Accelerator Architectures for Machine Learning (AAML)

Lecture 6: Digital DNN Accelerator

Tsung Tai Yeh
Department of Computer Science
National Yang-Ming Chiao Tung University

Acknowledgements and Disclaimer

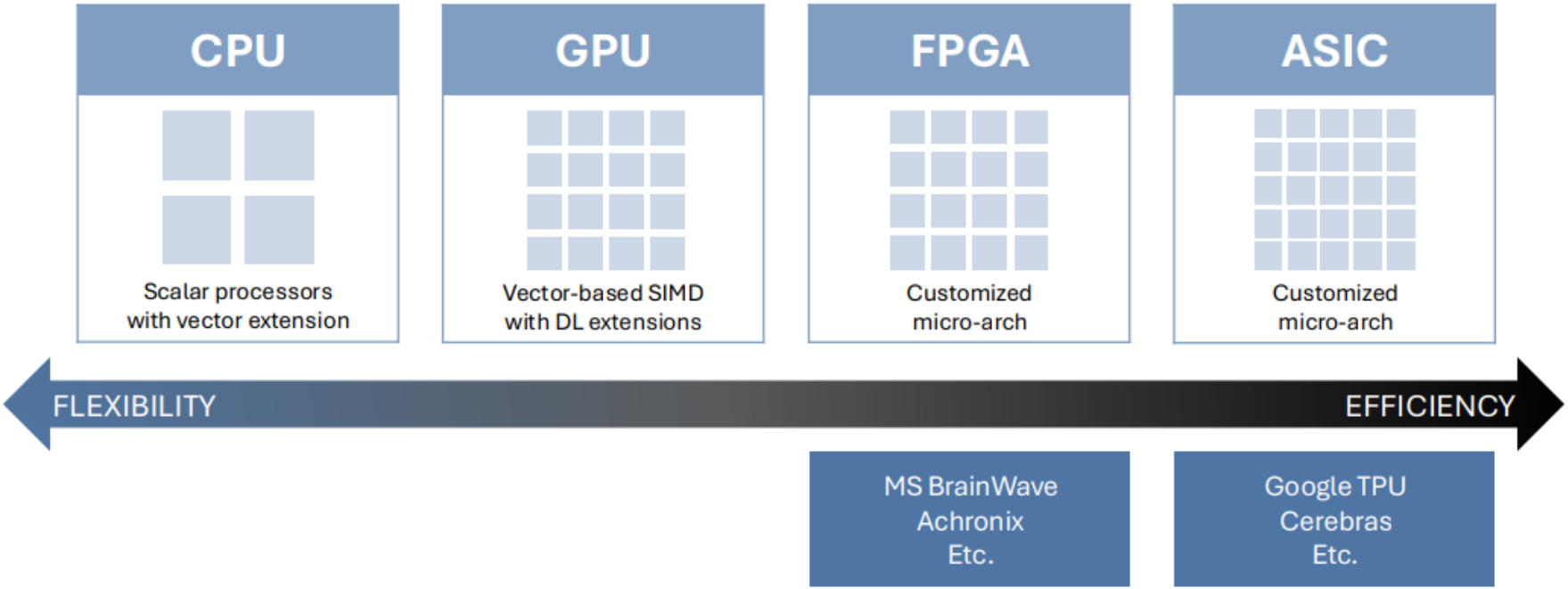
- Slides was developed in the reference with
  Joel Emer, Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, ISCA 2019 tutorial
  Efficient Processing of Deep Neural Network, Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, Morgan and Claypool Publisher, 2020
  Yakun Sophia Shao, EE290-2: Hardware for Machine Learning, UC Berkeley, 2020
  CS231n Convolutional Neural Networks for Visual Recognition, Stanford University, 2020
  CS224W: Machine Learning with Graphs, Stanford University, 2021

Outline

- Reconfigurable Deep Learning Accelerators
  - Reconfigurable FPGA
  - SambaNova Reconfigurable Dataflow Unit (RDU)
  - Coarse grained reconfigurable array (CGRA)
  - Meta MTIA AI Accelerator
  - Tenstorrent AI Processor
  - Wafer-scale AI chip -- Cerebras

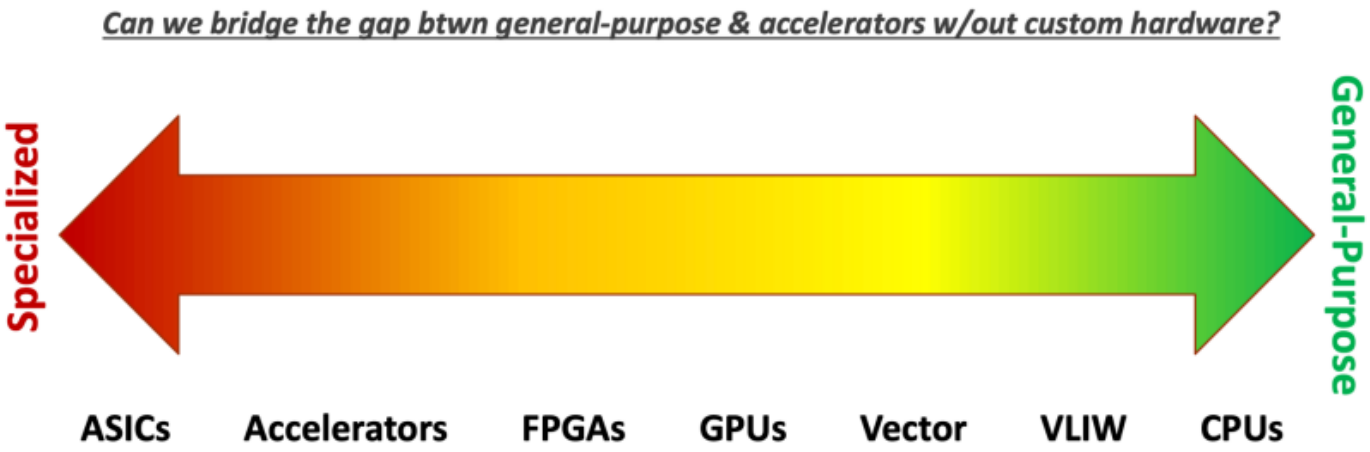
Reconfigurable Deep Learning on FPGA

Spectrum of Architectures for Deep Learning



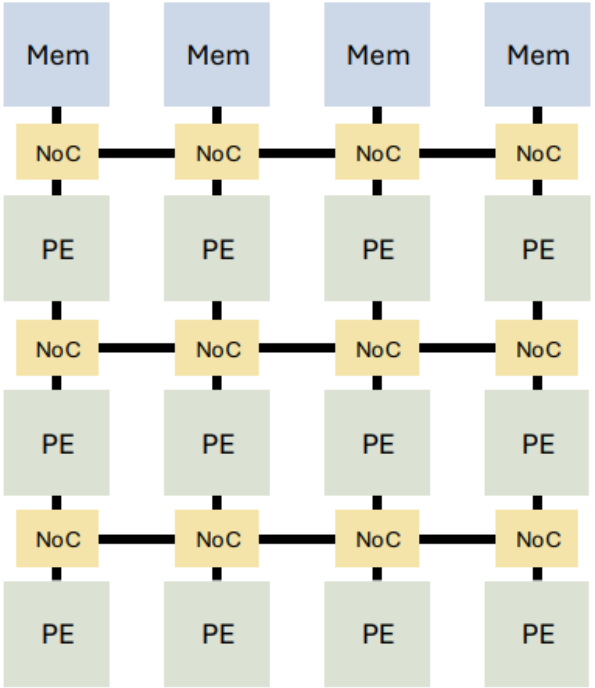
Why Reconfigurable Computing?

- AI accelerators improves 100X performance/energy compared to general-purpose processor
- But new hardware is sophisticated and expensive
  - Especially in cutting-edge manufacturing processes

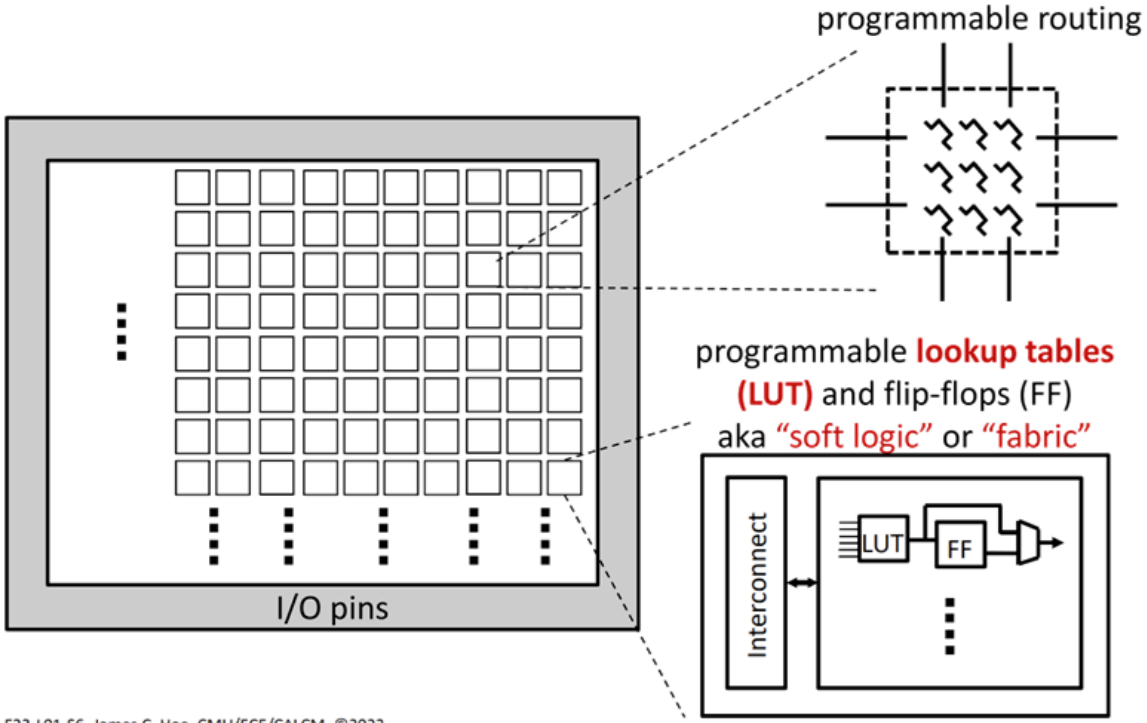


Reconfigurable Computing

- Basic idea
  - A spatial array of processing elements (PEs) & memories with a configurable network
  - Map your computation spatially onto the array
  - Goal: programmable with near ASIC efficiency

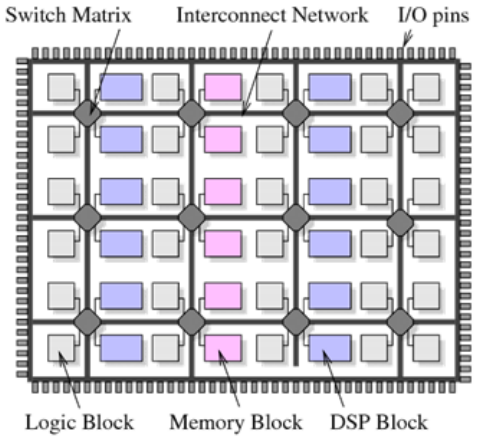


Basic FPGA Design

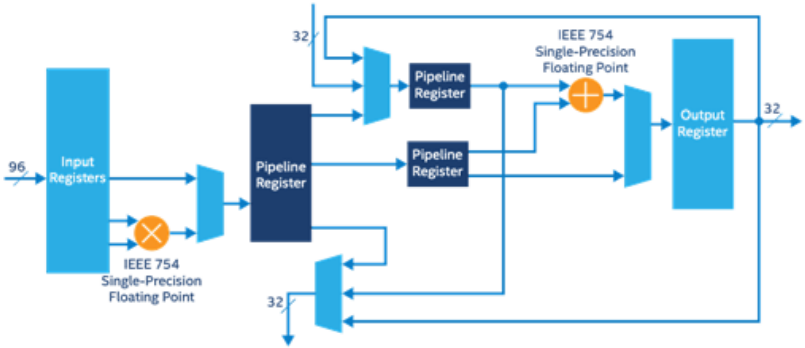


Modern FPGAs

- FPGAs are coarse-grain today
  - Hardened logic in LUTs
  - “DSP blocks” to implement wide add/mul efficiently
  - Dense memories distributed throughout fabric



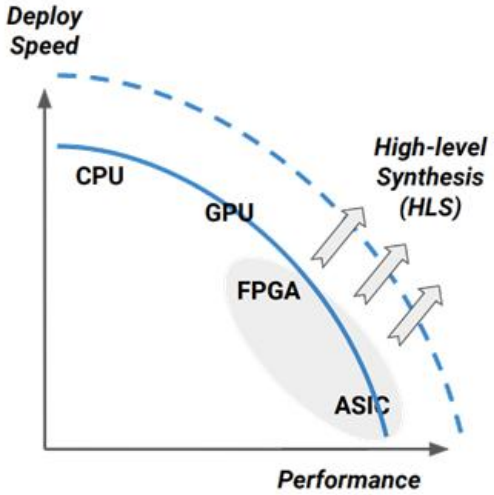
Mode Name	Mathematical Function
Multiplication Mode	X ´ Y
Adder or Subtract Mode	(X + Y) or (X-Y)
Multiply-Add/Subtract	(X ´ Y) + Z or (X ´ Y) - Z
Multiply Accumulate Mode	(X ´ Y) + Acc or (X ´ Y) – Acc
Vector One Mode	(X ´ Y) + Chain In

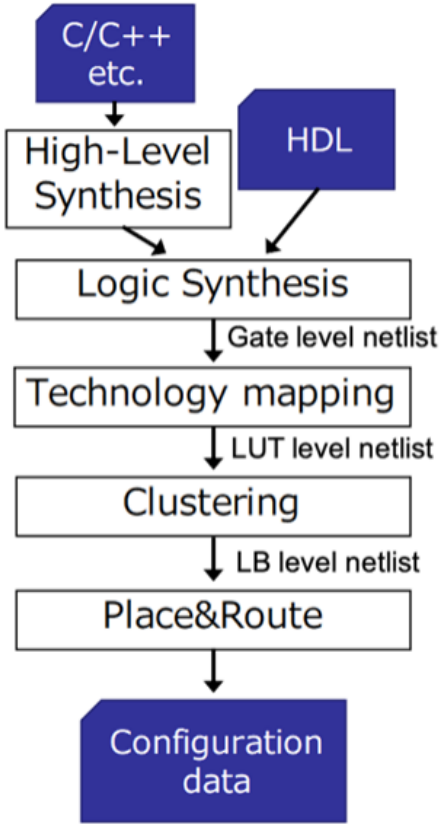


[Intel, Stratix 10]

High-Level Synthesis (HLS)

- Automated optimization and scheduling
- High portability against different PDK or PPA requirements
- Short design cycle



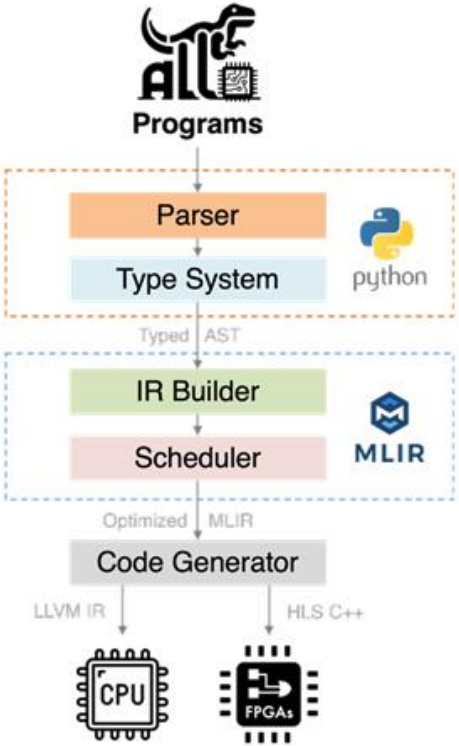


Challenges of HLS Accelerator Design

- **Time consuming**
  - Manual architecture and micro-architecture design, manual C/C++ code rewriting
- **Suboptimal**
  - Empirical parameter tuning, like parallel factors, buffer sizes, tiling sizes, etc..
- **Low flexibility**
  - Only support a small set of models

Accelerator Design Languages (ADLs)

- **Pythonic**
- **Maintainability**
  - Decoupled hardware customizations
- **Composability**
  - All the kernels, primitives, and schedules should be composable to form complex designs



[Allo, PLDI, 2024]

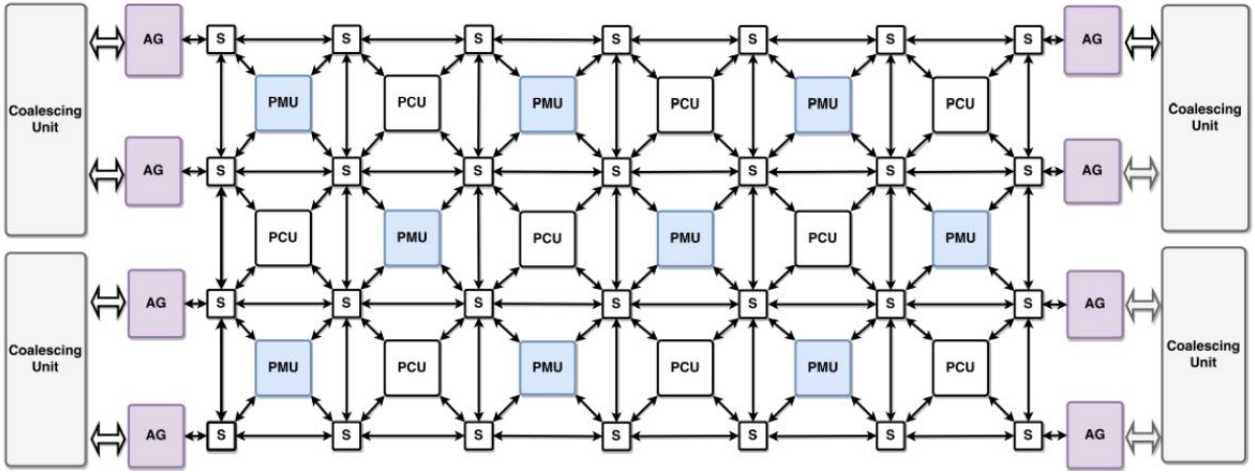
SambaNova Reconfigurable Dataflow Unit (RDU)

Plasticine Architecture

- **Plasticine architecture**
  - A reconfigurable architecture for parallel patterns (Raghu, ISCA 2017)
  - **Pattern Compute Unit (PCU)**
    - Reconfigurable pipeline with multiple stages of SIMD functional units (FUs)
  - **Pattern Memory Unit (PMU)**
    - A banked scratchpad memory
  - **The compiler**
    - Maps the computation of inner loops to PCUs
    - Most operands are transferred directly between FUs without scratchpad access or inter-PCU communication

Plasticine Architecture Overview

- Data access address calculation occurs while the PCU is working
- Each DRAM channel is accessed using several address generators (AG) on two sides of the chip
- Multiple AGs connect to an address coalescing unit for memory requests

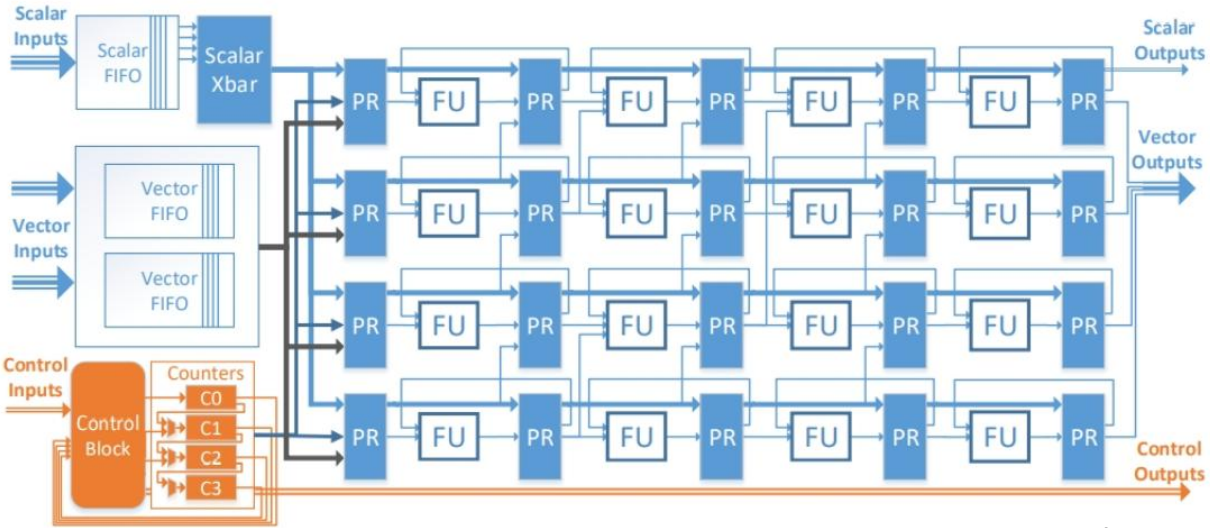


Raghu, ISCA 2017

1. **Scalar**: uses to communicate single words of data
2. Each **vector** communicates one word per line in the PCU
3. **Control** signals at the start or end of execution of a PCU

Plasticine PCU Architecture

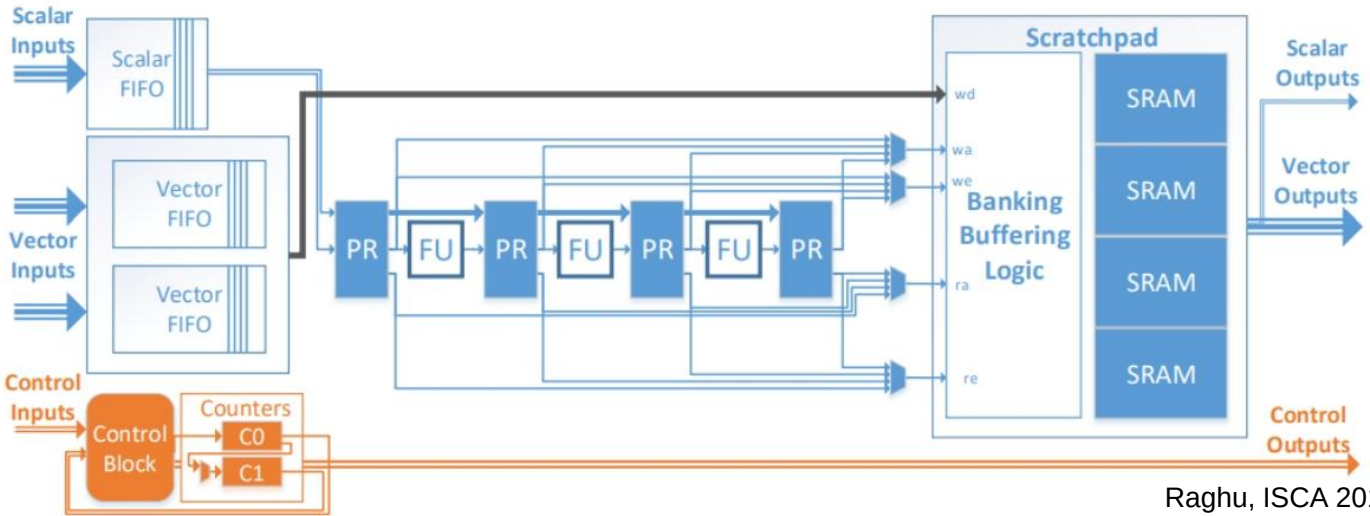
- **Pattern Compute Unit (PCU)**
  - Each stage's SIMD lane contains a FU and associated pipeline register (PR)



Raghu, ISCA 2017

Plasticine PMU Architecture

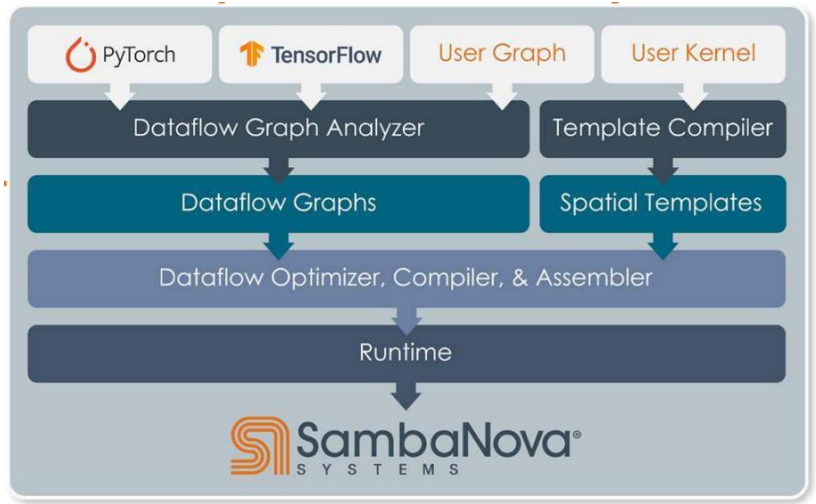
- **Pattern Memory Unit (PMU)**
  - Contains a scratchpad memory and address calculation
  - Calculates address only needs simple scalar math
  - Has simpler FUs than ones in PCUs



Raghu, ISCA 2017

Reconfigurable Dataflow Unit (RDU)

- **SambaNova RDU**
  - **Pattern Compute Units**
    - BF16 with FP32 accumulation
    - Support FP32, Int32, Int16, Int8
  - **Pattern Memory Unit**
    - Memory transformation
  - **Dataflow optimization**
    - Tiling
    - Nested pipelining
    - Operator parallel streaming

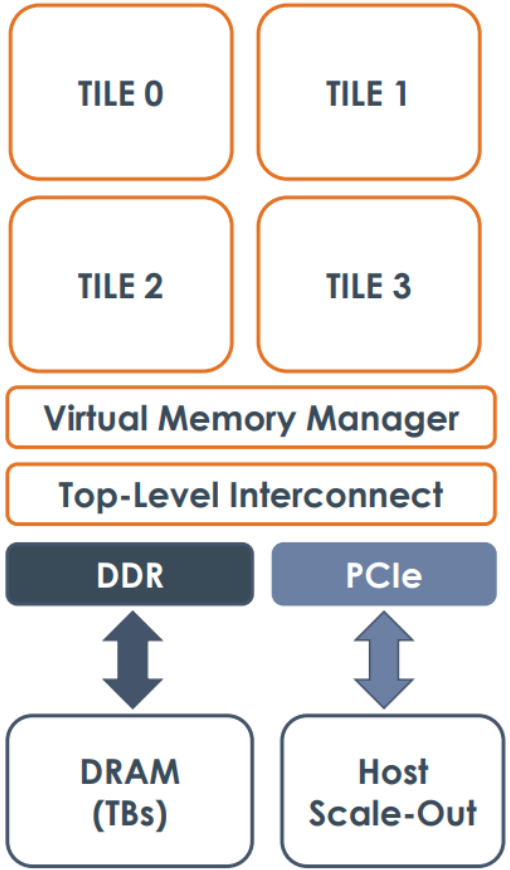


Dataflow Exploits Data Locality / Parallelism

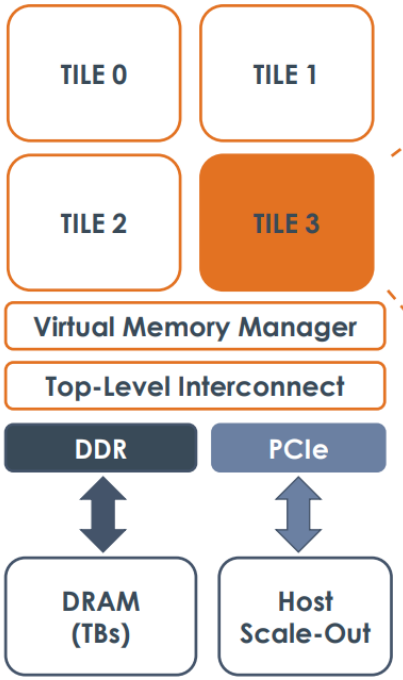
- **Software-hardware co-design architecture**
  - Dataflow captures data locality and parallelism
  - Flexible time and space scheduling to achieve higher utilization
  - Flexible memory system and interconnect to sustain high compute throughput
  - Custom dataflow pipeline

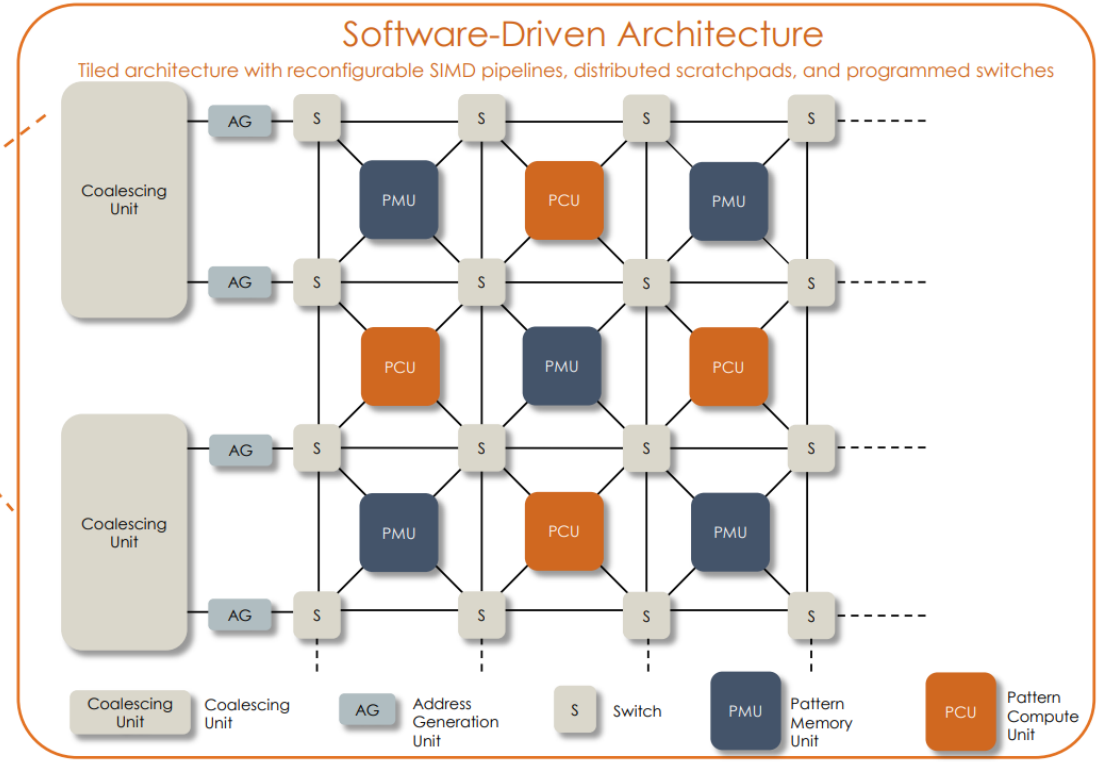
Chip and Architecture Overview

- **RDU Tile**
  - Compute and memory components
  - A programmable interconnect
- **Tile resource management**
  - Combine adjacent tiles to form a larger logical tile
  - Each tile controlled independently
  - Allow different applications on separate tiles concurrently
- **Memory access**
  - Direct access to TBs DDR4 off-chip memory
  - Memory-mapped access to host memory



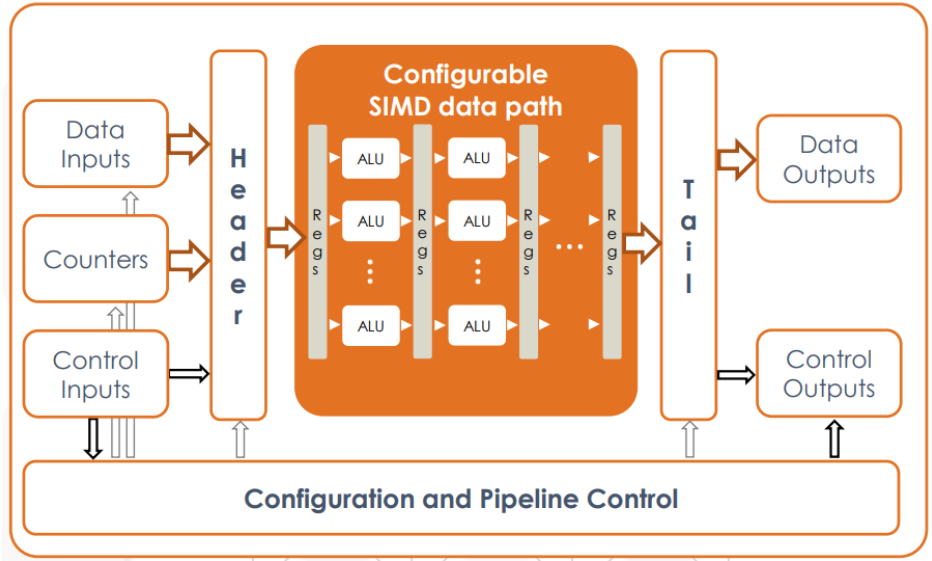
RDU Tile


TILE 0
TILE 1
TILE 2
TILE 3
Virtual Memory Manager
Top-Level Interconnect
DDR
PCIe
DRAM (TBs)
Host Scale-Out


Software-Driven Architecture
Tiled architecture with reconfigurable SIMD pipelines, distributed scratchpads, and programmed switches
Coalescing Unit
AG
S
PMU
PCU
Coalescing Unit
Address Generation Unit
Switch
Pattern Memory Unit
Pattern Compute Unit

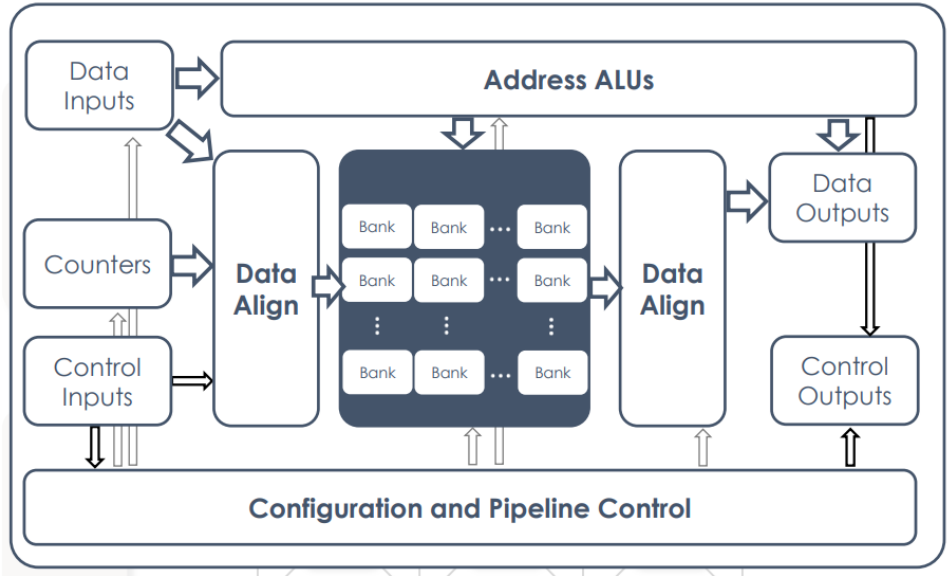
Pattern Compute Unit (PCU)

- **Pattern Compute Unit (PCU)**
  - Compute engine
- **Reconfigurable SIMD data path**
  - For dense and sparse tensor algebra in FP32, BF16, and integer data format
- **Programmable counters**
  - Program loop iterators
- **Tail unit**
  - Accelerates functions such as exp, sigmoid



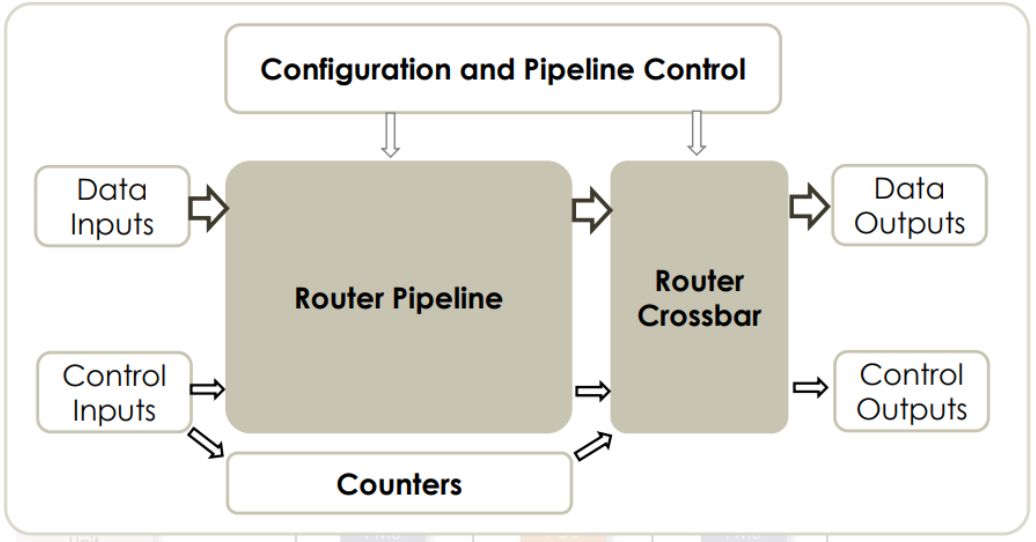
Pattern Memory Unit (PMU)

- **Pattern Memory Unit (PMU)**
  - **On-chip memory system**
  - **Banked SRAM arrays**
    - Write and read multiple high bandwidth SIMD data stream concurrently
  - **Address ALUs**
    - Address calculation for arbitrarily complex accesses
  - **Data align**
    - Tensor layout transformation



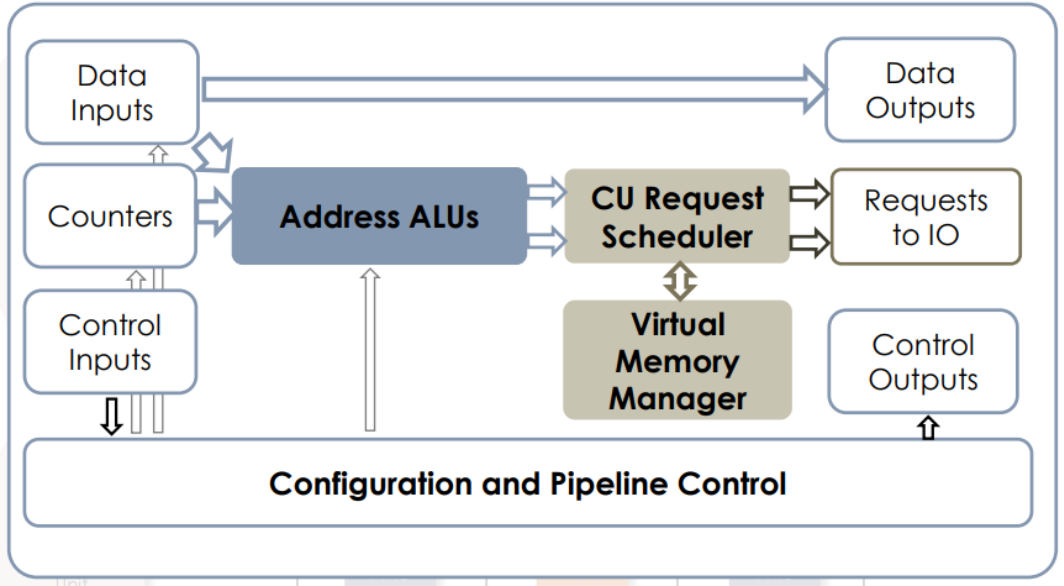
Switch and On-chip Interconnect

- **Switch**
  - Programmable packet-switched interconnect fabric
- **Independent data and control buses**
  - Suit different traffic classes
- **Programmable routing**
  - Flexible chip bandwidth allocation to concurrent stream
- **Programmable counters**
  - Outer loop iterators
  - On-chip metric collection



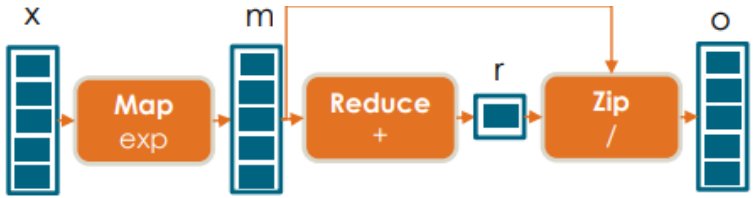
Interface to I/O Subsystem

- **Address ALUs**
  - Address calculation for arbitrarily complex accesses
- **Coalescing Units**
  - Enable transparent access to memories across RDUs and host memory
- **Address space manager**
  - Programmable, variable length segments

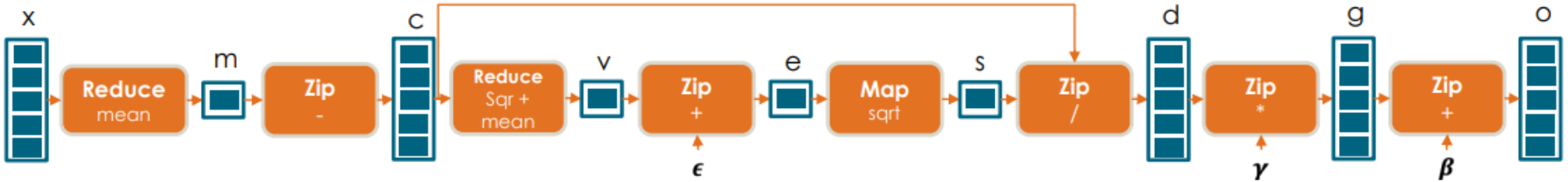


Operator Mapping

SOFTMAX: $\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$

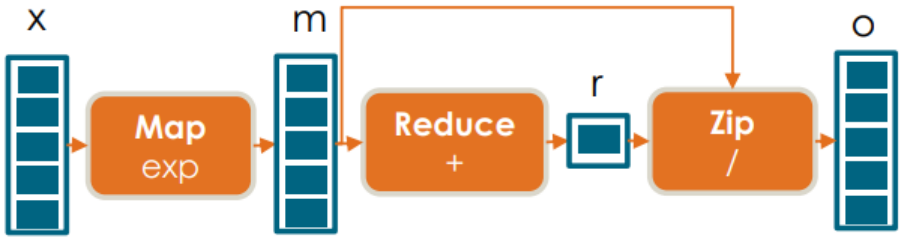


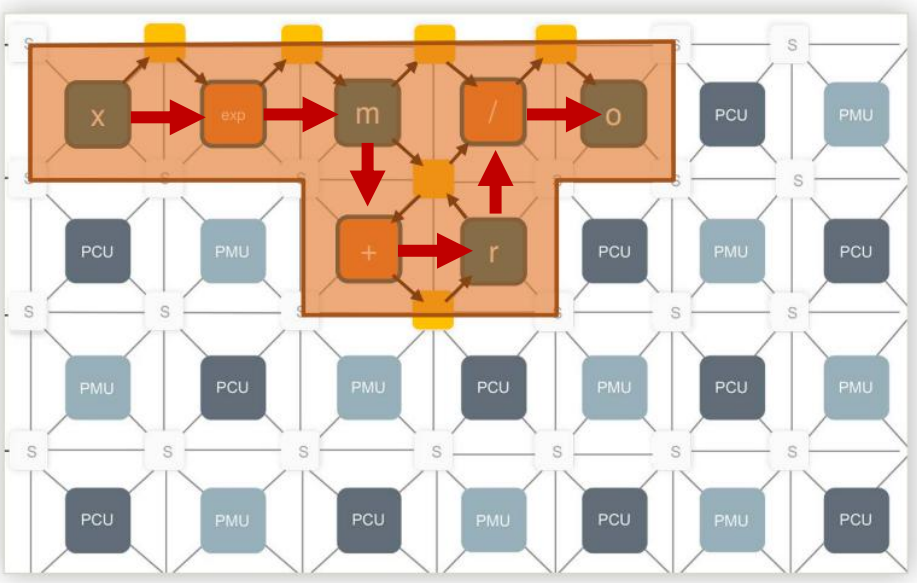
LAYERNORM: $y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta$



Operator Mapping (Softmax)

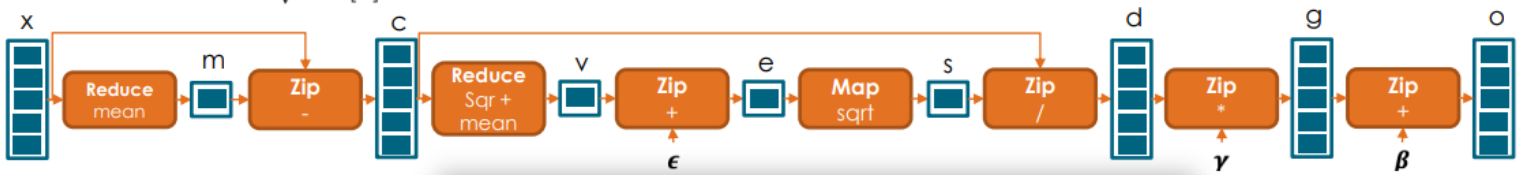
SOFTMAX: $\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$

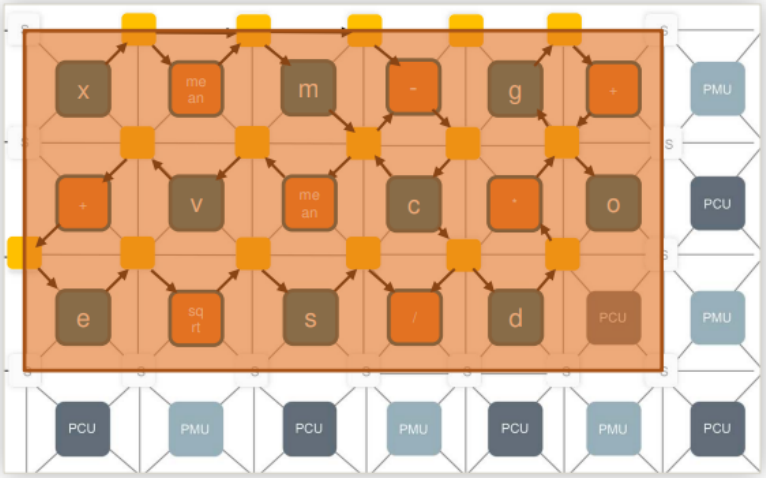




Pipelined in Space

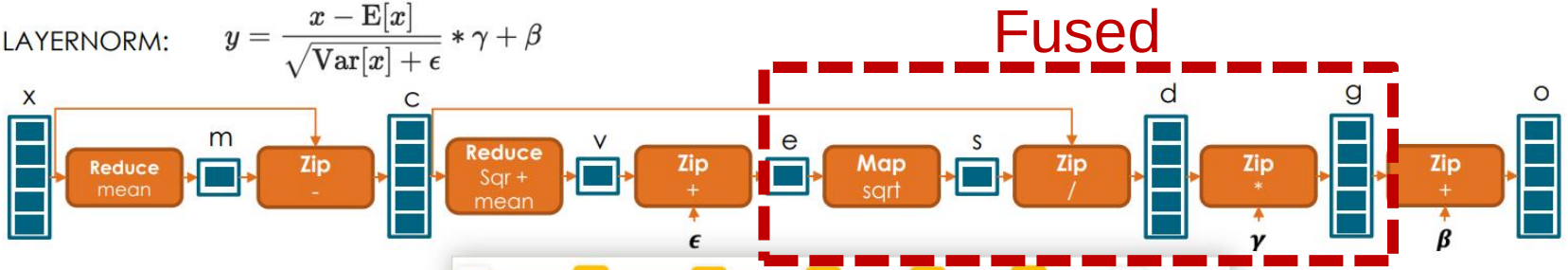
LAYERNORM: $y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta$

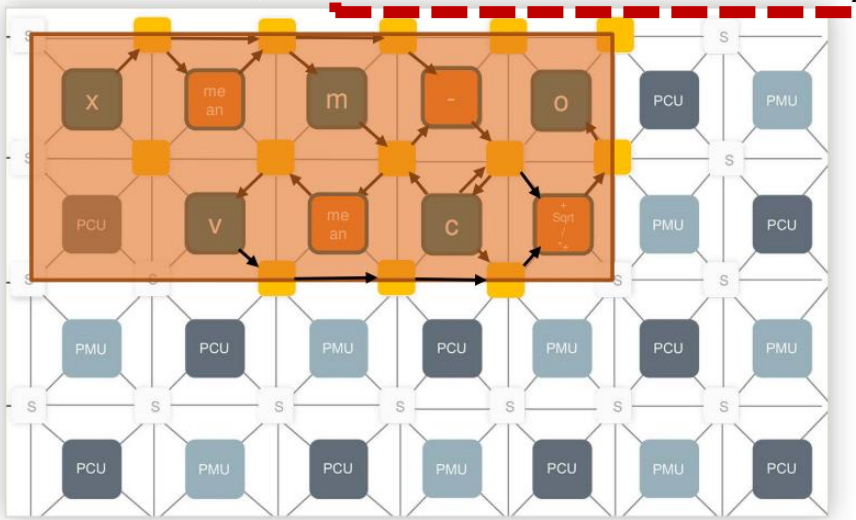




Pipelined in Space + Fused

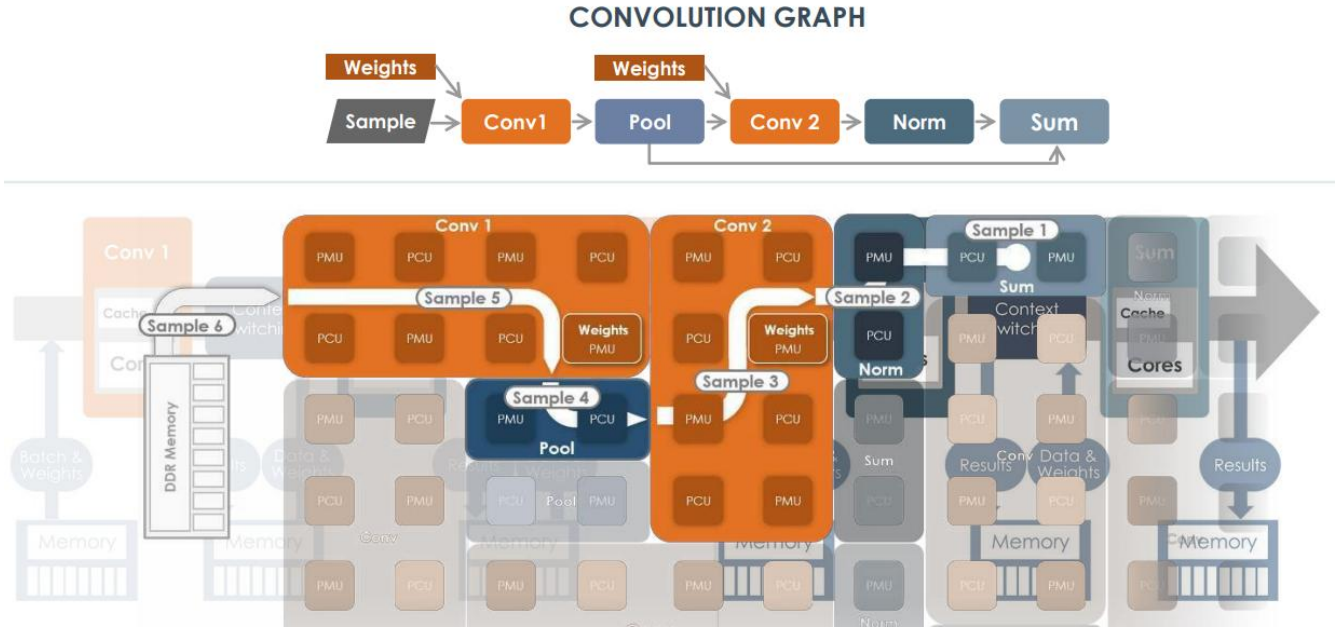
LAYERNORM: $y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta$





Spatial Dataflow within an RDU

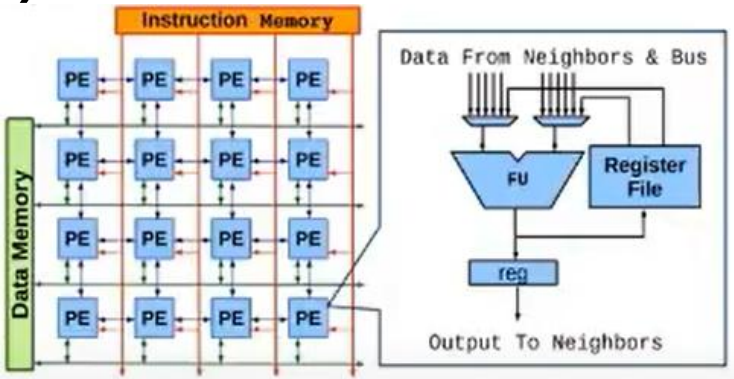
- **The dataflow removes**
  - Memory traffic and host communication overhead



CGRA

Coarse grained reconfigurable array (CGRA)

- **Coarse grained reconfigurable array (CGRA)**
  - Multiple **processing elements (PEs)**
  - Each PE has ALU-like functional unit
  - **Array configurations vary by**
    - Array size
    - Functional units
    - Interconnection network
    - Register file architectures
  - CGRAs can achieve **power-efficiency** of several 10s of GOps/sec per Watt (why?)
    - Samsung SRP processor (embedded and multimedia apps)



Key features of CGRA accelerators

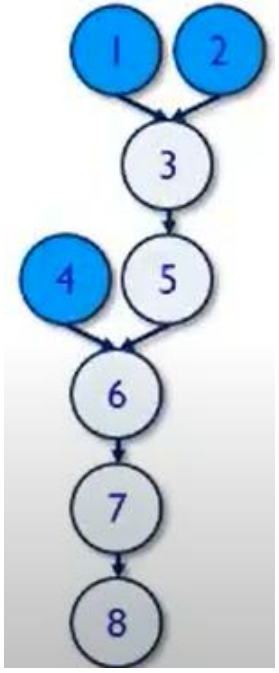
- **Software-pipelining execution mapping**
  - **Accelerate loops with low parallelism**
  - Loops with loop-carried dependence, loops with high branch divergence
- **Avoid von-Neumann architecture bottleneck**
  - CGRAs are not subjected to dynamic fetch and decoding of instructions
  - CGRA instructions are in a **pre-decoded form** in the instruction memory
  - **PE transfers data directly** among each another
  - Without going through a centralized registers and memory

Loop execution on the CGRA

Loop:

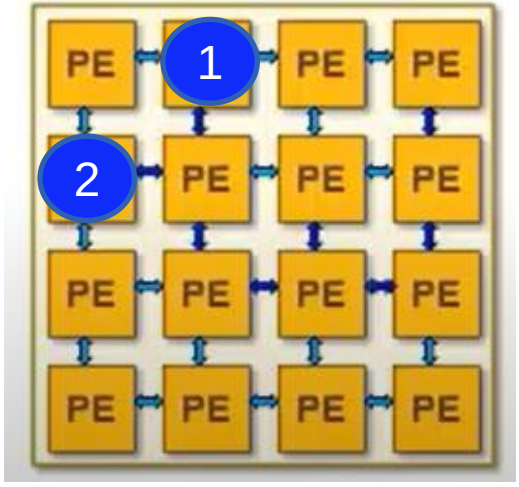
t1 = (a[i]+b[i]-k)*c[i]
d[i] = ~t1 & 0xFFFF

Data dependency graph



Mapping data dependency graph to CGRA

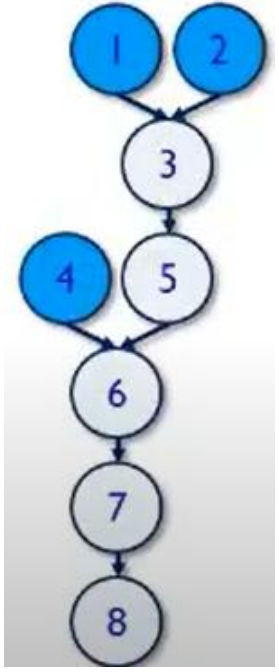
Execution time: 1



Loop execution on the CGRA

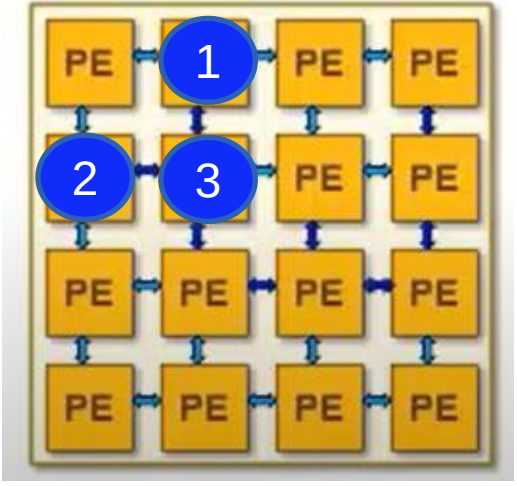
Loop:
t1 = (a[i]+b[i]-k)*c[i]
d[i] = ~t1 & 0xFFFF

Data dependency graph



Mapping data dependency graph to CGRA

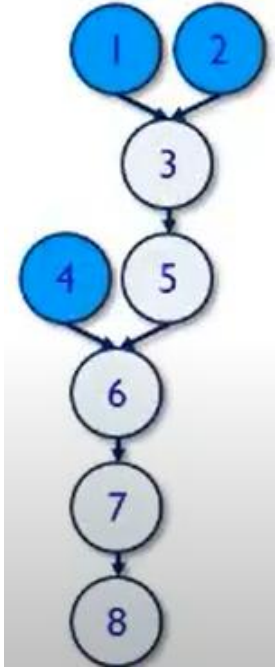
Execution time: 2



Loop execution on the CGRA

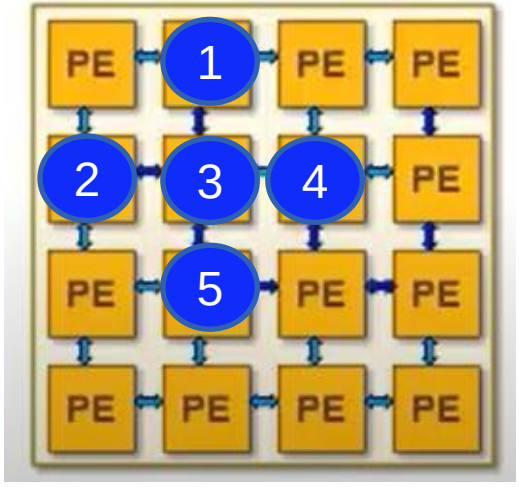
Loop:
t1 = (a[i]+b[i]-k)*c[i]
d[i] = ~t1 & 0xFFFF

Data dependency graph



Mapping data dependency graph to CGRA

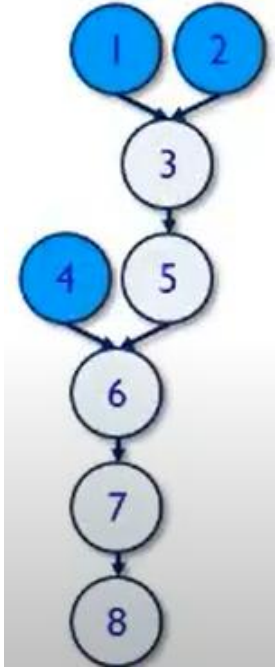
Execution time: 3



Loop execution on the CGRA

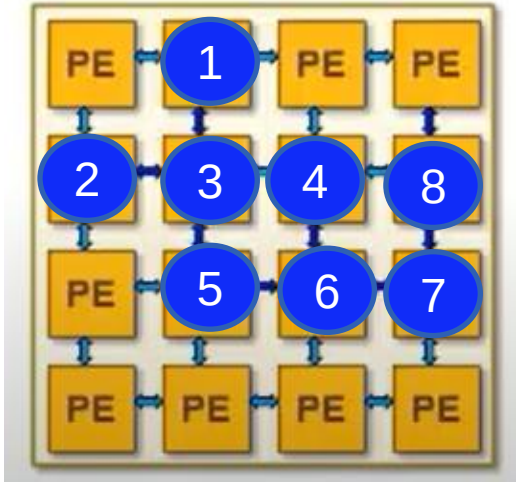
Loop:
t1 = (a[i]+b[i]-k)*c[i]
d[i] = ~t1 & 0xFFFF

Data dependency graph



Mapping data dependency graph to CGRA

Execution time: 6



Takeaway Questions

- What are hardware components used by RDU ?
  - (A) Pattern computer unit (PCU)
  - (B) Pattern memory unit (PMU)
  - (C) Interconnect network router
- What are features of CGRAs ?
  - (A) Customized PEs
  - (B) Software-pipelining execution mapping
  - (C) Reconfigurable dataflow

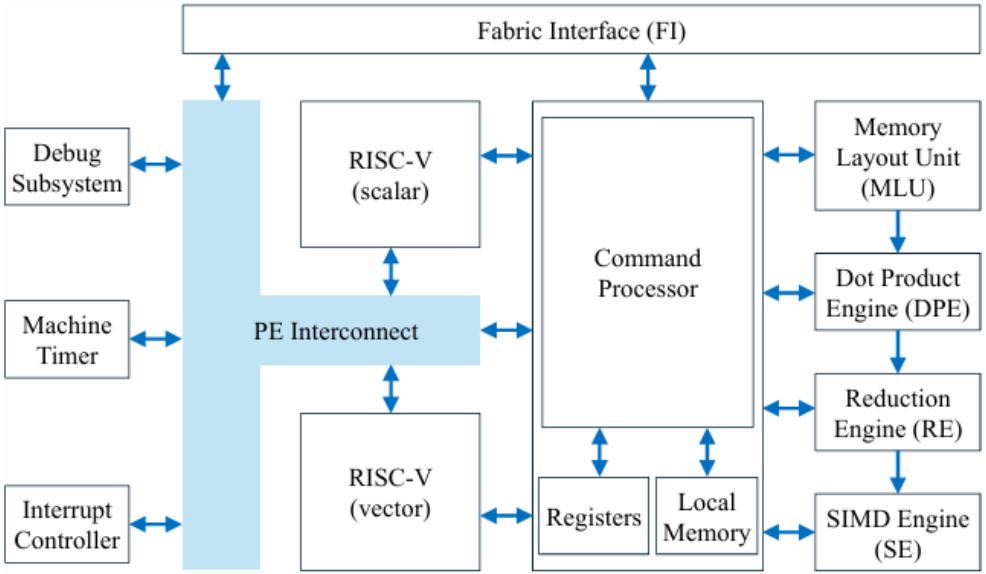
Meta MTIA2 ASIC

MTIA Model Inference Engine

- **MTIA Architecture**
  - 8x8 array of processing elements (PEs)
  - Network-on-chip (NoC) connects to a set of on-chip SRAMs
  - SRAMs are shared by the PEs and to off-chip memory
  - Control core is quad-core RISC-V processor

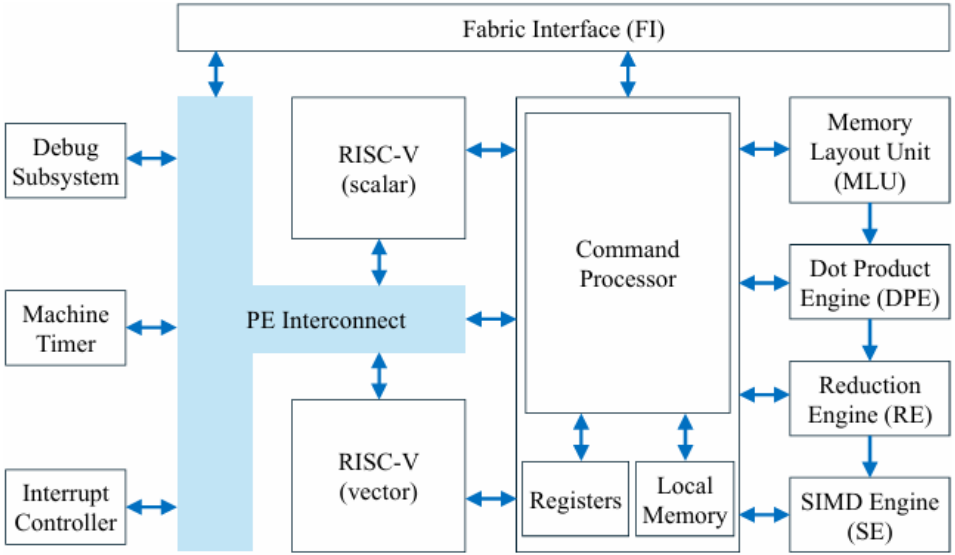
MTIA Model Inference Engine

- **MTIA PE**
  - Each PE includes 384 KB local memory
  - RISC-V core issues commands to Command Processor to offload computations to fixed-function units



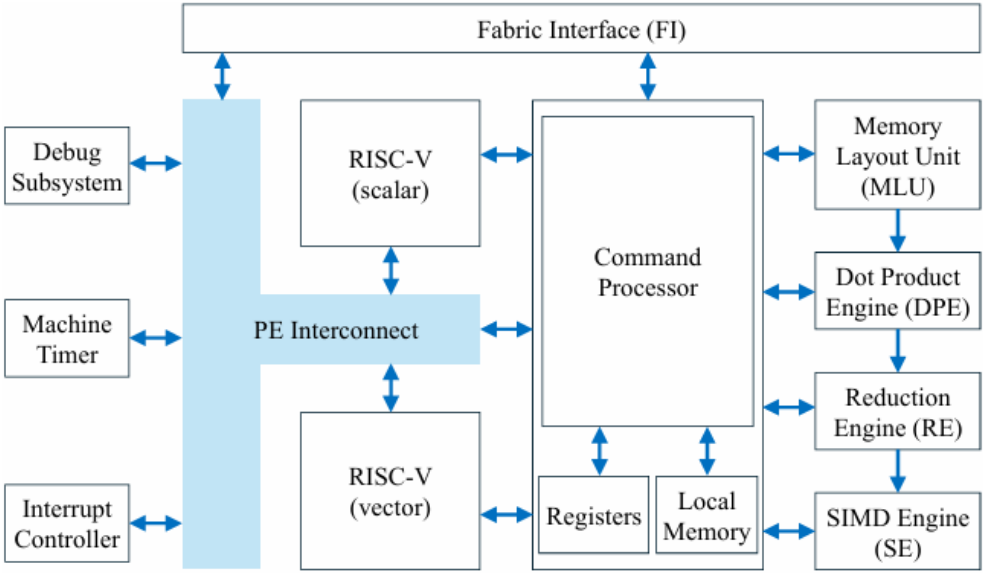
MTIA Model Inference Engine

- **MTIA PE**
  - RISC-V Vector extension (64B wide)
  - Memory Layout Unit (MLU) performs memory-layout transformation
  - Dot Product Engine (DPE) performs GEMM



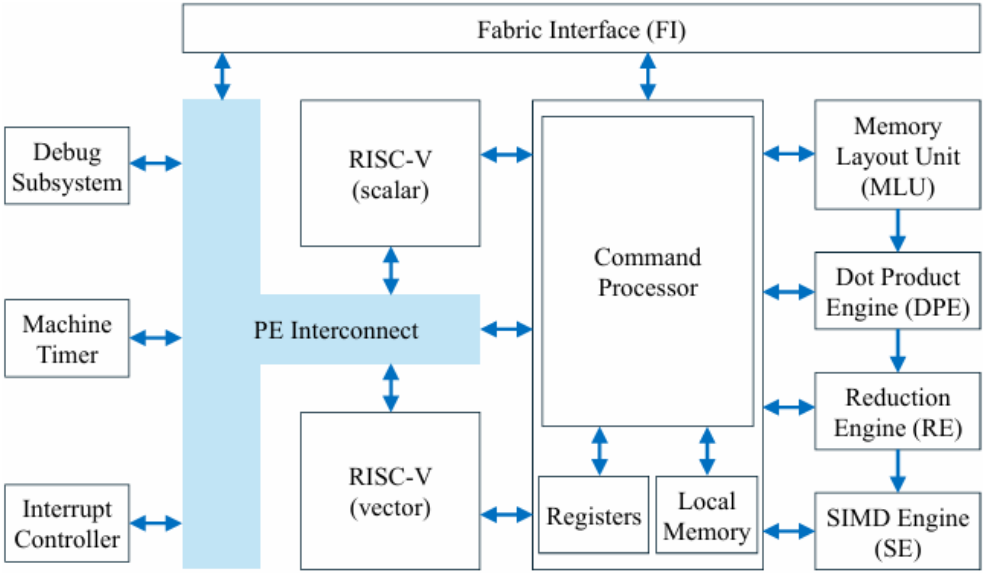
MTIA Model Inference Engine

- **MTIA PE**
  - Dot Product Engine (DPE) performs GEMM
    - Two 32 x 32B x 32 MAC tiles
    - 2.76 TFLOPS/s per PE with FP16/ BF16 and output in FP32 -> 2:4 sparsity for weights



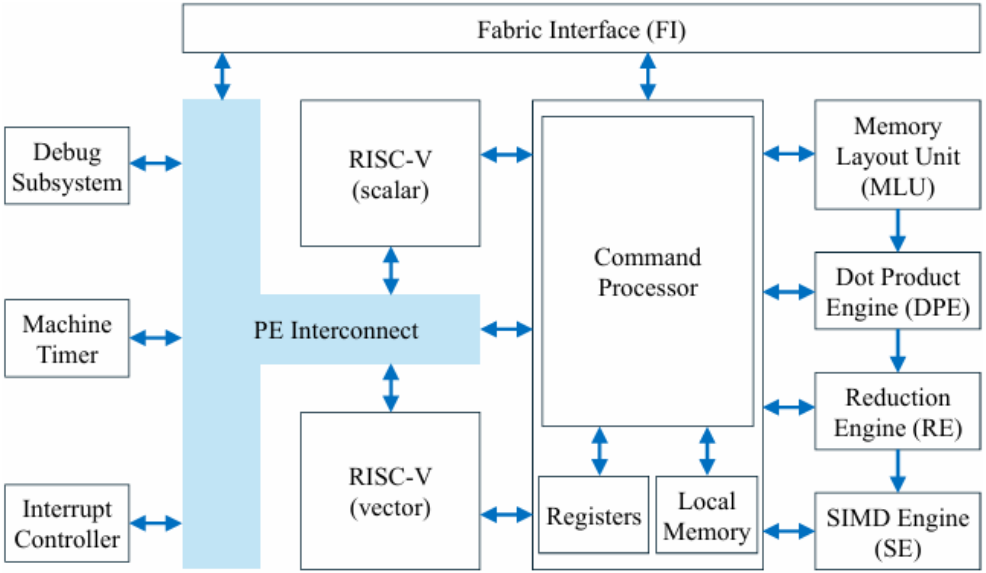
MTIA Model Inference Engine

- **MTIA PE**
  - Reduction Engine (RE)
    - Stores matrix multiply results as they are accumulated
  - SIMD Engine (SE)
    - Performs quantization and nonlinear funcs
    - Includes LUT for approximating nonlinear functions



MTIA Model Inference Engine

- **MTIA PE**
  - Fabric Interface (FI)
    - Acts as DMA engine to transfer data in and out of PE's local memory through NoC



MTIA Model Inference Engine

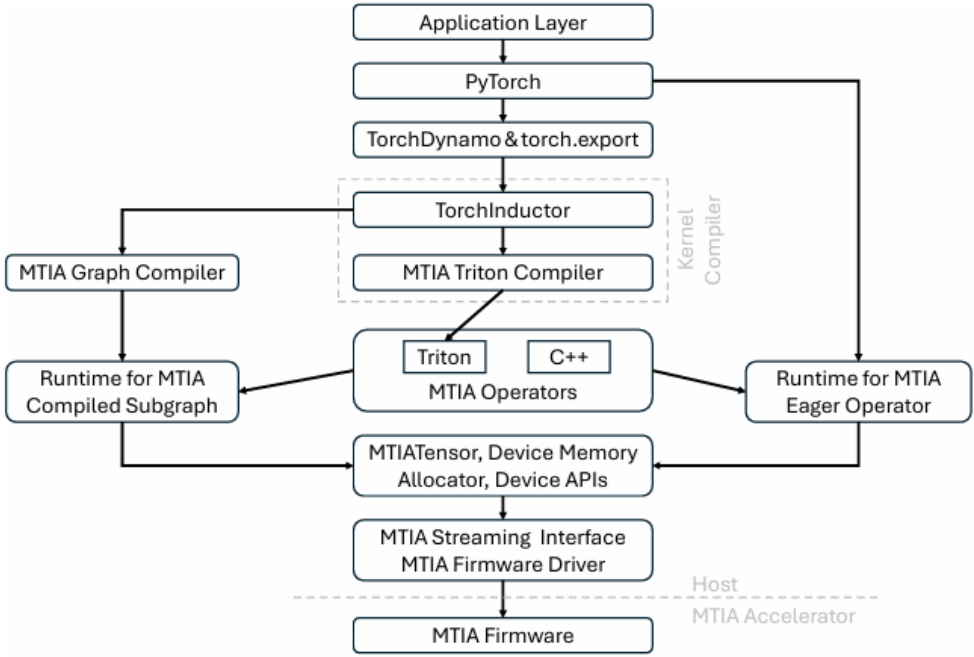
- **MTIA Unique Memory Hierarchy**
  - Uses a large SRAM (256 MB) backed by LPDDR DRAM
    - Avoids HBM to reduce cost and power consumption
    - Meet latency requirements of recommendation models
    - The recommendation models exhibit significant locality
    - Similar to Cerebras and Groq accelerator
    - SRAM provides 2.7TB/s of bandwidth
    - Performance drops sharply as models reach a complexity and size that exceeds the SRAM capacity

MTIA Model Inference Engine

- **New Feature in MTIA2i**
  - Dynamic INT8 quantization
    - Leveraging the reduction engine to identify the min and max values per batch
    - Channel-wise symmetric dynamic INT8 for FC layers
  - Compression
    - Supports lossless asymmetric numerical system (ANS) compression for weights
    - Achieves up to a 50% compression ratio

MTIA Model Inference Engine

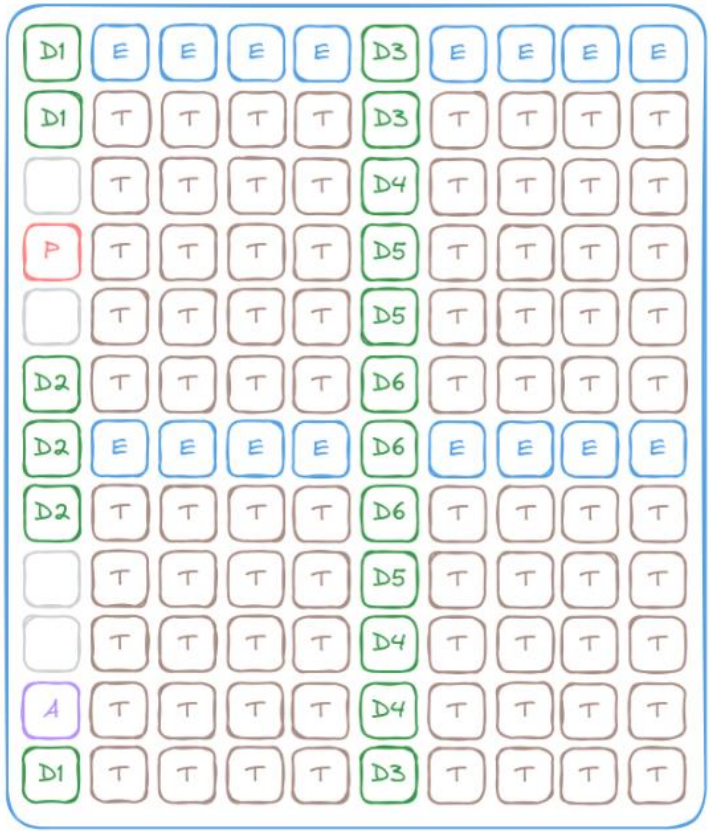
- **MTIA2i Software Stack**
  - Support PyTorch eager mode
  - TorchDynamo enables symbolic tracing to capture models with dynamic shapes
  - TorchInductor generates Triton codes for PyTorch operator and operator fusion



Tenstorrent AI Hardware

Tenstorrent AI Processor

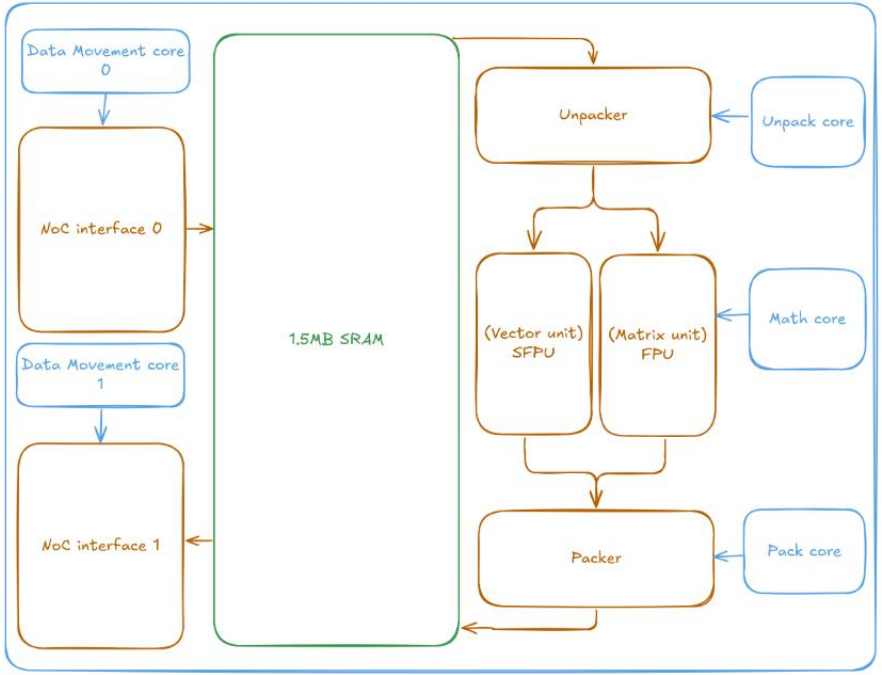
- **The NoC grid of Tenstorrent Wormhole**
  - Most compute elements called a Tensix core
  - D = DRAM
  - T = Tensix
  - E = Ethernet
  - A = ARC
  - P = PCI/e



https://clehaxze.tw/gemlog/2025/04-21-programming-tensotrrent-processors.gmi

Tenstorrent AI Processor

- **Each Tensix contains**
  - 5 “Baby” RISC-V CPUs
    - 5 stage pipelined, single issue CPU
    - Handle instruction dispatch
  - 2 NoC interface
  - A vector unit
  - A matrix/tensor unit



https://clehaxze.tw/gemlog/2025/04-21-programming-tensotrrent-processors.gmi

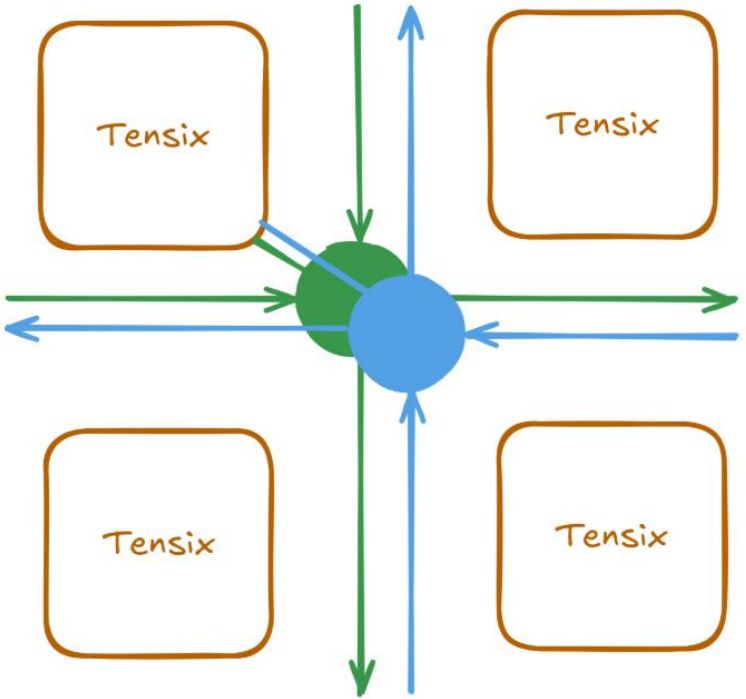
Tenstorrent AI Processor

- **The Dataflow of Tensix**
  - NoC 0 reads data from DRAM
  - Unpacker unpacks the data into a format that can be processed by the matrix/tensor unit
  - Matrix/tensor unit performs the computation
  - Packer packs the result back into a format for storage
  - NoC 1 sends the result to DRAM

https://clehaxze.tw/gemlog/2025/04-21-programming-tensotrrent-processors.gmi

Tenstorrent AI Processor

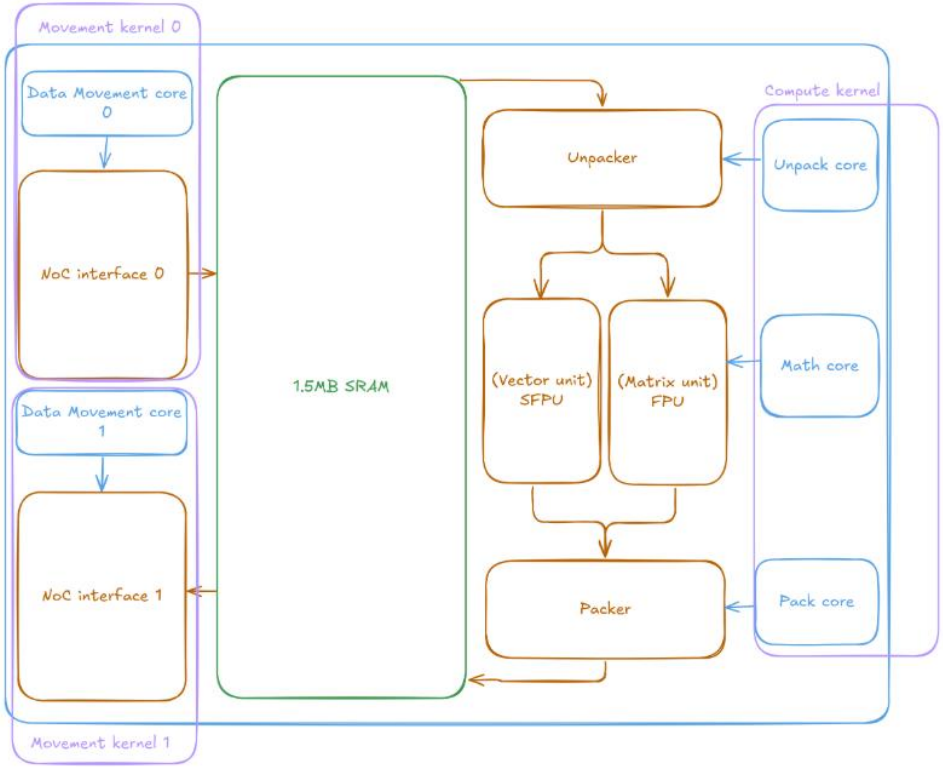
- **The Tensix NoCs**
  - The NoCs are full-duplex
  - Both NoCs supports sending and receiving data at the same time
  - NoC 0 running in the opposite direction of NoC1
  - The uni-directional design of the NoCs reduce power and area
  - The 2D torus topology ensures that every points on the chip remain accessible from every other point



https://clehaxze.tw/gemlog/2025/04-21-programming-tensotrrent-processors.gmi

Tenstorrent AI Processor

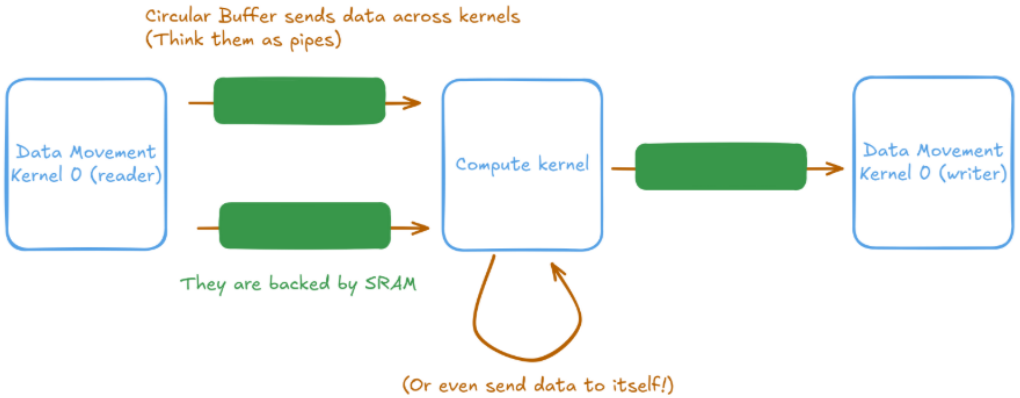
- **3 kernels on Tensix**
  - 2 data movement kernels can be developed separately
  - The 3 compute cores work cooperatively to perform the computation



https://clehaxze.tw/gemlog/2025/04-21-programming-tensotrrent-processors.gmi

Tenstorrent AI Processor

- **Kernel Synchronization**
  - Circular buffers backed by hardware mutexes and SRAM
  - Kernels wait for available space in the circular buffer, write data to it, and then mark that data as ready



https://clehaxze.tw/gemlog/2025/04-21-programming-tensotrrent-processors.gmi

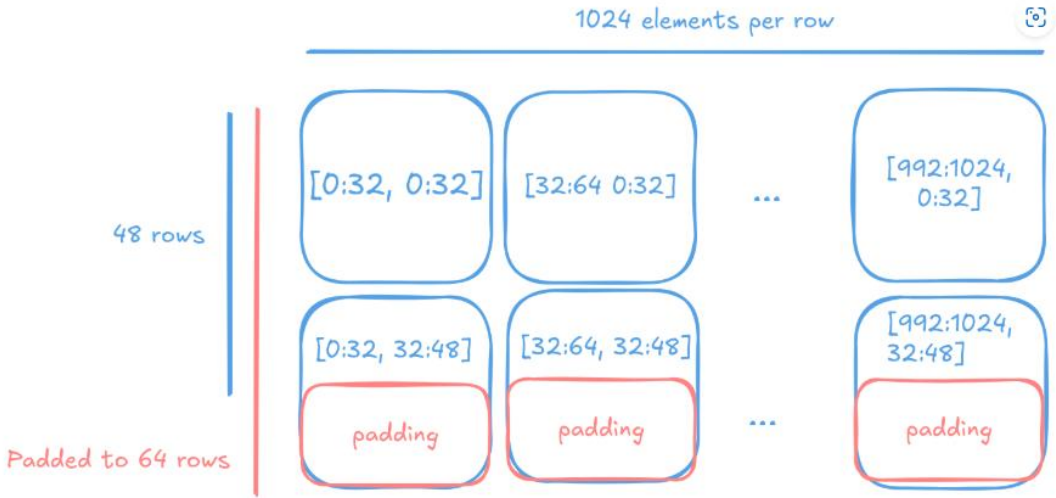
Tenstorrent AI Processor

- **SRAM, interleaved and shared buffers**
  - Allows data, intermediate tensor or operator buffers to live in SRAM
  - This reduces the need for expensive DRAM accesses
  - The “interleaved” mode is used for memory access
  - The “shared mode” reduces the distance and cross talk when accessing DRAM for certain operations

https://clehaxze.tw/gemlog/2025/04-21-programming-tensotrrent-processors.gmi

Tenstorrent AI Processor

- **Native tile based computing**
  - The Tensix natively performs operations on 32 x 32 tiles
  - 32 x 32 tiles are small enough for hardware to digest in a few cycles



https://clehaxze.tw/gemlog/2025/04-21-programming-tensotrrent-processors.gmi

Tenstorrent AI Processor

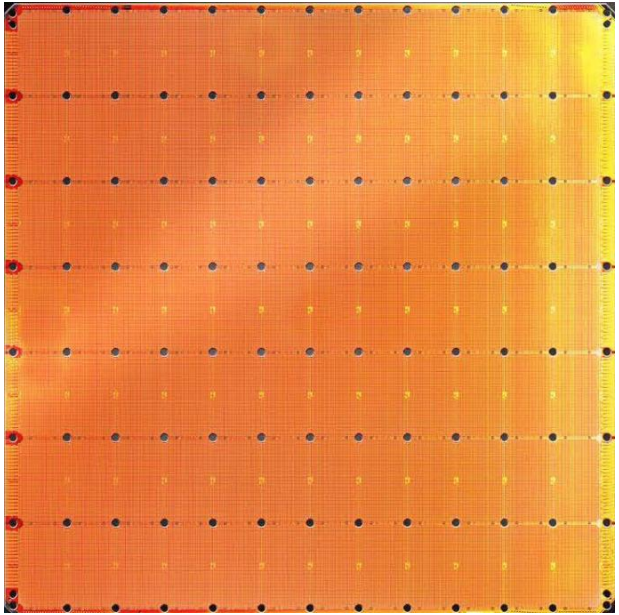
- **Cache Hierarchy**
  - There is no cache hierarchy on Tenstorrent chips
  - Provides direct access to SRAM across the entire chip
  - SRAM is not cache and no automatic caching occurs
  - Data must be explicitly brought into SRAM

https://clehaxze.tw/gemlog/2025/04-21-programming-tensotrrent-processors.gmi

Wafer-scale AI chip -- Cerebras

Largest AI chip

- 46,225 mm^2 silicon
- 1.2 trillion transistors
- 400,000 AI optimized cores
- 18 Gigabytes of on-chip memory
- 9 Pbyte/s memory bandwidth
- 100 Pbit/s fabric bandwidth
- TSMC 16 nm process



Cerebras WSE

21.1 Billion
Transistors
815 mm^2 silicon

GPU

Why big chips ?

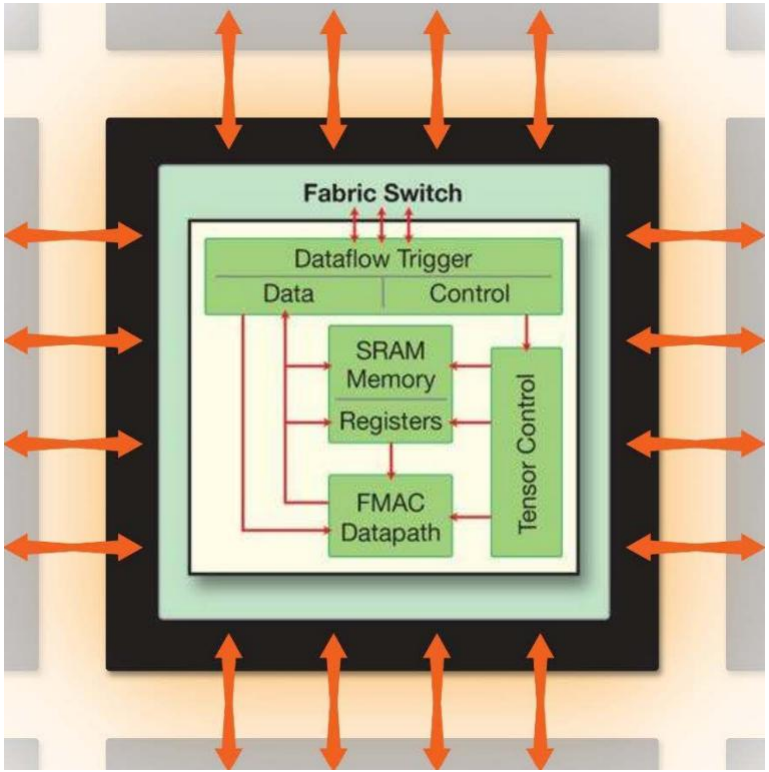
- Big chips process data more quickly
  - Cluster scale performance on a single chip
  - GB of fast memory 1 clock cycle from core
  - On-chip interconnect orders of magnitude faster than off-chip
  - Model-parallel, linear performance scaling
  - Training at scale, with any batch size, at full utilization

Cerebras Architecture

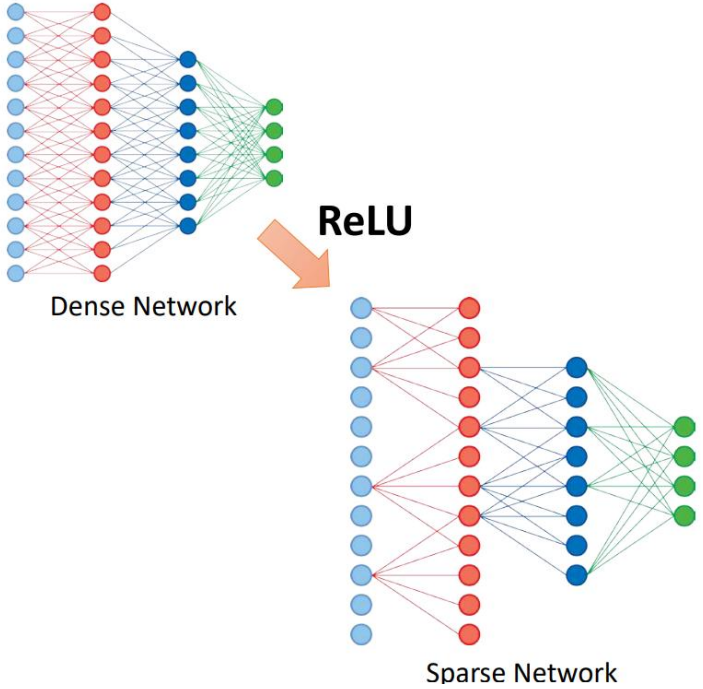
- Core optimized for neural network primitives
- **Flexible, programmable core**
  - NN models are evolving
- **Designed for sparse compute**
  - Workloads contain fine-grained sparsity (where are these sparsity from ?)
- **Local memory**
  - reusing weight & activations
- **Fast interconnect**
  - Layer-to-layer with high bandwidth and low latency

Cerebras programmable core

- Flexible cores optimized for **tensor operations**
  - General ops for control processing
  - e.g. arithmetic, logical, LD/ST, branch
  - Optimized tensor ops for data processing
  - **Tensor operands**
  - e.g. fmac [Z] = [Z], [W], a
    3D 3D 2D



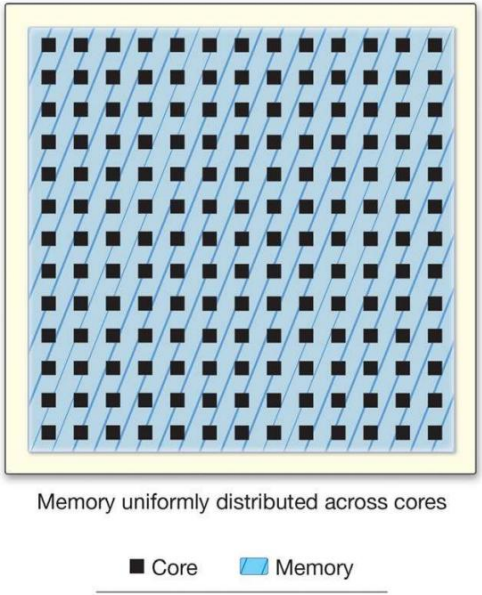
Sparse compute engine



- **Nonlinear activations** naturally create **fine-grained sparsity**
- **Dataflow scheduling in hardware**
  - Triggered by data
  - Filters out sparse zero data
  - Skips unnecessary processing
- Fine-grained execution datapaths
  - Small cores with independent instructions
  - Efficiently processes dynamic, non-uniform work

Cerebras memory architecture

- **Traditional memory designs**
  - Centralized shared memory is slow & far away
  - Requires high data reuse (caching)
  - Local weights and activations are local -> low data reuse
- **Cerebras memory architecture**
  - All memory is fully distributed along compute
  - Datapath has full performance from memory



High-bandwidth low-latency interconnect

- **2D mesh topology** effective for local communication
  - High bandwidth and low latency for local communication
  - All HW-based communication avoids SW overhead
  - Small single-word message

Challenges of wafer scale

- Building a 46,225 mm^2, 1.2 trillion transistor chip
- **Challenges include**
  - Cross-die connectivity
  - Yield
  - Thermal expansion
  - Package assembly
  - Power and cooling

Takeaway Questions

- What are challenges to build a large chip for NN applications ?
  - (A) Power and cooling
  - (B) Fault tolerance for defected dies
  - (C) Package assembly
- How does Cerebras tackle the DNN sparsity ?
  - (A) Customized sparse core
  - (B) Data-driven dataflow scheduling
  - (C) Filters out sparse zero data