Spatial Computing with AIR for Ryzen™ AI

@ASPLOS 2024

Jack Lo, Joe Melber, Kristof Denolf, Phil James-Roxby, Samuel Bayliss

@our team

Andra Bisca, Jeff Fifield, Alireza Khodamoradi, Eddie Richter, Erwei Wang, Stephen Neuendorffer, Gagandeep Singh, Ralph Wittig

Research and Advanced Development (RAD)

April 27, 2024

AMD

together we advance_

Tutorial Overview

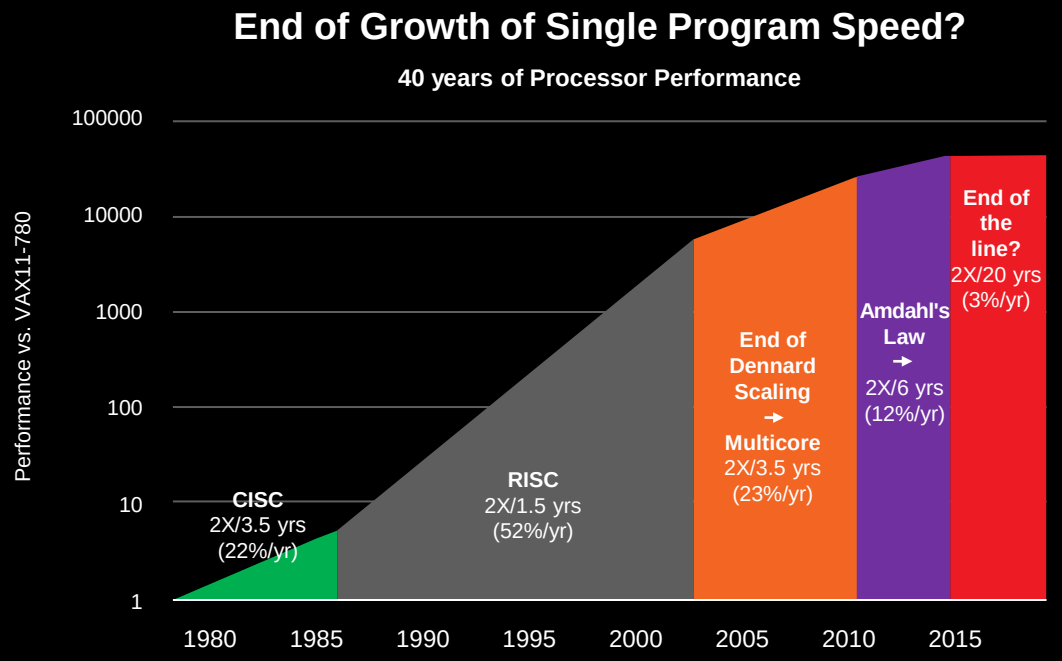
- Leveraging MLIR to Design for AI Engines on Ryzen™ AI

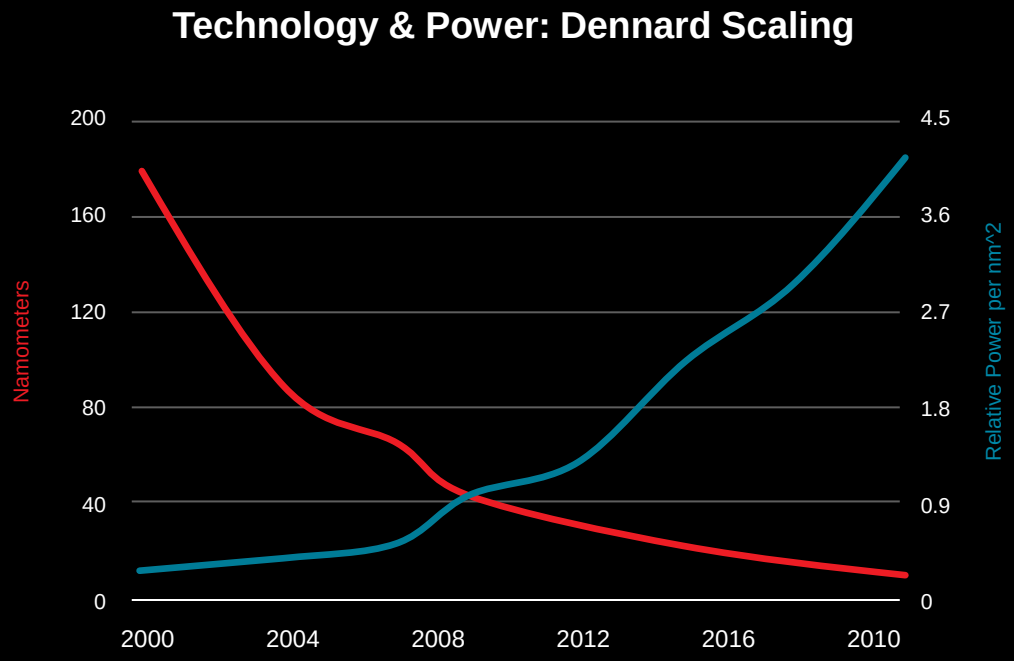
- https://github.com/Xilinx/mlir-aie/tree/main/programming_guide
- Section 1 - Basic AI Engine building blocks
  - Section 2 - Data Movement (Object FIFOs)
  - Section 3 - My First Program
  - Section 4 - Vector Programming & Performance Measurement
  - Section 5 - Example Vector Designs
  - Section 6 - Larger Example Designs

- https://github.com/Xilinx/mlir-aie/tree/main/programming_examples
  - basic
  - ml
  - vision
  - utils

Time	Topic
08:30am	Intro to spatial compute and explicit data movement
08:45am	"Hello World" from Ryzen AI
09:00am	Data movement on Ryzen AI with objectFIFOs
09:30am	Your First Program
09:50am	Exercise 1: Build and run your first program
10:00am	Break
10:30am	Exercise 2: Vector-Scalar Mul
10:40am	Tracing and performance analysis
11:10am	Exercise 3: Tracing vector-scalar
11:30am	Vectorizing on AIE
11:40am	Exercise 4: Vectorized vector-scalar
12:00pm	Dataflow and larger designs
12:15pm	Exercises
12:30pm	Close Tutorial

Heterogenous Architectures: Sustain Compute Growth





A New Golden Age for
Computer Architecture

→

A New Golden Age for
Compilers

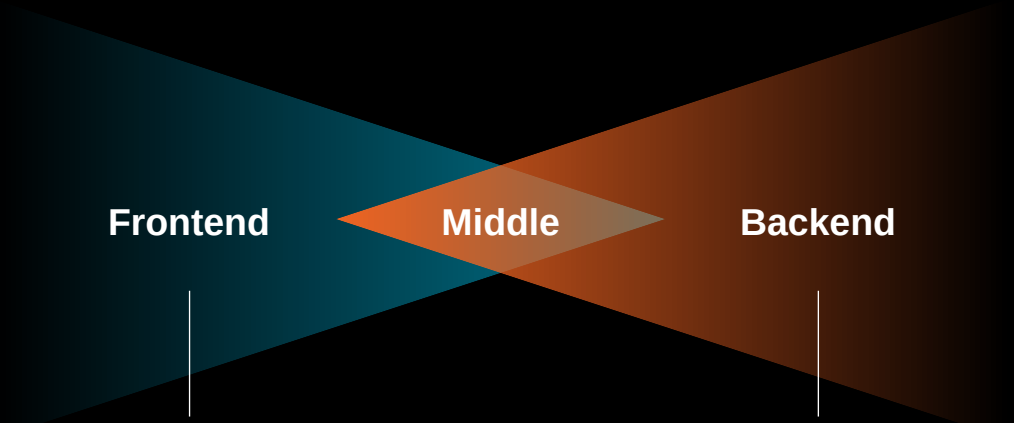
Hennessey and Patterson, ISCA 2018

MLIR: Multi-Level Intermediate Representation

Next-generation open-source compiler infrastructure

- LLVM core project

Well positioned to support this new golden age!



Multiple open-source frontend languages (not just ML!)

Exascale Fortran, Tensorflow, Pytorch -> C, HIP/CUDA, SYCL

Multiple open-source backends

LLVM, GPU, AI Engine, Programmable Logic

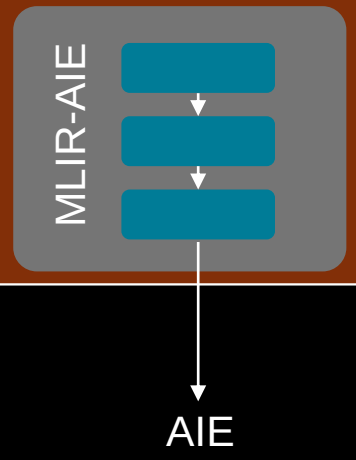
🦾 IRON: Open Access Toolkit for Performance Engineers for Close-to-Metal Programming of Fast and Efficient Designs

- Programmer experience first
  - Clear close-to-metal API with design examples
  - Debug and trace support
- Enable performance engineers to build fast and efficient specialized designs
- Multiple abstraction levels without hiding architectural features
- Leverages the open-source community and technology
  - Common open-source technology: mlir-aie, mlir-air, IREE

IRON

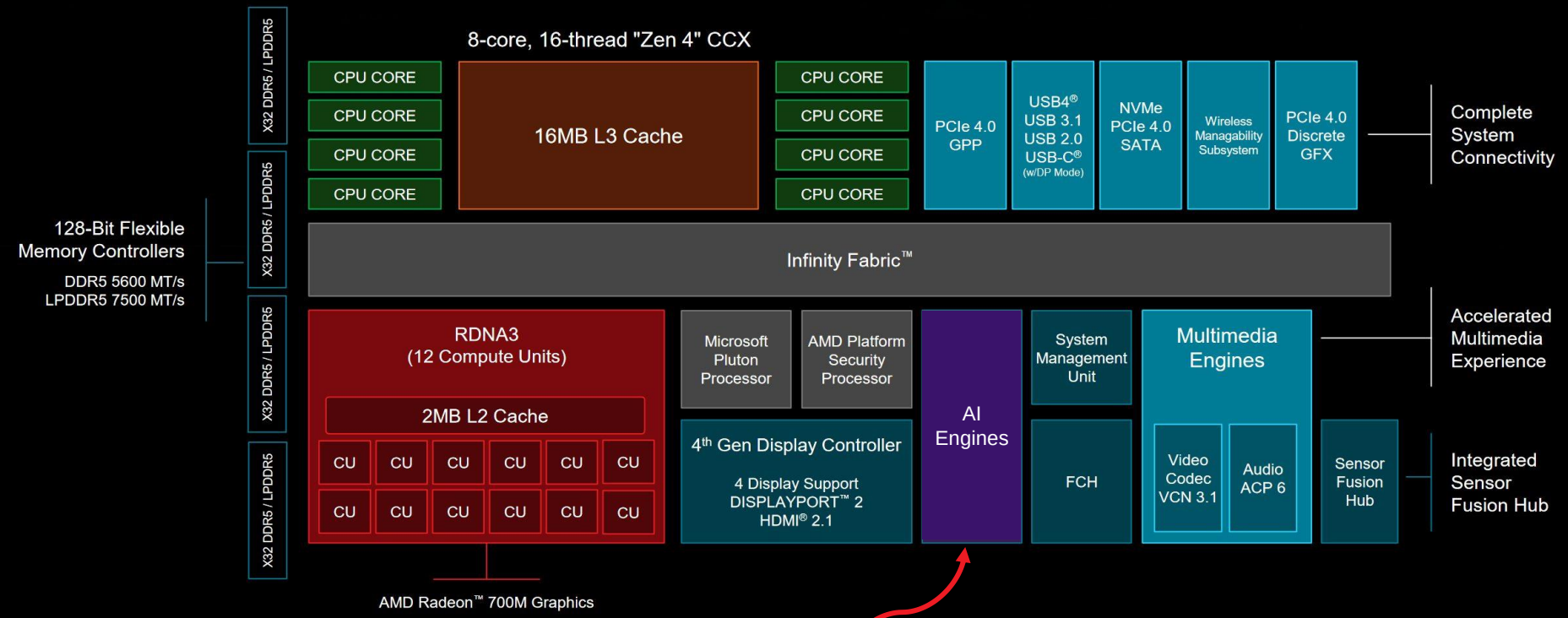
Toolkit for performance engineers

Close-to-Metal Programming with Python Language Bindings



AMD Ryzen™ AI Architecture

AMD Ryzen™ 7040 Series for Mobile – The 'Phoenix' SoC



3 AMD Ryzen™ 7040 | Aug 2023

* Certain capabilities and features dependent upon OEM enablement

AMD

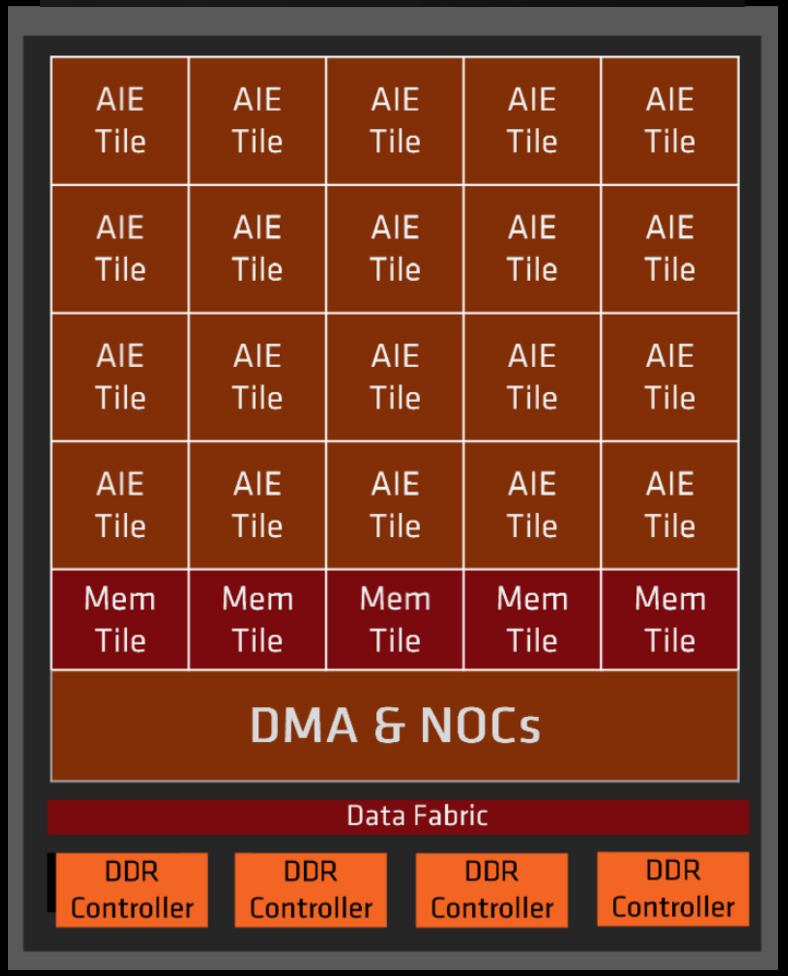
Spatial Hardware

[Hot Chips '23: Mahesh Subramony & David Kramer, AMD Ryzen 7040 Series Mobile Processor]

Spatial Compute, In Stores Today

AI Engine inside!

AMD RYZEN AI



Best Buy, Longmont, CO

Intro to Spatial Compute and Explicit Data Movement

Rescue Workers Crunching AMDs Overdue Paper

AMD Paperwork Warehouse

Worker: Paperwork Cruncher

[freepik.com]

How to Efficiently Move Paperwork to Array of Workers

AMD Paperwork Warehouse

Worker Office

[freepik.com]

Add Storage Room on Each Level in Worker's Office

AMD Paperwork Warehouse

Storage Room

Worker Office

[freepik.com]

Add Data Movement Acceleration (DMA) in Different Locations

AMD Paperwork Warehouse

DMA-L3 DMA-L3 DMA-L3 DMA-L3

Storage Room

DMA-L2

Worker Office

DMA-L1

DMA-L1

DMA-L1

DMA-L1

DMA-L1

DMA-L1

[freepik.com]

AI Engine Basic Building Blocks

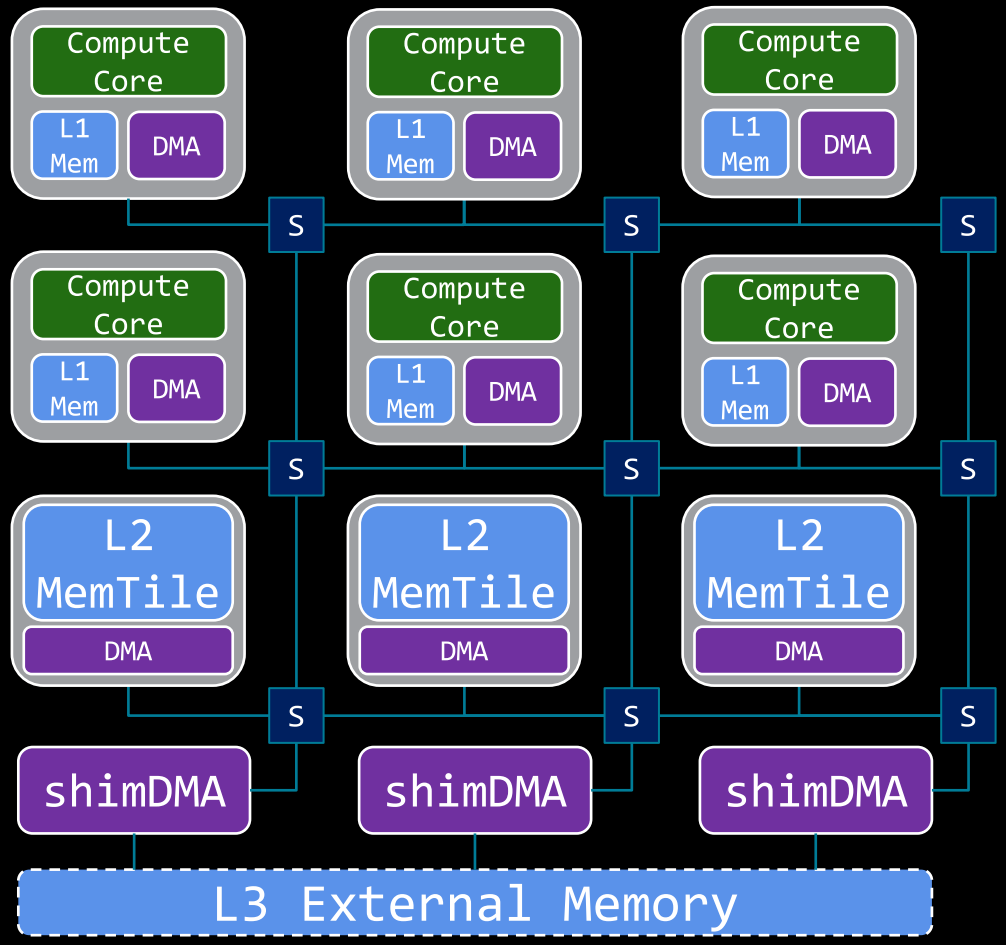
AI Engine Array: Modular and Scalable Architecture with Independent Concurrent Processing and Explicit Data Movement

The AI Engine array has

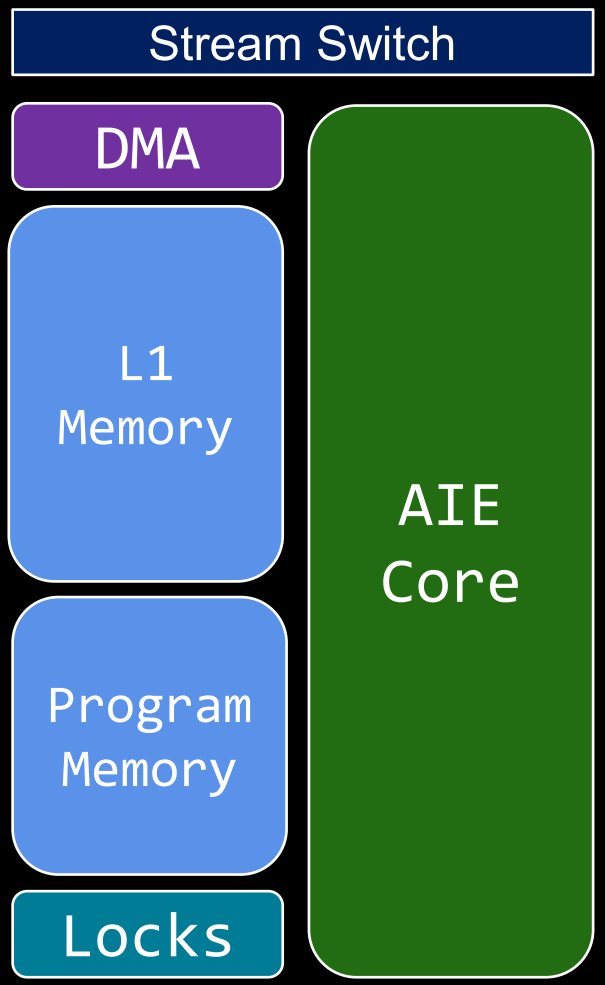
- Spatially distributed compute and memory
- Multi-level memory hierarchy
- Explicitly scheduled, decoupled data movement
  - Direct communication with neighbor L1 Mem ***or***
  - Packet-switched interconnect

The AI Engine array enables

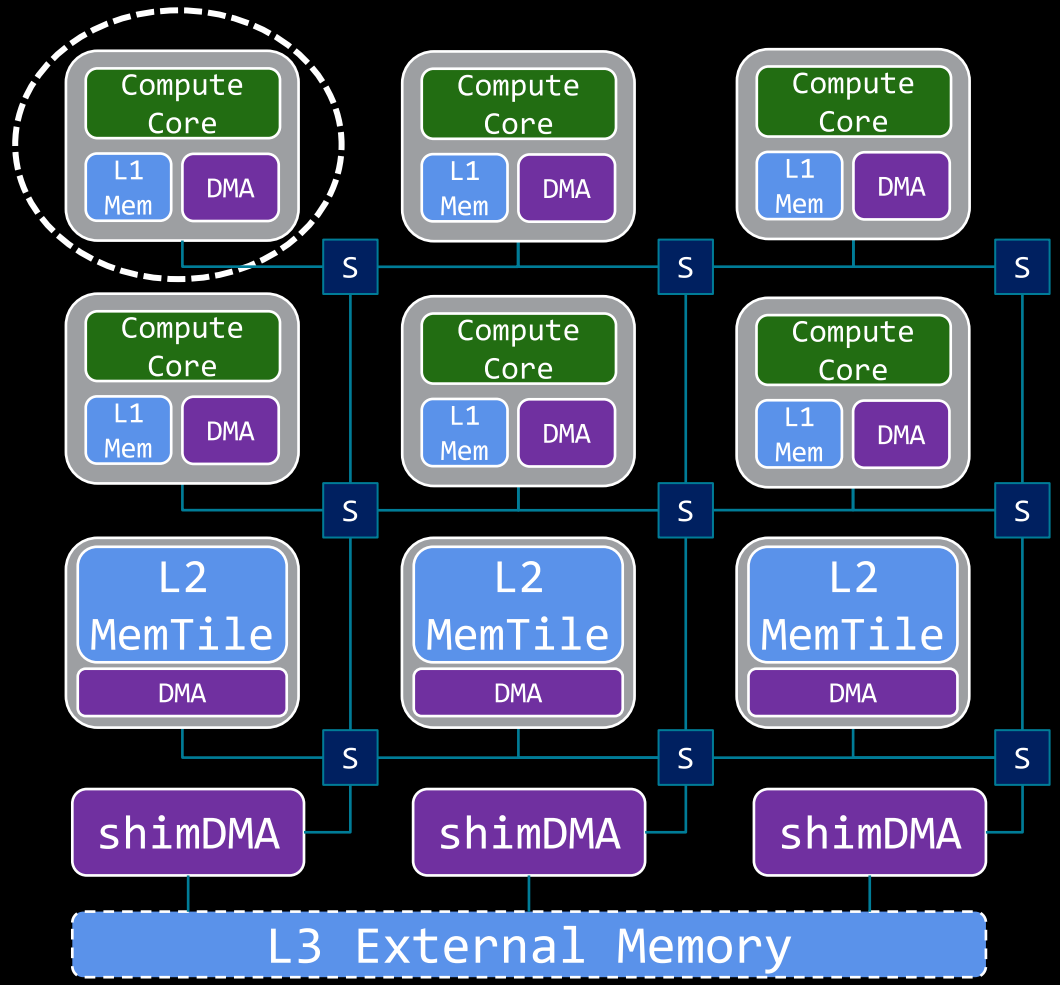
- Independent tasks on each core
- Dataflow between tasks
- Distribute an input data space to cores
- Coordinate towards output data space



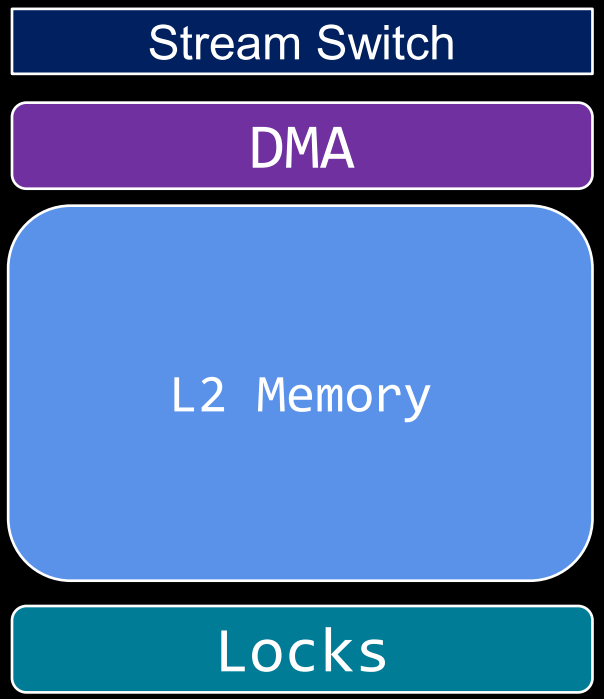
AIE Compute Tile Architecture



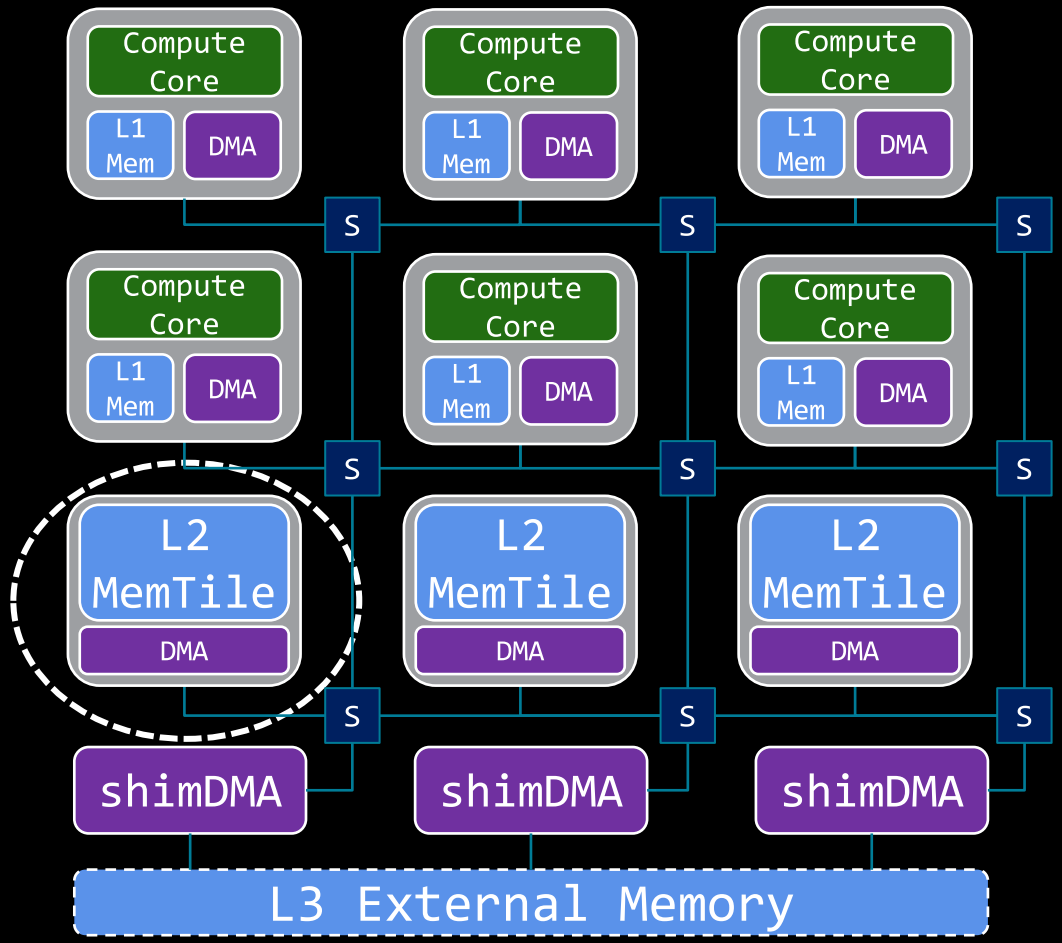
ComputeTile = tile(0, 3)



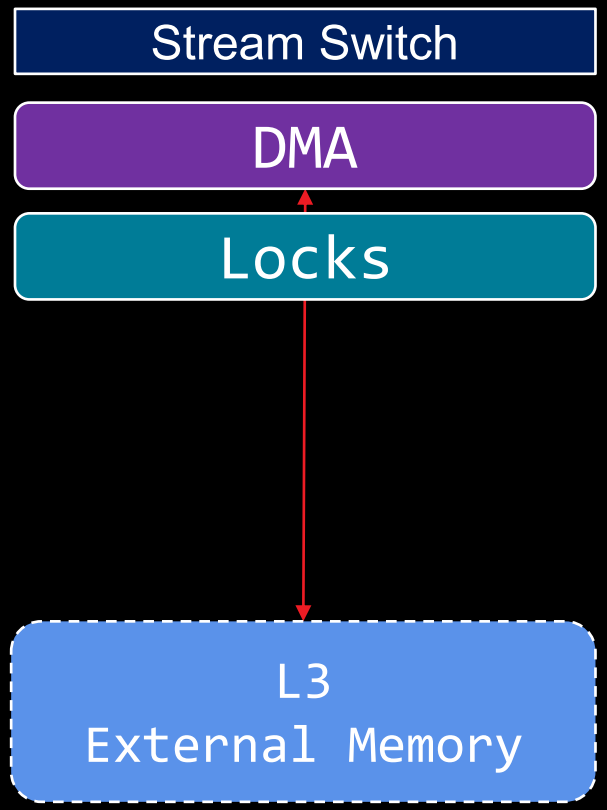
AIE Mem Tile Architecture



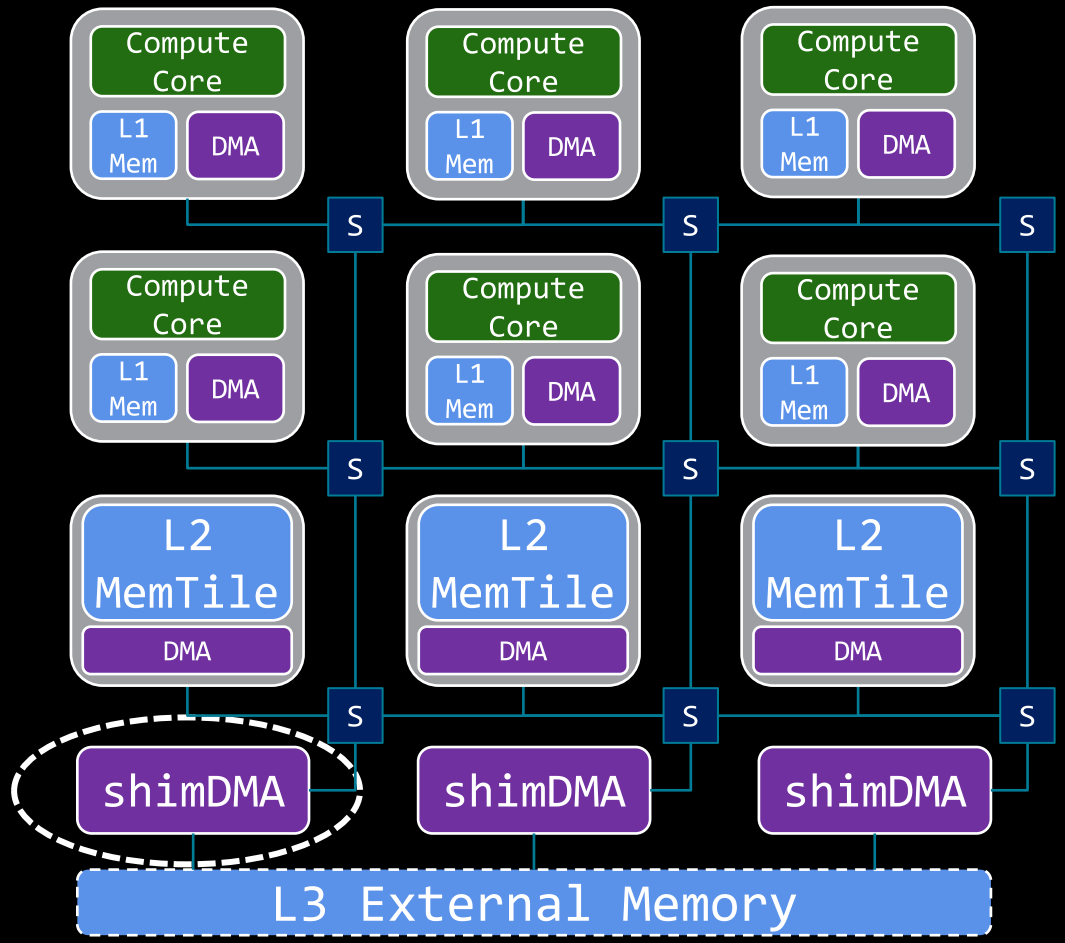
MemTile = tile(0, 1)



AIE Shim Tile Architecture



ShimTile =
tile(0, 0)



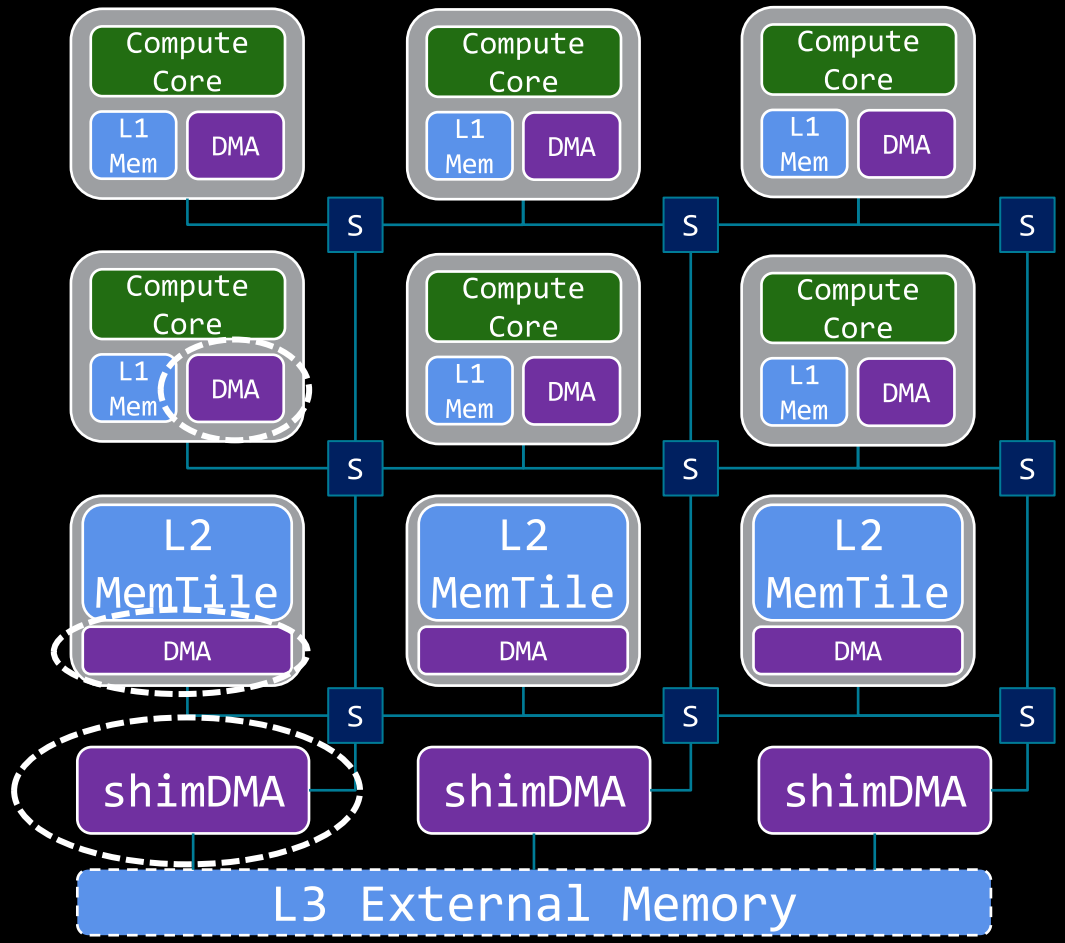
Walkthrough of a Simple Python Source File (aie2.py)

```
from aie.dialects.aie import * # primary mlir-aie dialect definitions
from aie.extras.context import mlir_mod_ctx # mlir-aie context


# AI Engine structural design function
def mlir_aie_design():

  # Device declaration - aie2 device NPU
  @device(AIEDevice.npu)
  def device_body():

    # Tile(s) declarations
    ComputeTile1 = tile(1, 3)
    ComputeTile2 = tile(2, 3)
    ComputeTile3 = tile(2, 4)


# Declares that subsequent code is in mlir-aie context
with mlir_mod_ctx() as ctx:
  mlir_aie_design() # Call design function within the mlir-aie context
  print(ctx.module) # Print the python-to-mlir conversion to stdout
```

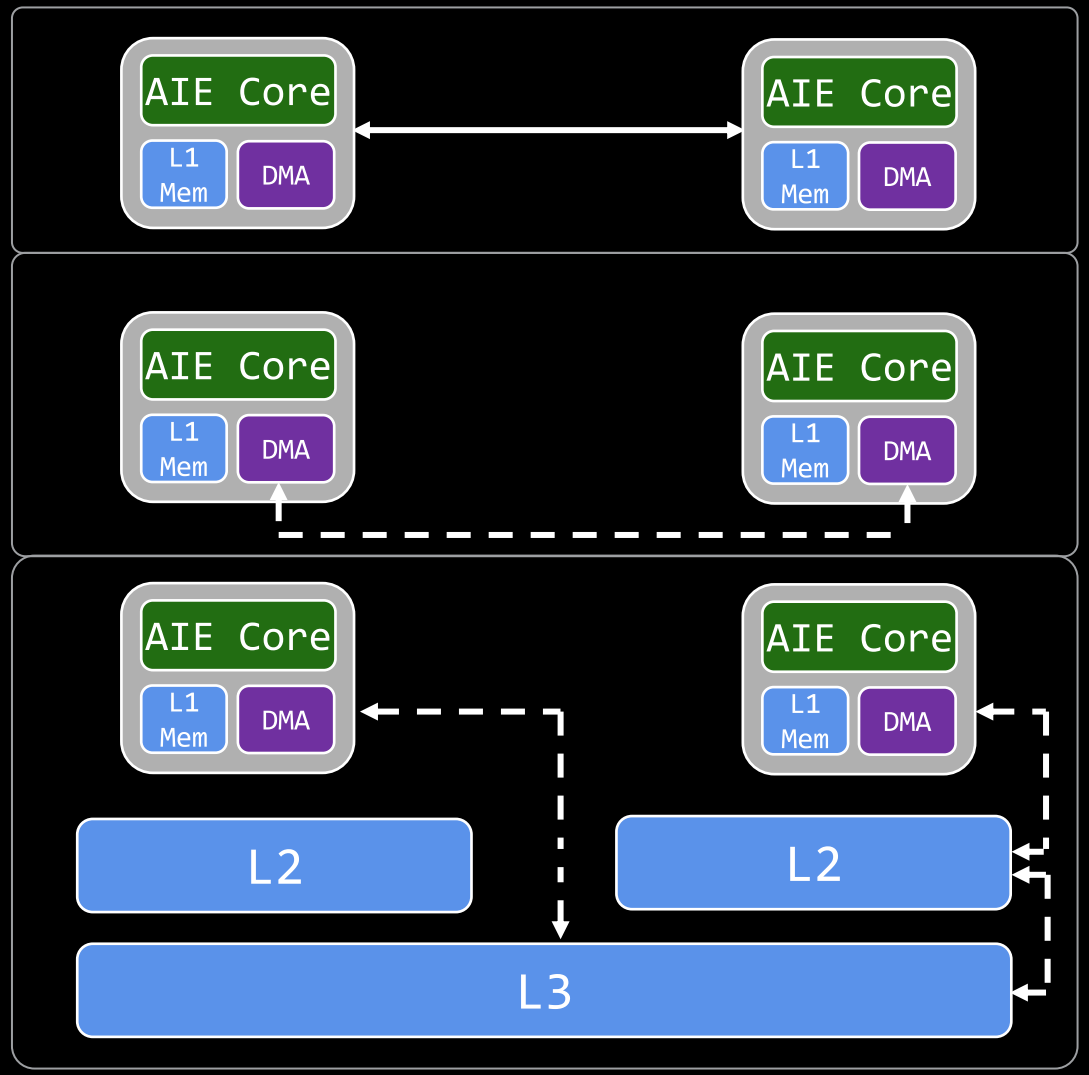
Data Movement on Ryzen™ AI with objectFifos

Spatial Compute = Data Movement a First Class Concern

DMA Feature	Tile DMA	MemTile DMA	ShimDMA
Max Addressing Dimension	3D	4D	3D
Zero-padding	N/A	Yes	N/A
Num Channels (R / W)	2 / 2	6 / 6	2 / 2
Num Buffer Descriptors	16	48	16
Num Semaphore Locks	16	64	16



Data Movement Scenarios

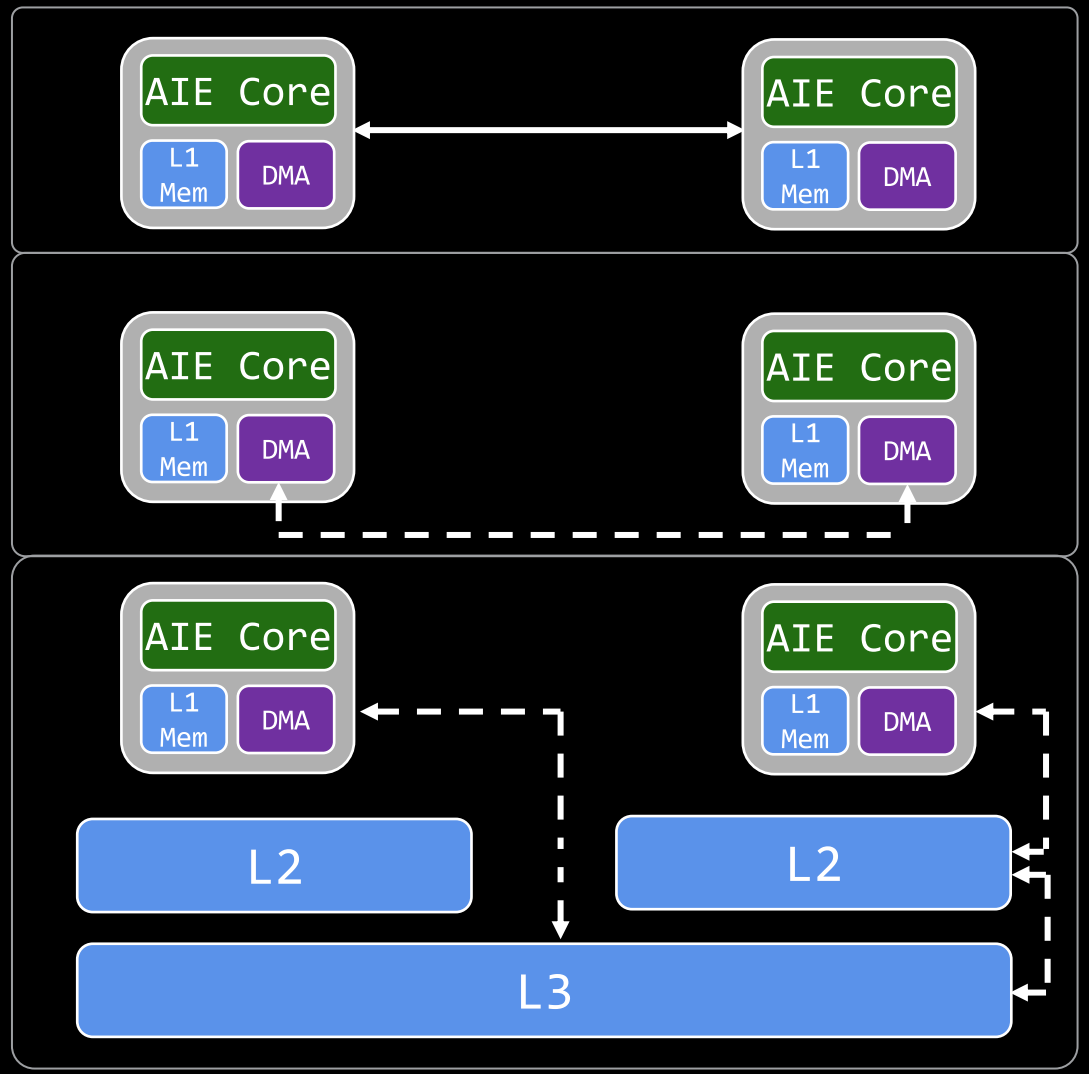


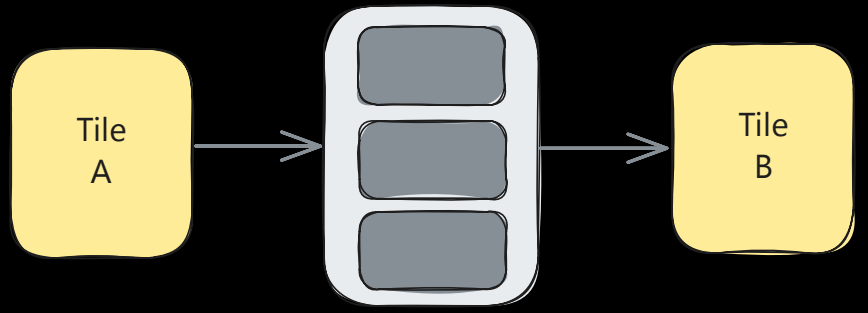
Peer Mem-To-Mem
Shared Memory

Peer Mem-To-Mem
DMAs + Routing

Hierarchical Mem-To-Mem
DMAs + Routing

Requirement for a Primitive That Encapsulates Data Allocation, Data Access, Movement and Routing





ObjectFifo Is the High-Level MLIR-AIE Data Movement Primitive

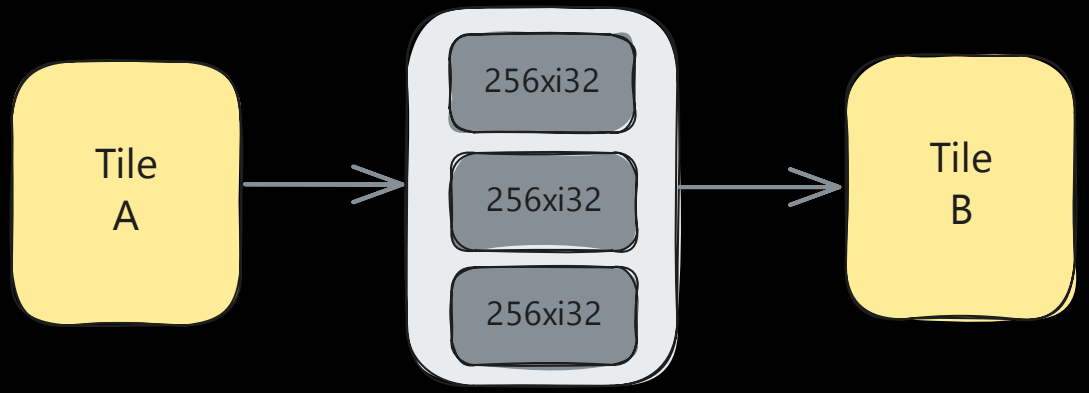
```
class object_fifo:
        def __init__(
                self,
                name,
                producerTile,
                consumerTiles,
                depth,
                datatype,
        )
```

Data Allocation

Data Access

Data Movement & Routing

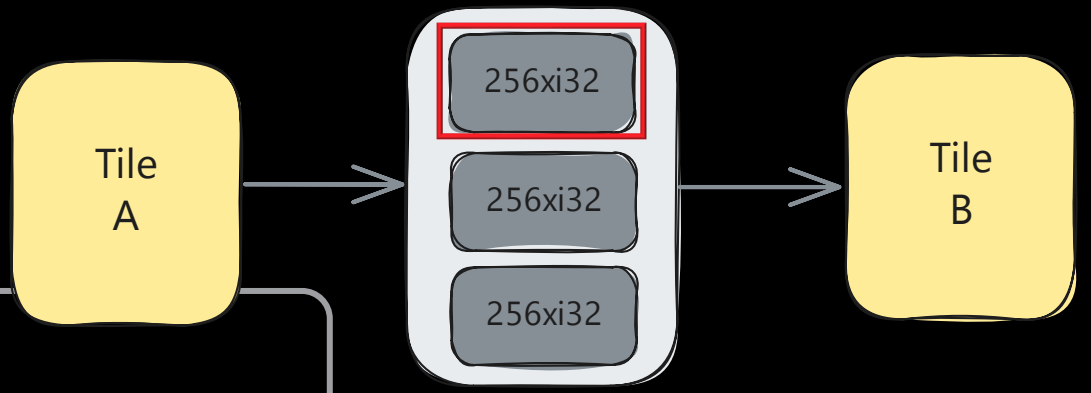
ObjectFifo Is the High-Level MLIR-AIE Data Movement Primitive



Python:

```
A = tile(1, 3)
B = tile(2, 4)
of0 = object_fifo("objfifo0", A, B, 3, T.memref(256, T.i32()))
```

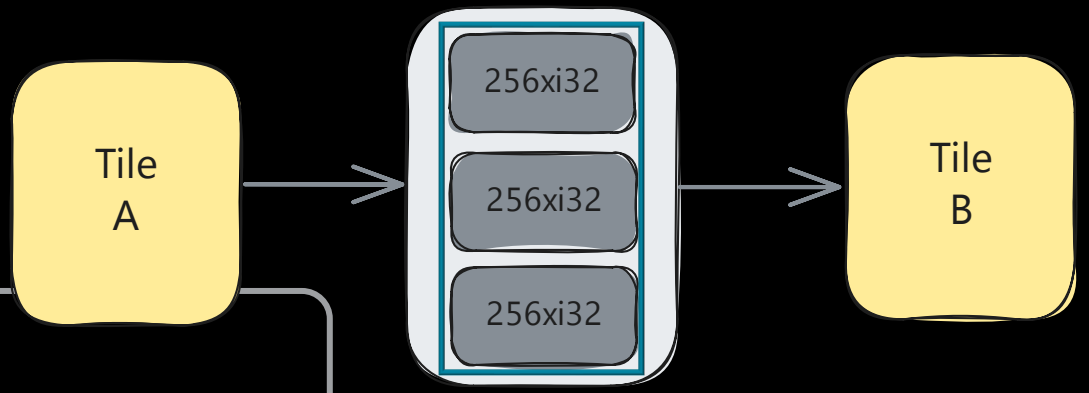
Synchronized Accesses to the ObjectFifo Enable a Deadlock-Free Schedule



Python:

```
A = tile(1, 3)
B = tile(2, 4)
of0 = object_fifo("objfifo0", A, B, 3, T.memref(256, T.i32()))

@core(A)
def core_body():
        for _ in range_(3):
                elem0 = of0.acquire(ObjectFifoPort.Produce, 1)
                call(test_func, [elem0])
                of0.release(ObjectFifoPort.Produce, 1)
                yield_([])
```

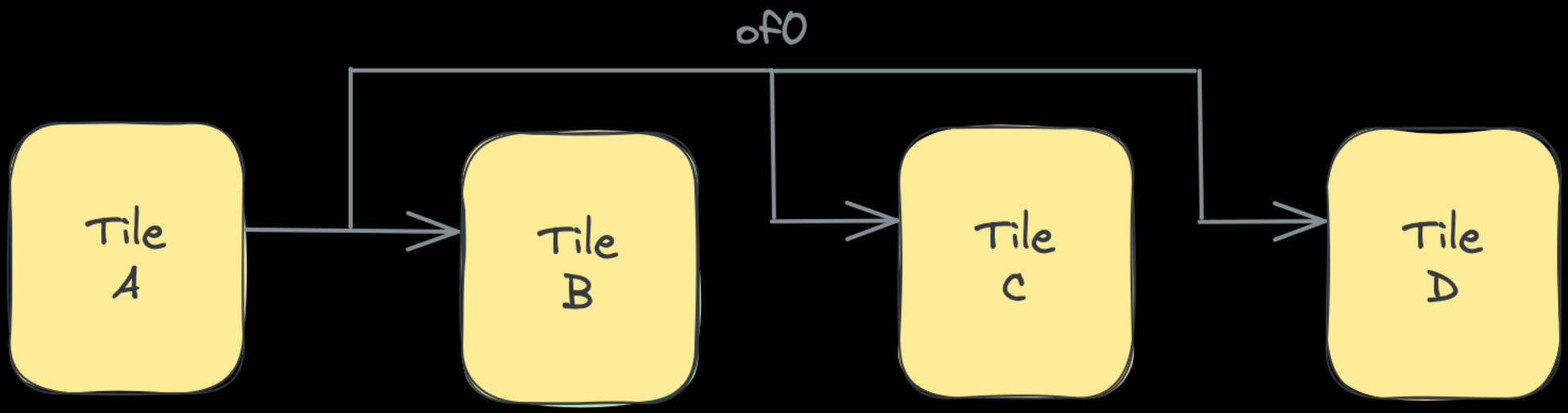
Synchronized Accesses to the ObjectFifo Enable a Deadlock-Free Schedule



Python:

```
A = tile(1, 3)
B = tile(2, 4)
of0 = object_fifo("objfifo0", A, B, 3, T.memref(256, T.i32()))

@core(B)
def core_body():
        elems = of0.acquire(ObjectFifoPort.Consume, 3)
        call(test_func2, [elems[0]])
        call(test_func2, [elems[1]])
        call(test_func2, [elems[2]])
        of0.release(ObjectFifoPort.Consume, 3)
```

Broadcast with ObjectFifo

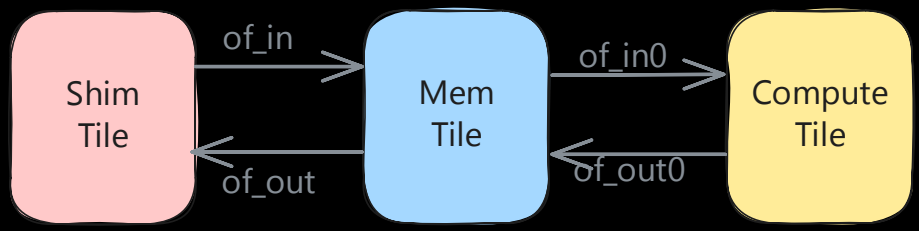


Python:

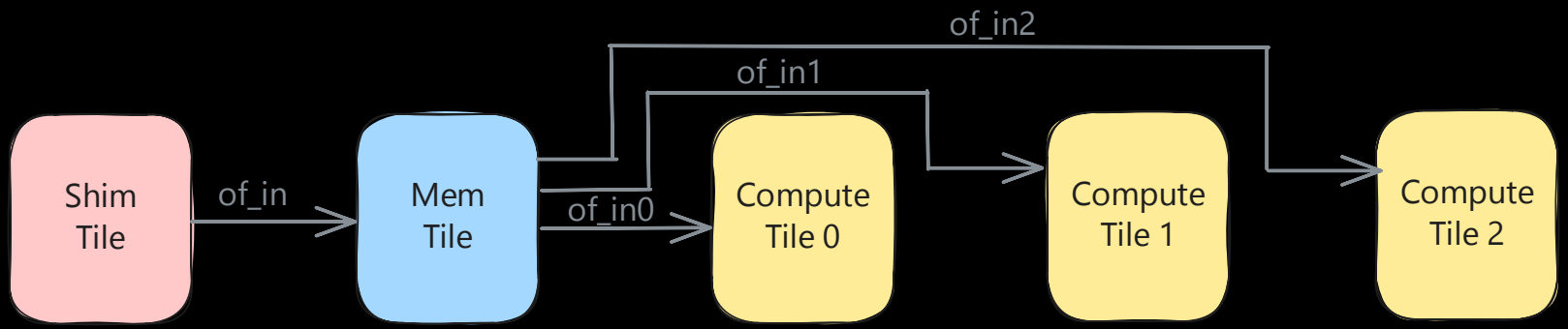
```
A = tile(1, 1)
B = tile(1, 3)
C = tile(2, 3)
D = tile(3, 3)
of0 = object_fifo("objfifo0", A, [B, C, D], 3, T.memref(256, T.i32()))
```

Some ObjectFifo Patterns

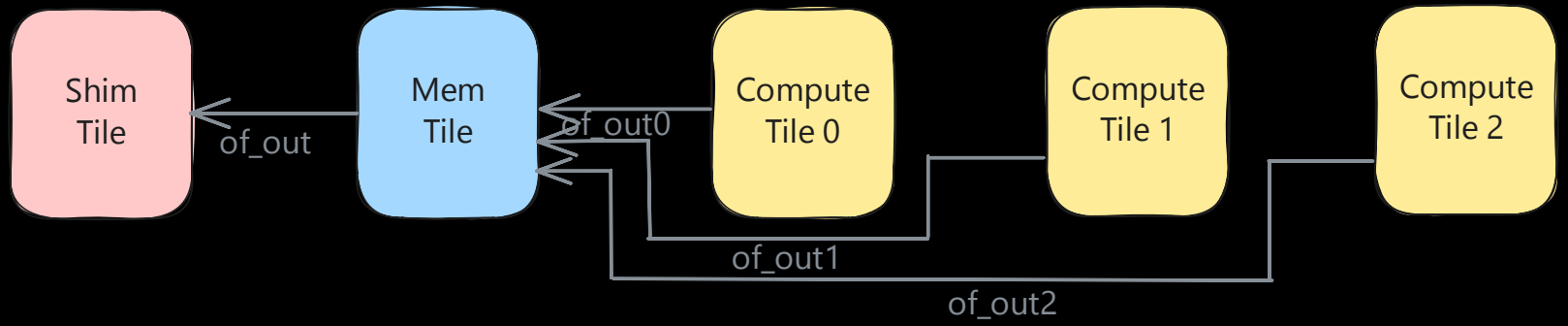
- Hierarchical Data Movement



- Distribute from MemTile



- Join in MemTile



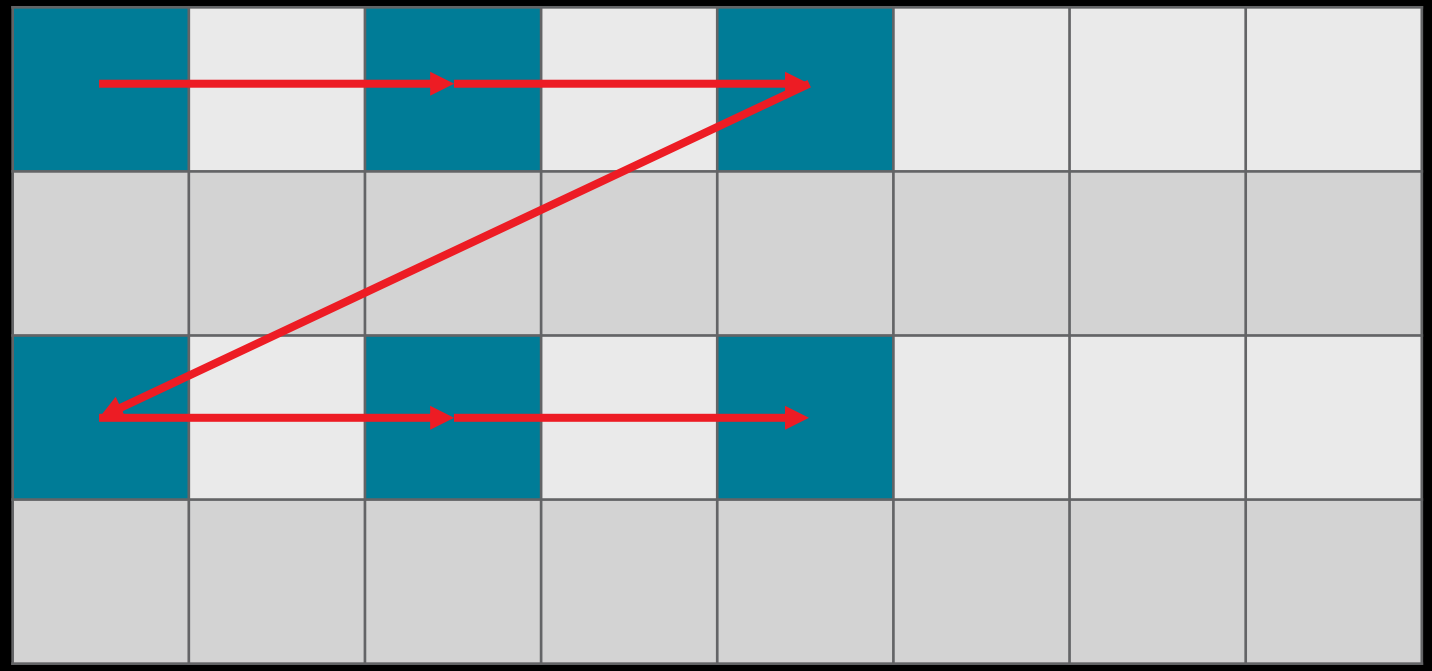
- Checkout the programming guide for more patterns and details!

Data Layout Transformations on The Fly

```
A = tile(1, 1)
B = tile(1, 3)
of0 = object_fifo("objfifo0", A, B, 3, T.memref((4, 8), T.i32()),
                                       [(2, 16), (3, 2),],
                                )
```

Dimension 0: Length 3, Stride 2

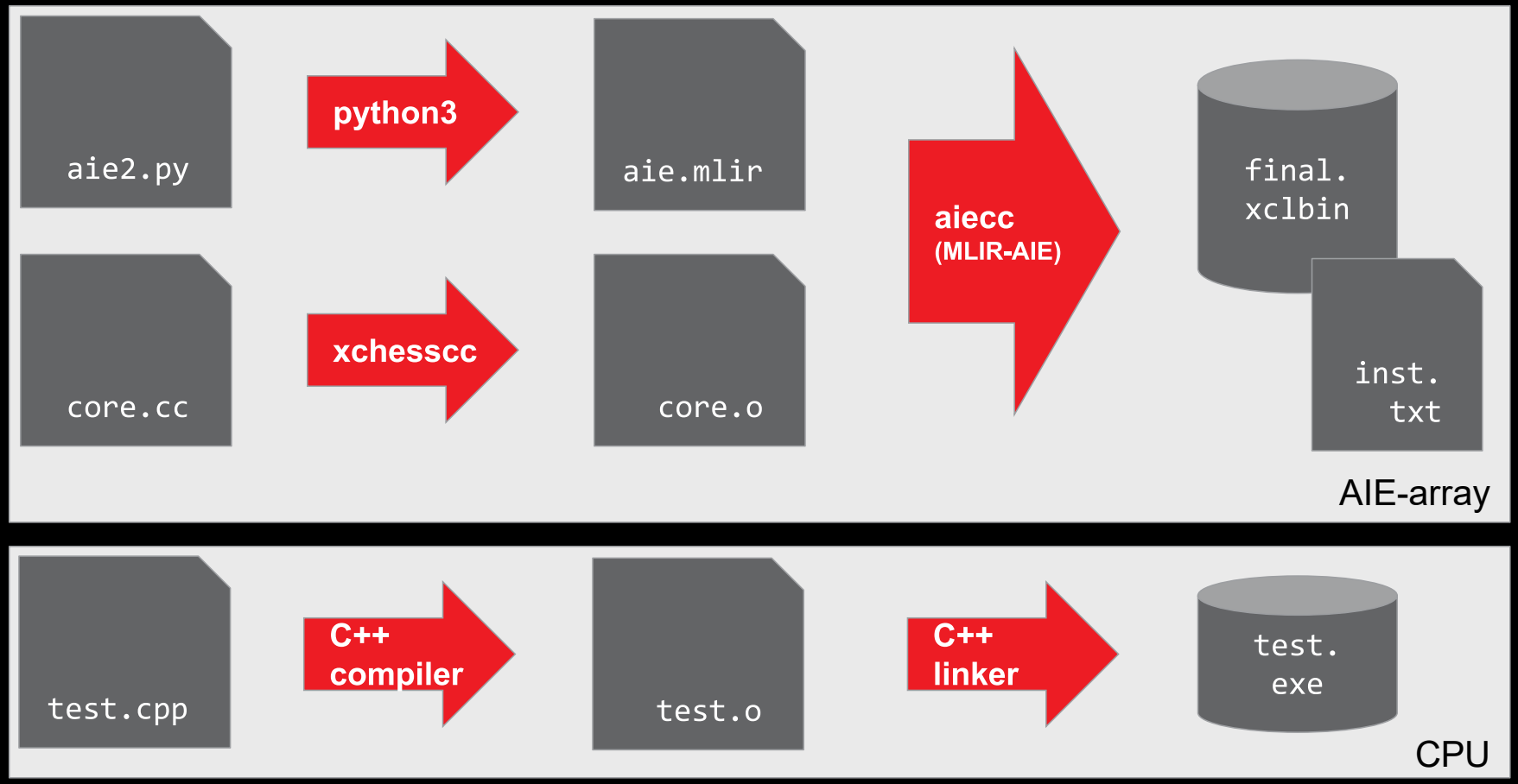
Dimension 1: Length 2, Stride 16



```
int *buffer;
for(int dim1 = 0; dim1 < 2; dim1++)   // size_1
  for(int dim0 = 0; dim0 < 3; dim0++) // size_0
    // access/store element at/to index:
    buffer[
      dim1 * 16  // stride_1
      + dim0 * 2 // stride_0
    ];
```

Your First Program

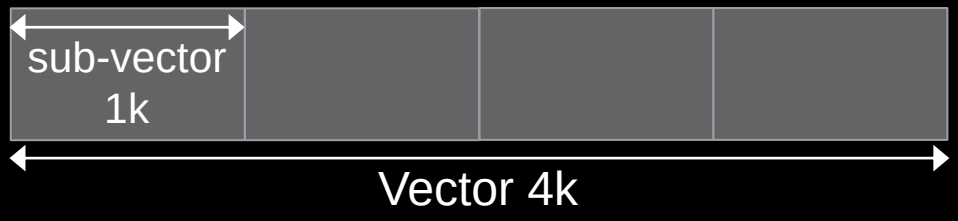
From Source Code to Binaries



Source Code

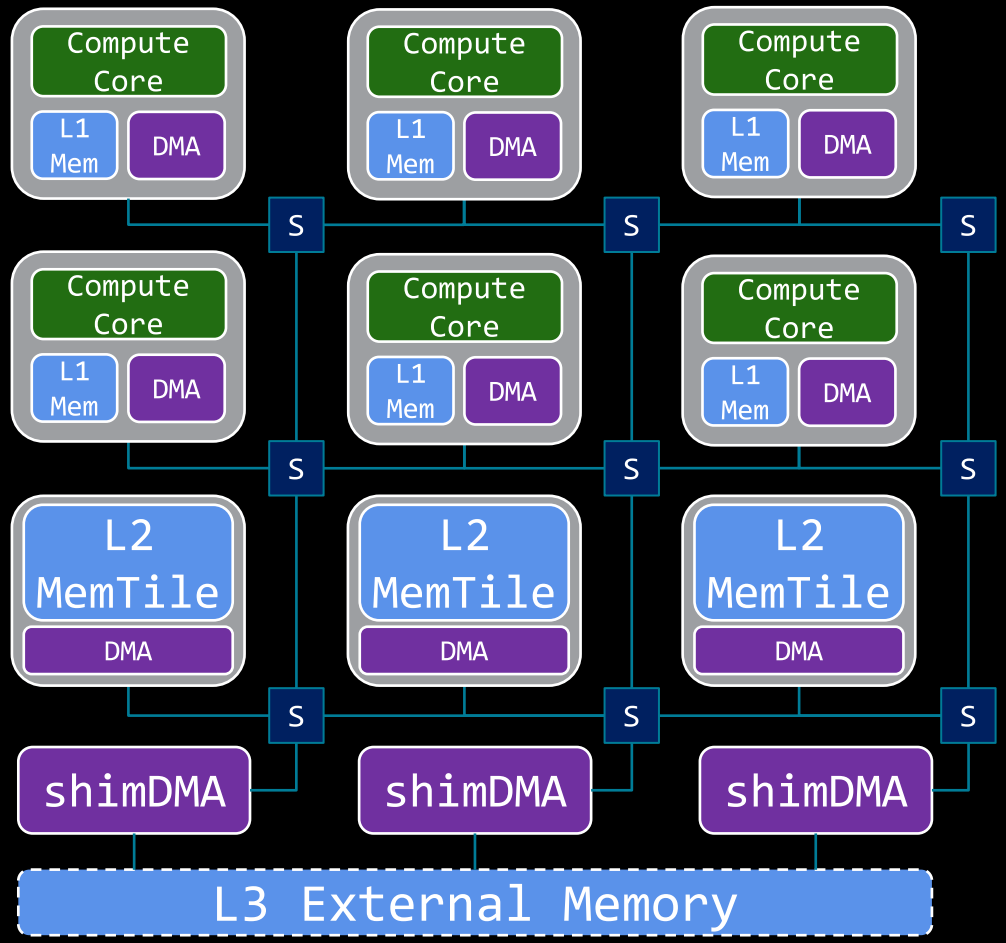
Ryzen™ AI Binaries

Hello World
vector_scalar: c = a*factor

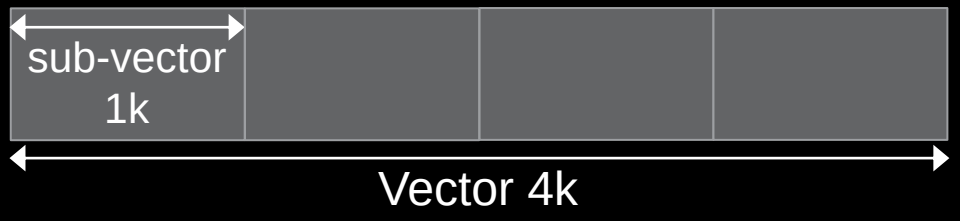


AIE Core Code (Scalar Version)

```
void scale_int32(int32_t *a, int32_t *c,
                 int32_t factor) {

 for (int i = 0; i < 1024; i++) {
  c[i] = factor * a[i];
 }
}
```



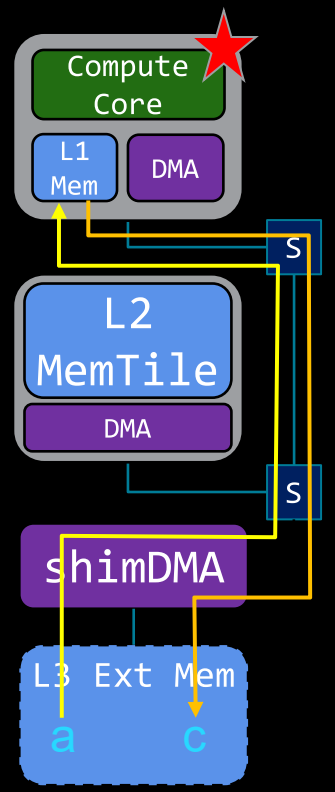
Hello World: Data Movement and Compute Steps vector_scalar: c = a*factor



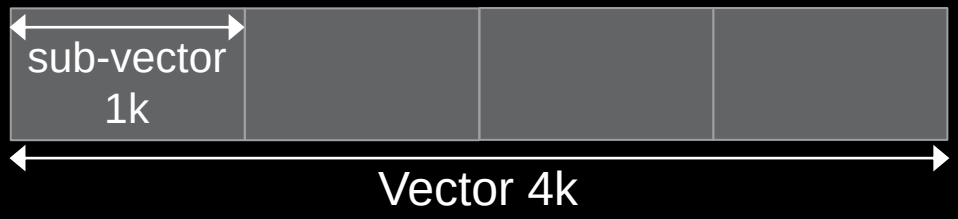
- Read “a” data from external memory and
- Push data onto streaming interconnect with Shim DMA
- Pull data off streaming interconnect with AIE tile DMA

- Vector scale operation in the AIE Core

- Push data from L1 tile memory onto streaming interconnect with AIE tile DMA
- Pull data off streaming interconnect with Shim DMA
- Write “c” back to DDR



Hello World
vector_scalar: c = a*factor



AIE Core Code (Scalar Version)

```
void scale_int32(int32_t *a, int32_t *c,
                 int32_t factor) {

 for (int i = 0; i < 1024; i++) {
  c[i] = factor * a[i];
 }
}
```

AIE Array Code

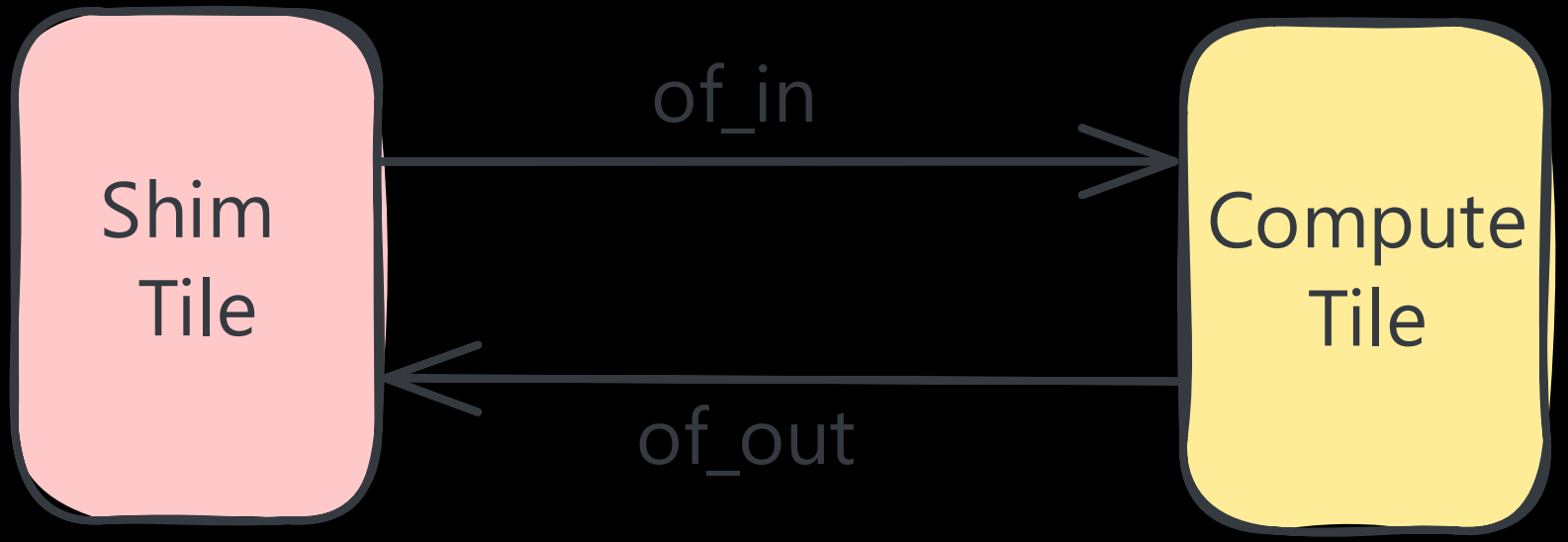
```
def my_vector_scalar():

  @device(AIEDevice.npu)
  def device_body():

    # AIE Core Function declarations
    scale_scalar = external_func("scale_scalar",inputs=[memRef_ty, memRef_ty])

    # Tile declarations
    ShimTile = tile(0, 0)
    ComputeTile2 = tile(0, 2)

    # AIE-array data movement with object fifos
    of_in = object_fifo("in", ShimTile, ComputeTile2, 2, memRef_ty)
    of_factor = object_fifo("infactor", ShimTile, ComputeTile2, 2, scalar_ty)
    of_out = object_fifo("out", ComputeTile2, ShimTile, 2, memRef_ty)

    # Set up compute tiles
    @core(ComputeTile2, "scale.o")
    def core_body():     # Number of sub-vector "tile" iterations
      for _ in for_(4):
        elem_factor = of_factor.acquire(ObjectFifoPort.Consume, 1)
        elem_out = of_out.acquire(ObjectFifoPort.Produce, 1)
        elem_in = of_in.acquire(ObjectFifoPort.Consume, 1)
        call(scale_scalar, [elem_in, elem_out, elem_factor, 1024])
        of_in.release(ObjectFifoPort.Consume, 1)
        of_out.release(ObjectFifoPort.Produce, 1)
        of_factor.release(ObjectFifoPort.Consume, 1)

    # To/from AIE-array data movement
    @FuncOp.from_py_func(tensor_ty, scalar_ty, tensor_ty)
    def sequence(A, F, C):
      ipu_dma_memcpy_nd(metadata="out", bd_id=0, mem=C, sizes=[1, 1, 1, 4096])
      ipu_dma_memcpy_nd(metadata="in", bd_id=1, mem=A, sizes=[1, 1, 1, 4096])
      ipu_dma_memcpy_nd(metadata="infactor", bd_id=2, mem=F, sizes=[1, 1, 1, 1])
      ipu_sync(column=0, row=0, direction=0, channel=0)
```

Connect to our Ryzen™ AI mini-PCs

- Join our local WIFI:
  - SSID: Iron_AIE or Iron_AIE_5G
  - Password: radaiewifi
- Connect to one of our mini-PCs using the machine name for your username on the paper
  - `ssh ….` or launch visual studio code session and connect
  - Username: see paper
  - Password: see paper (hint: radaie)/
- Source the setup script in: cd /home/<username>/ASPLOS; source setup.sh

hostname	IP	users
radaiedemo1	192.168.1.101	user1 → user10
radaiedemo2	192.168.1.102	user11 → user20
radaiedemo3	192.168.1.103	user21 → user30
radaiedemo4	192.168.1.104	user31 → user40
radaiedemo5	192.168.1.105	user41 → user50

Exercise 1: Build and run your first program



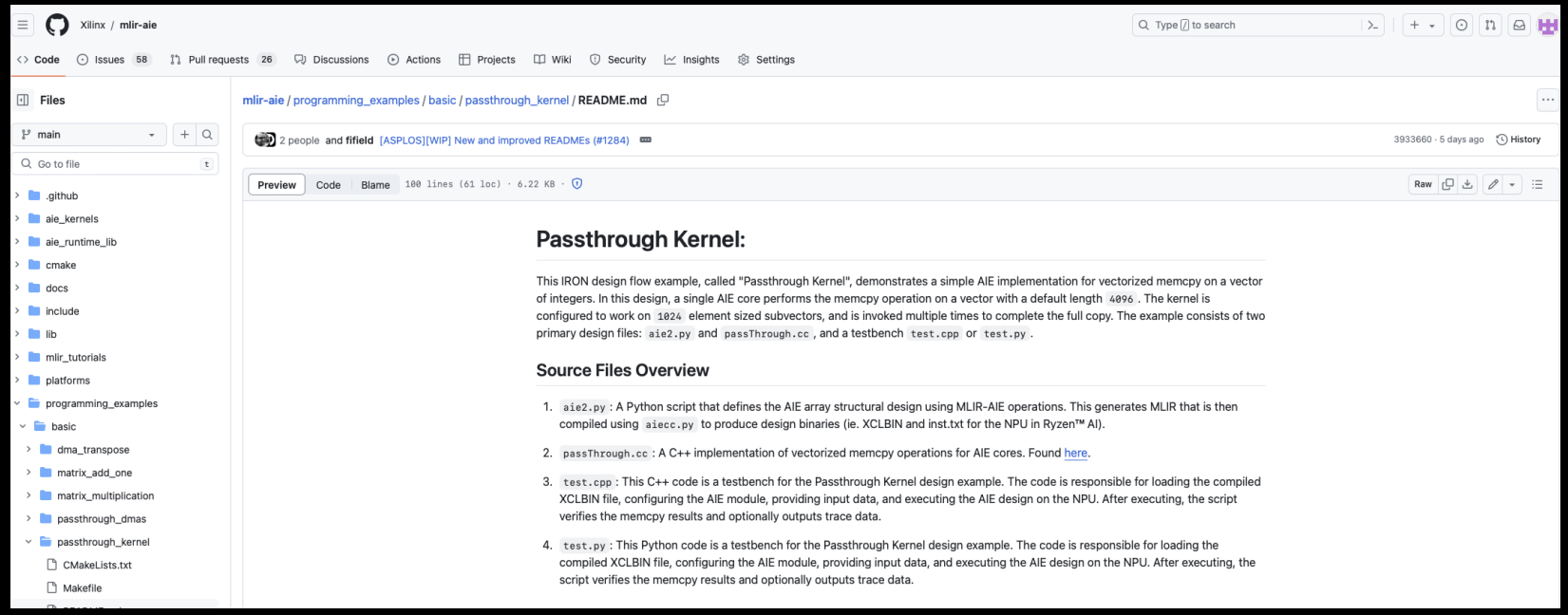
Passthrough Kernel

of_out = of_in

Run the passthrough_kernel on Hardware

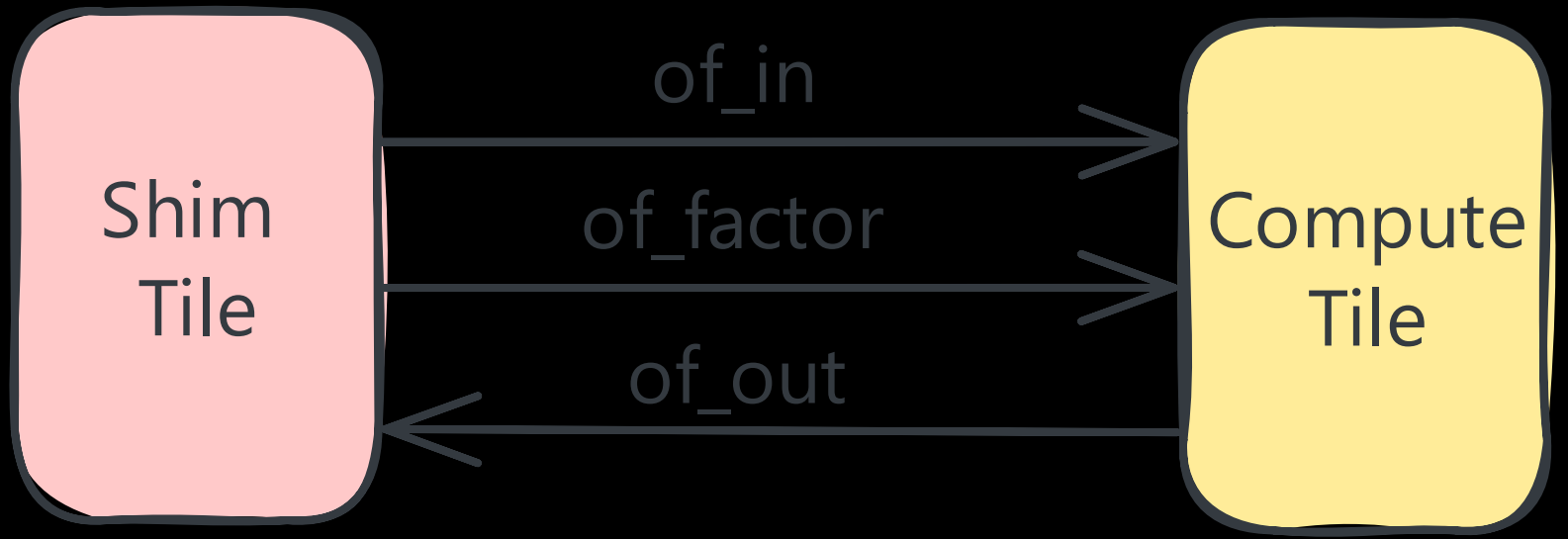
- Navigate in your browser to https://github.com/Xilinx/mlir-aie/tree/main/programming_examples
  - Contains all programming examples for completing the IRON AIE programming guide
  - Navigate to basic → passthrough_kernel (https://github.com/Xilinx/mlir-aie/tree/main/programming_examples/basic/passthrough_kernel)

  - On your local mlir-aie install, change directory to mlir-aie/programming_examples/basic/passthrough_kernel folder
  - Follow the instructions in the Usage section to build and run the design on the HW



Break: 10am – 10:30am

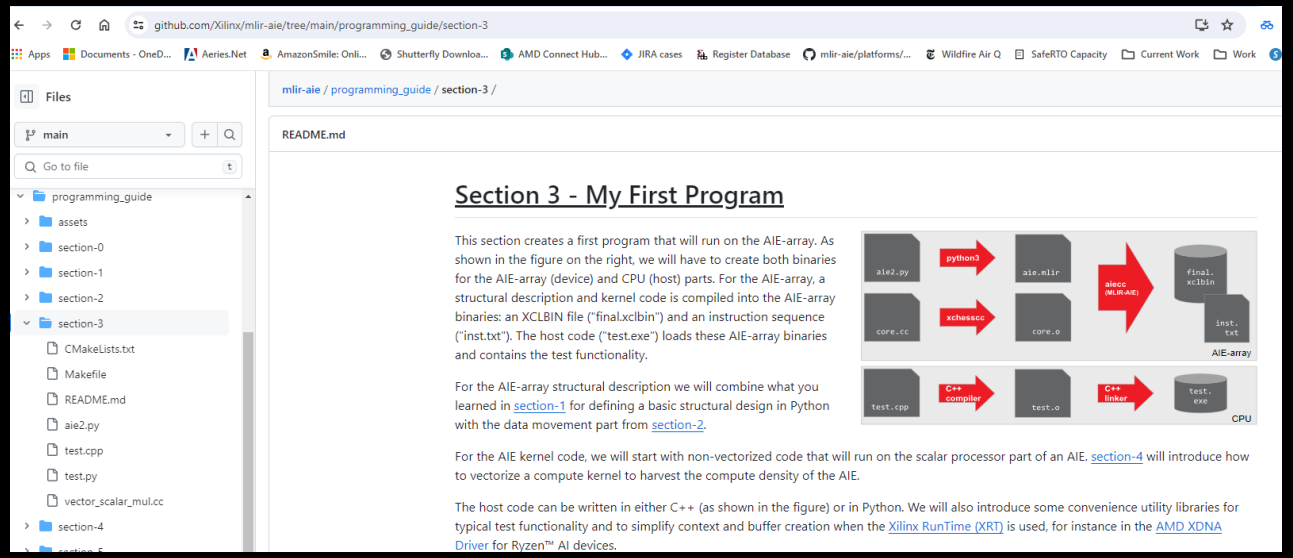
Exercise 2: Vector Scalar Mul



of_out = of_in * of_factor

Run the Vector Scalar Multiply Example on Hardware

- If you haven't yet, navigate in your browser to https://github.com/Xilinx/mlir-aie/tree/main /programming_guide
  - The remaining exercises will be guided by instructions here
  - Work at your own pace
  - Come back later to explore more
- Navigate now to Section 3 of our Programming Guide
  - (https://github.com/Xilinx/mlir-aie/tree/main/programming_guide/section-3)
  - On your local mlir-aie install, change the directory to mlir-aie/programming_guide/section-3 folder
  - Follow the instructions in Section 3 for building and running this design



Tracing and Performance Analysis

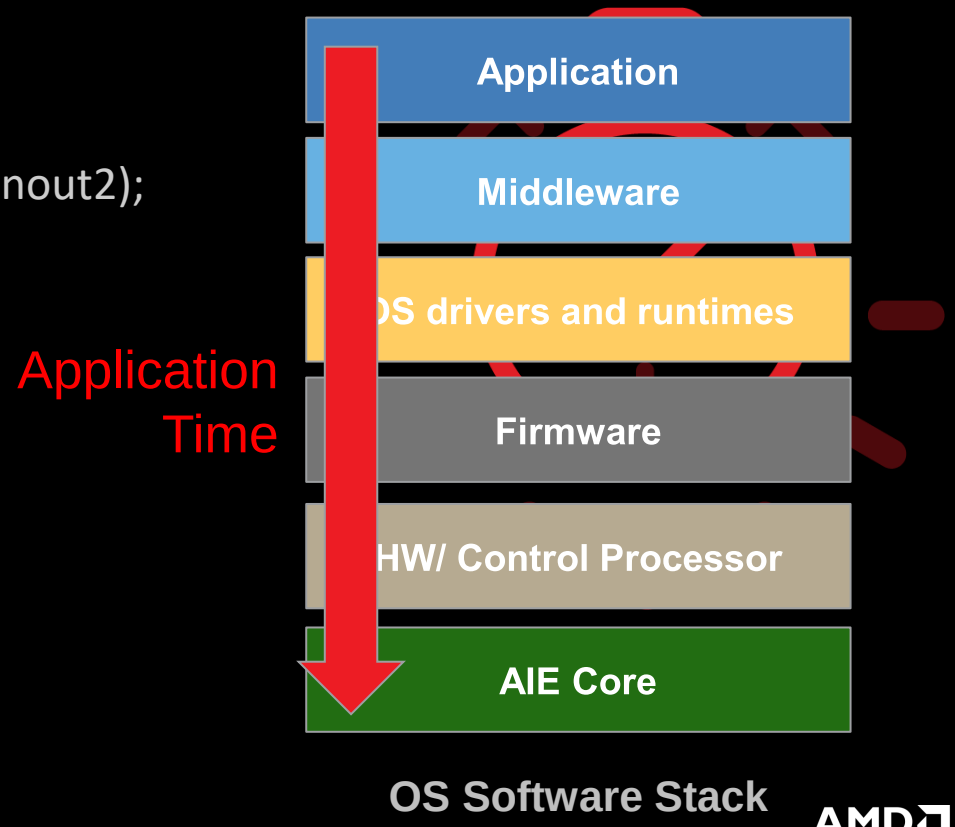
Fe: Fast and efficient Designs

- Measuring performance is key to building efficient designs
- Different ways to measure performance
  - Latency
  - Throughput
  - Power & power efficiency
  - Hardware utilization
- Area of ongoing research & development

Fe: Fast and efficient Designs with Timers

- “Wall clock” time is a helpful metric to measure the overall performance of our application
  - Gives upper bounds AIE application
    - Include software stack overhead like OS/ cache, kernel drivers, and communication overheads

```
auto start = std::chrono::high_resolution_clock::now();
auto run = kernel(bo_instr, instr_v.size(), bo_inout0, bo_inout1, bo_inout2);
run.wait();
auto stop = std::chrono::high_resolution_clock::now();

bo_inout2.sync(XCL_BO_SYNC_BO_FROM_DEVICE);
```



Fe: Fast and efficient Designs with Timers

- Multiple iterations
  - Helps smooth out variability of runs
- Warmup
  - Removes higher initial runtimes from average calculation

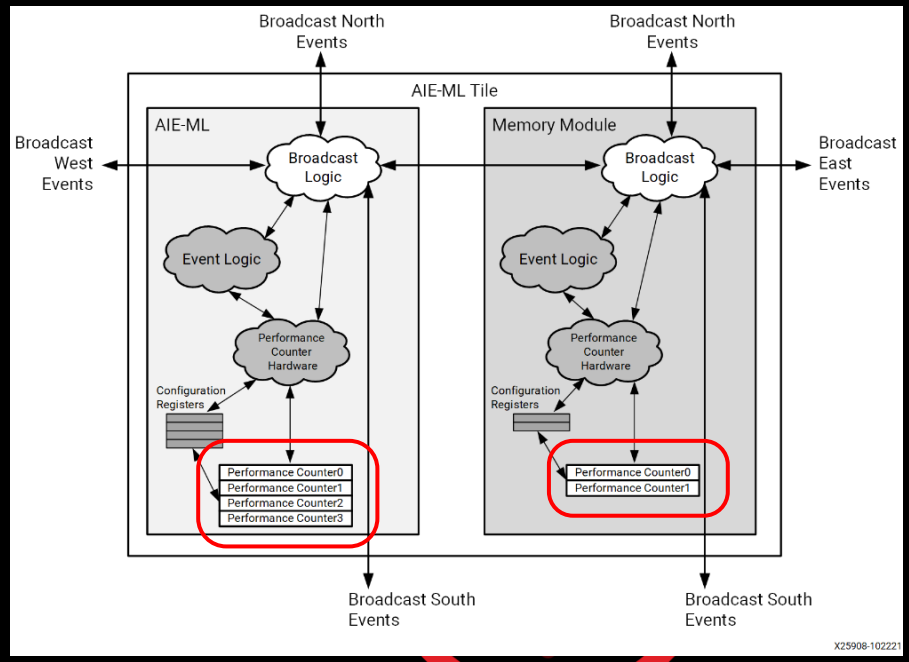
```
for (unsigned iter = 0; iter < num_iter; iter++) {
  <... kernel run code ...>
  if (iter < n_warmup_iterations) {
    /* Warmup iterations do not count towards average runtime. */
    continue;
  }
  <... verify and measure timers ...>

  npu_time_total += npu_time;
  npu_time_min = (npu_time < npu_time_min) ? npu_time : npu_time_min;
  npu_time_max = (npu_time > npu_time_max) ? npu_time : npu_time_max;
}

std::cout << "Avg NPU time: " << npu_time_total / n_iterations << "us." << std::endl;
std::cout << "Min NPU time: " << npu_time_min << "us." << std::endl;
std::cout << "Max NPU time: " << npu_time_max << "us." << std::endl;
```

Fe: Fast and efficient Designs with (Other) Timers

- 1x 64-bit free running timer in all 4 tile types
  - Tile Core, Tile Memory, Mem Tile, Shim Tile

- Core Tile has 6 counters (tile core – 4, tile memory – 2)
  - Can configure counters to trigger off events
  - Useful for reporting runtime counts of specific events

- However, a full trace of events over time (via trace event packets) gives a more complete cycle-accurate performance picture

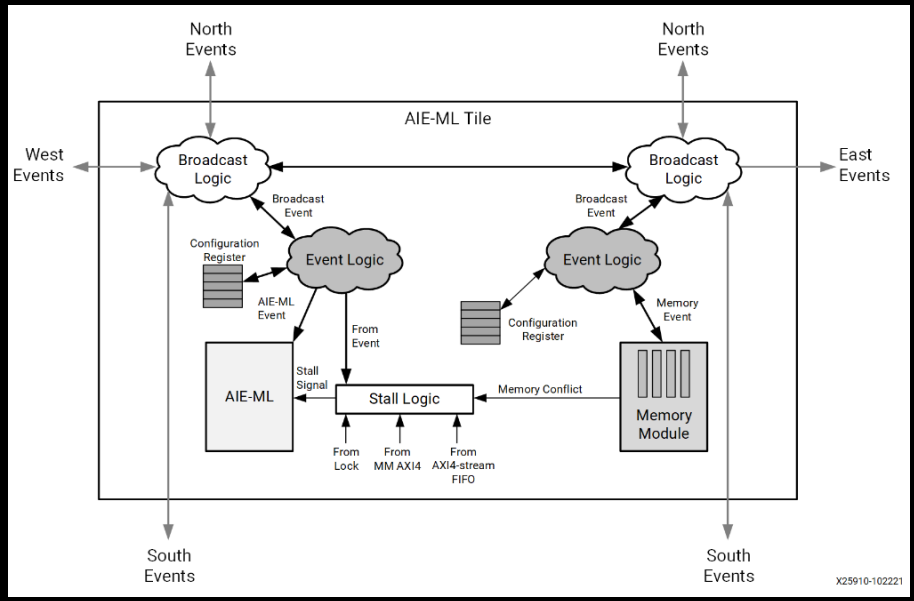


Fe: Fast and efficient Designs with Trace

- Trace Units
  - Present in all 4 tile types
    - Tile Core, Tile Memory, MemTile, ShimTile
  - Reconfigure to monitor up to 8 events (out of >100 events)
  - Generate trace packets based on events
  - Supported tile core trace modes
    - Event-time – cycle accurate event packets tracking event changes
    - Event-PC – cycle accurate event packets + PC counter every cycle
    - Execution-Trace – tracks program execution flow (not cycle accurate)

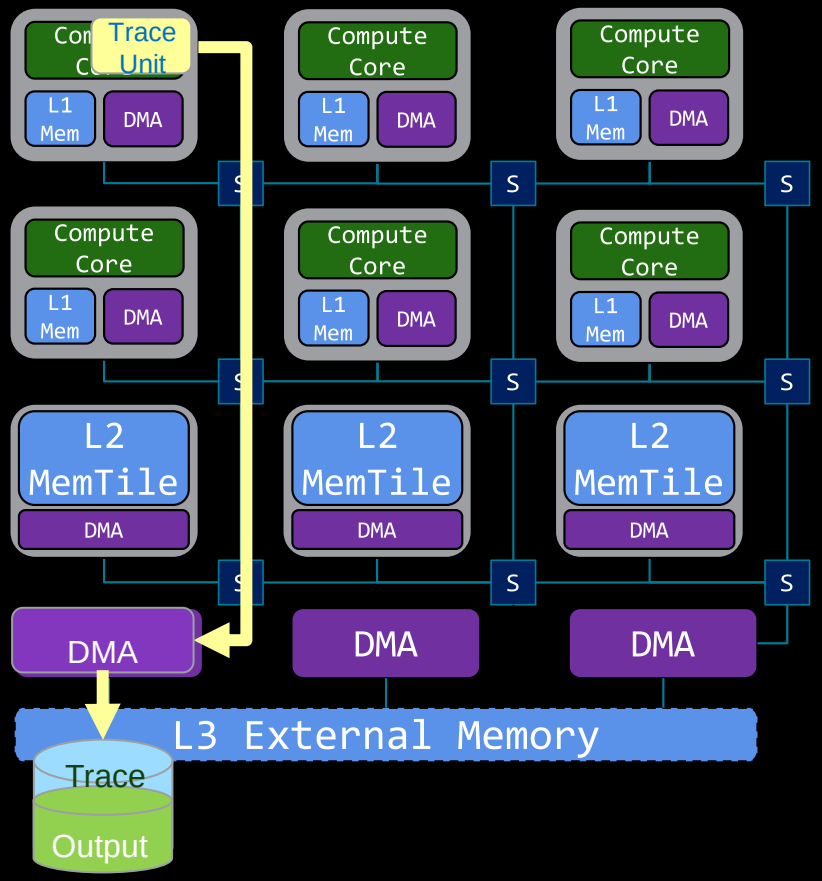
- Trace events monitoring and generation do not affect performance, however …
  - Routing of packets through stream switch uses routing resource
  - Moving data to DDR (L3) uses ShimDMA channel

- Vitis supports advanced profiling and execution trace for integrated software debugging
  - IRON provide hooks at the metal layer to configuring hardware for trace enabling custom trace processing



Fe: Fast and efficient Designs with Trace

- Trace → visualizing the relevant AIE events for our application
  - Cycle accurate waveform for hardware trace events
    - Start of the kernel (event0), end of the kernel (event1), active port transfer, lock stalls, memory stalls, stream stalls, vector instructions, etc



aie2.py

```
if enableTrace:
    trace_utils.configure_simple_tracing_aie2(
        ComputeTile2,
        ShimTile,
        ddr_id=2,
        size=trace_size,
        offset=4096 * 4,  # offset in bytes
    )

if enableTrace:
        flow(ComputeTile2, WireBundle.Trace, 0,
             ShimTile, WireBundle.DMA, 1)
```

Config Trace and ShimDMA

Route Trace Packets

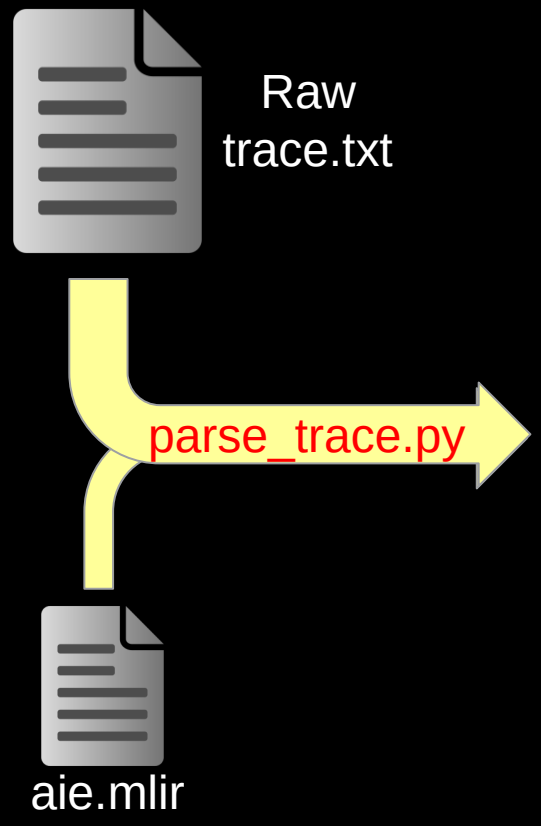
test.cpp

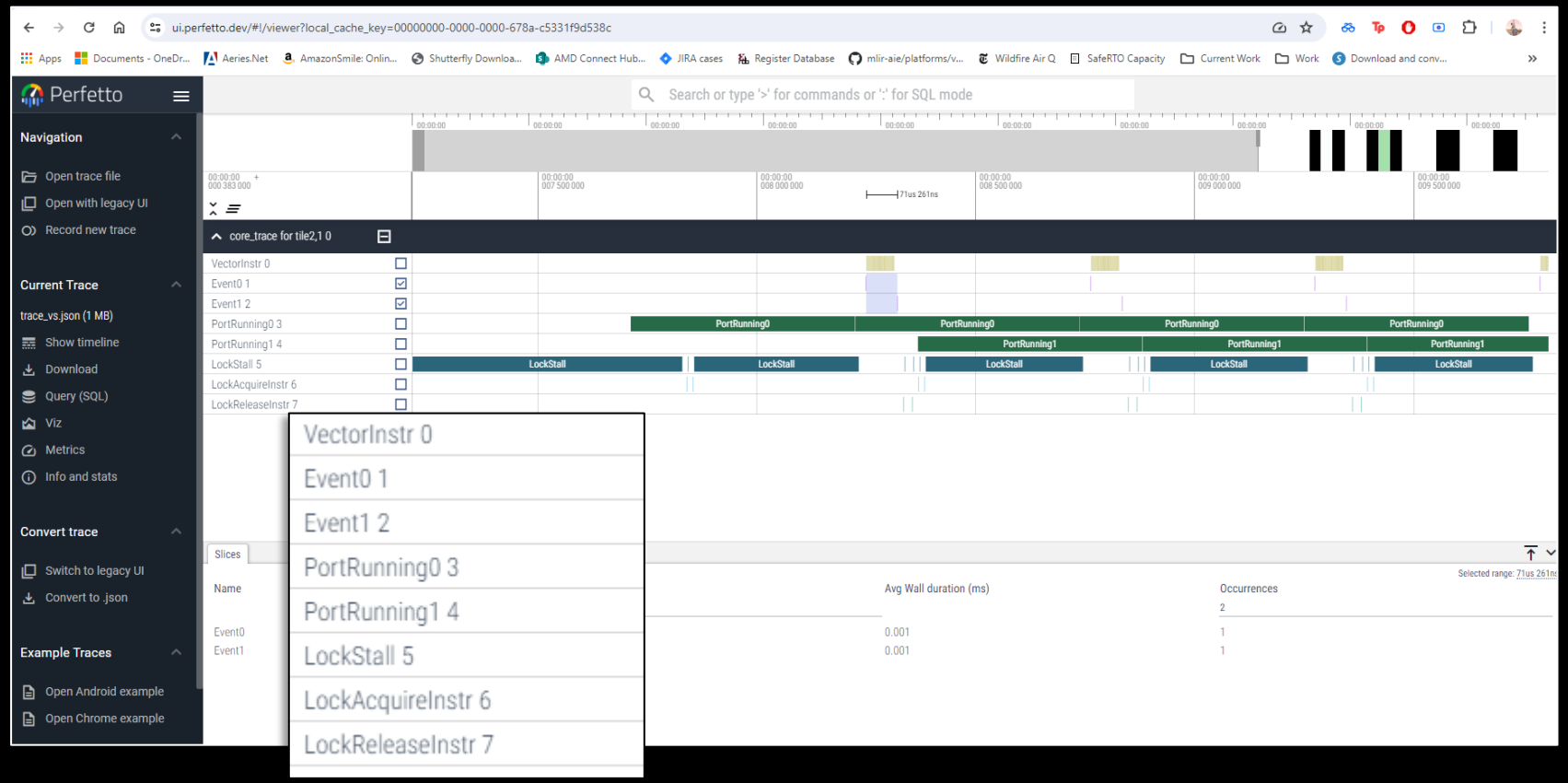
```
if (trace_size > 0) {
  test_utils::write_out_trace(((char *)bufOut) + IN_SIZE,
                              trace_size, vm["trace_file"].as<std::string>());
}
```

Write trace data to file

Fe: Fast and efficient Designs with Trace

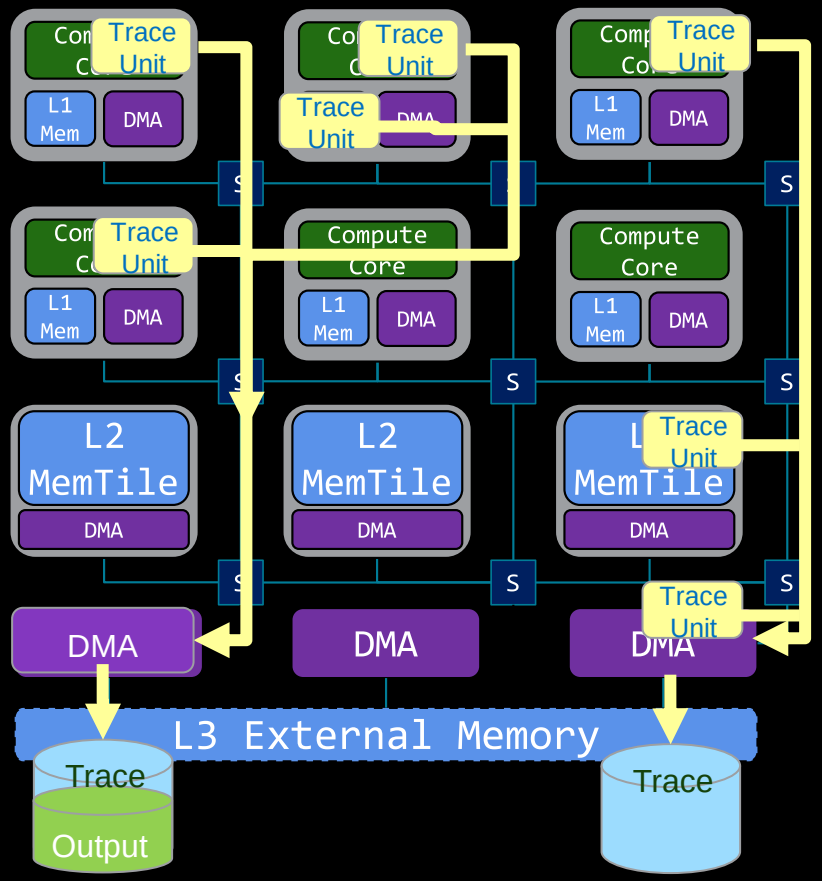
- Use trace parser (parse_trace.py) to convert raw trace.txt file to waveform json (trace.json)
- View results (trace.json) in web browser (http://ui.perfetto.dev)





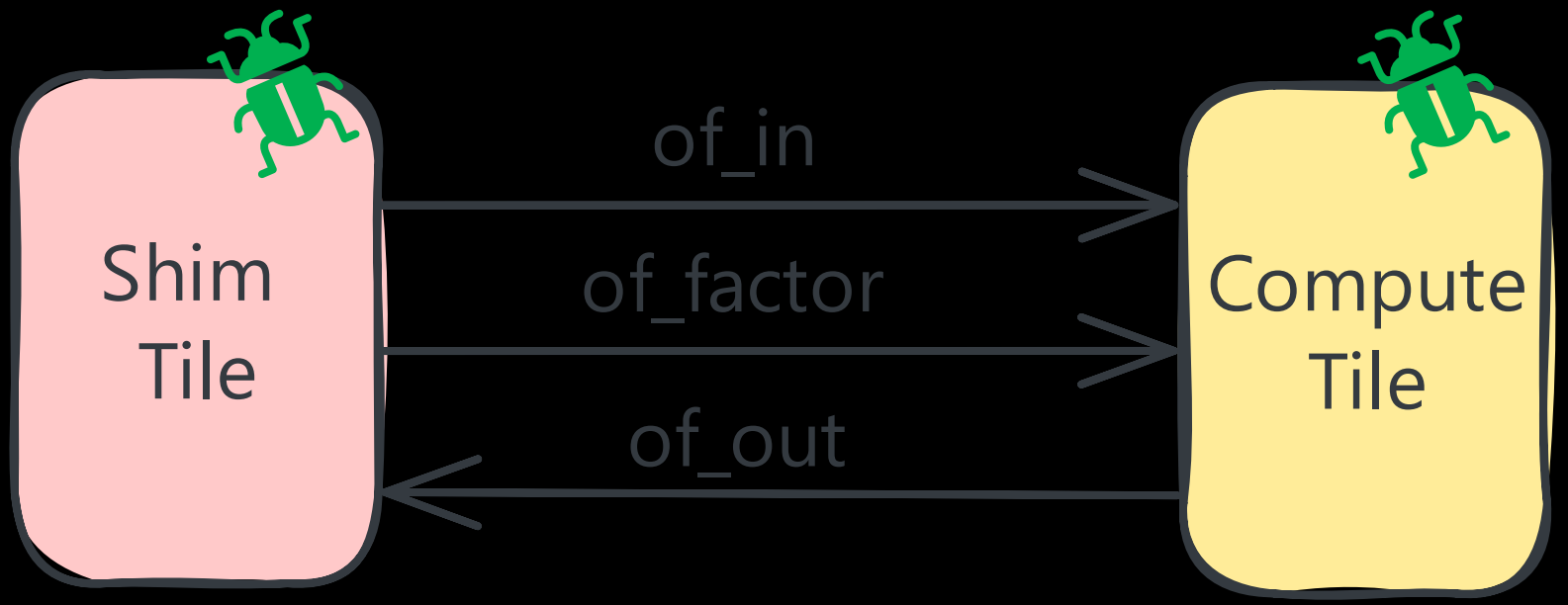
Fe: Fast and efficient Designs with Trace

- IRON currently provides abstractions for tracing single AIE core tile
- Support for multi-tile tracing and trace auto-routing is coming soon



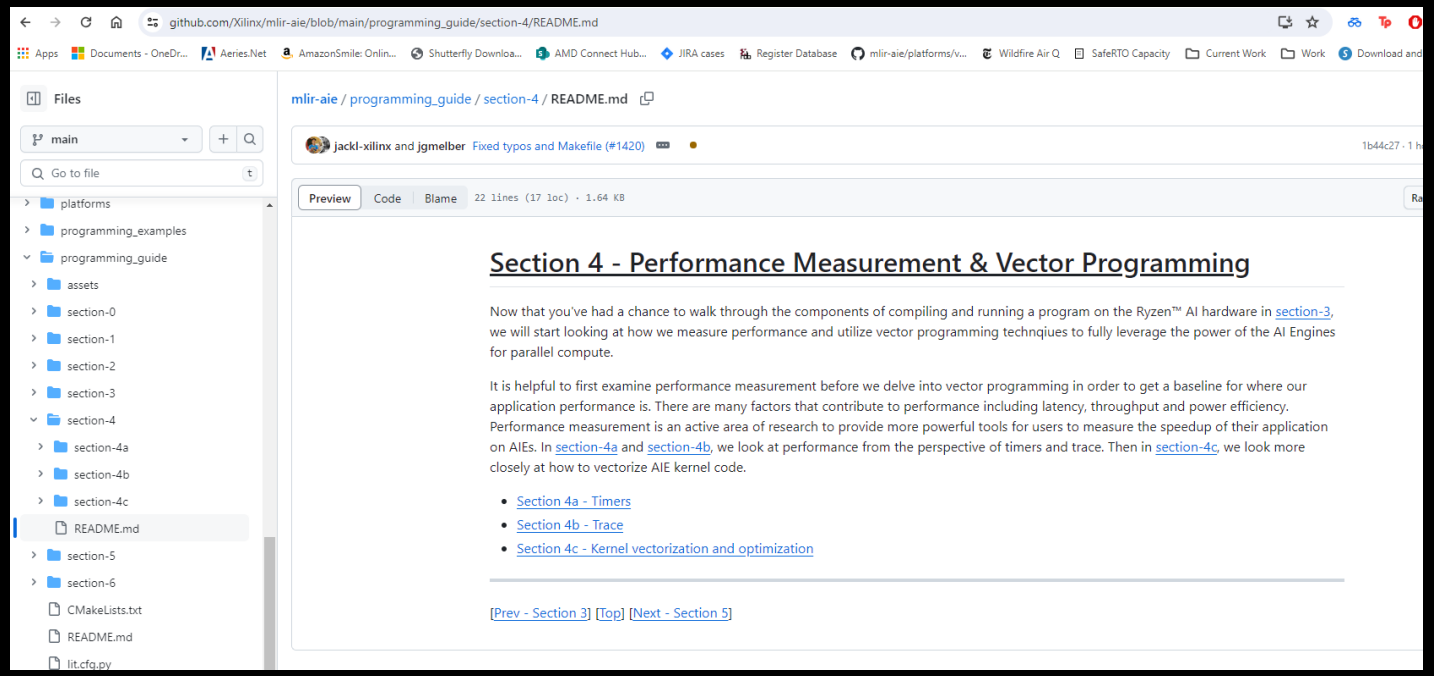
- Other Use Cases
  - Route packets from multiple tiles to single ShimDMA
  - Routing of Tile Memory trace packets
  - Routing of MemTile trace packets
  - Routing of ShimTile trace packets
  - Multi ShimDMA config → unified output trace buffer

Exercise 3: Tracing Vector Scalar Mul

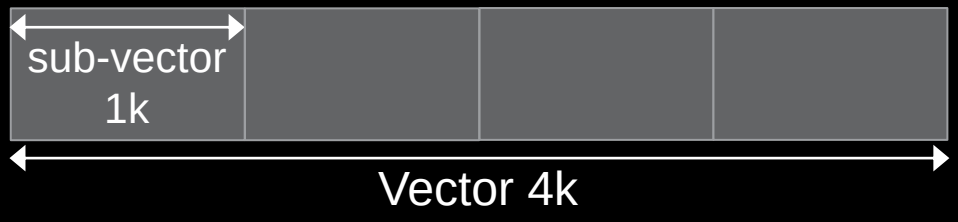


Timers and Trace Exercises

- Navigate now to Section 4 of our Programming Guide
  - (https://github.com/Xilinx/mlir-aie/tree/main/programming_guide/section-4)
  - On your local mlir-aie install, change directory to mlir-aie/programming_guide/section-4/section-4a folder
  - Follow the instructions in Section 4
    - Section-4a – Timers
    - Section-4b – Trace

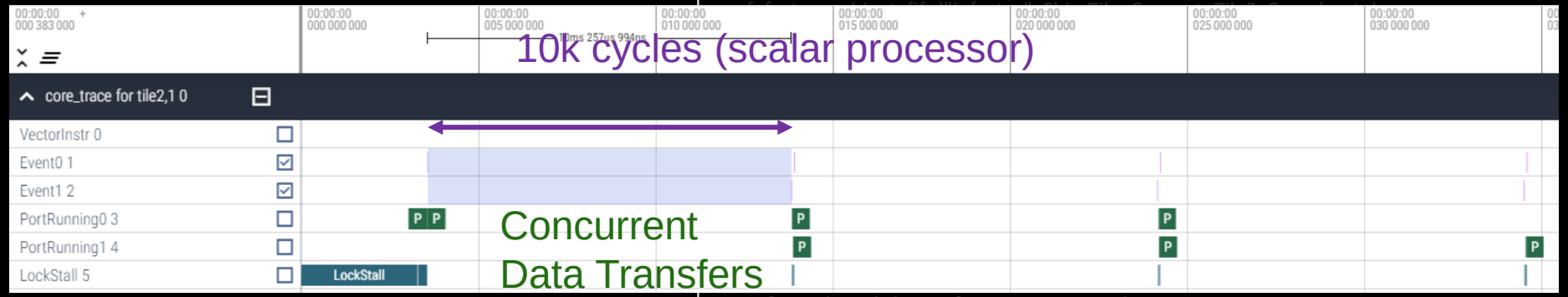


Hello World
vector_scalar: c = a*factor



AIE Array Code

```
def my_vector_scalar():

  @device(AIEDevice.npu)
  def device_body():

    # AIE Core Function declarations
    scale_scalar = external_func("scale_scalar",inputs=[memRef_ty, memRef_ty])

    # Tile declarations
    ShimTile = tile(0, 0)
    ComputeTile2 = tile(0, 2)

    # AIE-array data movement with object fifos
    of_in = object_fifo("in", ShimTile, ComputeTile2, 2, memRef_ty)
```



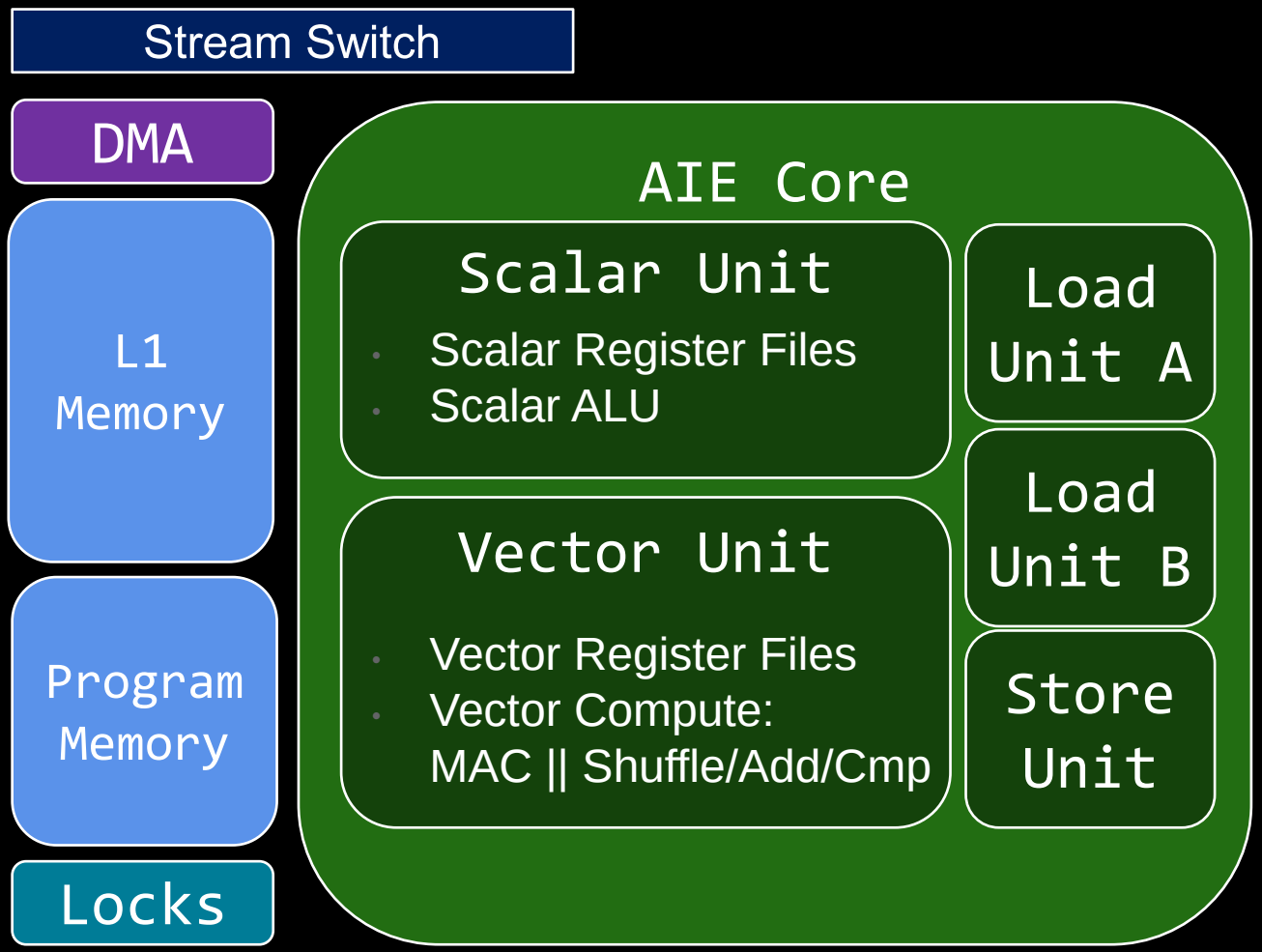

```
    of_in.release(ObjectFifoPort.Consume, 1)
    of_out.release(ObjectFifoPort.Produce, 1)
    of_factor.release(ObjectFifoPort.Consume, 1)

  # To/from AIE-array data movement
  @FuncOp.from_py_func(tensor_ty, scalar_ty, tensor_ty)
  def sequence(A, F, C):
    ipu_dma_memcpy_nd(metadata="out", bd_id=0, mem=C, sizes=[1, 1, 1, 4096])
    ipu_dma_memcpy_nd(metadata="in", bd_id=1, mem=A, sizes=[1, 1, 1, 4096])
    ipu_dma_memcpy_nd(metadata="infactor", bd_id=2, mem=F, sizes=[1, 1, 1, 1])
    ipu_sync(column=0, row=0, direction=0, channel=0)
```

AIE Core Code (Scalar Version)

```
void scale_int16(int16_t *a, int16_t *c,
                 int16_t factor) {

  for (int i = 0; i < 1024; i++) {
    c[i] = factor * a[i];
  }
}
```

Vectorizing on AIE

AIE Compute Tile Architecture: Compute Details



Precision 1	Precision 2	Acc Lanes	Bits per Acc Lane	MACs
int 8	int 4	32	32	512
int 8	int 8	32	32	256
int 16	int 8	32	32	128
int 16	int 16	32	32	64
int 32	int 16	16	64	32
int 32 [1]	int 32	16	64	16
bfloat 16 [3]	bfloat 16	16	SPFP 32 [2]	128

1.int32 x int32 can be emulated. The operation should have half the performance of int32 x int16 and there should be 16 multiplications per cycle.
2.Single precision floating point (SPFP) per the IEEE standard.
3.float32 x float32 can be emulated. Emulation deviates from the IEEE-754 standard.

AIE-ML AIE API Manual: https://www.xilinx.com/htmldocs/xilinx2023_2/aiengine_api/aie_api/doc/

AIE Compute Tile: Compute Dense, SW Programmable Signal Processor with Zero Loop Overhead on Counters and Buffer Pointer Auto Increment


```
void processing_int32(int32_t *in, int32_t *out, int32_t parameter) {

 for (int i = 0; i < iLoopBound; i++) {
  for (int j = 0; j < jLoopBound; j++) {
   aie::vector< int32_t, 16> vectorOfData = aie::load_v<16>(in++);
   aie::accum<acc64, vec_factor> cout =
                          aie::vectorProcessing(vectorOfData, parameter);
   aie::store_v(out++,cout);
  }
 }

}
```

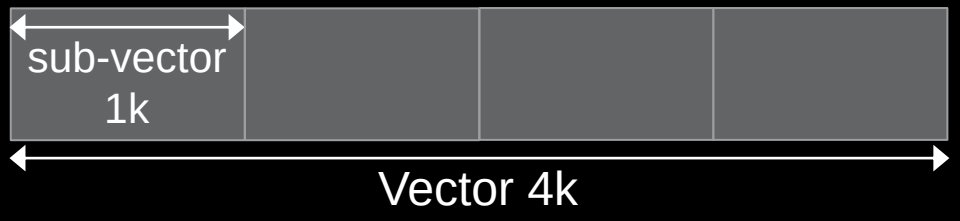
Zero counter overhead on nested loop

Zero pointer increment overhead

Enable back-to-back vector ops

AIE-ML AIE API Manual: https://www.xilinx.com/htmldocs/xilinx2023_2/aiengine_api/aie_api/doc/

IRON: Vectorizing vector_scalar: c = a*factor

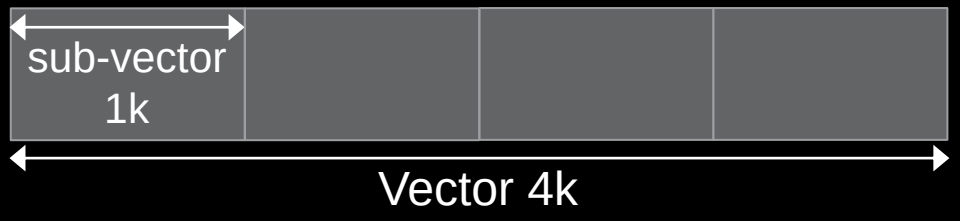


1. Find vector operation, its vector and accumulator size:
   mul/mac, int16 x int16 → vector size 32
   → acc type 32

AIE Core Code (Scalar Version)

```
void scale_int16(int16_t *a, int16_t *c,
                 int16_t factor) {

  for (int i = 0; i < 1024; i++) {
    c[i] = factor * a[i];
  }
}
```

IRON: Vectorizing vector_scalar: c = a*factor

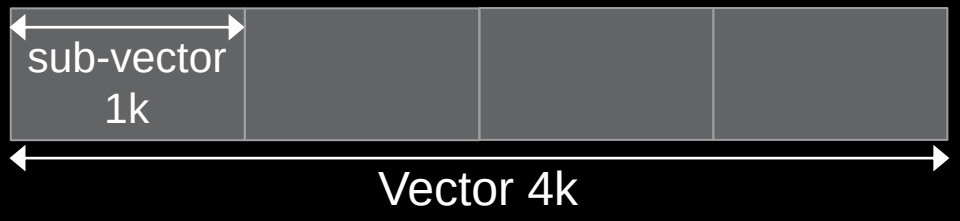


1. Find vector operation, its vector and accumulator size:
   mul/mac, int16 x int16 → vector size 32
   → acc type 32

AIE Core Code (Vectorizing WIP)

```
void scale_int16(int16_t *a, int16_t *c,
                 int16_t factor) {

  for (int i = 0; i < 1024/32; i++) {
    aie::accum<acc32, 32> cout = aie::mul(a, factor);
  }
}
```

IRON: Vectorizing vector_scalar: c = a*factor

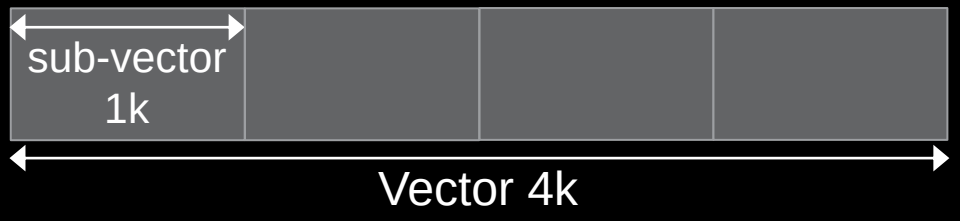


AIE Core Code (Vectorizing WIP)

```
void scale_int16(int16_t *a, int16_t *c,
                 int16_t factor) {

  for (int i = 0; i < 1024/32; i++) {
    aie::vector< int16_t, 32> a0 = aie::load_v<32>(a);
    a += 32;
    aie::accum<acc32, 32> cout = aie::mul(a, factor);
  }
}
```

1. Find vector operation, its vector and accumulator size:
   mul/mac, int16 x int16 → vector size 32
   → acc type 32
2. Load input into vector register and increment input pointer

IRON: Vectorizing vector_scalar: c = a*factor

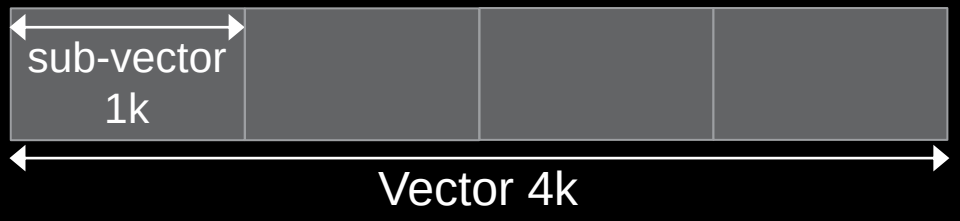


AIE Core Code (Vectorizing WIP)

```
void scale_int16(int16_t *a, int16_t *c,
                 int16_t factor) {

  for (int i = 0; i < 1024/32; i++) {
    aie::vector< int16_t, 32> a0 = aie::load_v<32>(a);
    a += 32;
    aie::accum<acc32, 32> cout = aie::mul(a0, factor);
    aie::store_v(c,cout.to_vector<int16_t>(0));
    c += 32;
  }
}
```

1. Find vector operation, its vector and accumulator size:
   mul/mac, int16 x int16 → vector size 32
   → acc type 32
2. Load input into vector register and increment input pointer
3. Cast and store the result from the accumulator into the output and increment the output pointer

IRON: Vectorizing vector_scalar: c = a*factor

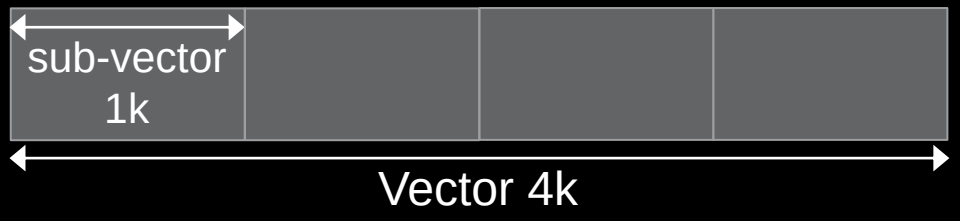


AIE Core Code (Vectorizing WIP)

```
void scale_int16(int16_t *a, int16_t *c,
                 int16_t factor) {

  for (int i = 0; i < 1024/32; i++)
    chess_prepare_for_pipelining chess_loop_range(16, ) {
    aie::vector< int16_t, 32> a0 = aie::load_v<32>(a);
    a += 32;
    aie::accum<acc32, 32> cout = aie::mul(a0, factor);
    aie::store_v(c,cout.to_vector<int16_t>(0));
    c += 32;
  }
}
```

1. Find vector operation, its vector and accumulator size:
   mul/mac, int16 x int16 → vector size 32
   → acc type 32
2. Load input into vector register and increment input pointer
3. Cast and store the result from the accumulator into the output and increment the output pointer
4. Add optional pragmas to further optimize design

IRON: Vectorizing vector_scalar: c = a*factor

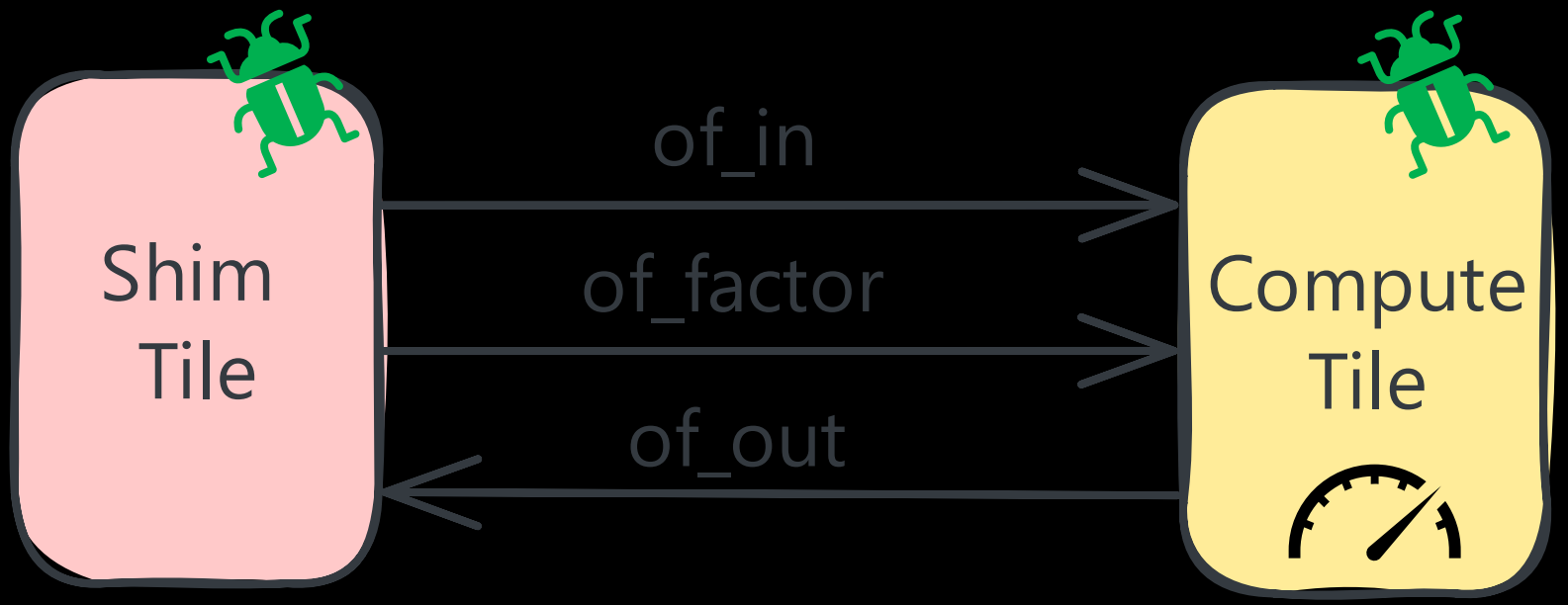


AIE Core Code (Vectorized)

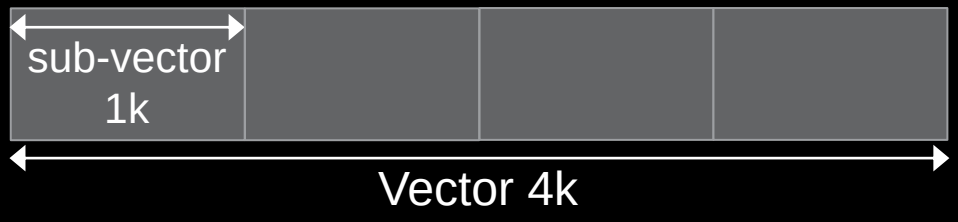
```
void scale_int16(int16_t *a, int16_t *c,
                 int16_t factor) {

  for (int i = 0; i < 1024/32; i++)
     chess_prepare_for_pipelining chess_loop_range(16, ) {
    aie::vector< int16_t, 32> a0 = aie::load_v<32>(a);
    a += 32;
    aie::accum<acc32, 32> cout = aie::mul(a0, factor);
    aie::store_v(c,cout.to_vector<int16_t>(0));
    c += 32;
  }
}
```

1. Find vector operation, its vector and accumulator size:
   mul/mac, int16 x int16 → vector size 32
   → acc type 32
2. Load input into vector register and increment input pointer
3. Cast and store the result from the accumulator into the output and increment the output pointer
4. Add optional pragmas to further optimize design

Exercise 4: Tracing *Vectorized* Vector Scalar Mul

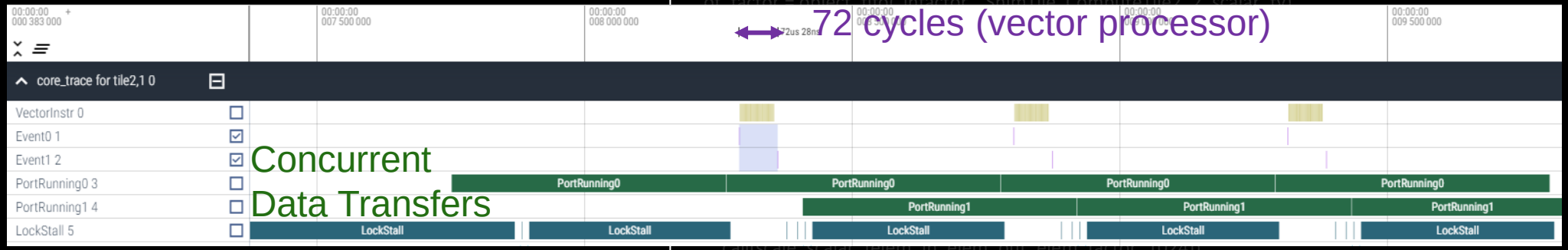


Hello World Vectorized vector_scalar: c = a*factor



AIE Array Code

```
def my_vector_scalar():

  @device(AIEDevice.npu)
  def device_body():

    # AIE Core Function declarations
    scale_scalar = external_func("scale_scalar",inputs=[memRef_ty, memRef_ty])

    # Tile declarations
    ShimTile = tile(0, 0)
    ComputeTile2 = tile(0, 2)

    # AIE-array data movement with object fifos
    of_in = object_fifo("in", ShimTile, ComputeTile2, 2, memRef_ty)
    of_factor = object_fifo("infactor", ShimTile, ComputeTile2, 2, scalar_ty)
                                                                  call(scale_scalar, [elem_in, elem_out, elem_factor, 1024])
    of_in.release(ObjectFifoPort.Consume, 1)
    of_out.release(ObjectFifoPort.Produce, 1)
    of_factor.release(ObjectFifoPort.Consume, 1)

  AIE-array data movement
  from_py_func(tensor_ty, scalar_ty, tensor_ty)
  nce(A, F, C):
  _memcpy_nd(metadata="out", bd_id=0, mem=C, sizes=[1, 1, 1, 4096])
  _memcpy_nd(metadata="in", bd_id=1, mem=A, sizes=[1, 1, 1, 4096])
  _memcpy_nd(metadata="infactor", bd_id=2, mem=F, sizes=[1, 1, 1, 1])
  (column=0, row=0, direction=0, channel=0)
```

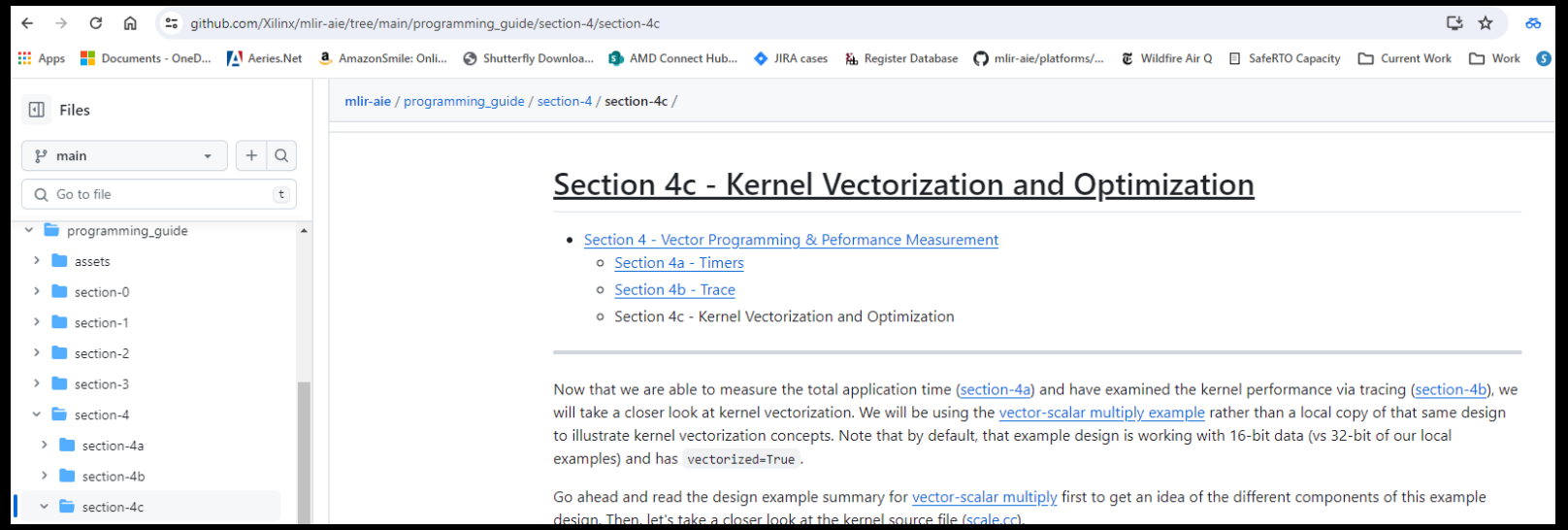


AIE Core Code (Vector Version)

```
void scale_int16(int16_t *a, int16_t *c,
                 int16_t factor) {

  for (int i = 0; i < 1024/32; i++)
     chess_prepare_for_pipelining chess_loop_range(16, ) {
    aie::vector< int16_t, 32> a0 = aie::load_v<32>(a);
    aie::accum<acc32, vec_factor> cout = aie::mul(a0, factor);
    aie::store_v(c,cout.to_vector<int16_t>(0));
  }
}
```

Vectorization Exercises

- Navigate now to Section 4c of our Programming Guide
  - (https://github.com/Xilinx/mlir-aie/tree/main/programming_guide/section-4/section-4c)
  - On your local mlir-aie install, change directory to mlir-aie/programming_guide/section-4/section-4c folder
  - Follow the instructions in Section-4c up to Vectorization Exercises
- NOTE – We will be working with programming_examples/basic/vector_scalar_mul example directly
  - e.g., Edit programming_examples/basic/vector_scalar_mul/aie2.py to set vectorized=False
  - e.g., Edit aie_kernels/aie2/scale.cc to comment/ uncomment pragmas



Dataflow and Larger Designs

Section 5: Example Vector Designs

Basic

Design name	Data type	Description
Vector Scalar Add	i32	Adds 1 to every element in vector
Vector Scalar Mul	i32	Returns a vector multiplied by a scale factor
Vector Reduce Add	bfloat16	Returns the sum of all elements in a vector
Vector Reduce Max	bfloat16	Returns the maximum of all elements in a vector
Vector Reduce Min	bfloat16	Returns the minimum of all elements in a vector
Vector Exp	bfloat16	Returns a vector representing e^x of the inputs

Machine Learning Kernels

Design name	Data type	Description
Eltwise Add	bfloat16	An element by element addition of two vectors
Eltwise Mul	i32	An element by element multiplication of two vectors
ReLU	bfloat16	Rectified linear unit (ReLU) activation function on a vector
Softmax	bfloat16	Softmax operation on a matrix
Single core GEMM	bfloat16	A single core matrix-matrix multiply
Multi core GEMM	bfloat16	A matrix-matrix multiply using 16 AIEs with operand broadcast. Uses a simple "accumulate in place" strategy
GEMV	bfloat16	A vector-matrix multiply returning a vector
Conv2D	i8	A single core 2D convolution for CNNs
Conv2D+ReLU	i8	A Conv2D with a ReLU fused at the vector register level

Section 6: Larger Example Designs

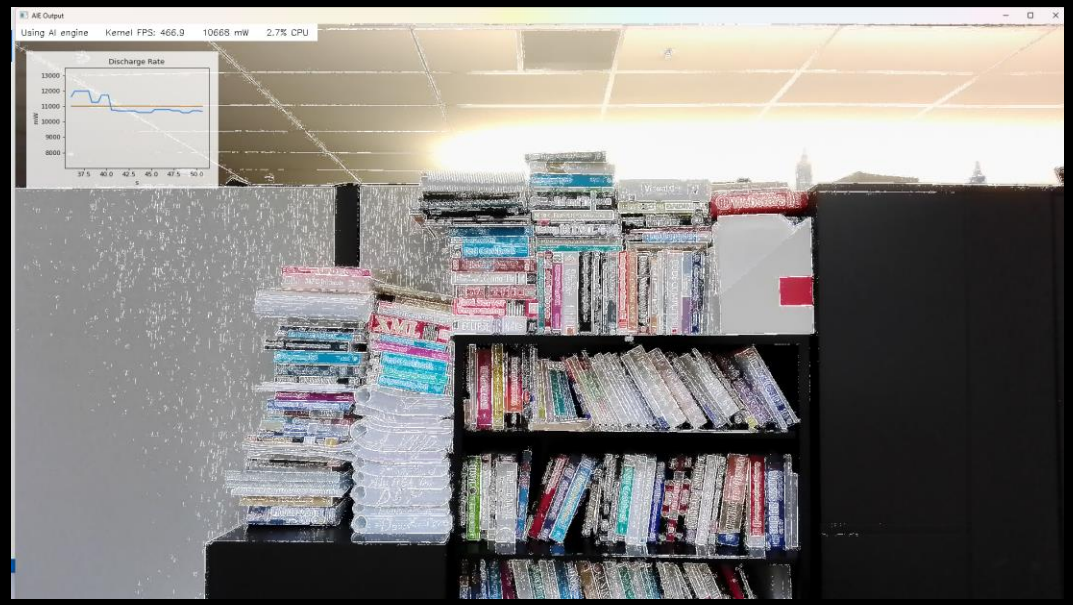
Vision Kernels

Design name	Data type	Description
Vision Passthrough	i8	A simple pipeline with just one `passThrough` kernel. This pipeline's main purpose is to test whether the data movement works correctly to copy a greyscale image.
Color Detect	i32	This multi-kernel, multi-core pipeline detects colors in an RGBA image.
Edge Detect	i32	A mult-kernel, multi-core pipeline that detects edges in an image and overlays the detection on the original image.
Color Threshold	i32	A mult-core data-parallel implementation of color thresholding of a RGBA image.

Machine Learning Designs

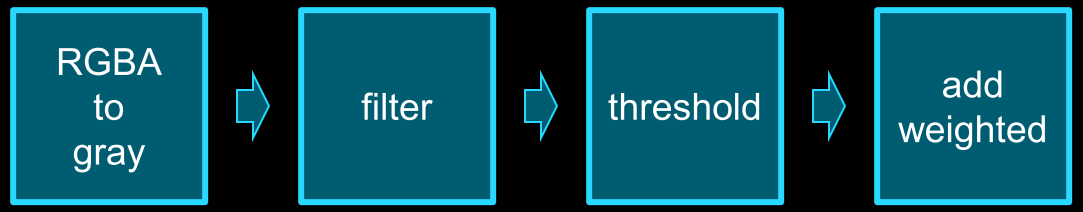
Design name	Data type	Description
bottleneck	ui8	A Bottleneck Residual Block is a variant of the residual block that utilises three convolutions, using 1x1, 3x3 and 1x1 filter sizes, respectively. The use of a bottleneck reduces the number of parameters and computations.
resnet	ui8	ResNet with offloaded conv2_x bottleneck blocks. The implementation features kernel fusion and dataflow optimizations highlighting the unique architectural capabilties of AI Engines.

Motivation: Edge Detect Without Burning Up Your Laptop

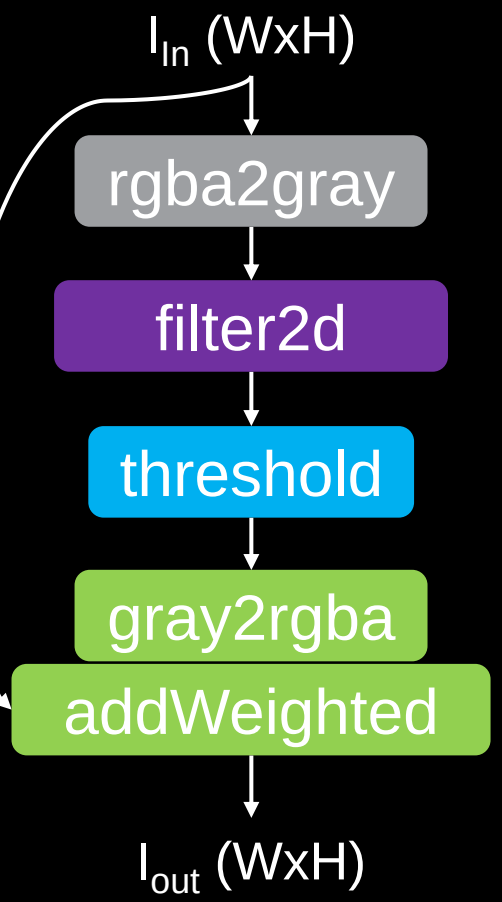


- Edge Detect Pipeline Running on AI-Engine-Enabled Ryzen™ AI Laptop
- Kernel and pipeline designs developed with IRON and compiled with mlir-aie toolflow
- Power Draw Drop: (throttled)
  ~ 1 W
- Increase in Kernel Rate:
  80 FPS → 130 FPS

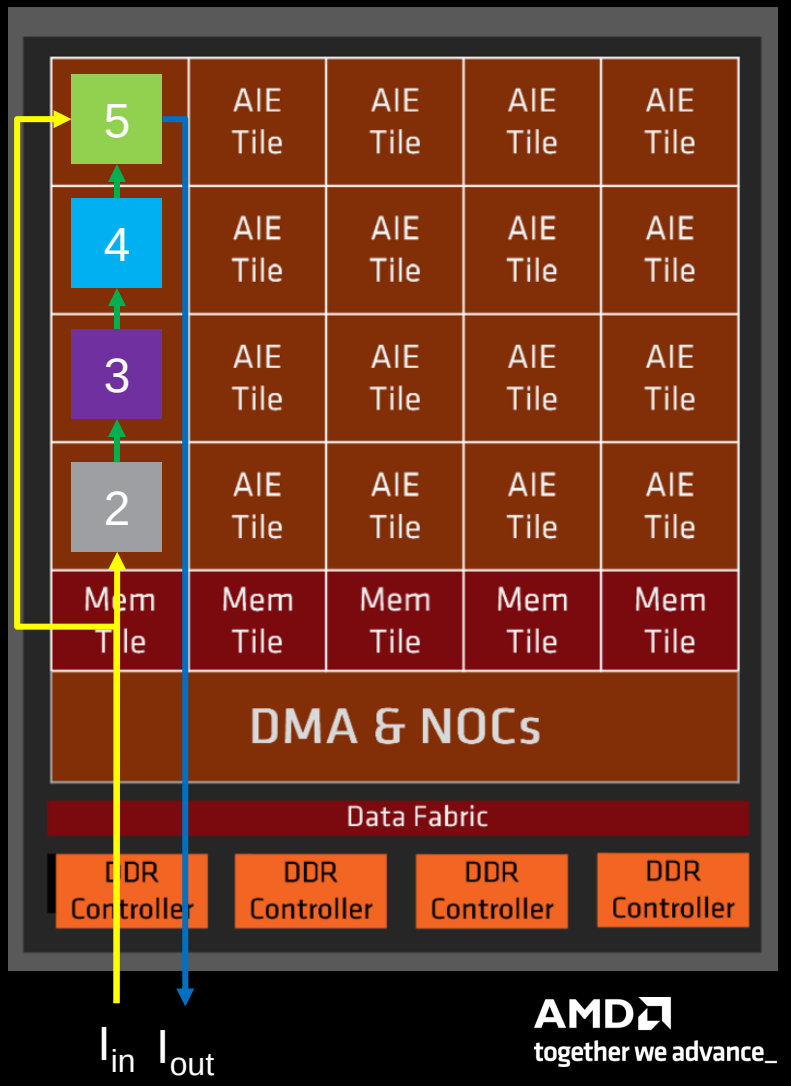
Edge Detect Pipeline



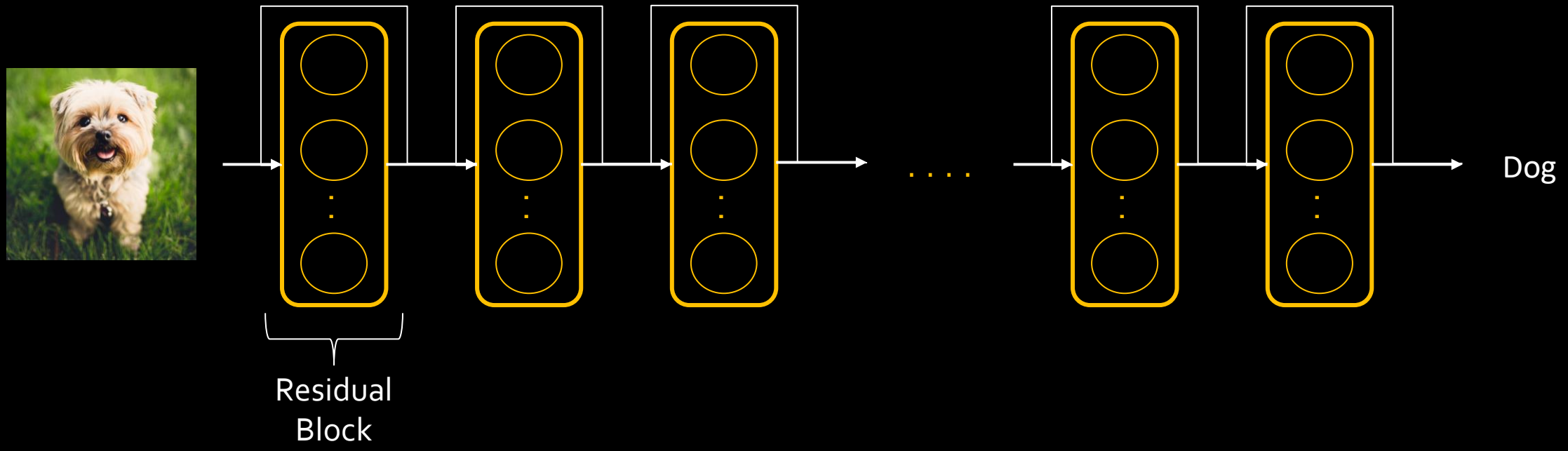
Edge Detect Vision Pipeline on 1 Ryzen™ AI Column



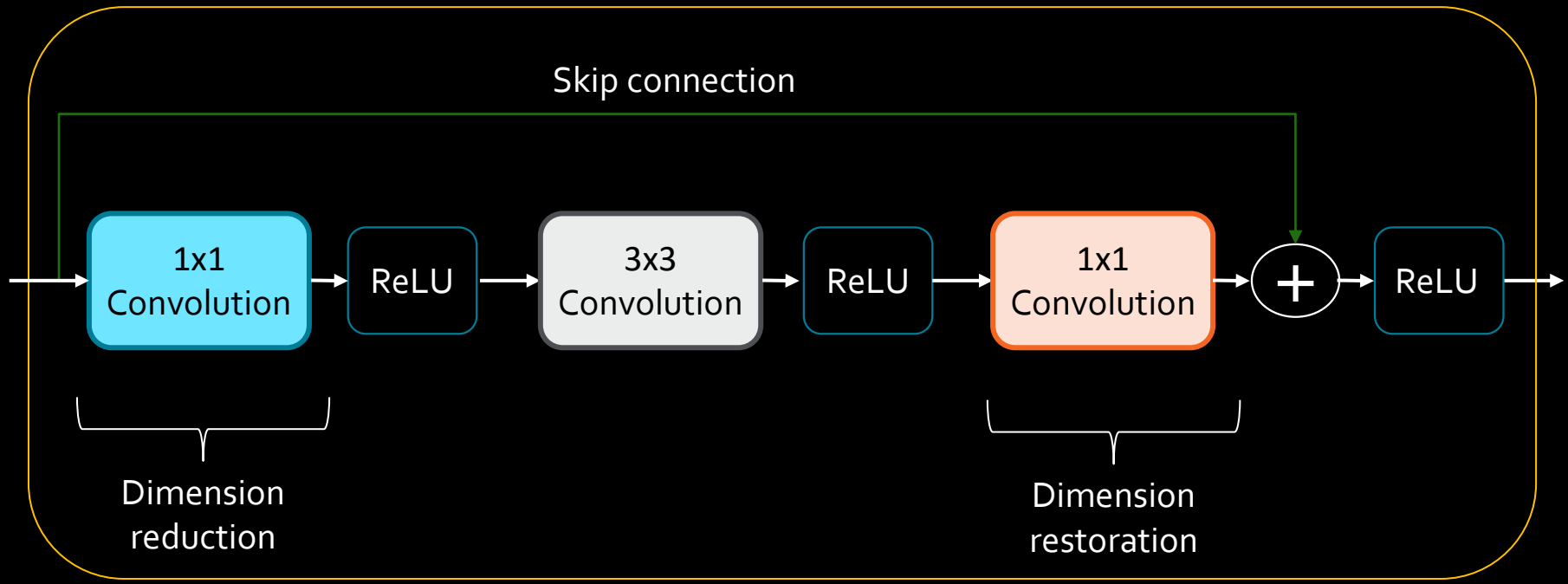

```
def edgeDetect():
@device(AIEDevice.ipu)
def deviceBody():

# AIE Core Function declarations

# Tile declaration
ShimTile = tile(0, 0)
MemTile = tile(0, 1)
ComputeTile2 = tile(0, 2)
ComputeTile3 = tile(0, 3)
ComputeTile4 = tile(0, 4)
ComputeTile5 = tile(0, 5)

# AIE-array data movement with object fifos
# input RGBA broadcast + memtile for skip
inOF_L3L2 = object_fifo("inOF_L3L2",ShimTile,[CompTile2, MemTile],[2, 2, 7],IB_ty)
inOF_L2L1 = object_fifo("inOF_L2L1",MemTile,CompTile5,7,IB_ty)
object_fifo_link(inOF_L3L2, inOF_L2L1)

# output RGBA
outOF_L2L3 = object_fifo("outOF_L2L3",MemTile,ShimTile,2,IB_ty)
outOF_L1L2 = object_fifo("outOF_L1L2",ComputeTile5,
MemTile,2,IB_ty)
object_fifo_link(outOF_L1L2, outOF_L2L3)

# between computeTiles
OF_2to3 = object_fifo("OF_2to3",CompTile2,CompTile3,4,I_ty)
OF_3to4 = object_fifo("OF_3to4",CompTile3,CompTile4,2,I_ty)
OF_4to5 = object_fifo("OF_4to5",CompTile4,CompTile5,2,I_ty)
OF_5to5 = object_fifo("OF_5to5",CompTile5,CompTile5,1,IB_ty)
```



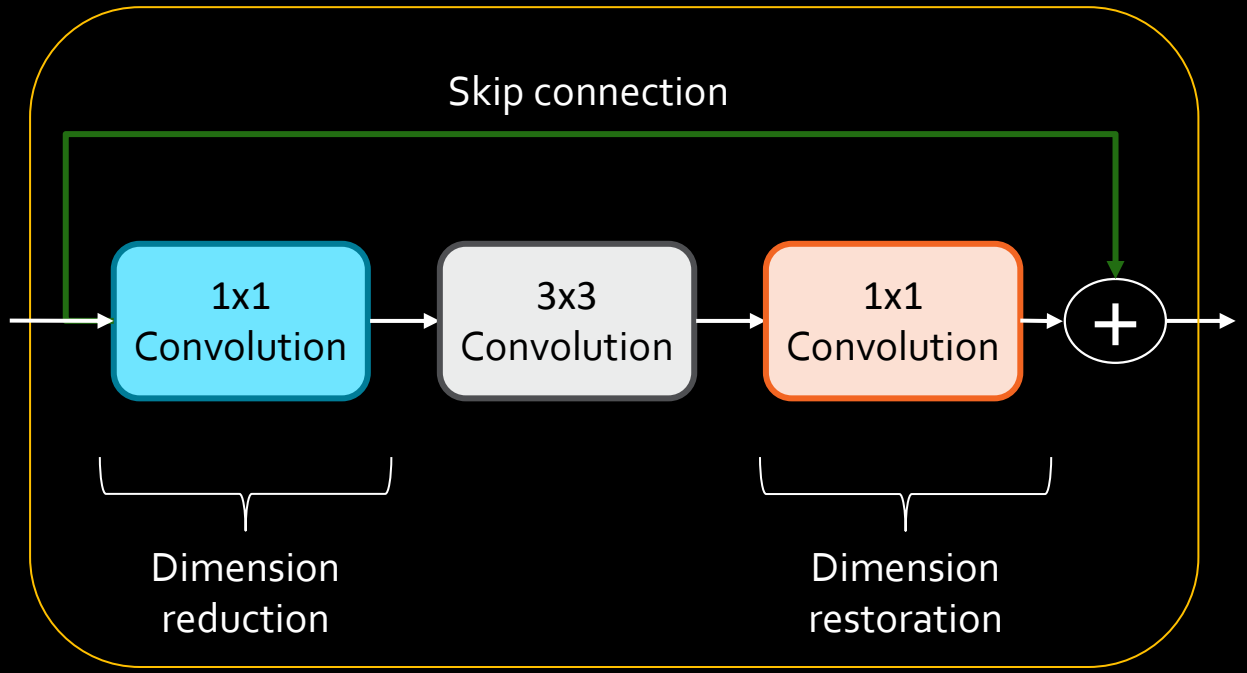
ResNet: Residual Network



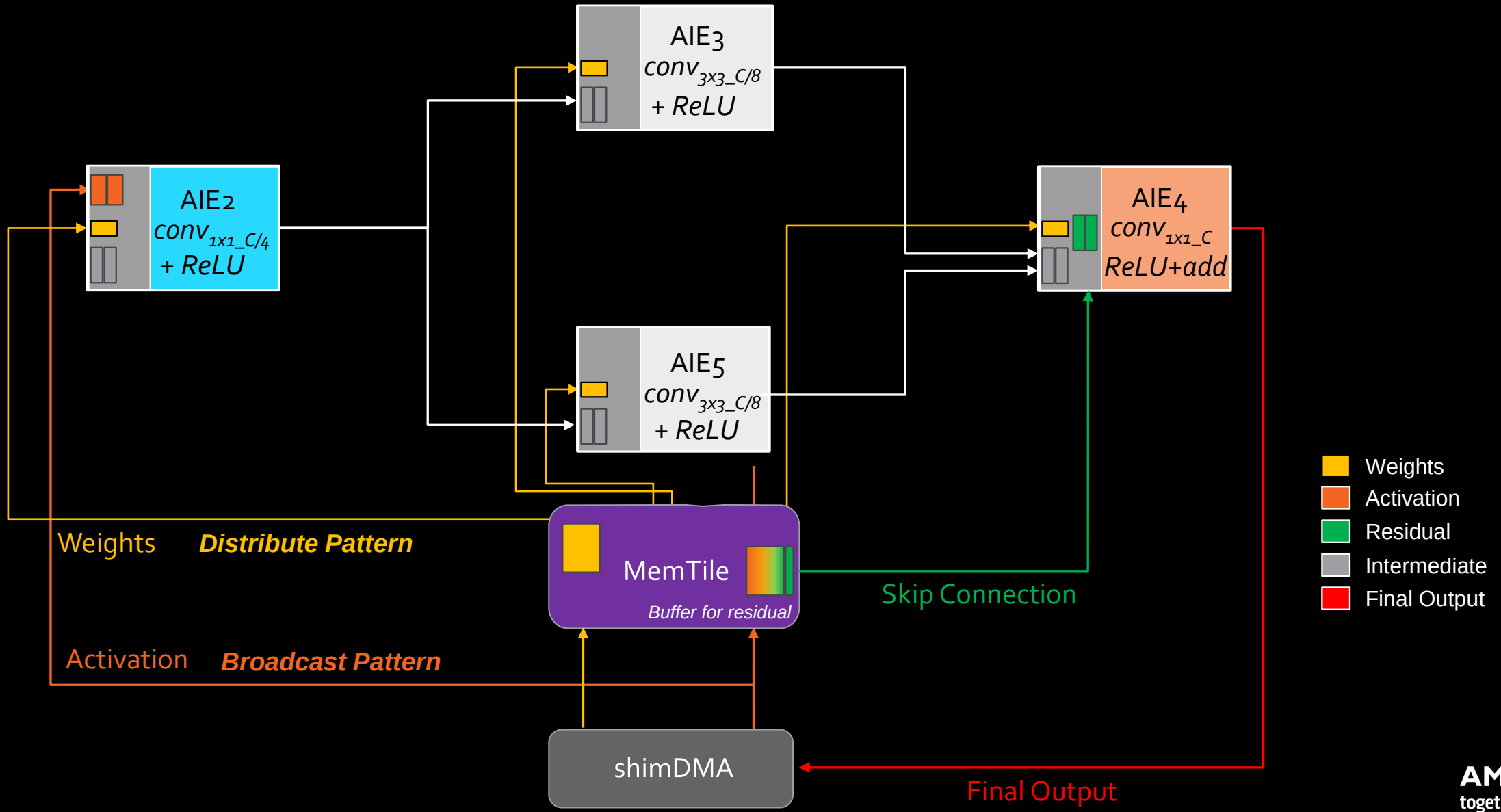
Bottleneck Block



Bottleneck Block: Kernel Fusion



Spatial Dataflow Mapping: 1 Bottleneck on 1 NPU Column



Exercise 5: Programming Examples

Basic

Design name	Data type	Description
Vector Scalar Add	i32	Adds 1 to every element in vector
Vector Scalar Mul	i32	Returns a vector multiplied by a scale factor
Vector Reduce Add	bfloat16	Returns the sum of all elements in a vector
Vector Reduce Max	bfloat16	Returns the maximum of all elements in a vector
Vector Reduce Min	bfloat16	Returns the minimum of all elements in a vector
Vector Exp	bfloat16	Returns a vector representing e^x of the inputs

Machine Learning Kernels

Design name	Data type	Description
Eltwise Add	bfloat16	An element by element addition of two vectors
Eltwise Mul	i32	An element by element multiplication of two vectors
ReLU	bfloat16	Rectified linear unit (ReLU) activation function on a vector
Softmax	bfloat16	Softmax operation on a matrix
Single core GEMM	bfloat16	A single core matrix-matrix multiply
Multi core GEMM	bfloat16	A matrix-matrix multiply using 16 AIEs with operand broadcast. Uses a simple "accumulate in place" strategy
GEMV	bfloat16	A vector-matrix multiply returning a vector
Conv2D	i8	A single core 2D convolution for CNNs
Conv2D+ReLU	i8	A Conv2D with a ReLU fused at the vector register level

Resources to Continue Developing for Ryzen AI

- Repository Link
  - AIR MLIR-AIE: https://github.com/Xilinx/mlir-aie/

- AMD Documentation
  - Versal ACAP AIE-ML Architecture Manual: https://docs.xilinx.com/r/en-US/am020-versal-aie-ml
  - AIE-ML AIE API Manual: https://www.xilinx.com/htmldocs/xilinx2023_2/aiengine_api/aie_api/doc/
  - Ryzen™ AI: https://www.amd.com/en/products/ryzen-ai
  - MLIR: https://mlir.llvm.org

- Organizers
  - Kristof Denolf: kristof.denolf@amd.com
  - Joe Melber: joseph.melber@amd.com
  - Jack Lo: jack.lo@amd.com
  - Phil James-Roxby: phil.james-roxby@amd.com
  - Sam Bayliss: samuel.bayliss@amd.com

Copyright and Disclaimer

- The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate releases, for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

- THIS INFORMATION IS PROVIDED 'AS IS." AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION


AMD