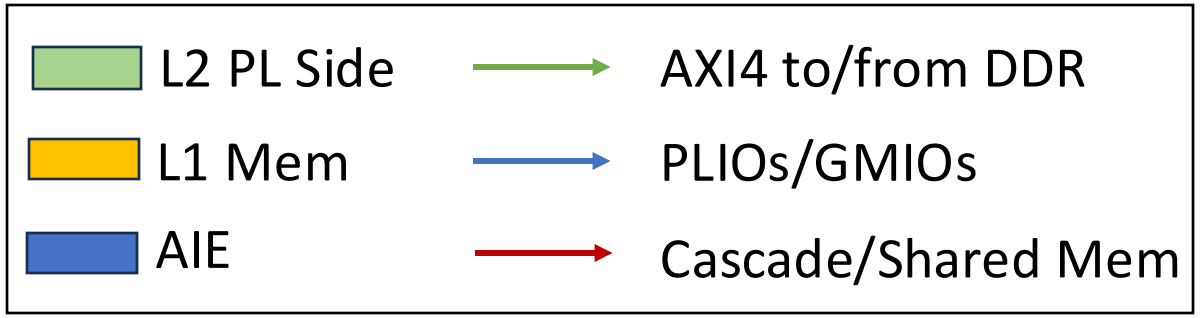


GEMM Example: ARIES MLIR-Based Middle End

- **Local optimizations & automation**

  Conceptual graph to hardware features

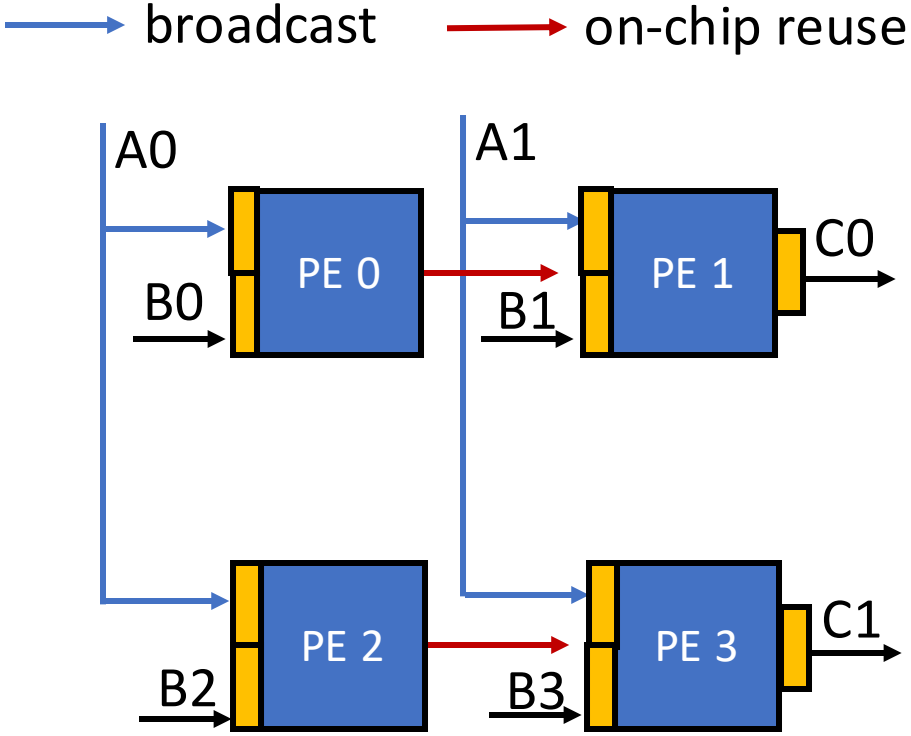


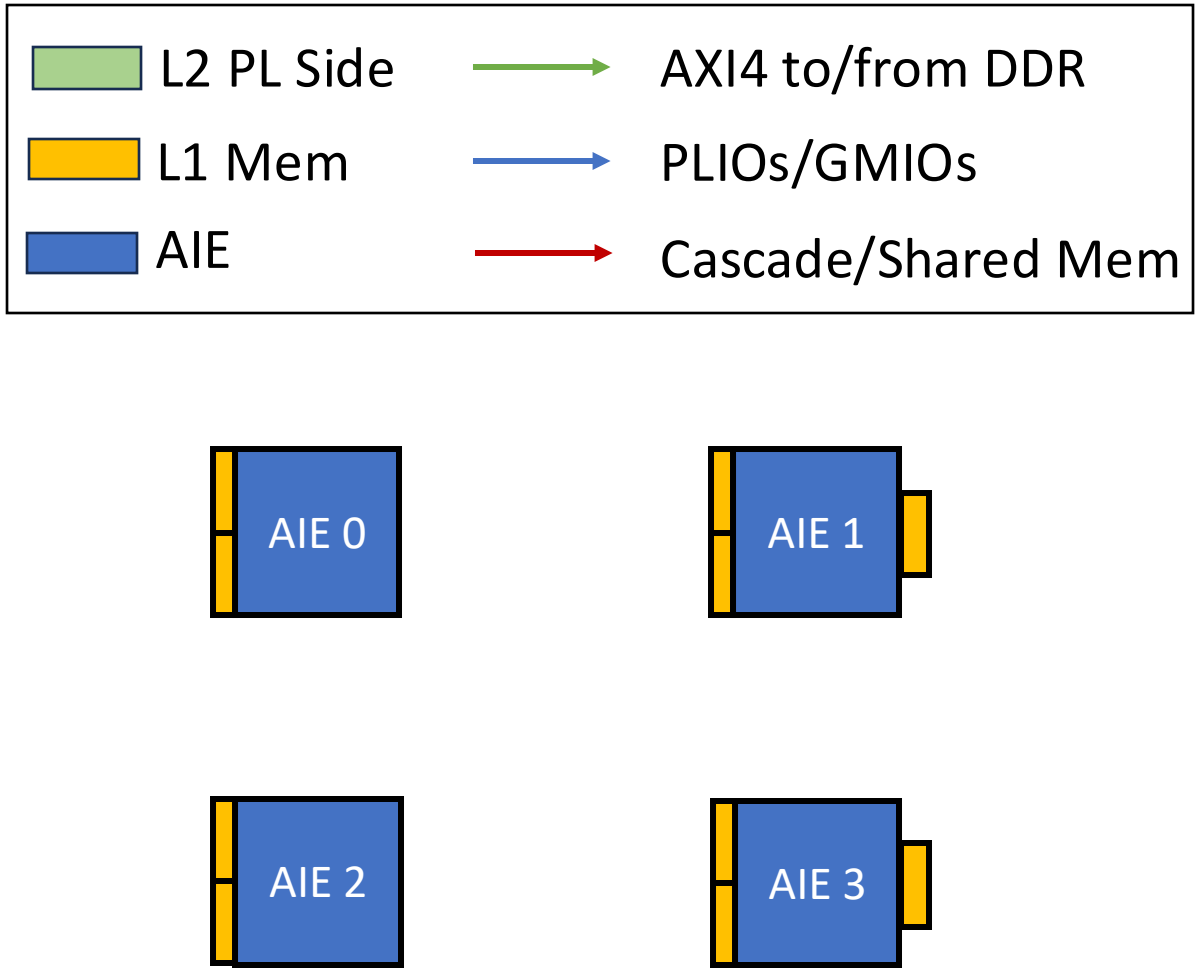


GEMM Example: ARIES MLIR-Based Middle End

- **Local optimizations & automation**

  Conceptual graph to hardware features

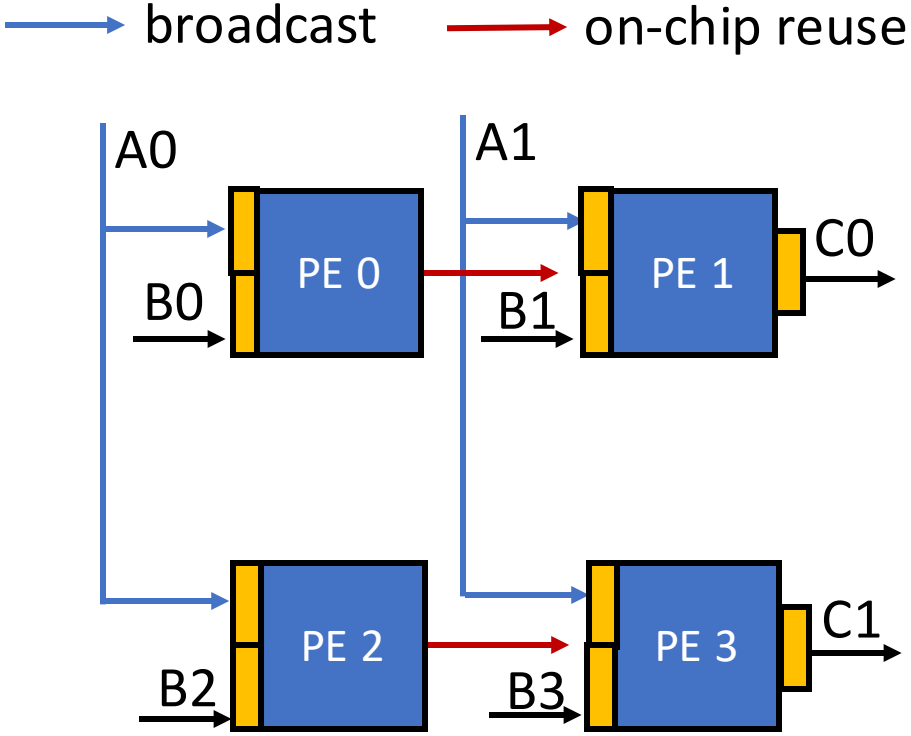




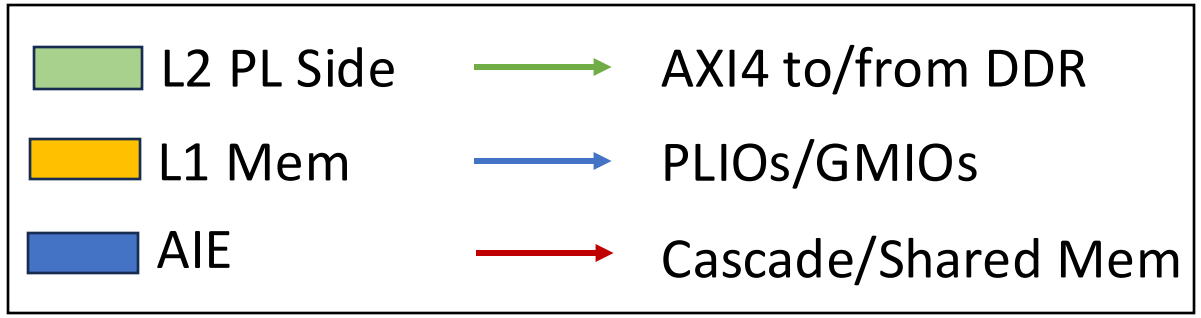
GEMM Example: ARIES MLIR-Based Middle End

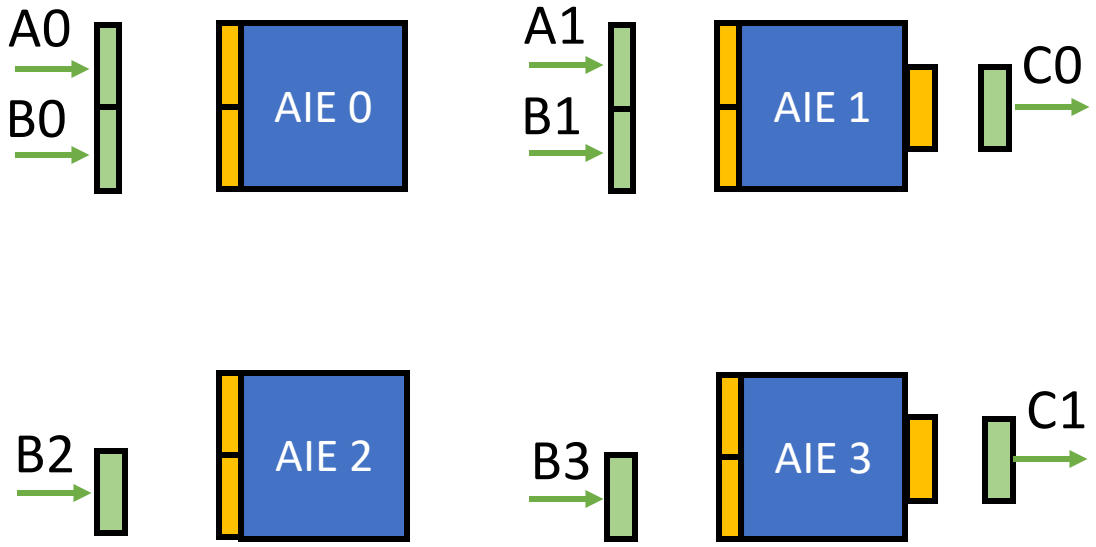
- **Local optimizations & automation**

  Conceptual graph to hardware features



Extract L2 Buffer

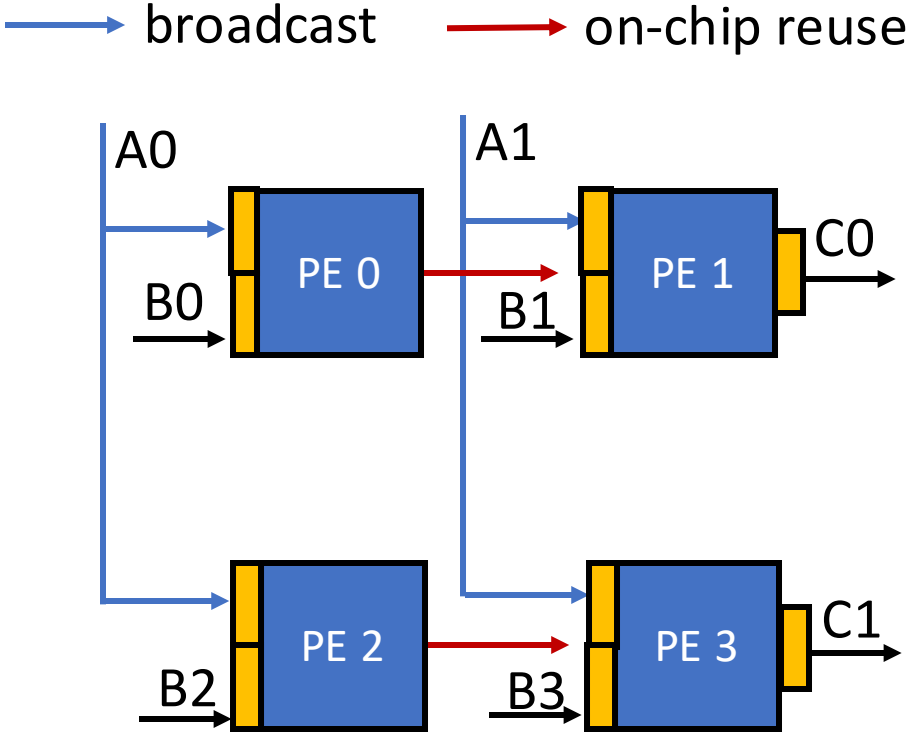




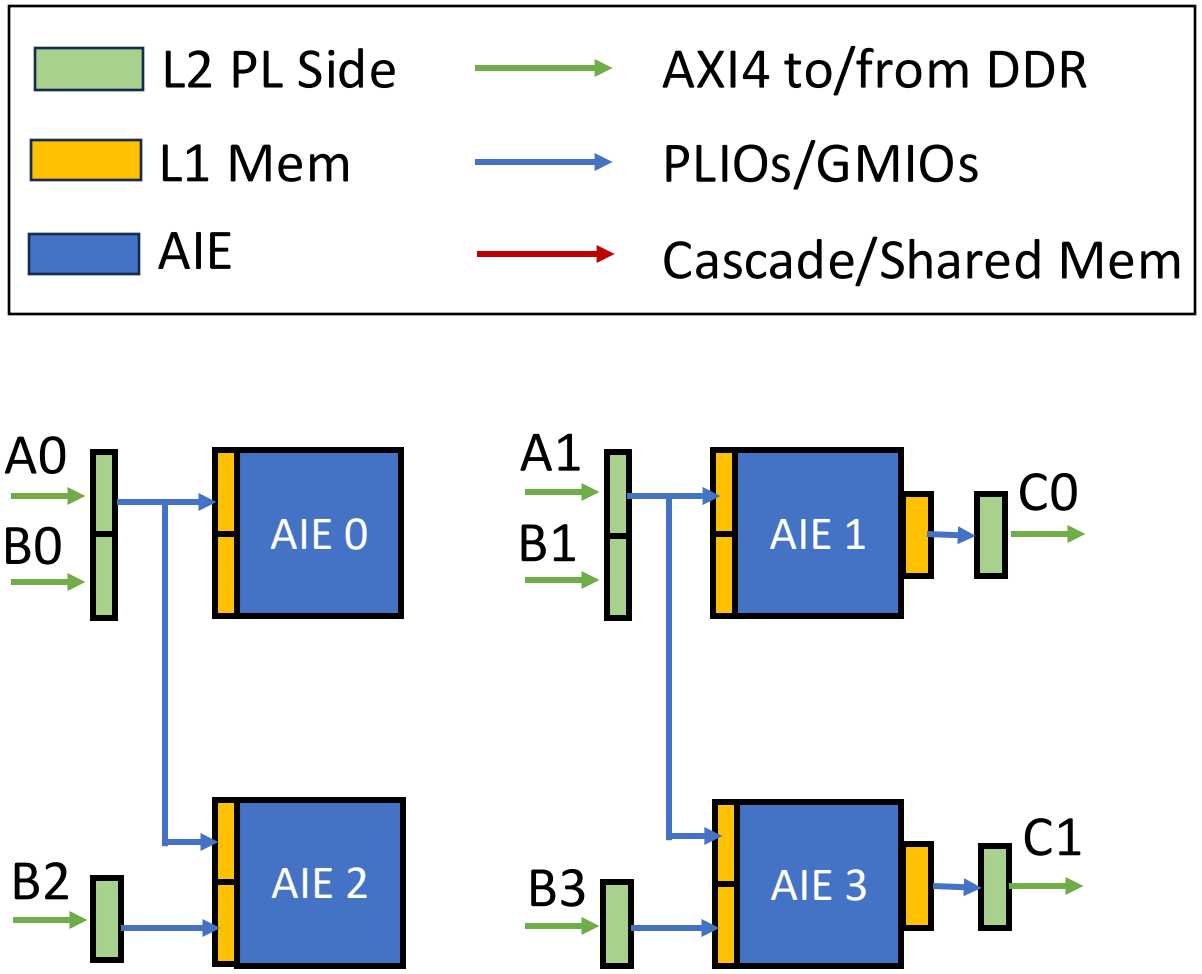
GEMM Example: ARIES MLIR-Based Middle End

- **Local optimizations & automation**

  Conceptual graph to hardware features



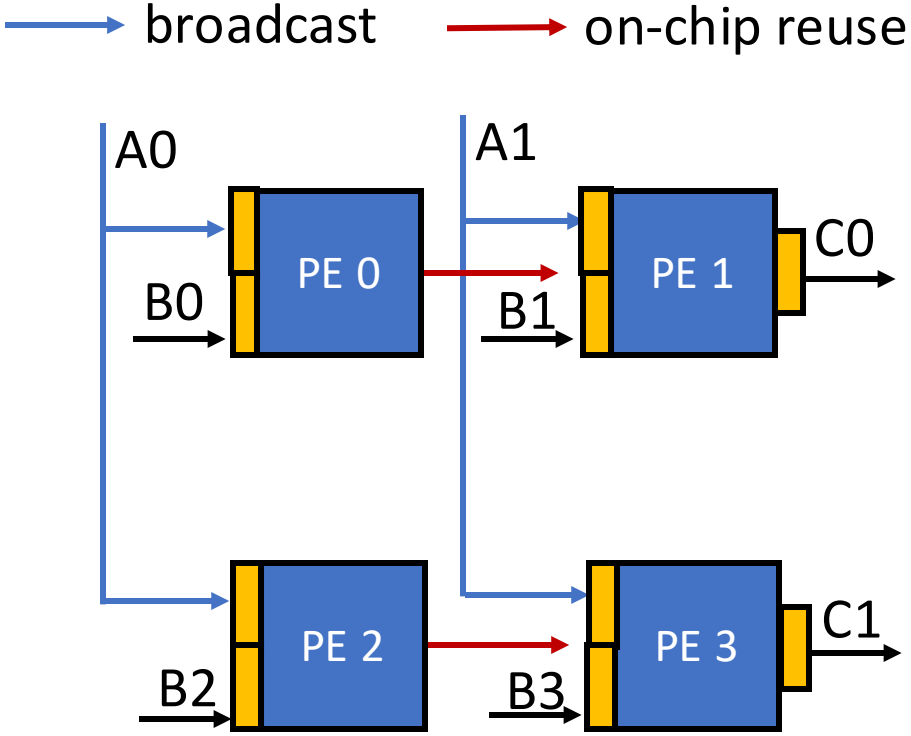
Materialize connections from/to AIE



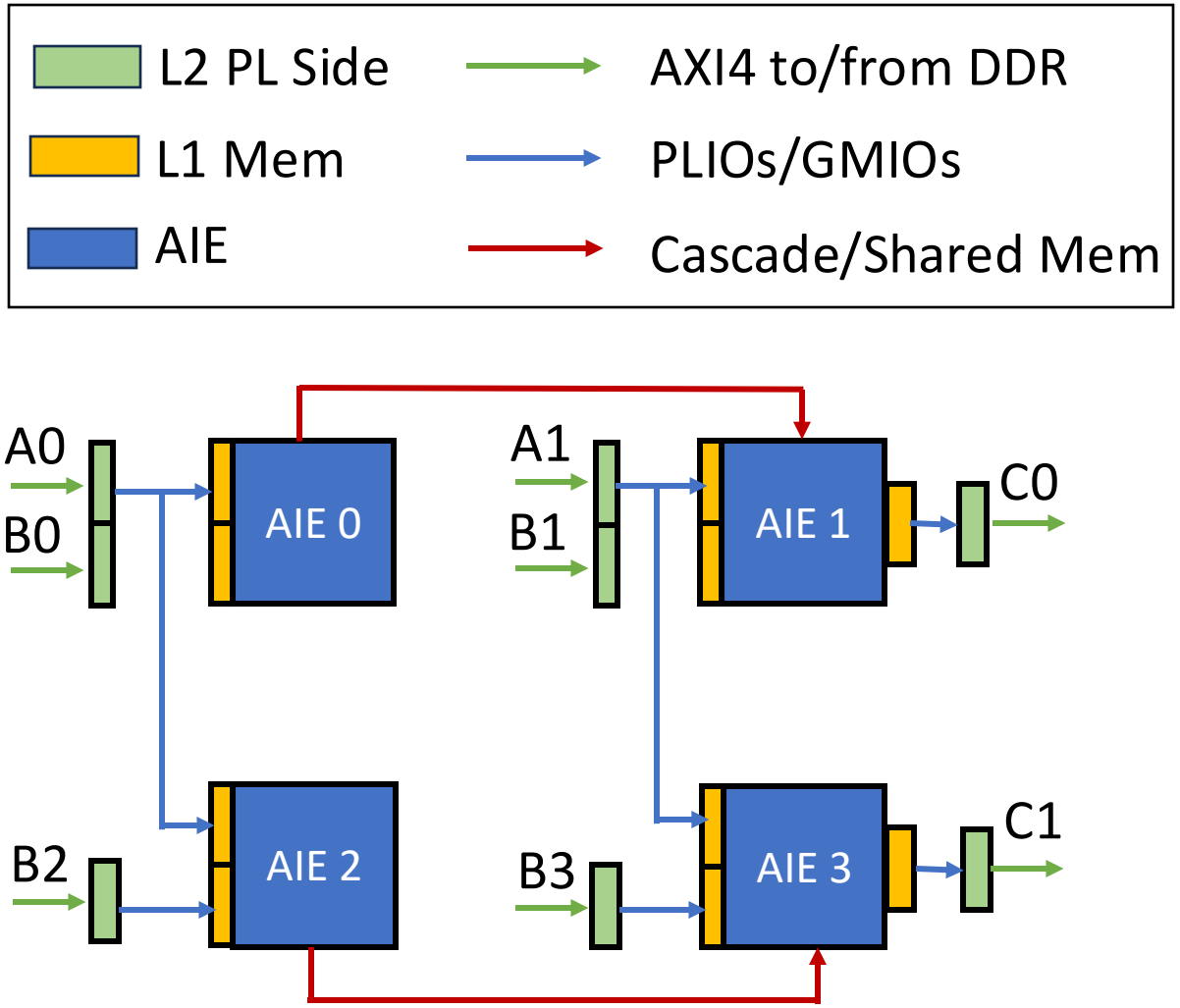
GEMM Example: ARIES MLIR-Based Middle End

- **Local optimizations & automation**

  Conceptual graph to hardware features



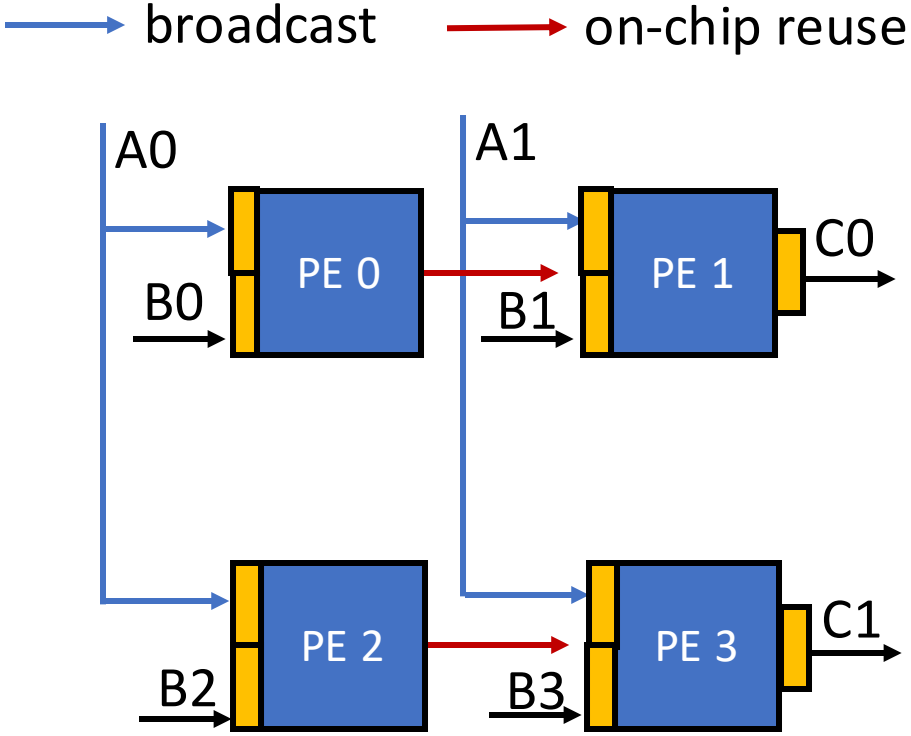
Materialize connections within AIE



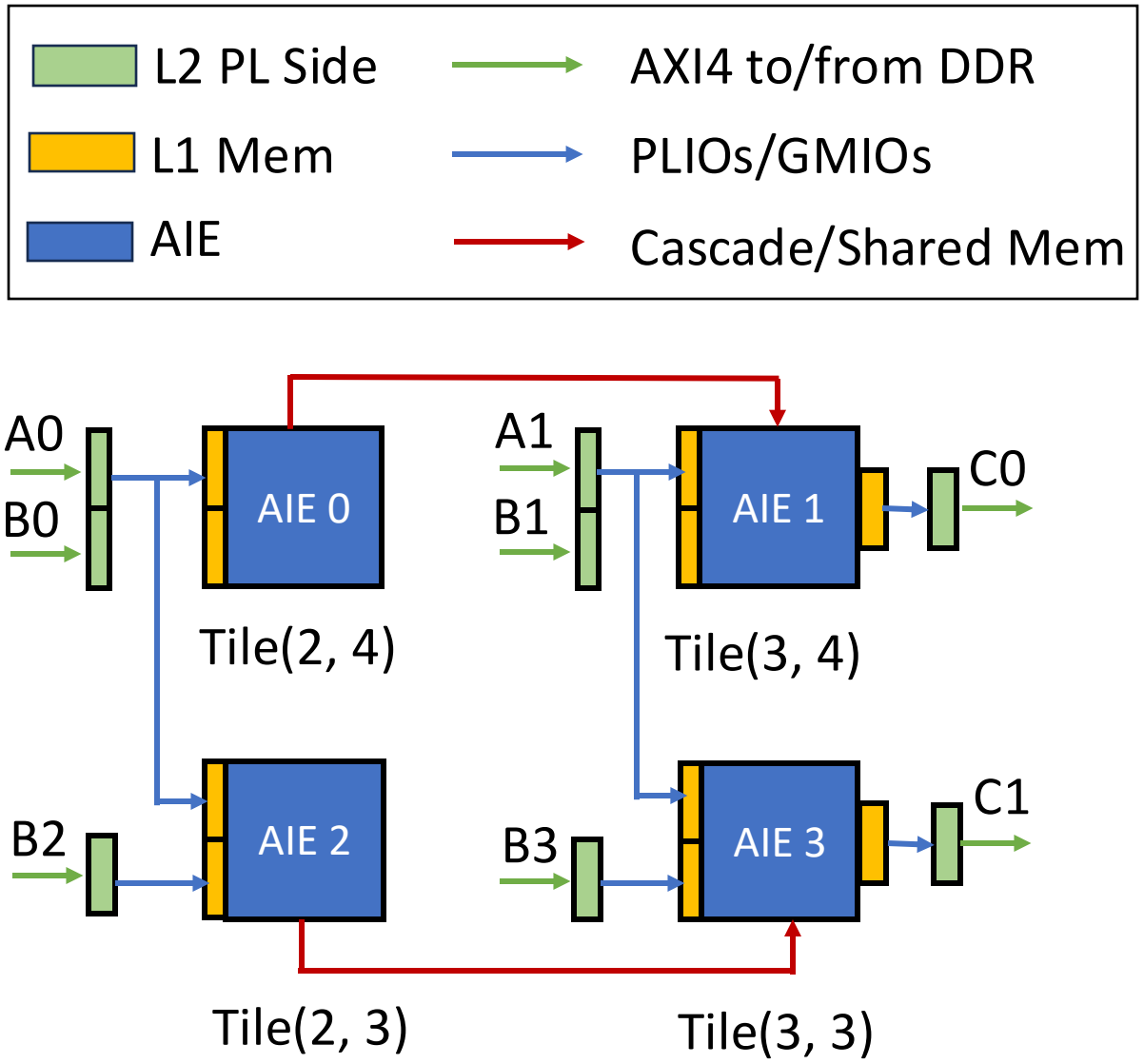
GEMM Example: ARIES MLIR-Based Middle End

- **Local optimizations & automation**

  Conceptual graph to hardware features



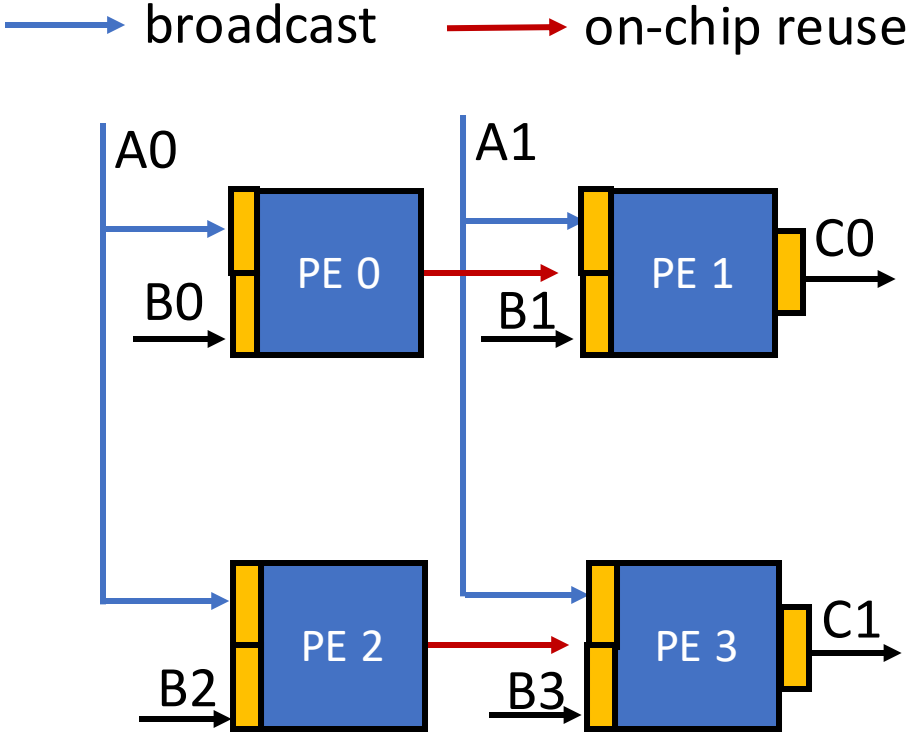
Core placement



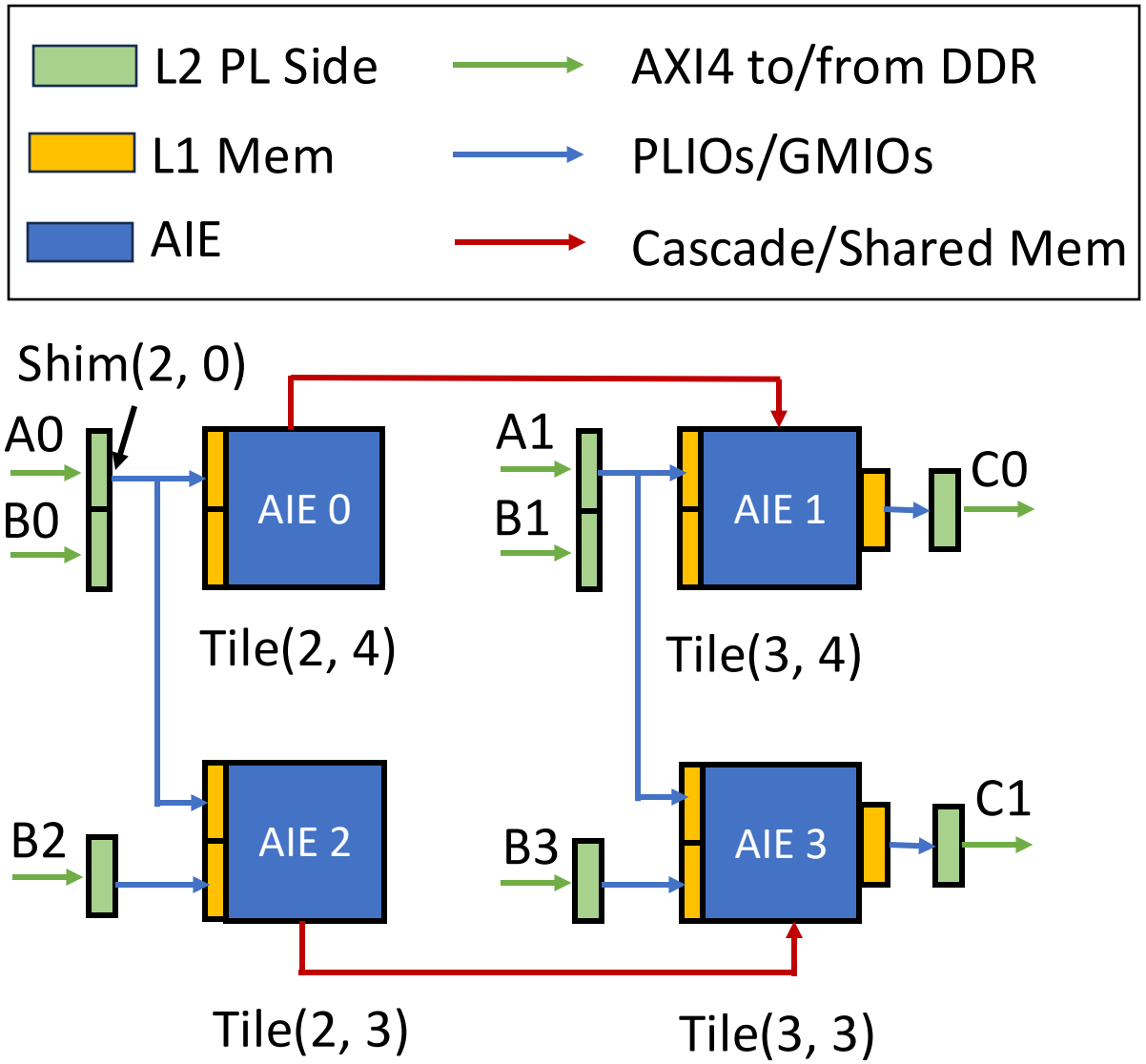
GEMM Example: ARIES MLIR-Based Middle End

- **Local optimizations & automation**

  Conceptual graph to hardware features



ShimDMA/IO placement



Experiment Setup

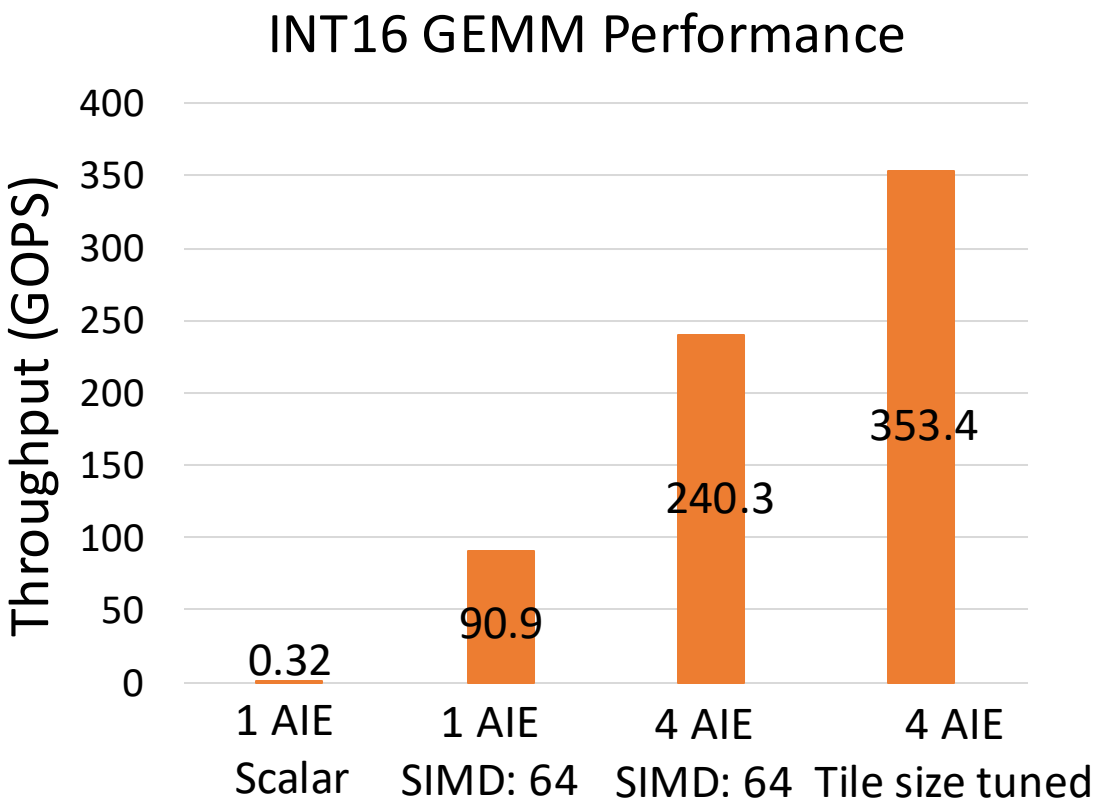
Specification	NPU	Versal
Platform	AMD Ryzen™ 9 7940HS CPU	AMD Versal 7nm VCK190
Frequency	AIE-ML-1L@1GHz	AIE@1GHz, PL@220MHz
Software Tools	MLIR-AIE, Vitis 2023.2	Vitis 2023.1
AIEs	20 AIE-MLs	400 AIEs (no Mem-tiles)
L1 Memory	$20 \times 64\text{KB} = 1.25\text{MB}$	$400 \times 32\text{KB} = 12.5\text{MB}$
L2 Memory	$5 \times 512\text{KB}$ Mem-tiles	~20MB SRAMs in PL side

Experiment Results

- NPU Backend: Performance and Lines of Code Comparison for Constructing AIE Array

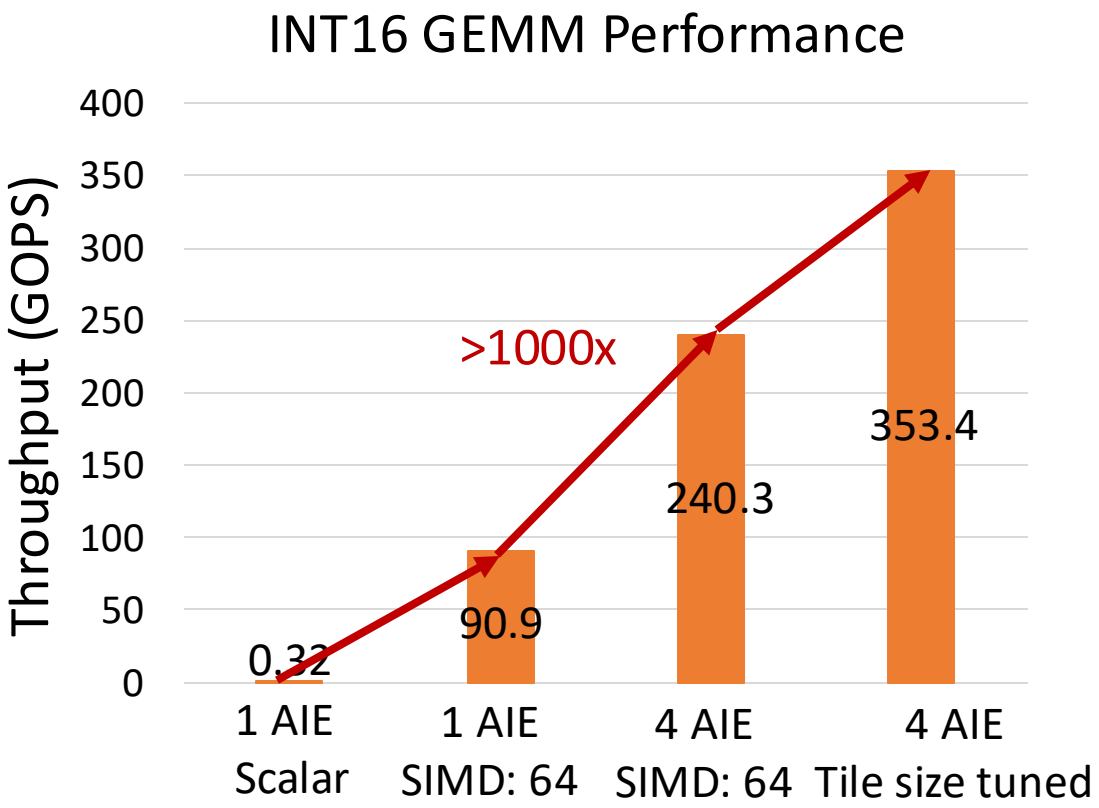
Experiment Results

- NPU Backend: Performance and Lines of Code Comparison for Constructing AIE Array



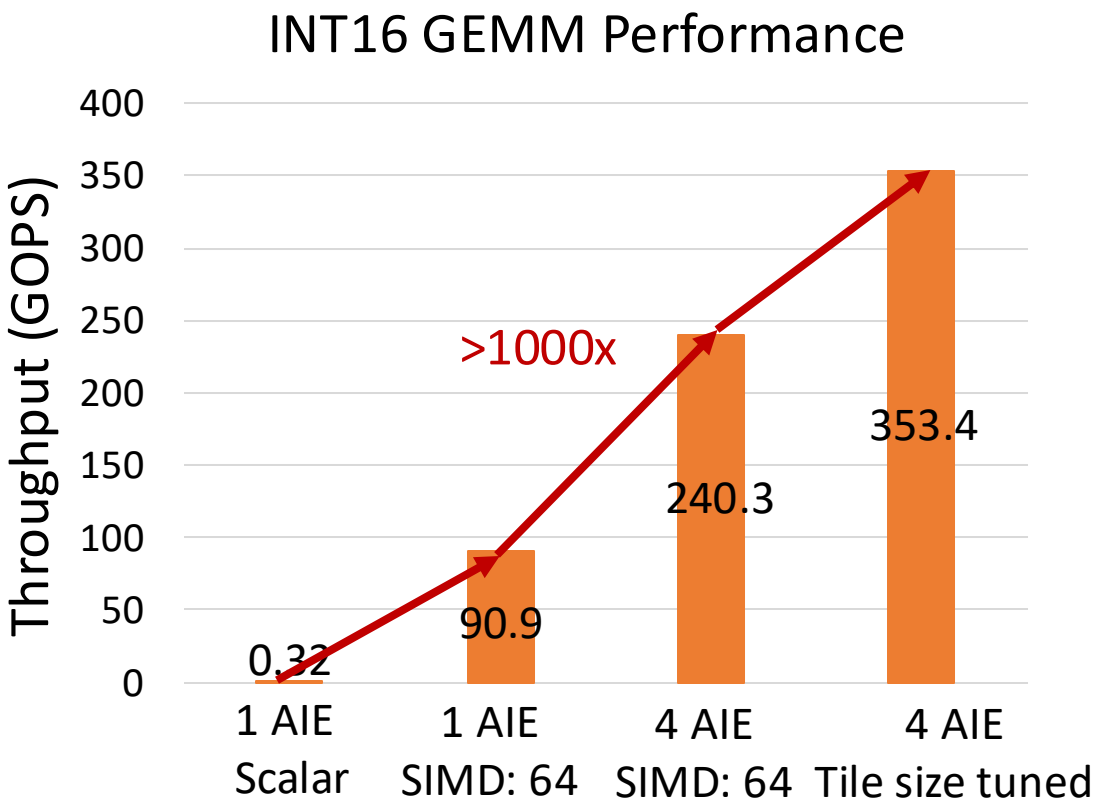
Experiment Results

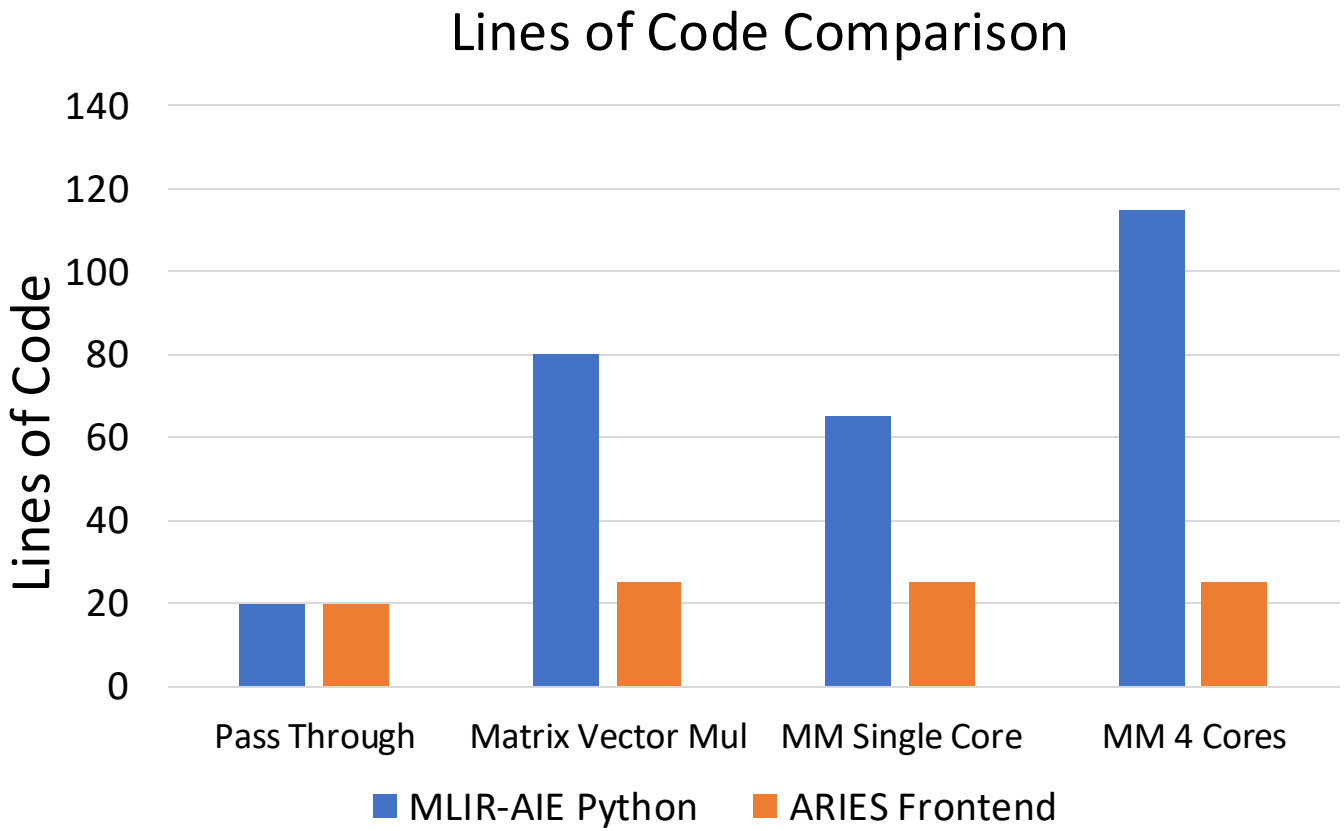
- NPU Backend: Performance and Lines of Code Comparison for Constructing AIE Array



Experiment Results

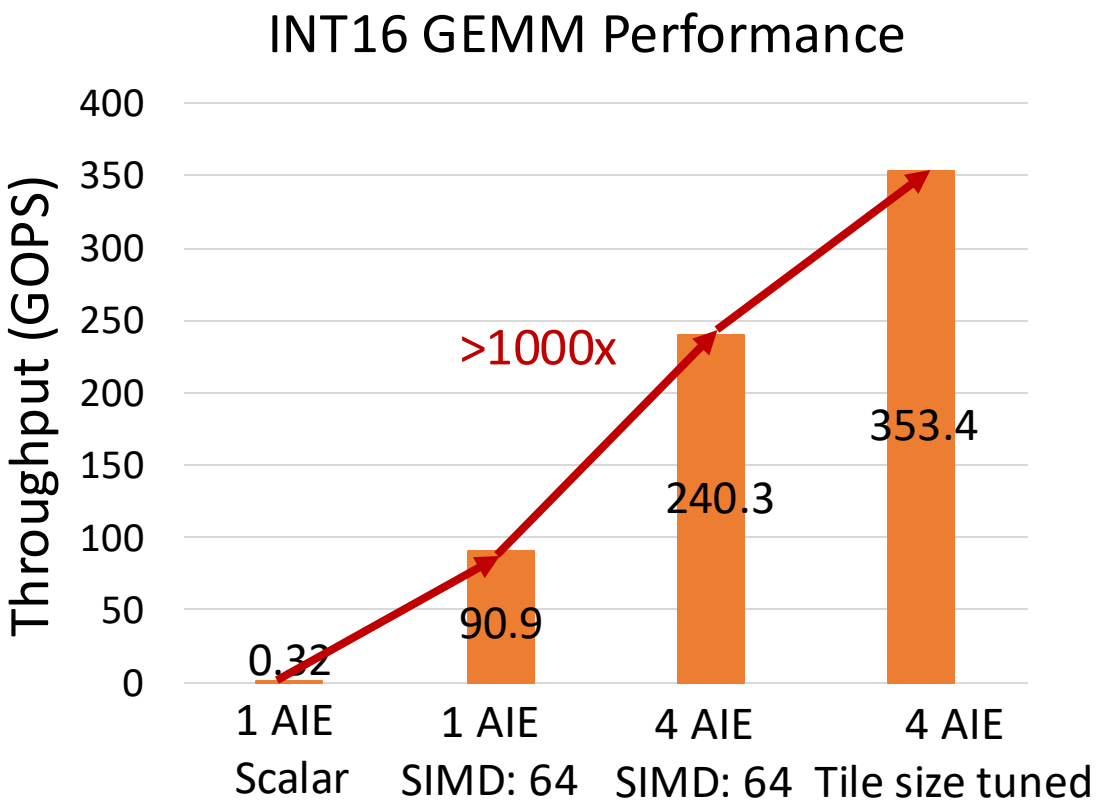
- NPU Backend: Performance and Lines of Code Comparison for Constructing AIE Array

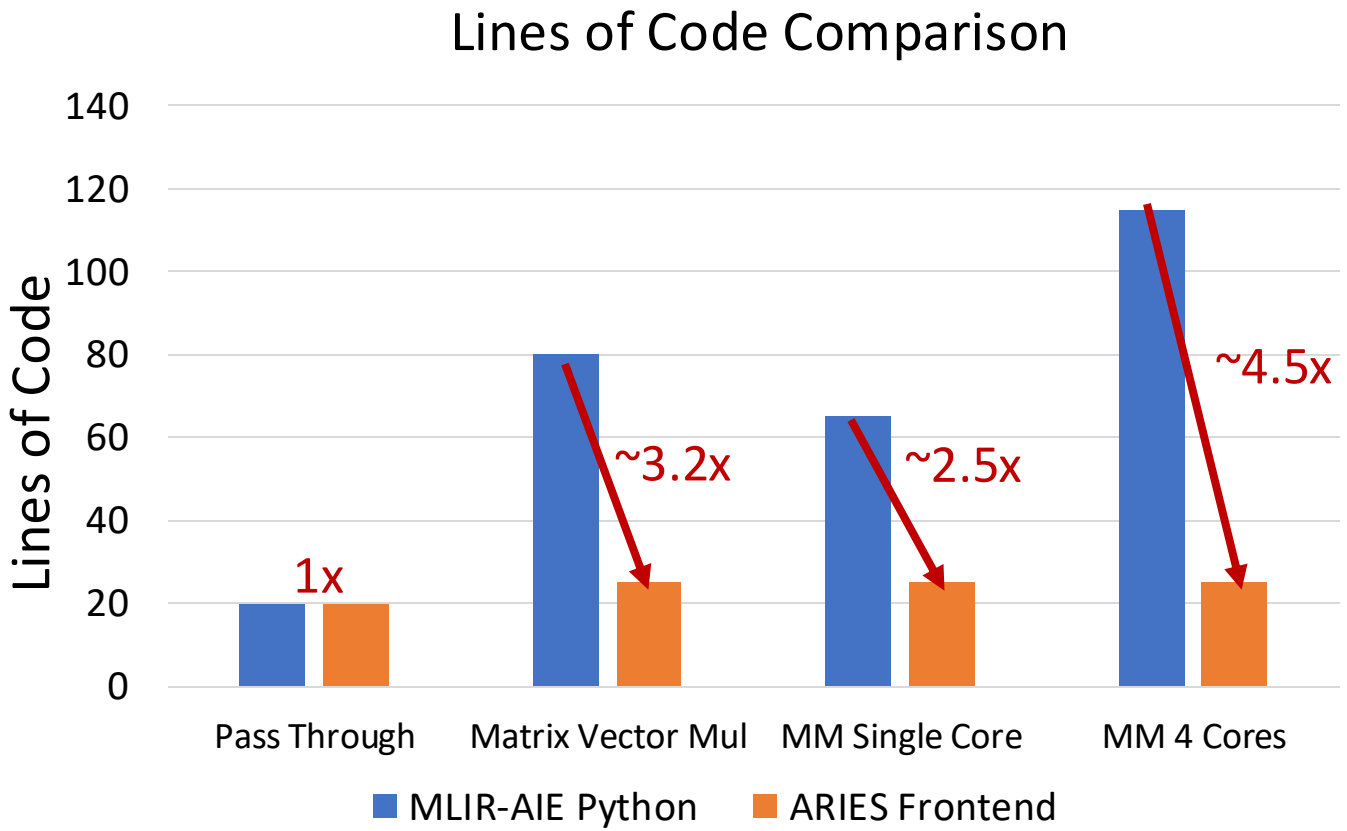




Experiment Results

- NPU Backend: Performance and Lines of Code Comparison for Constructing AIE Array

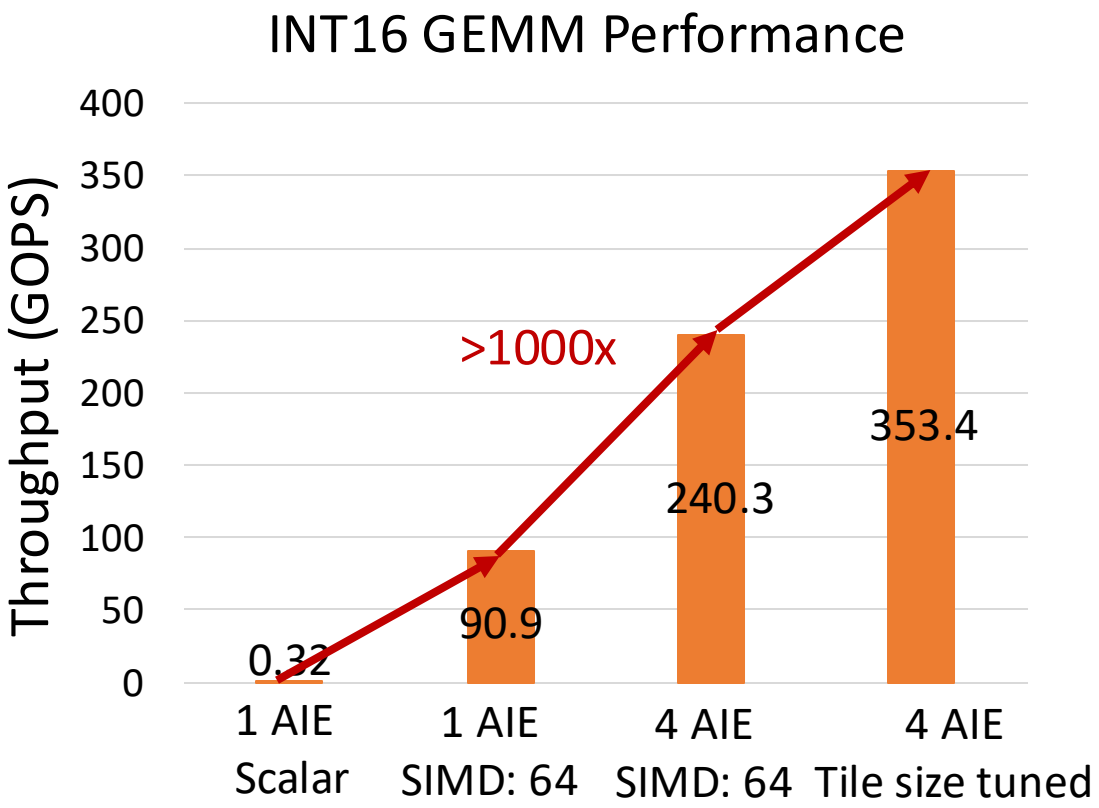


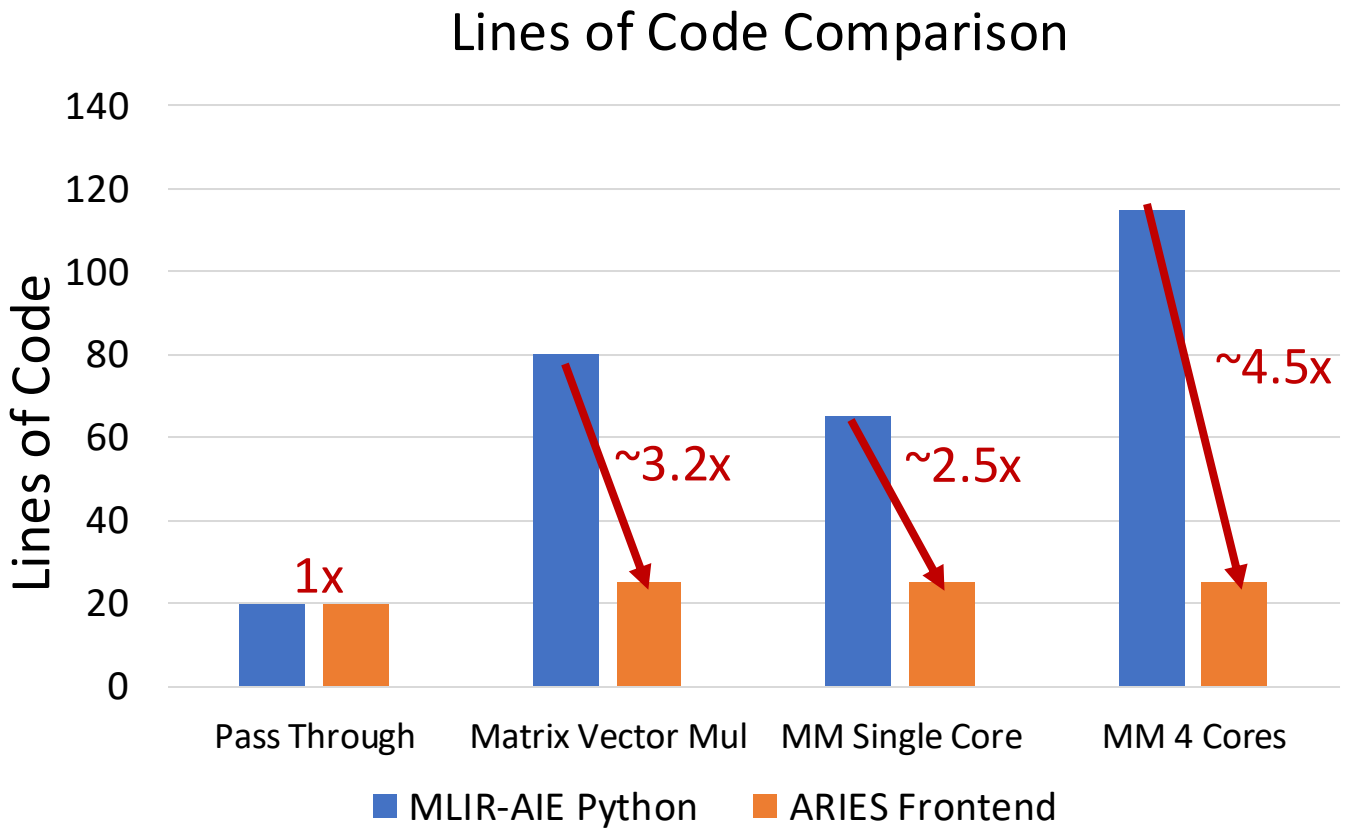


Experiment Results

- NPU Backend: Performance and Lines of Code Comparison for Constructing AIE Array

  ARIES provides simplified abstractions for data movement



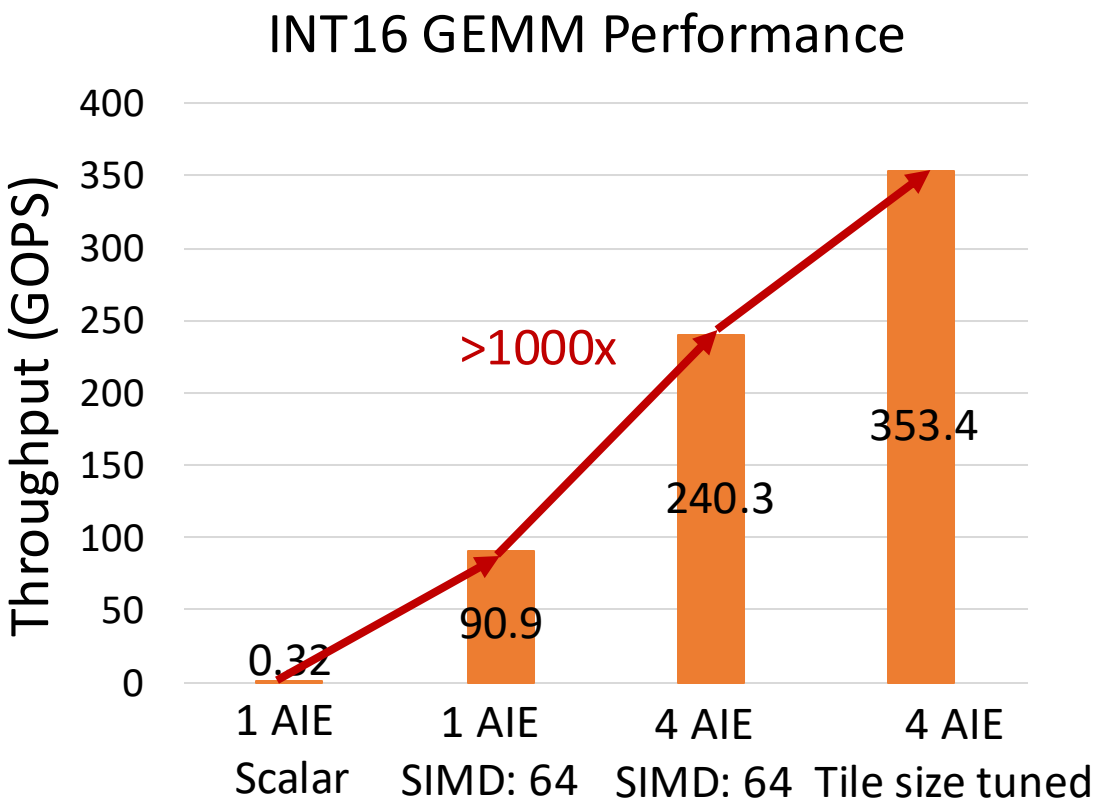


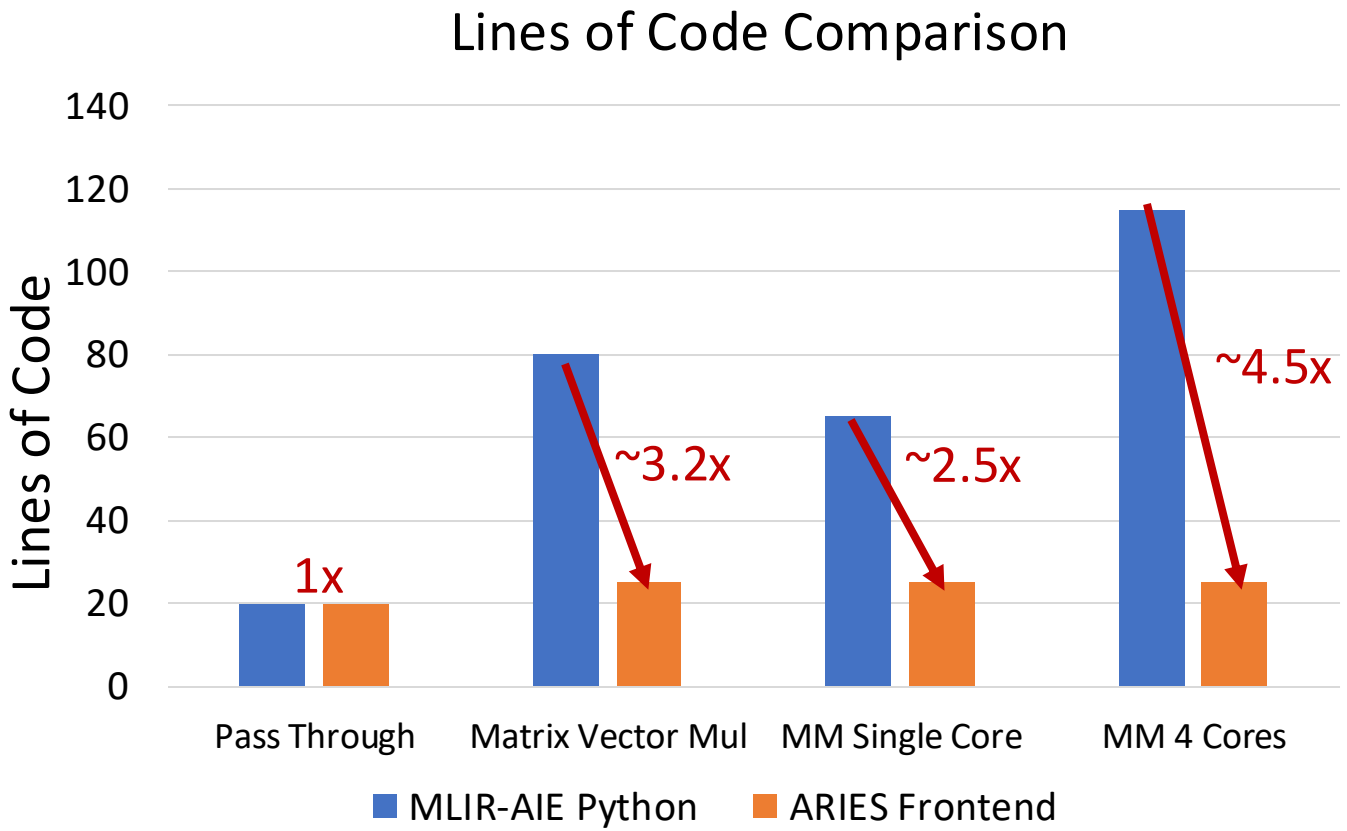
Experiment Results

- NPU Backend: Performance and Lines of Code Comparison for Constructing AIE Array

  ARIES provides simplified abstractions for data movement

  Users are free of the detailed hardware controls (Tiles, locks, bd_ids, ShimDMAs)



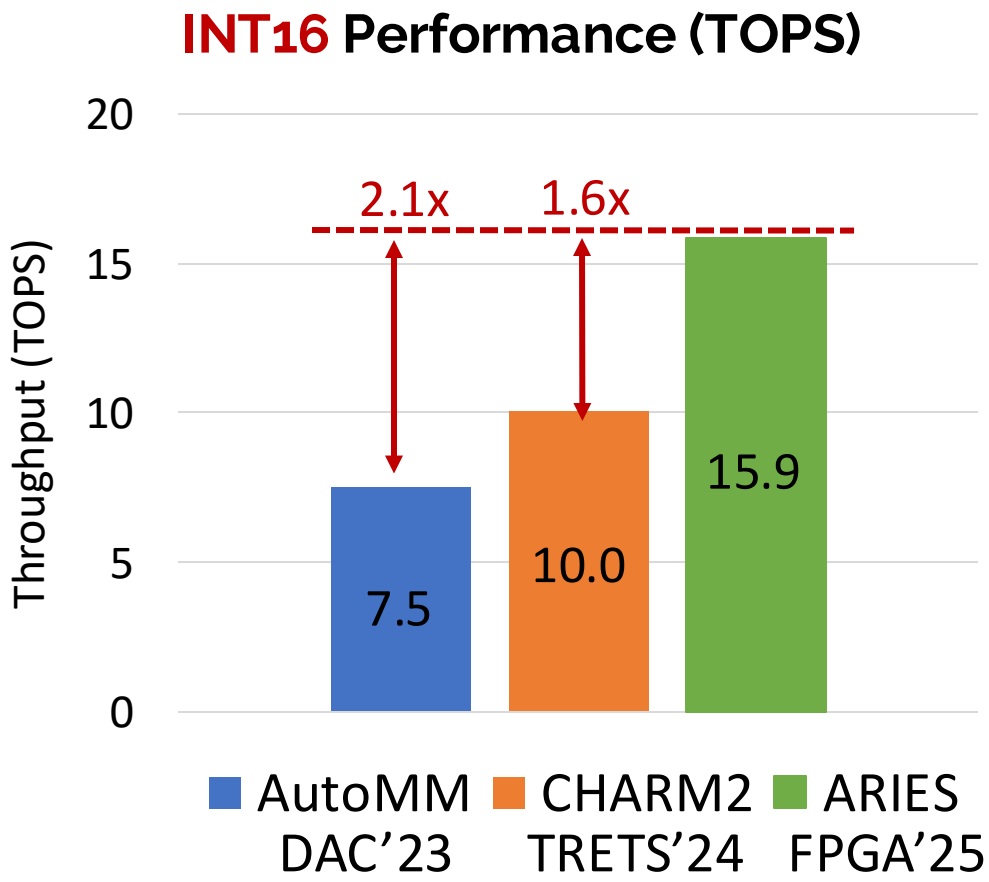


Experiment Results

- Versal Backend: GEMM Performance

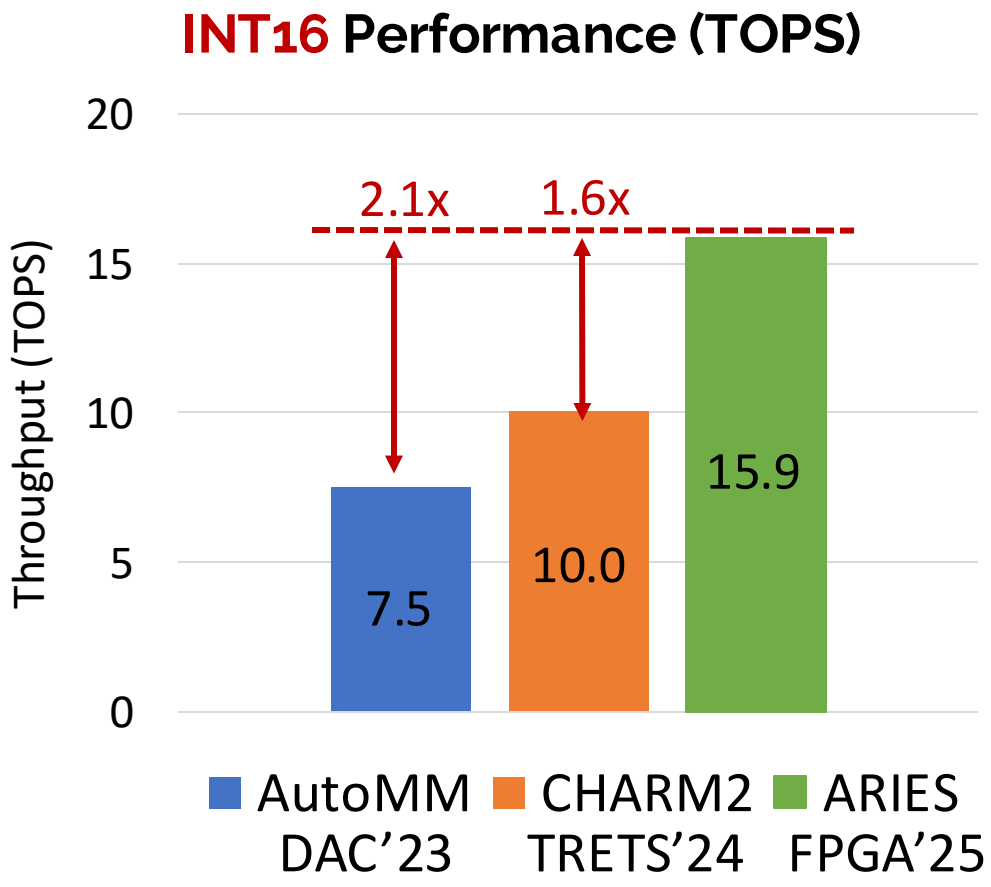
Experiment Results

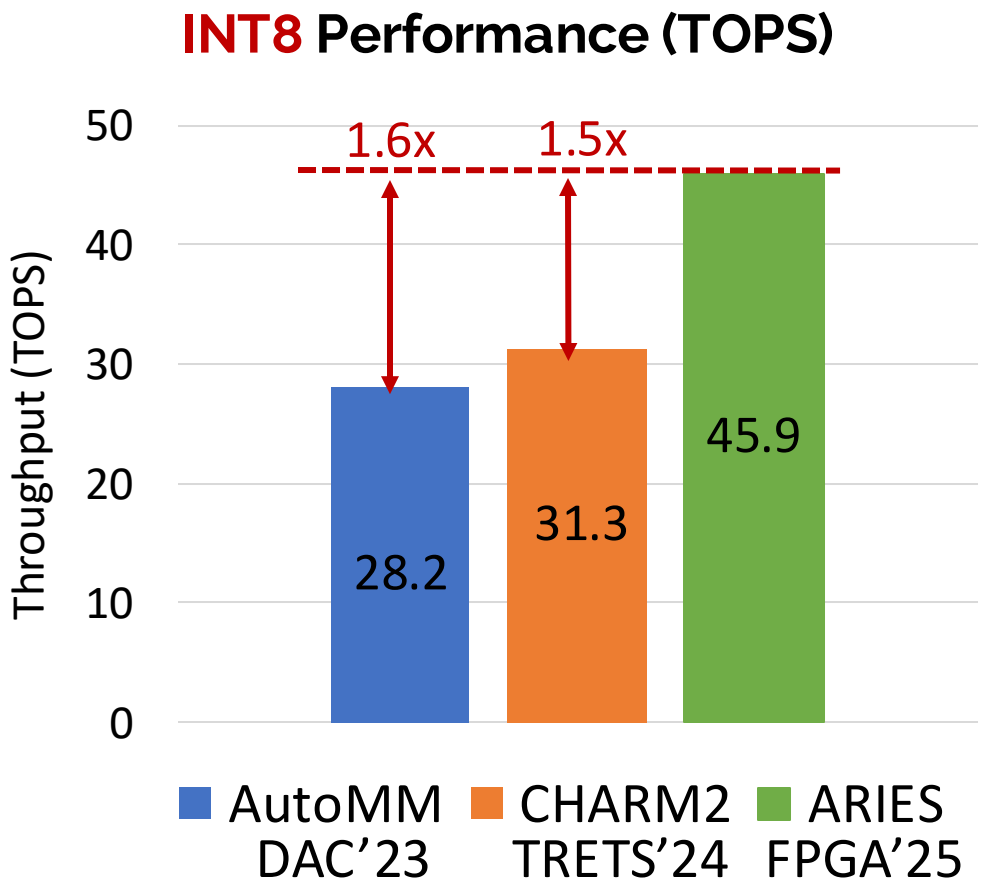
- Versal Backend: GEMM Performance



Experiment Results

- Versal Backend: GEMM Performance



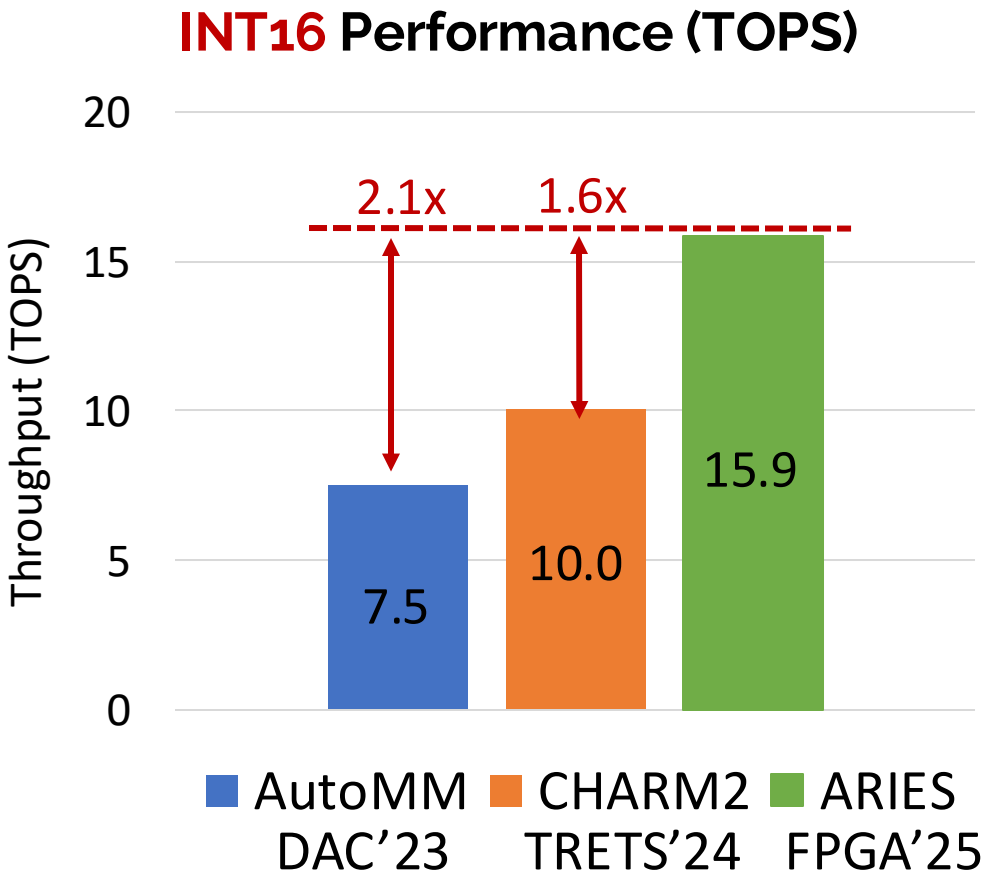


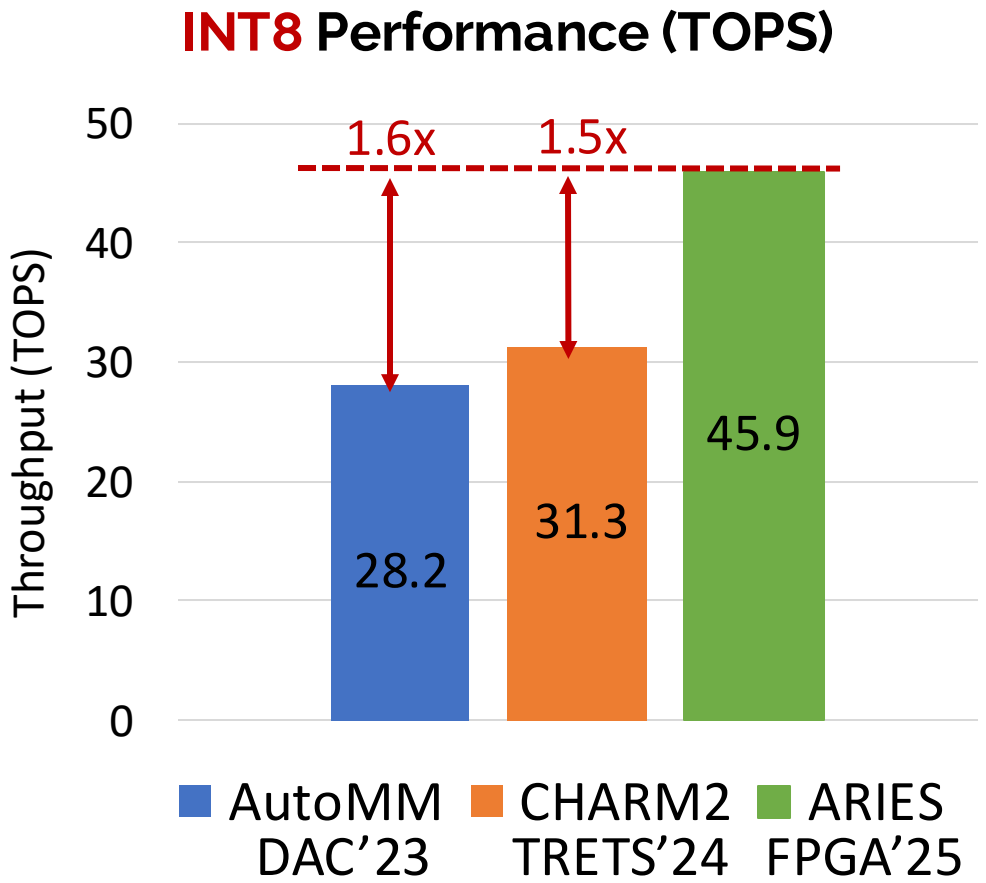
Experiment Results

- Versal Backend: GEMM Performance

15.9 INT16 TOPS, 2.1x over AutoMM

45.9 INT8 TOPS, 1.6x over AutoMM





Experiment Results

- Versal Backend: GEMM Performance

Experiment Results

- Versal Backend: GEMM Performance

① Higher Comm. Efficiency between AIE & PL

- Finer-grained data transfer mechanism & wider PLIO port width are applied

Experiment Results

- Versal Backend: GEMM Performance

① Higher Comm. Efficiency between AIE & PL

- Finer-grained data transfer mechanism & wider PLIO port width are applied

② More efficient AIE core and IO placement

DType	Work	PLIOs	AIEs	TOPS
INT16	**ARIES**	**164**	**352 (88%)**	**15.86**
	CHARM	120	288 (72%)	10.03
	AutoMM	120	288 (72%)	7.51
INT8	**ARIES**	**152**	**320 (80%)**	**45.94**
	CHARM	104	192 (48%)	31.31
	AutoMM	104	192 (48%)	28.15

Experiment Results

- Versal Backend: GEMM Performance

① Higher Comm. Efficiency between AIE & PL

- Finer-grained data transfer mechanism & wider PLIO port width are applied

- The unified representation enables hardware aware optimizations

→ Config IO in AIE & Corresponding adjustment in PL

② More efficient AIE core and IO placement

DType	Work	PLIOs	AIEs	TOPS
INT16	**ARIES**	**164**	**352 (88%)**	**15.86**
	CHARM	120	288 (72%)	10.03
	AutoMM	120	288 (72%)	7.51
INT8	**ARIES**	**152**	**320 (80%)**	**45.94**
	CHARM	104	192 (48%)	31.31
	AutoMM	104	192 (48%)	28.15

Experiment Results

- Versal Backend: GEMM Performance

① Higher Comm. Efficiency between AIE & PL

- Finer-grained data transfer mechanism & wider PLIO port width are applied

→ The unified representation enables hardware aware optimizations

→ Config IO in AIE & Corresponding adjustment in PL

② More efficient AIE core and IO placement

DType	Work	PLIOs	AIEs	TOPS
INT16	**ARIES**	**164**	**352 (88%)**	**15.86**
	CHARM	120	288 (72%)	10.03
	AutoMM	120	288 (72%)	7.51
INT8	**ARIES**	**152**	**320 (80%)**	**45.94**
	CHARM	104	192 (48%)	31.31
	AutoMM	104	192 (48%)	28.15

→ ARIES end-to-end flow provides good extensibility and reusability

→ Optimizations can be implemented by adding or replacing a single pass in ARIES with other automations reused

Experiment Results

- Versal Backend: Multilayer perceptron (MLP)

Experiment Results

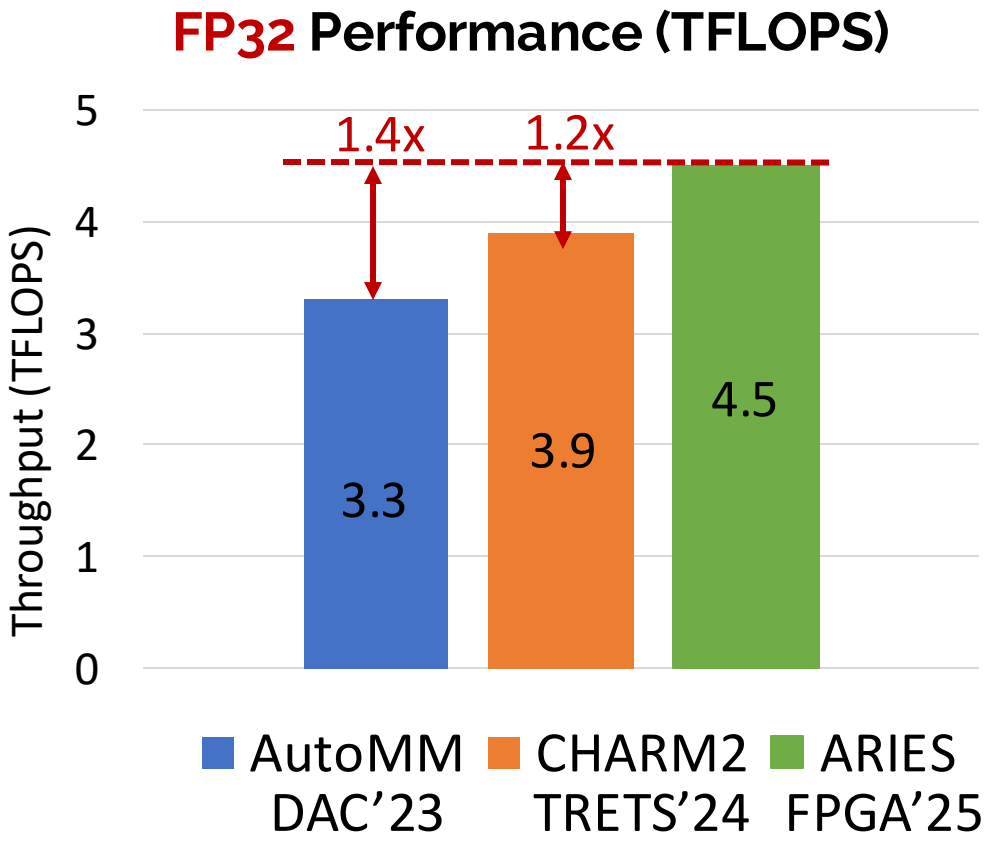
- Versal Backend: Multilayer perceptron (MLP)

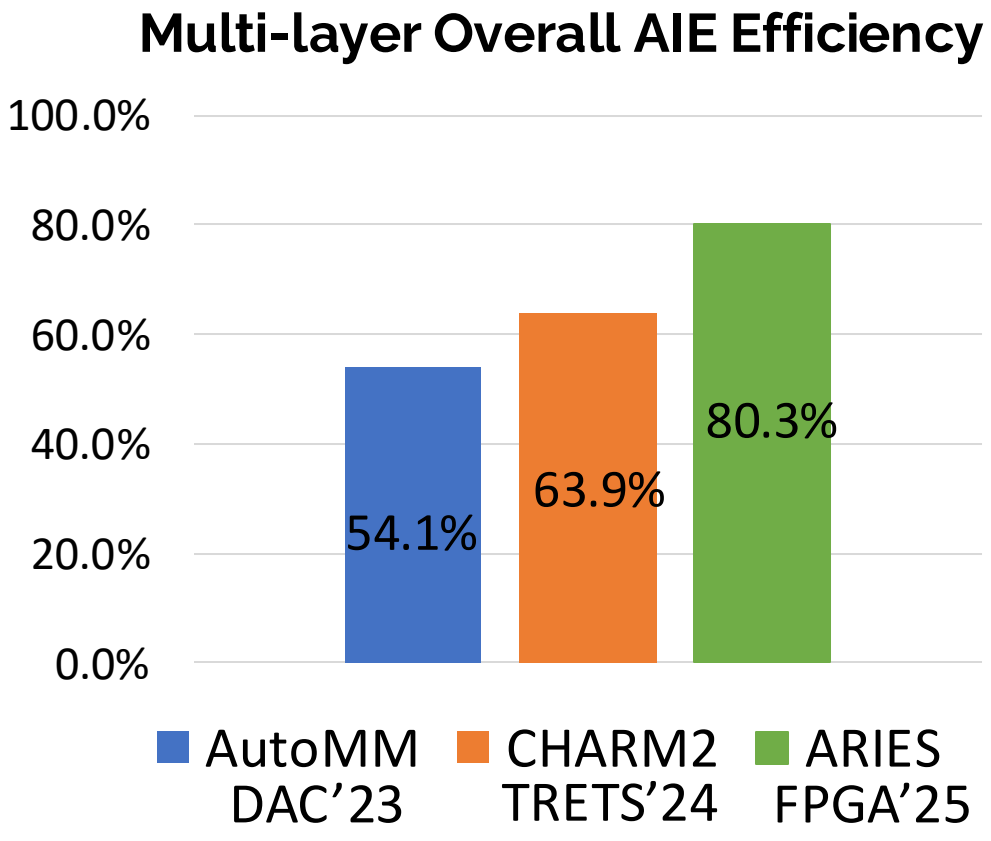
Parameter	Value
#Head	96
Head Dim	128
Embed Dim	12288
MLP Ratio	4

Experiment Results

- Versal Backend: Multilayer perceptron (MLP)

Parameter	Value
#Head	96
Head Dim	128
Embed Dim	12288
MLP Ratio	4





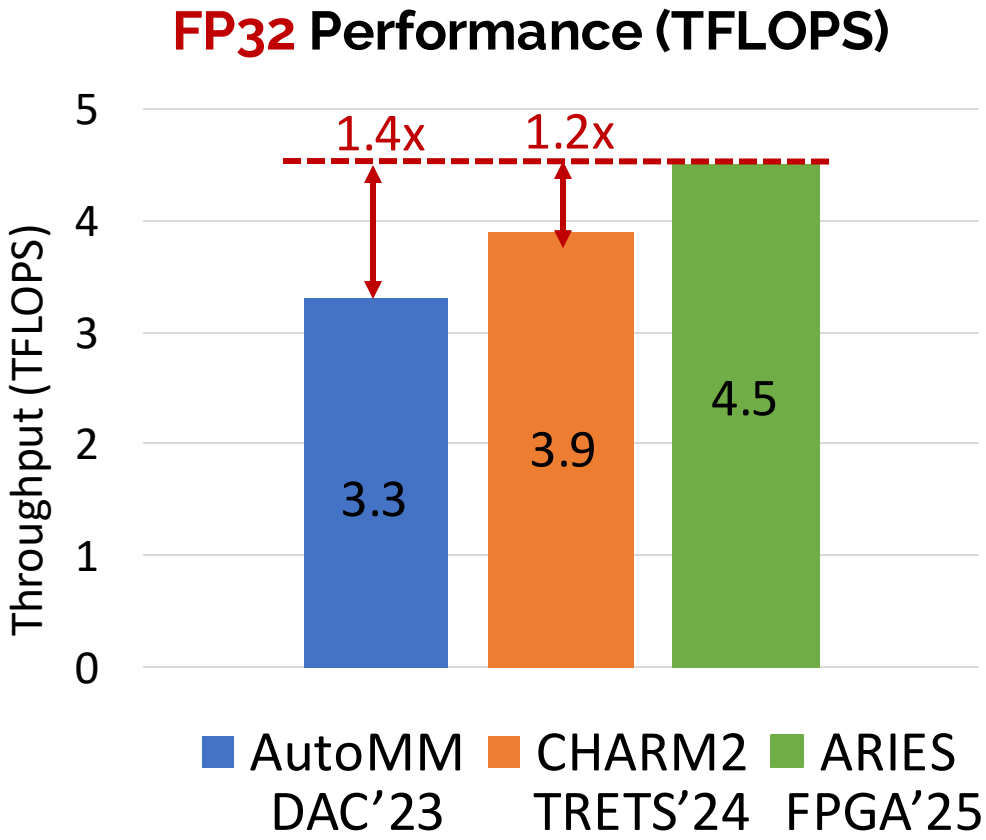
Experiment Results

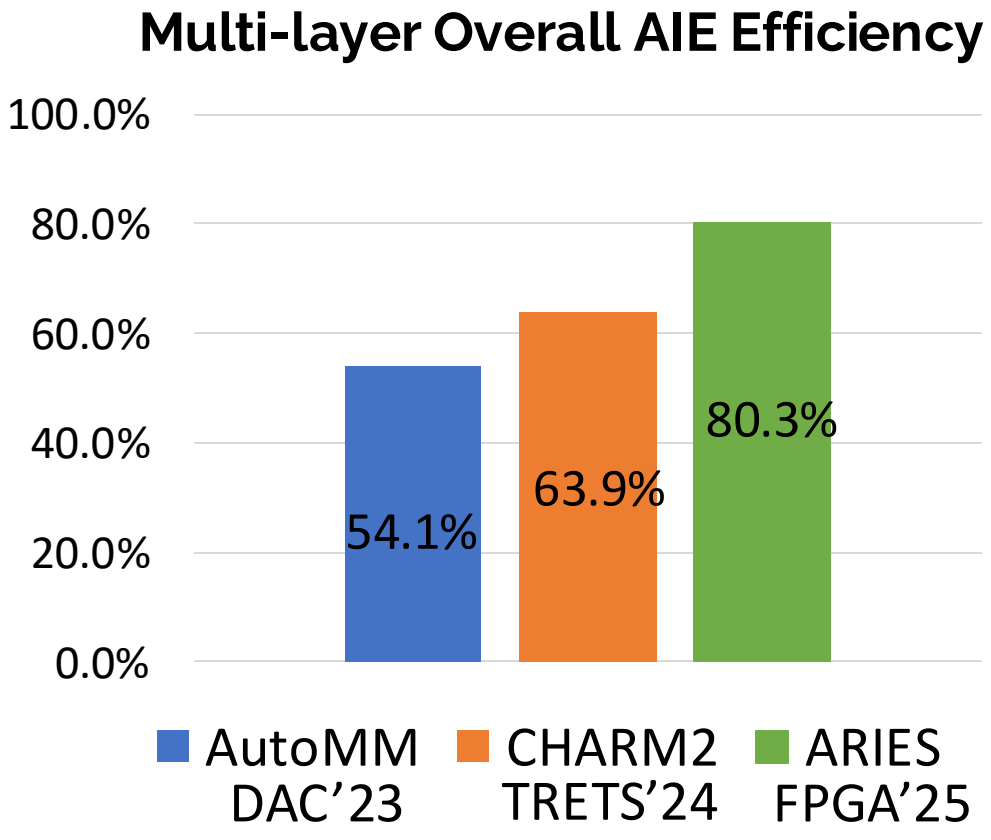
- Versal Backend: Multilayer perceptron (MLP)

4.5 TFLOPS overall throughput

1.4x gain over AutoMM

Parameter	Value
#Head	96
Head Dim	128
Embed Dim	12288
MLP Ratio	4





Experiment Results

- Versal Backend: Tensor Operations Performance

Experiment Results

- Versal Backend: Tensor Operations Performance

① $TTM : C(i, j, k) += A(i, j, l) \times B(l, k)$

② $TTMc : D(i, j, k) += A(i, l, m) \times B(l, j) \times C(m, k)$

③ $MTTKRP : D(i, j) += A(i, k, l) \times B(k, j) \times C(l, j)$

Experiment Results

- Versal Backend: Tensor Operations Performance

① $TTM : C(i,j,k) += A(i,j,l) \times B(l,k)$

② $TTMc : D(i,j,k) += A(i,l,m) \times B(l,j) \times C(m,k)$

③ $MTTKRP : D(i,j) += A(i,k,l) \times B(k,j) \times C(l,j)$

~30 Lines of code

Experiment Results

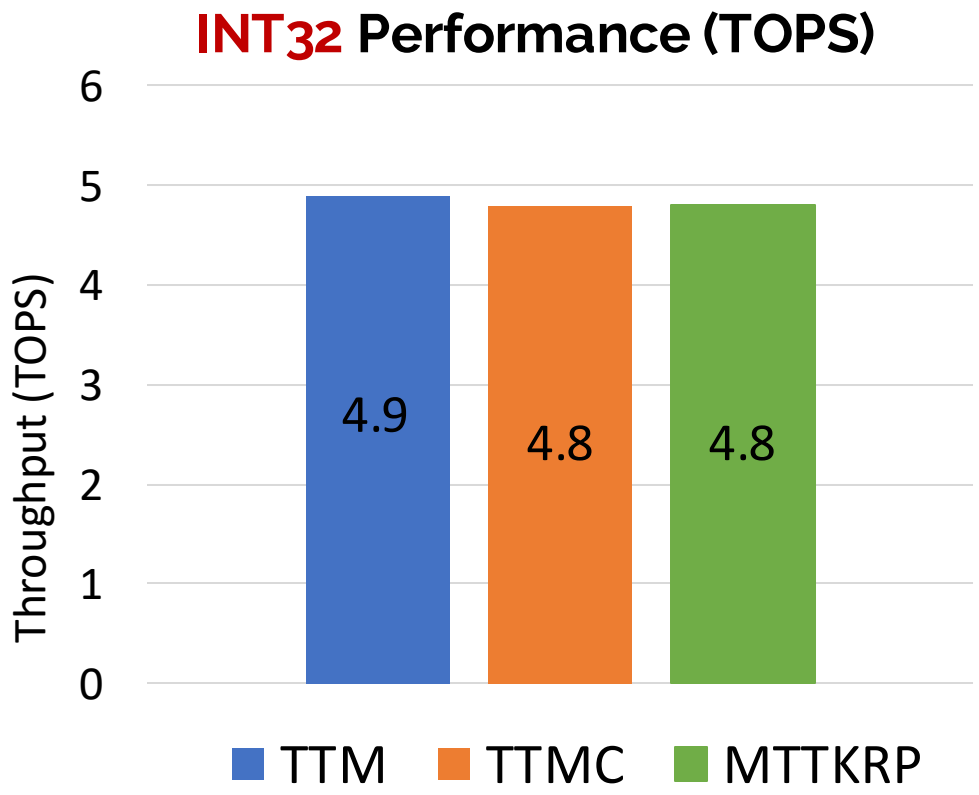
- Versal Backend: Tensor Operations Performance

① $TTM : C(i,j,k) += A(i,j,l) \times B(l,k)$

② $TTMc : D(i,j,k) += A(i,l,m) \times B(l,j) \times C(m,k)$

③ $MTTKRP : D(i,j) += A(i,k,l) \times B(k,j) \times C(l,j)$

~30 Lines of code



Experiment Results

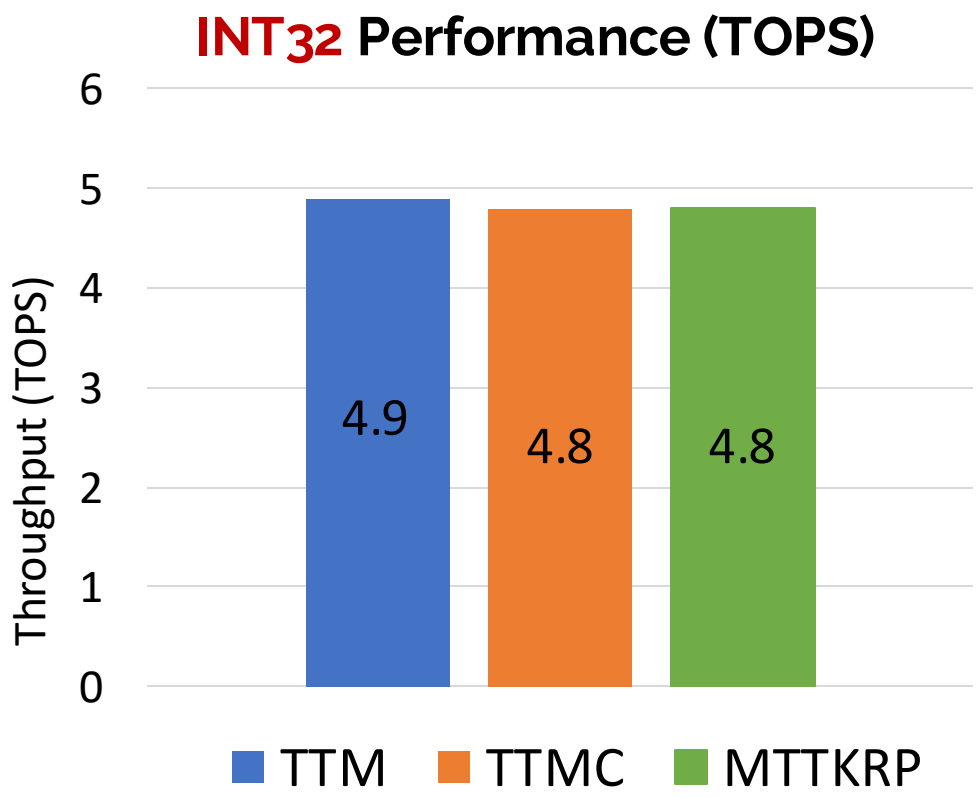
- Versal Backend: Tensor Operations Performance

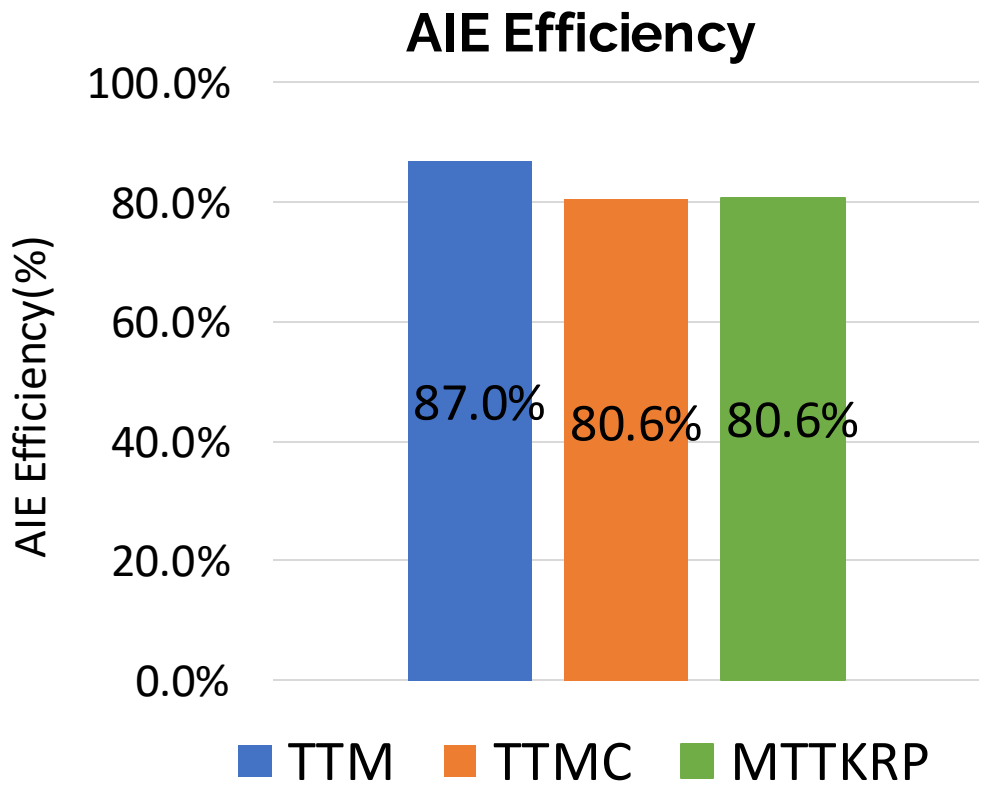
① $TTM : C(i,j,k) += A(i,j,l) \times B(l,k)$

② $TTMc : D(i,j,k) += A(i,l,m) \times B(l,j) \times C(m,k)$

③ $MTTKRP : D(i,j) += A(i,k,l) \times B(k,j) \times C(l,j)$

~30 Lines of code





Key Takeaways

- ARIES is the first work that proposes a unified representation for the heterogeneous system that enables holistic optimization

- ARIES provides a simplified abstraction for constructing AIE that greatly improves the productivity reliving users from detailed hardware controls

- ARIES end-to-end flow has good extensibility and reusability providing the opportunities for potential optimizations and design space exploration

ARIES Open-source GitHub Repo

- https://github.com/arc-research-lab/Aries

Thank You & Welcome to Questions

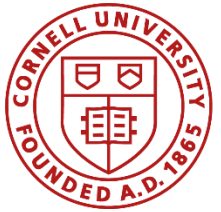
ARIES: An Agile MLIR-Based Compilation Flow for Reconfigurable Devices with AI Engines

The ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA' 25)

Jinming Zhuang*, Shaojie Xiang*†, Hongzheng Chen†, Niansong Zhang†, ZhuopingYang,
Tony Mao†, Zhiru Zhang† and Peipei Zhou

Brown University; Cornell University†
Equal Contribution*

BROWN



Cornell University

National Science Foundation

Semiconductor Research Corporation

AMD

Thank You & Welcome to use ARIES

ARIES: An Agile MLIR-Based Compilation Flow for Reconfigurable Devices with AI Engines

The ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA' 25)

Jinming Zhuang*, Shaojie Xiang*†, Hongzheng Chen†, Niansong Zhang†, ZhuopingYang,
Tony Mao†, Zhiru Zhang† and Peipei Zhou

Brown University; Cornell University†
Equal Contribution*

National Science Foundation

Semiconductor Research Corporation

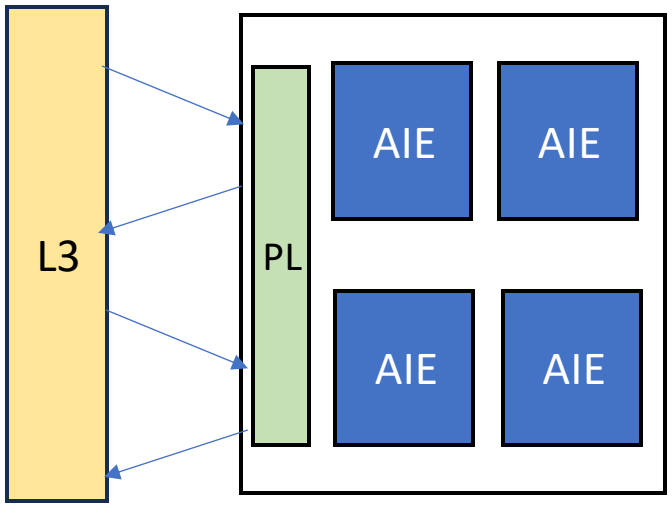
GEMM Example: ARIES Python-Based Frontend

- **ARIES Tile Programming Model:**

⑤ Construct multi-layer applications: `@task_top()`

```
@task_top()
def top(A: float32[32, 64], B: float32[64, 64],
        C: float32[32, 64], D: float32[64, 64],
        E: float32[32, 128]):
    grid0 = (1, 2, 2)
    grid1 = (1, 2, 4)
    tile_size = (32, 32, 32)
    gemm_task = gemm[grid0, tile_size, id=0](A, B, C)
    gemm_task = gemm[grid1, tile_size, id=0](C, D, E)
    return gemm_task
```

One-monolithic overlay



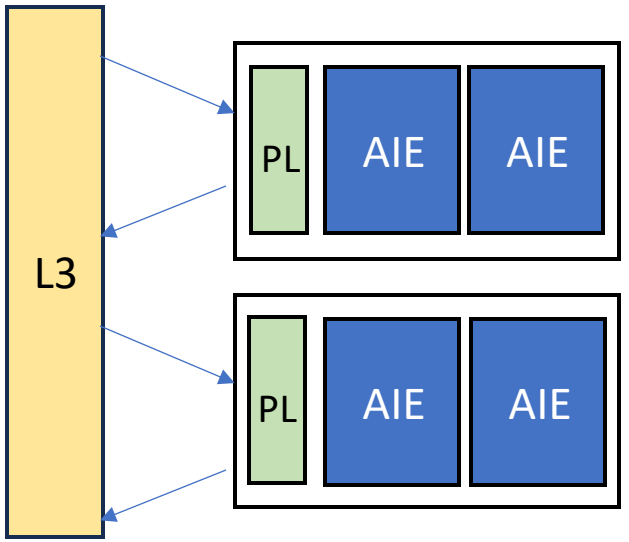
GEMM Example: ARIES Python-Based Frontend

- **ARIES Tile Programming Model**

⑤ Construct multi-layer applications: `@task_top()`

```
@task_top()
def top(A: float32[32, 64], B: float32[64, 64],
        C: float32[32, 64], D: float32[32, 64],
        E: float32[64, 64], F: float32[32, 128]):
    grid0 = (1, 2, 2)
    grid1 = (1, 2, 4)
    tile_size = (32, 32, 32)
    gemm_task0 = gemm[grid0, tile_size, id=0](A, B, C)
    gemm_task1 = gemm[grid1, tile_size, id=1](D, E, F)
    return gemm_task, gemm_task1
```

Spatial Acc design comm via L3



GEMM Example: ARIES Python-Based Frontend

- **ARIES Tile Programming Model**

⑤ Construct multi-layer applications with PL: `@task_pl()`

```
@task_pl()
def softmax(CIN: float32[32, 64], CO: float32[32, 64]):
    # Find max
    # Exp sum
    # Division
    return

@task_top()
def top(A: float32[32, 64], B: float32[64, 64],
        C: float32[32, 64], D: float32[32, 64]):
    grid0 = (1, 2, 2)
    tile_size = (32, 32, 32)
    gemm_task = gemm[grid0, tile_size](A, B, C)
    softmax_task = softmax(C, D)
    return gemm_task, softmax_task
```

Spatial Acc design comm via L3