Accelerator Architectures for Machine Learning

Lecture 6: Digital DNN Accelerator

Tsung Tai Yeh

Tuesday: 3:30 – 6:20 pm

Classroom: ED-302

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
  CS7960 Neuromorphic Architectures, University of Utah, 2019

Outline

- DaDianNao
- GraphCore IPU
- Wafer-scale AI chip – Cerebras
- SambaNova Reconfigurable Dataflow Unit (RDU)
- Coarse grained reconfigurable array (CGRA)

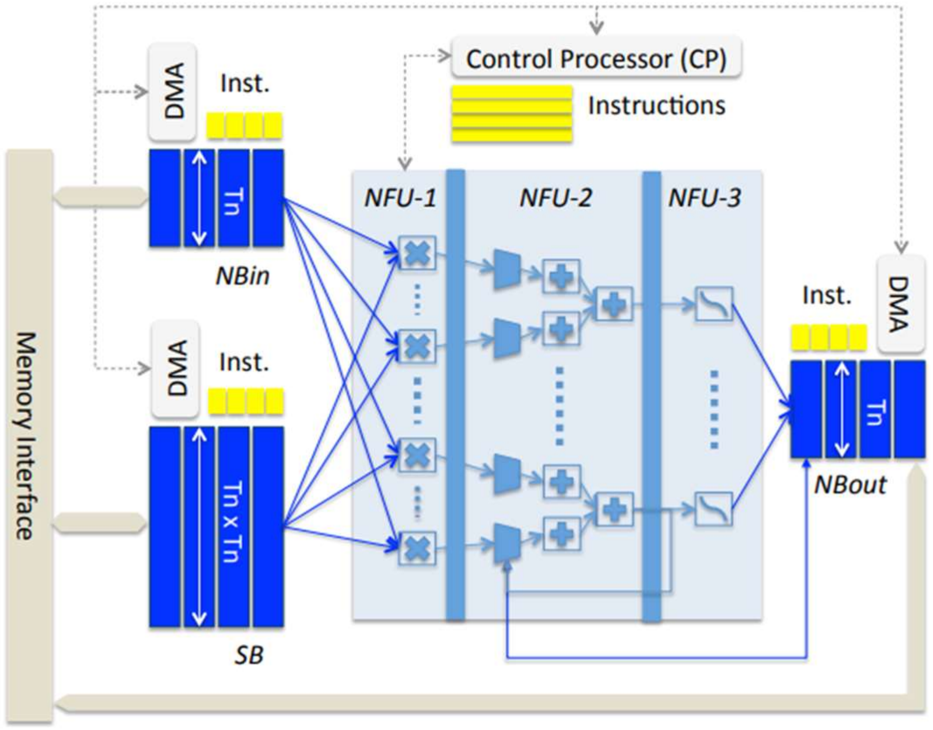
DianNao

DianNao

- Bottleneck
  - The access of large sets of input/weights/outputs
- Design methodology
  - Tiling is used to maximize reuse of the data that is brought into buffers
  - Place data in memory and prefetch “tile” into buffers
  - Each data type has a different requirement -> multiple buffers
  - The ALUs must be time-multiplexed across several neurons

DianNao Architecture

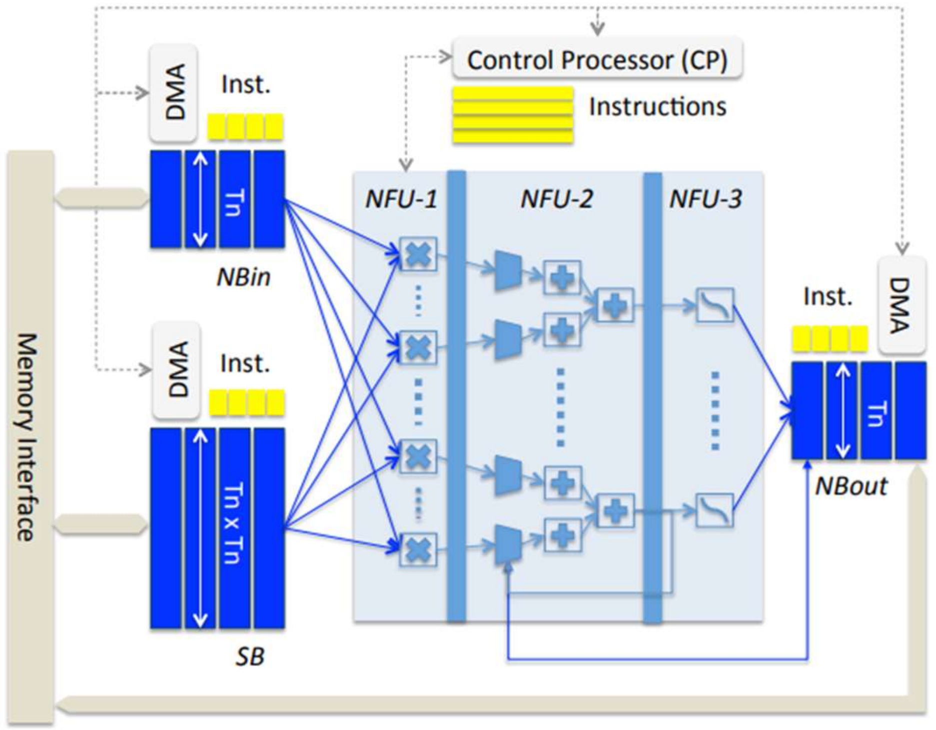
- **DianNao**
  - The NFU can handle 16 neurons in parallel (if a neuron has > 16 inputs?)
  - **Split buffers** – inputs/output/weight
  - **Staggered pipeline** – NFU-1 (pooling) NFU-2 (CONV), NFU-3 (ACT.)
  - The compute unit has 256 parallel multipliers (16 inputs for 16 neurons)
  - 16 adder-trees per neuron
  - Each tree has 15 adders to aggregate the results of 16 multipliers



DianNao, ASPLOS, 2014

DianNao Architecture

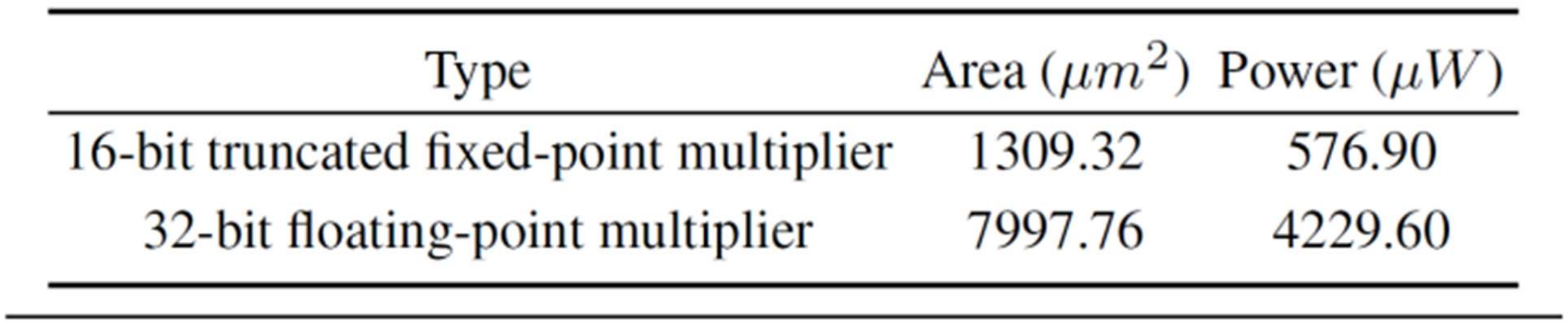
- **DianNao**
  - The sum-of-products is sent to the activation function
  - The ACT -> piecewise linear interpolation
  - The non-linear function is split into 16 linear segments
  - A look-up table tracks the end-points of each segment -> 8-stage pipeline
  - Control processor has instructions that specify how data is loaded/accessed in buffers



DianNao, ASPLOS, 2014

DianNao

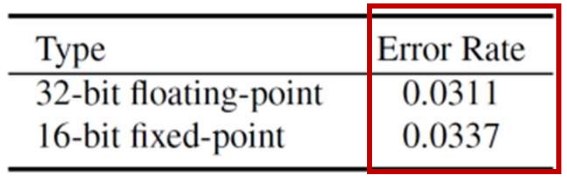
- **DaDianNao**
  - Use 16b fixed-point arithmetic, reduce area/power by ~7x when using 32b float-point math



Type	Area (μm^2)	Power (μW)
16-bit truncated fixed-point multiplier	1309.32	576.90
32-bit floating-point multiplier	7997.76	4229.60

Table 2. *Characteristics of multipliers.*

Fixed Point Error Rates



Type	Error Rate
32-bit floating-point	0.0311
16-bit fixed-point	0.0337

Table 1. *32-bit floating-point vs. 16-bit fixed-point accuracy for MNIST (metric: error rate).*

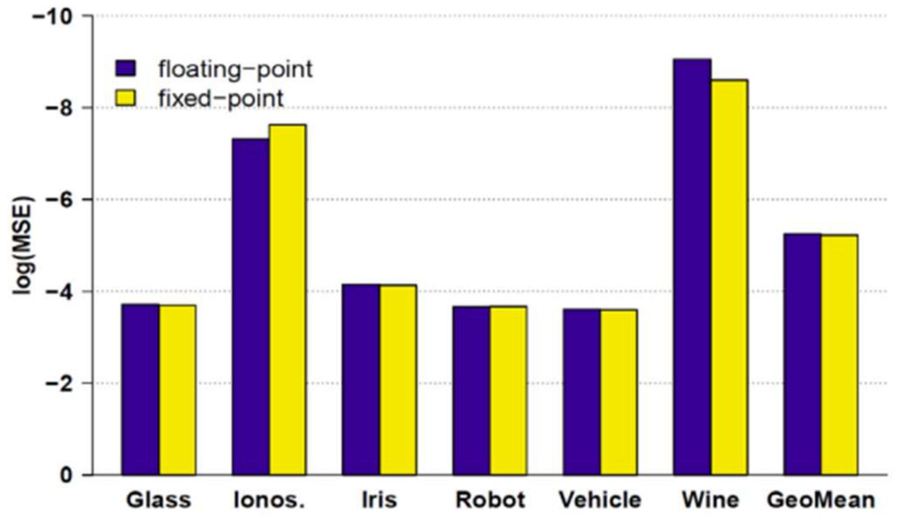


Figure 12. *32-bit floating-point vs. 16-bit fixed-point accuracy for UCI data sets (metric: log(*Mean Squared Error*)).*

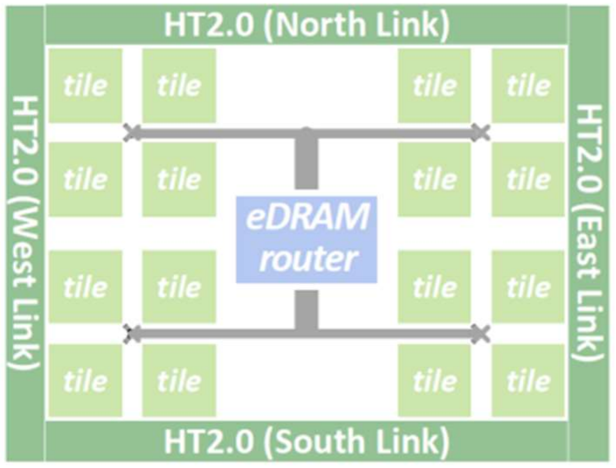
Other details

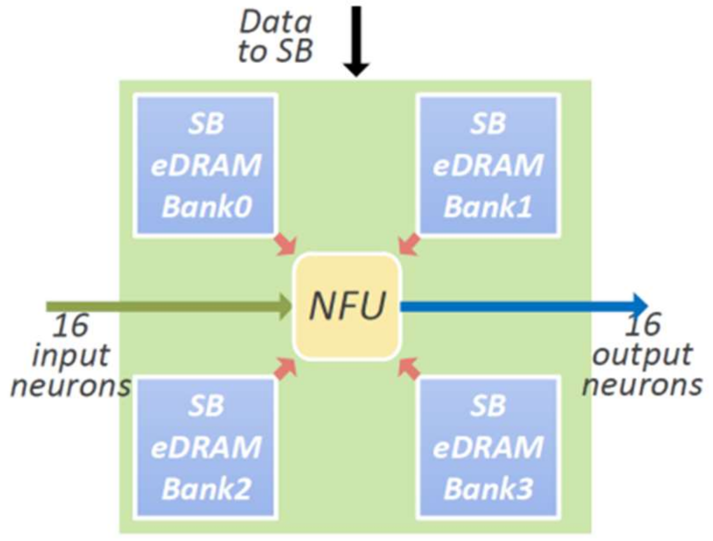
All these improvement are because DienDao removes the memory access bottleneck

- The NFU is composed of 8 pipeline stages
- Peak activity is nearly 500 GOP/s
- 44 KB of RAM capacity
- Buffers are about 60% of area/power, while NFU is ~30%
- Energy is 21X better than a SIMD baseline where has high cost of memory accesses
- Tiling to reduce memory traffic
- Big performance boots as well: higher computational density, tiling, prefetching

DaDianNao Philosophy

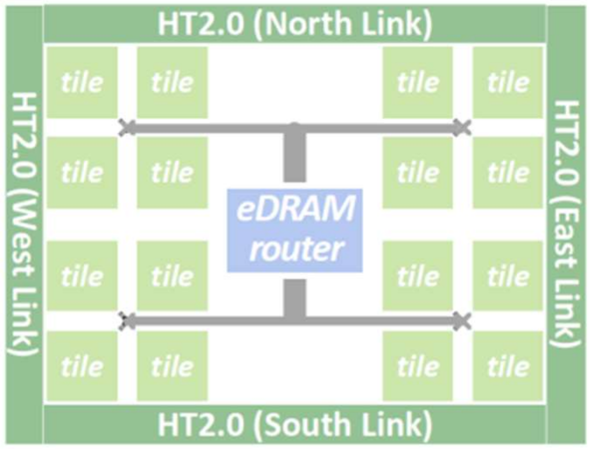
- Avoid going to off-chip memory altogether
- Keeps the weights on-chip in eDRAM banks (why eDRAM not SRAM?)
- Many chips as required to keep all weights in on-chip eDRAM bank
- Every operation is spread across several “tiles” to maximize parallelism

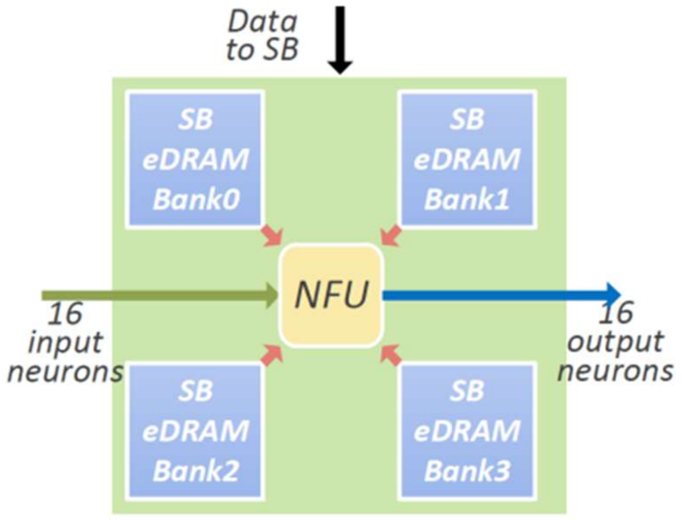




DaDianNao Philosophy II

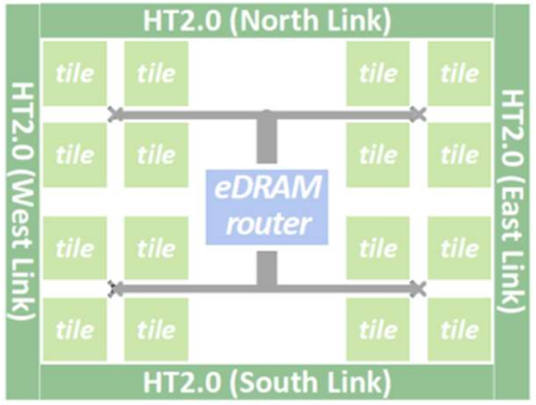
- Near data processing -> a computation is performed on the NFU next to the eDRAM that has the necessary weight
- When those neurons produce their output
- These outputs are broadcast to all the tiles that need these as inputs to the next layer

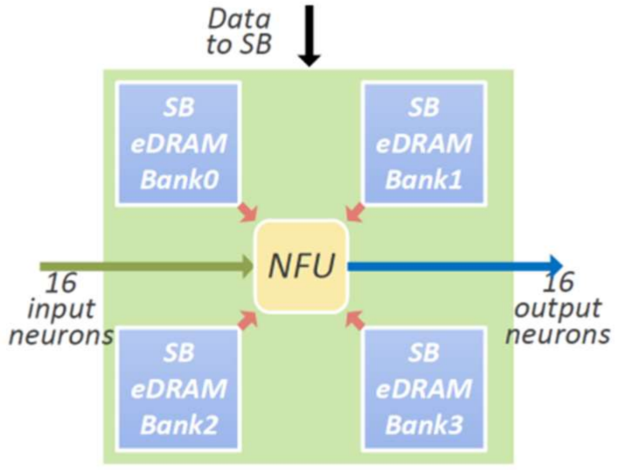




DaDianNao Philosophy III

- A chip has 16 tiles that share two 2 MB eDRAM banks (input from previous layer and the output of the current layer)
- 32 MB eDRAM banks for weights
- The NFU needs to receive 512 B/cycle to stay busy during the classifier layer -> each eDRAM bank has a read width of 512 bytes (wiring overhead)





DaDianNao Layouts

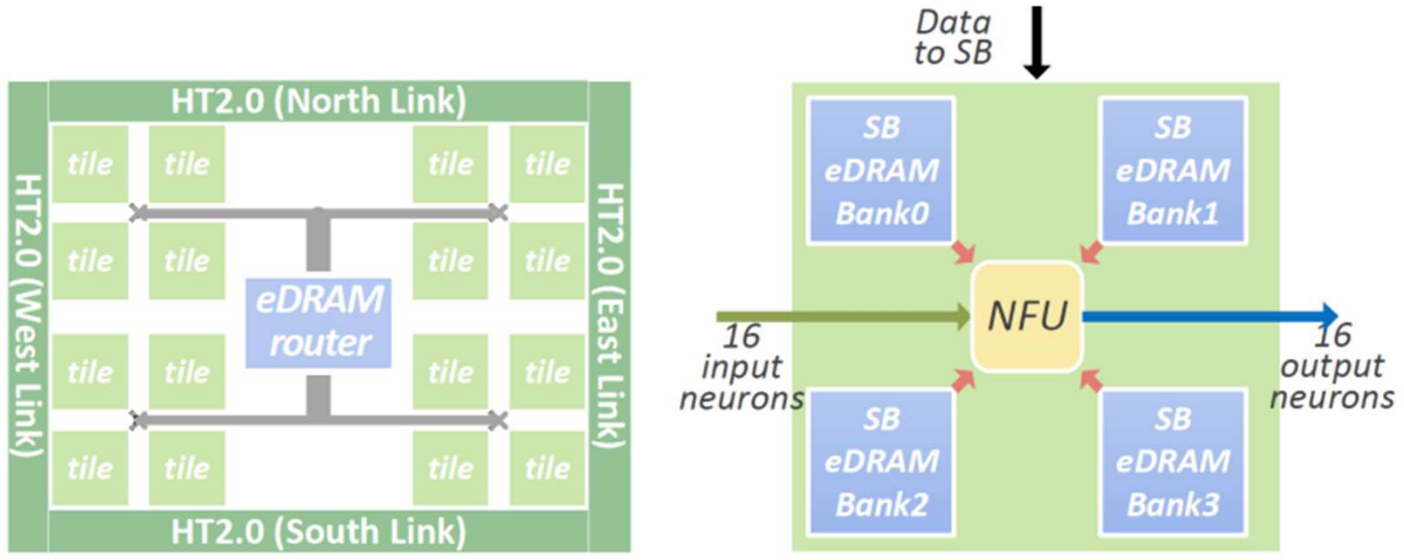


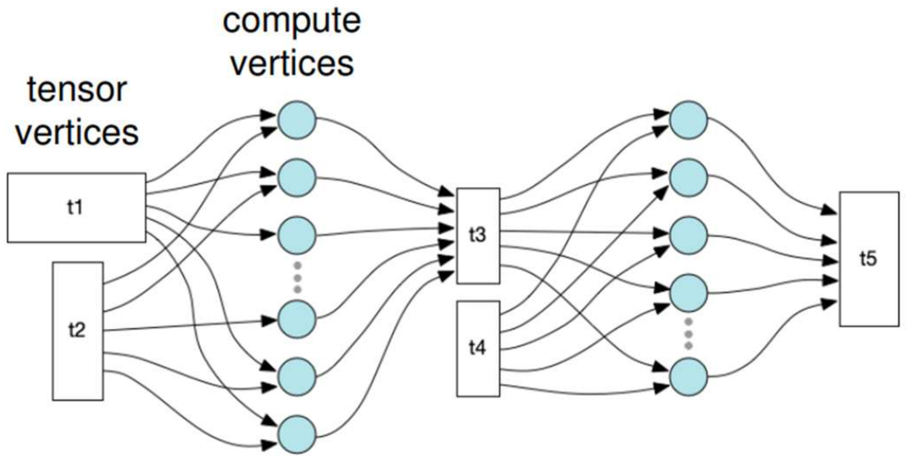
Figure 5: *Tile-based organization of a node (left) and tile architecture (right). A node contains 16 tiles, two central eDRAM banks and fat tree interconnect; a tile has an NFU, four eDRAM banks and input/output interfaces to/from the central eDRAM banks.*

Each eDRAM bank size is 512 KB (3 cyc); central eDRAM bank is 2MB (10 cyc); total node storage is 36 MB; HT bw is 6.4 x 4 GB/s (80ns).

GraphCore IPU

GraphCore IPUs

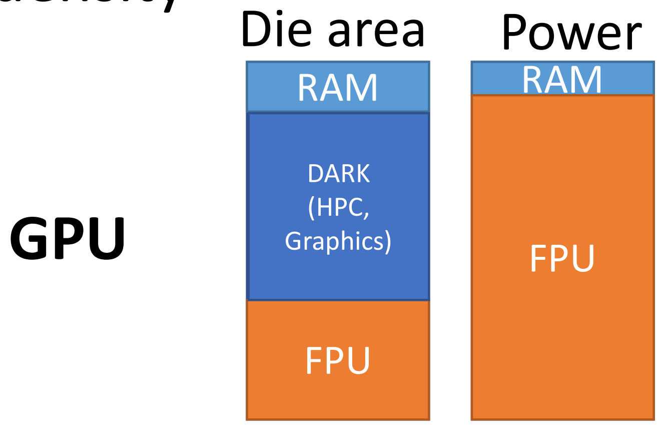
- **GraphCore Intelligent Processing Units (IPUs)**
  - Unlike GPU that is dedicated to accelerate large dense matrix
  - IPUs supports **dynamic sparse training** and unstructured computation such as path tracing in 3D computer graphics
  - Multiple **tile processors**
  - **Poplar programming model**
    - Dedicated compiler (PopC)
    - Mapping compute graph to tile processors
    - Compute kernels (Codelets)

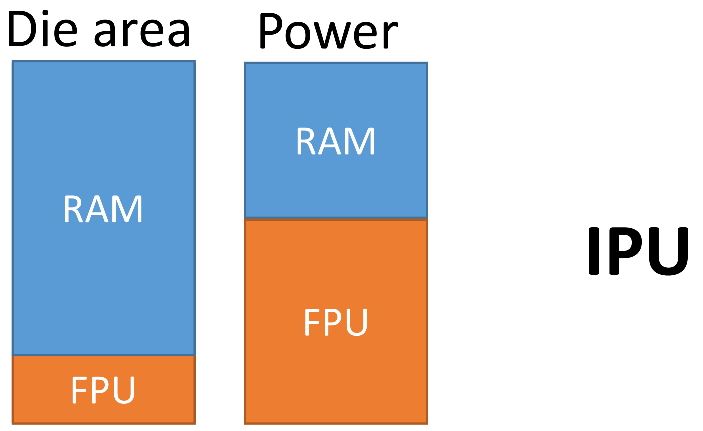


https://hc33.hotchips.org/assets/program/conference/day2/HC2021.Graphcore.SimonKnowles.v04.pdf

Graphcore IPU Approach

- Post-Dennard, **the silicon is power-limited**
  - we can put more logic on the die than we can power (dark silicon)
- **IPU architecture approach**
  - Replace dark silicon logic with on-chip RAM that has lower power density

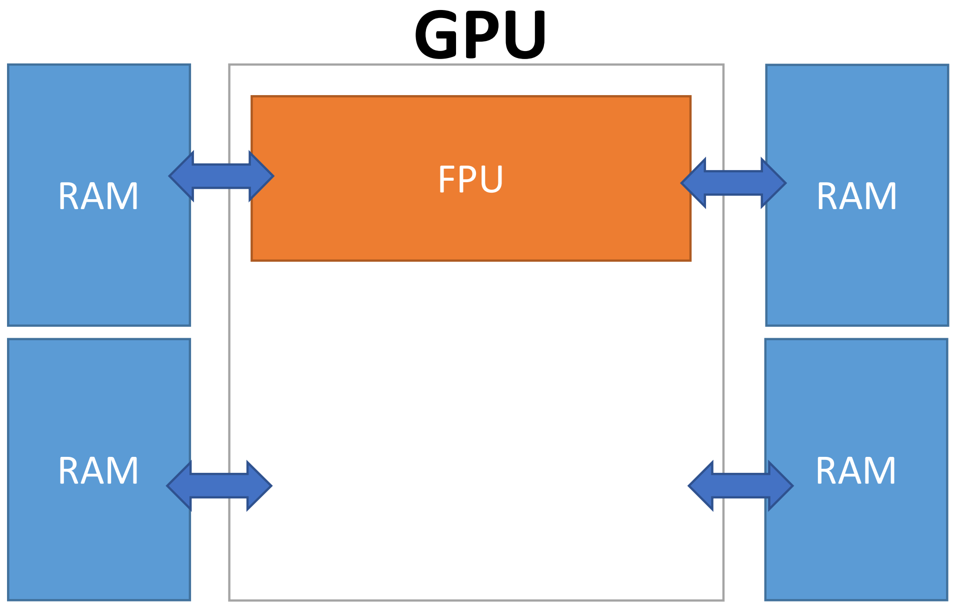


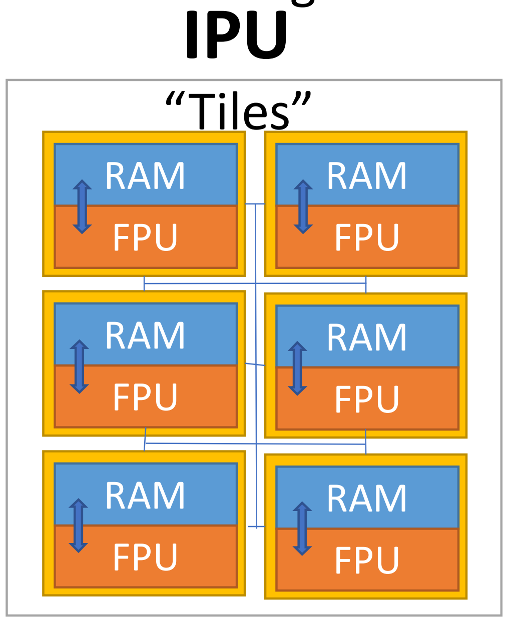


From: Knowles, Simon. *Designing Processors for Intelligence*. 2017. *UC Berkeley EECS Events*, https://www.youtube.com/watch?v=7XtBZ4Hsi_M.

Graphcore IPU approach

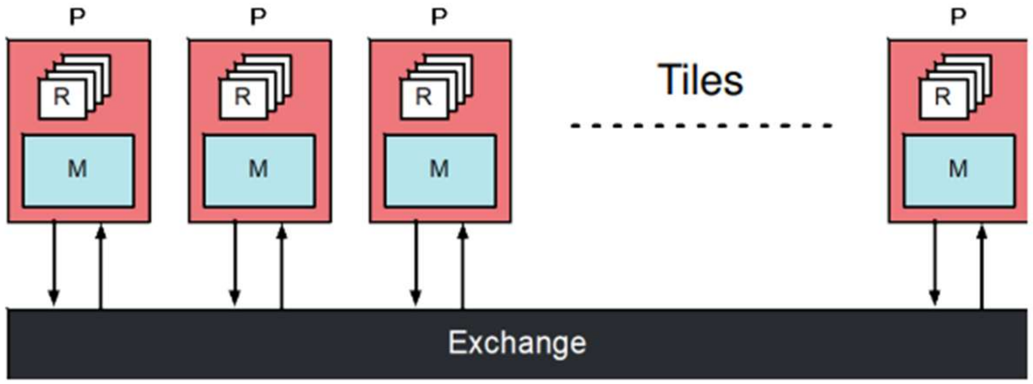
- **GPU approach**
    - Shared memory model with caches and memory hierarchy to reduce latency
- **IPU approach**
    - Move as much memory as possible into the chip local to the logic





Graphcore IPU Abstraction

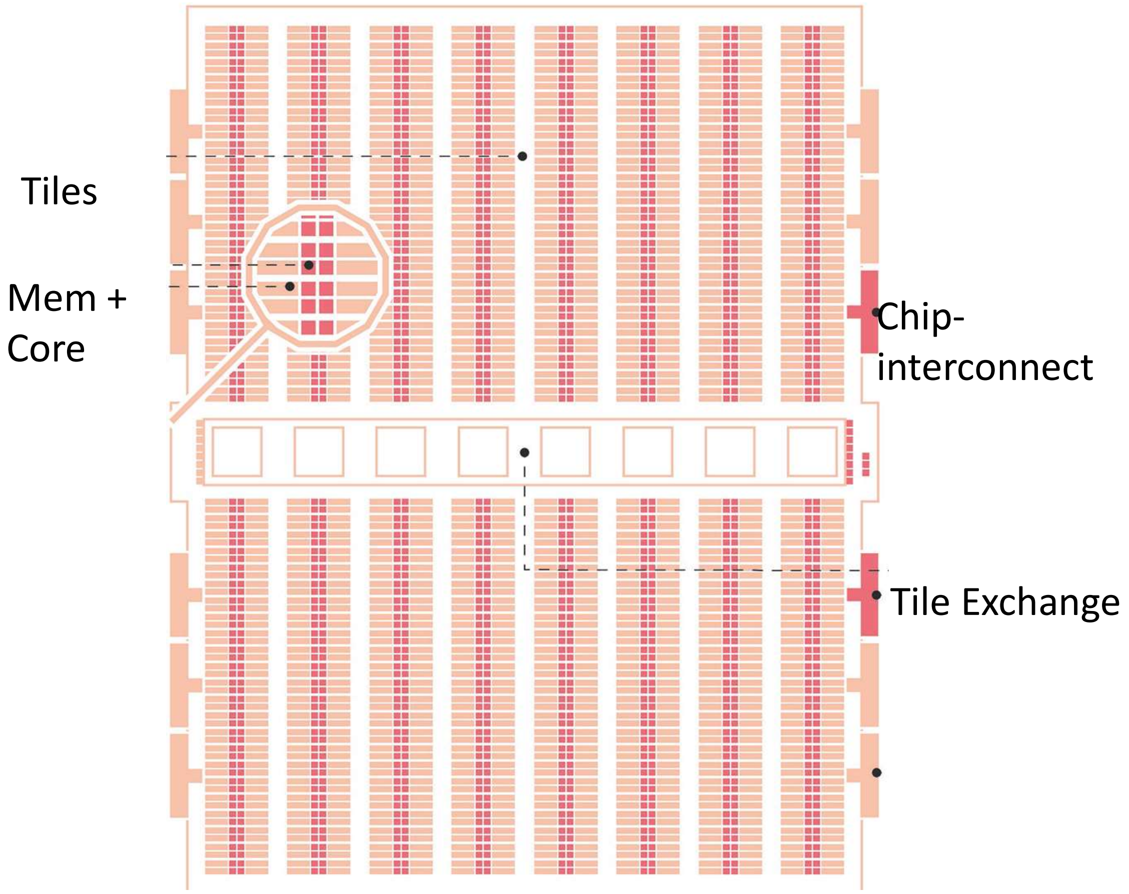
- Tile processors
  - Each tile is a multi-threaded processor and has its local memory
  - Tiles communicate through all-to-all, stateless exchange
- A tensor vertex
  - Can be distributed over many tiles



Distributed memory architecture

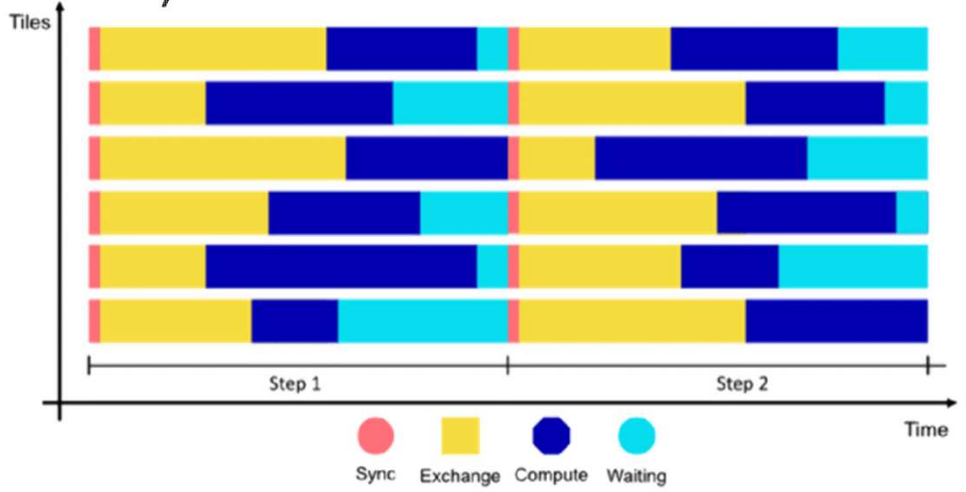
- 1472 tiles with 6 threads sharing 624 KiB of local SRAM
- Total of 896 MiB and 250Tflop/s in 8832 worker threads
- 7.8TB/s exchange between tiles
- Tiles have no shared memory or caches

Graphcore Architecture White paper, https://www.graphcore.ai/products/ipu



Execution Model

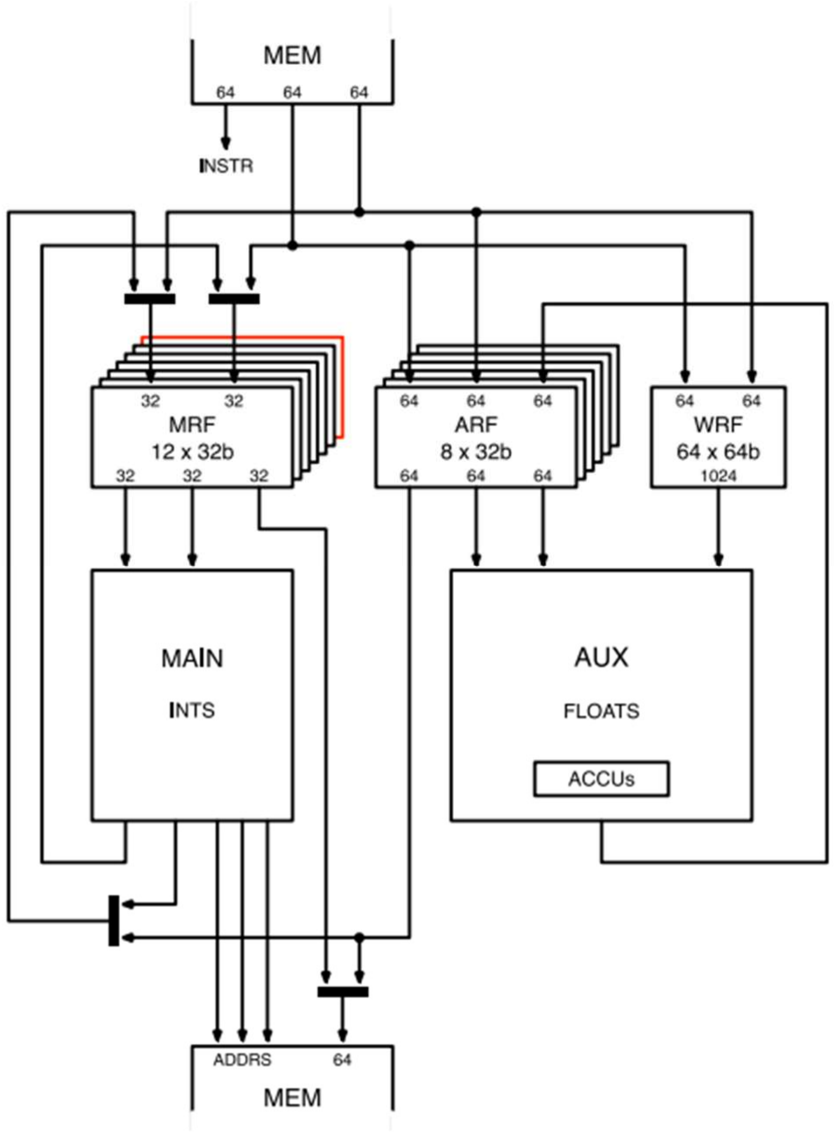
- Tile workers execute instructions independently in parallel (**MIMD**),
- Wait for sync, followed by all-to-all data exchange phase (hardware implementation of **Bulk Synchronous Parallel (BSP)** Model)
- No concurrency hazards (races, deadlocks etc.)
- Compiler faces hard job of scheduling and *load-balancing* compute-chunks on tile workers



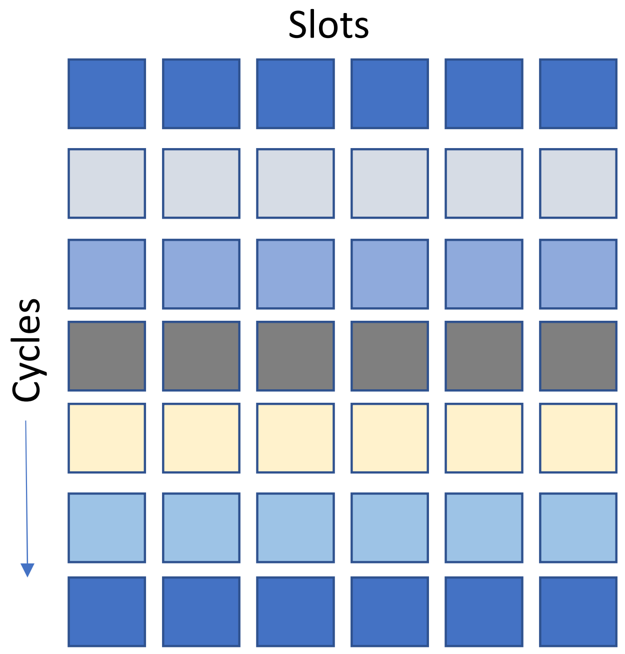
Jia, Zhe, et al. 'Dissecting the Graphcore IPU Architecture via Microbenchmarking'. *ArXiv:1912.03413*, Dec. 2019. *arXiv.org*, http://arxiv.org/abs/1912.03413.

Tile Processor

- 32b instructions, single or dual issue
- **Two execution paths:**
    - **MAIN:**
        - Control flow, integer/address arithmetic, load/store to/from either path
    - **AUX**:
        - Floating-point arithmetic for tensor operations + special instructions like log, tanh, PRNG etc.



Tile Processor

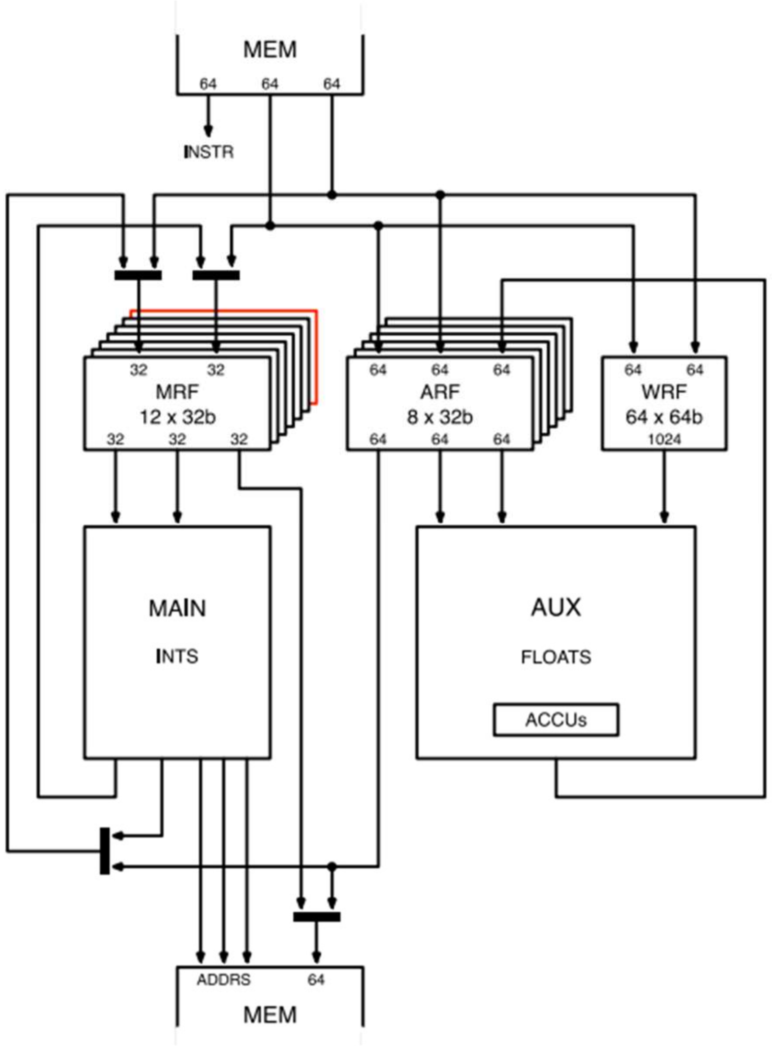


- Fine-grained multithreading that switches between 6 threads on every cycle in round-robin fashion
    - Issued worker programs run in a slot at 1/6 of the clock, so they can't see the pipeline, i.e., mem access, branches etc. all appear to take one cycle per instruction
    - This makes worker execution simple for the compiler to predict for easier load balancing

'Graphcore IPUs'. *2021 IEEE Hot Chips 33 Symposium (HCS)*, 2021, pp. 1–25. *IEEE Xplore*, https://doi.org/10.1109/HCS52781.2021.9567075.

N + 1 barrel threading

- 7 program contexts
- 6 round-robin pipeline slots
- **The supervisor program**
  - A fragment of the control program
  - Orchestrating the update of vertices
  - Execute in all slots not yielded to workers
  - Dispatch workers by **RUN** instruction
- A **worker program** is a codelet updating a vertex
  - Execute in 1 slot at 1/6 of clock
  - Returns its slot to the supervisor by **EXIT** instruction



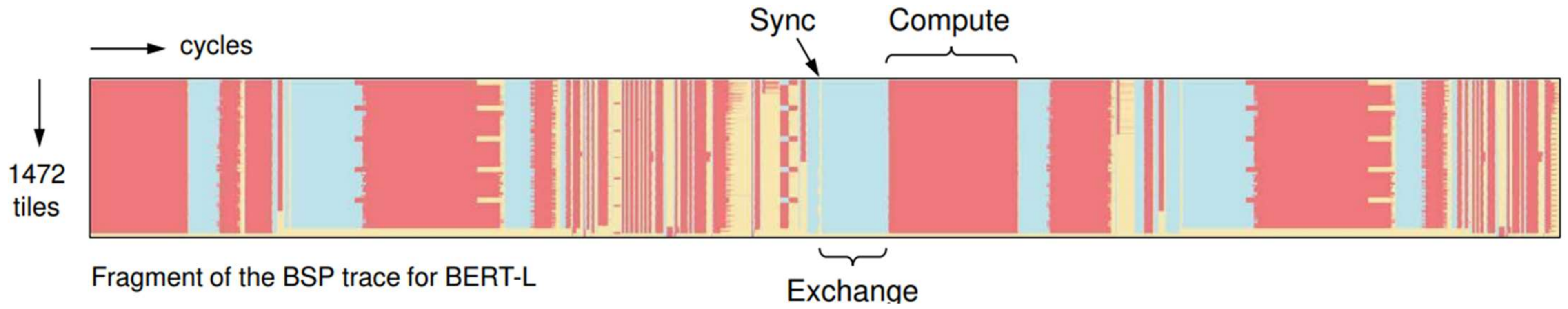
Sparse Load/Store

- **Large on-die SRAM memory**
  - 896 MiB on-die SRAM at 47TB/s (data-side)
  - Access arbitrarily-structured data which fits on chip
- **Ld/St instructions**
  - Support sparse gather in parallel with arithmetic at full speed via compact pointer lists
  - 16b absolute offsets to a base
  - 4b cumulative delta offsets to a base

‘Graphcore IPUs’. *2021 IEEE Hot Chips 33 Symposium (HCS)*, 2021, pp. 1–25. *IEEE Xplore*, https://doi.org/10.1109/HCS52781.2021.9567075.

Global Program Order

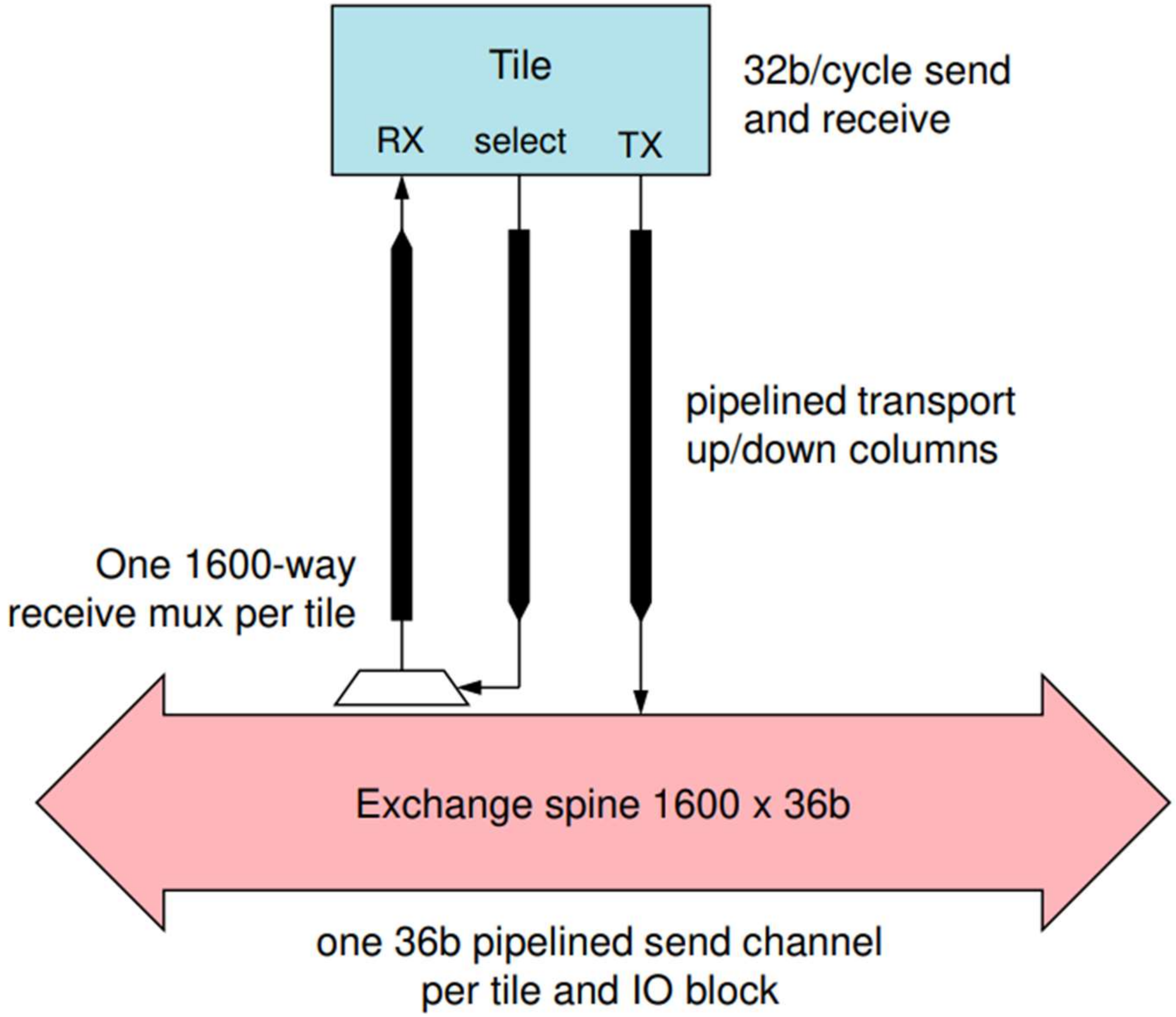
- **Tile processors**
  - Execute asynchronously until they need to exchange data
  - Each tile executes a list of atomic codelets in one compute phase
- **Bulk Synchronous Parallel**
  - Repeat {Sync; Exchange; Compute}
  - Hardware global sync. In ~150 cycles on chip, 15 ns/hop between chips



Fragment of the BSP trace for BERT-L

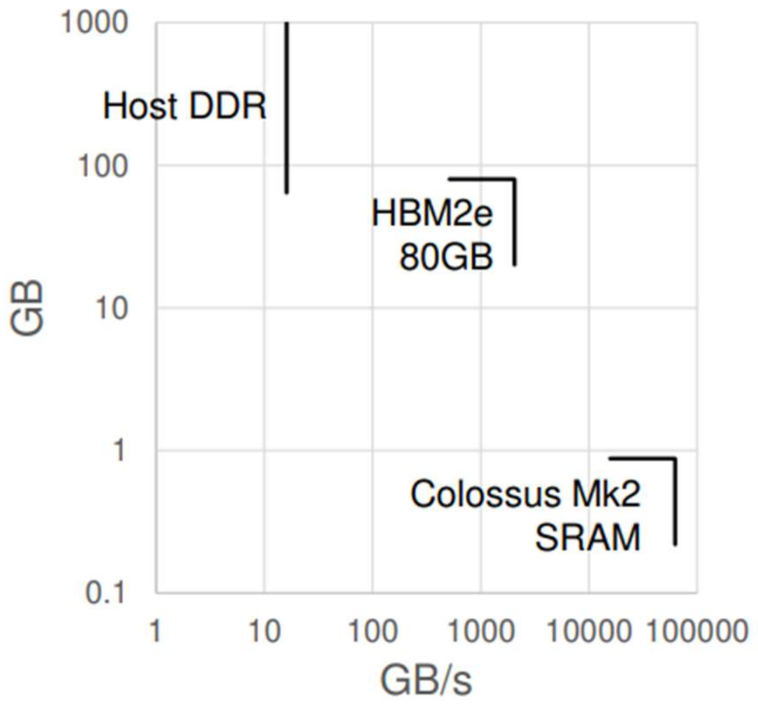
Exchange Mechanics

- **IPU POPLAR compiler**
  - Schedule transmit, receive and select at precise cycles from sync
  - Knowing all pipeline delays
- **Data movement**
  - At full bandwidth
  - No queues, arbiters, or packet overheads


Tile
RX
select
TX
32b/cycle send and receive
pipelined transport up/down columns
One 1600-way receive mux per tile
Exchange spine 1600 x 36b
one 36b pipelined send channel per tile and IO block

Why no HBM Memory ?

- **Memory bandwidth** limits how fast AI can complete
- **GPU and TPU**
    - Solve for bandwidth and capacity using HBM
    - HBM is expensive, capacity-limited, and adds 100W+ to the processor thermal envelope
- **IPU**
    - Solves for bandwidth with SRAM, and for capacity with DDR



IPU hardware helps software

- Simple mechanisms allow software evolution
    - Native graph abstraction
    - Codelet-level parallelism
    - Pipeline-oblivious threads
    - BSP removes concurrency hazards
    - Stateless all-to-all exchange
    - Cacheless, uniform, near/far memory

‘Graphcore IPUs’. *2021 IEEE Hot Chips 33 Symposium (HCS)*, 2021, pp. 1–25. *IEEE Xplore*, https://doi.org/10.1109/HCS52781.2021.9567075.

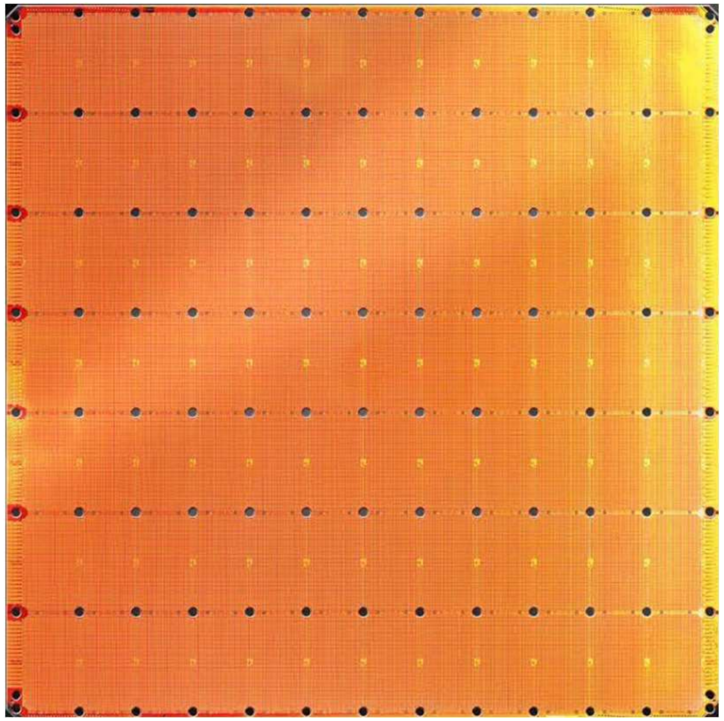
Takeaway Questions

- How does DaDianDao hide memory access latency ?
  - (A) Increase the size of the memory
  - (B) Increase the number of eDRAM banks
  - (C) Increase the number of tiles
- Why GraphCore IPU employ large SRAM instead of HBM?
  - (A) Achieve high bandwidth
  - (B) Large memory capacity
  - (C) Save silicon area

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

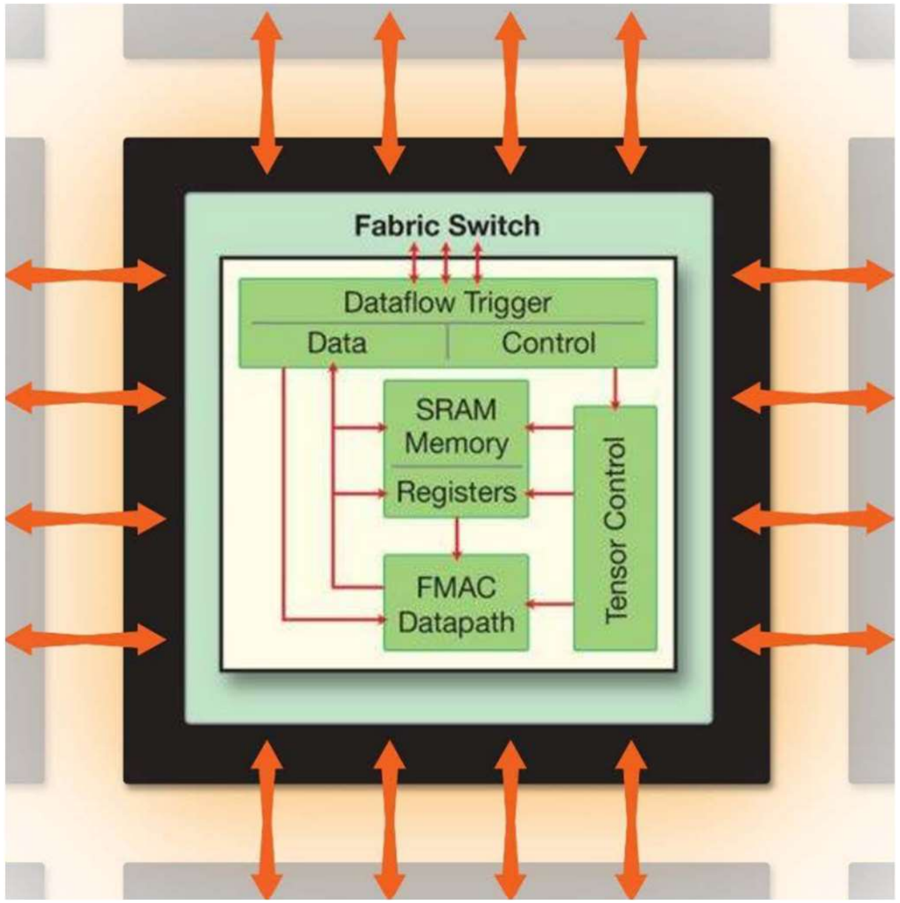
Cerebras Architecture

- 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

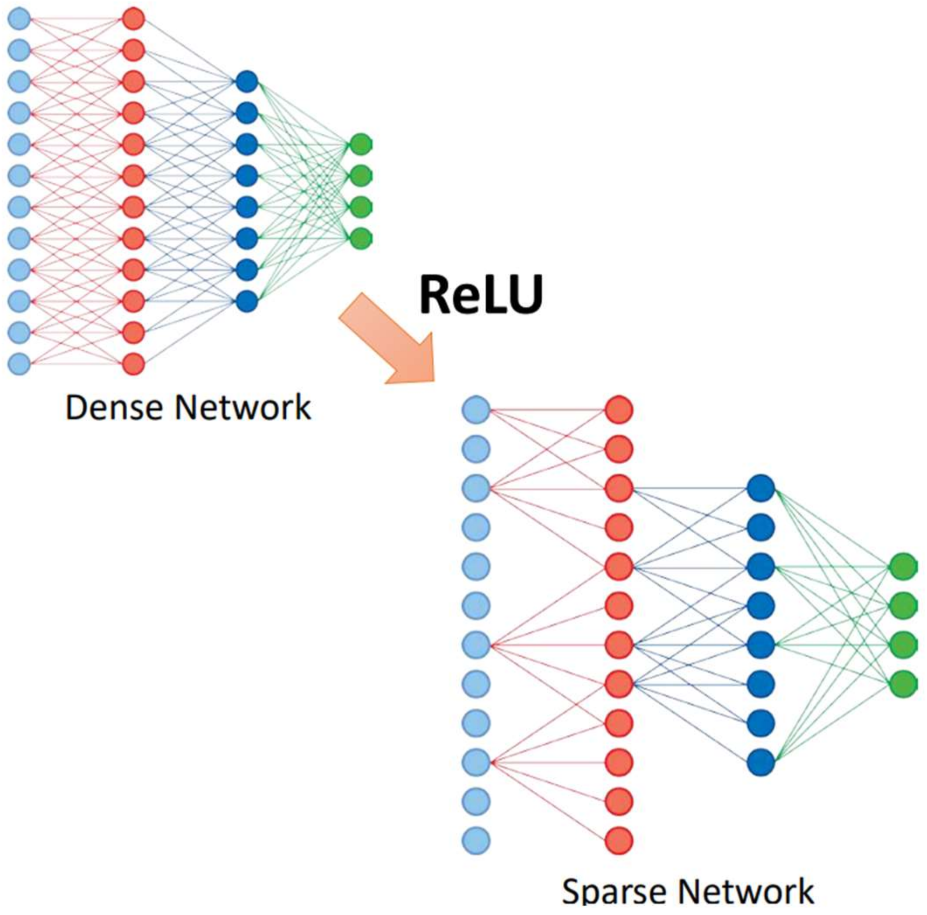
Cerebras programmable core

- 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



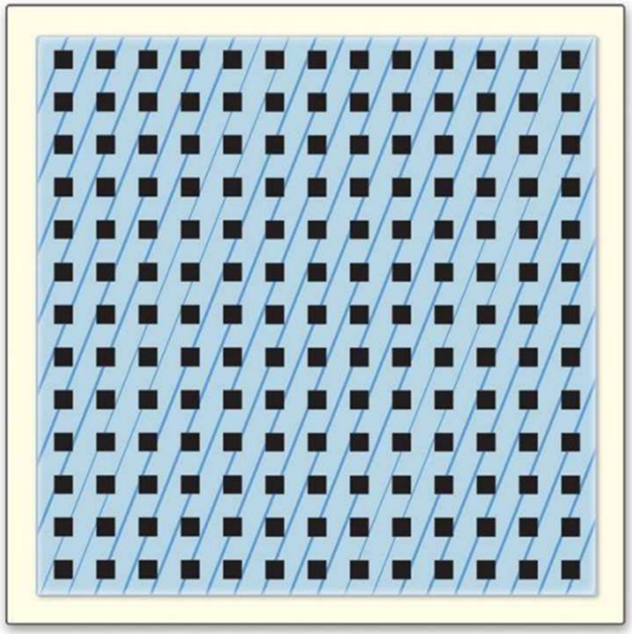
Sparse compute engine



- **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


Memory uniformly distributed across cores

■ Core ▨ Memory

High-bandwidth low-latency interconnect

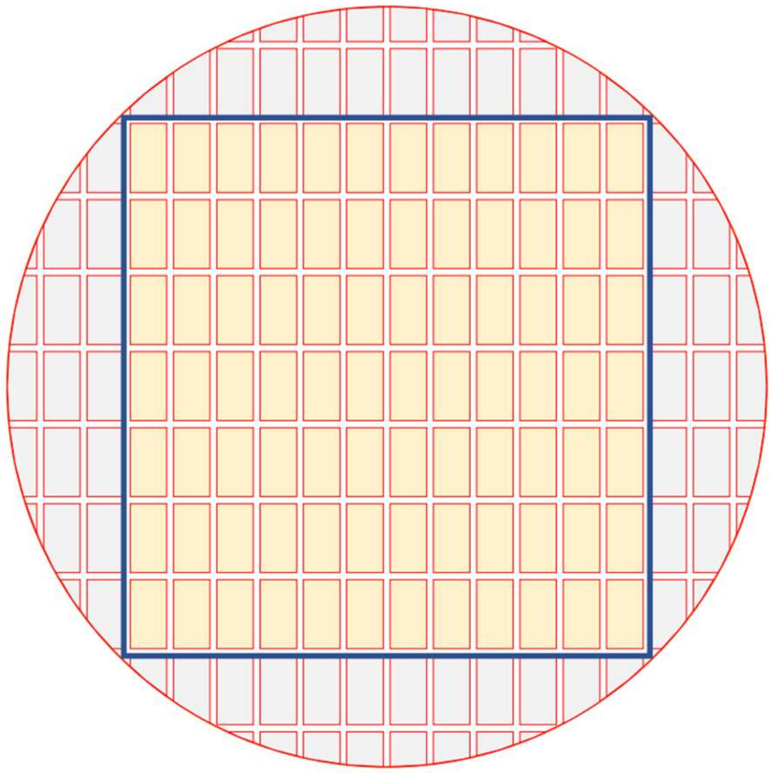
- **2D mesh topology** effective for local communication
  - High bandwidth and low latency for local communication
  - All HW-based communication avoids SW overhead
  - Small single-word message

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

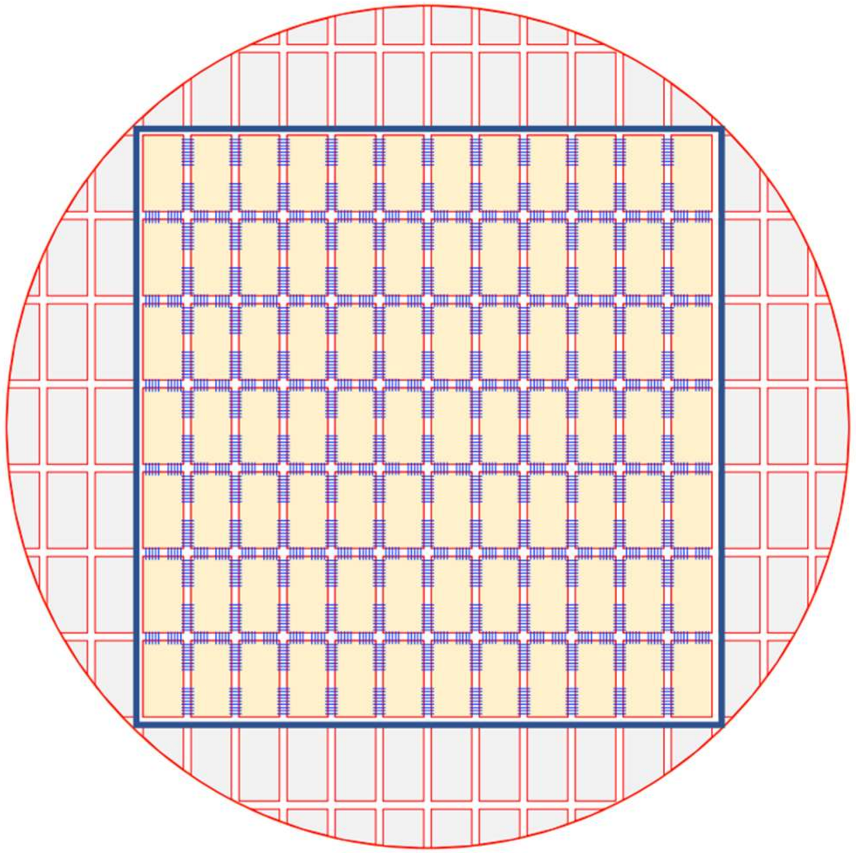
Challenge 1: cross die connectivity

- Standard fabrication process requires die to be independent
- Scribe line separates each die
- Scribe line used as mechanical barrier for die cutting for test structures



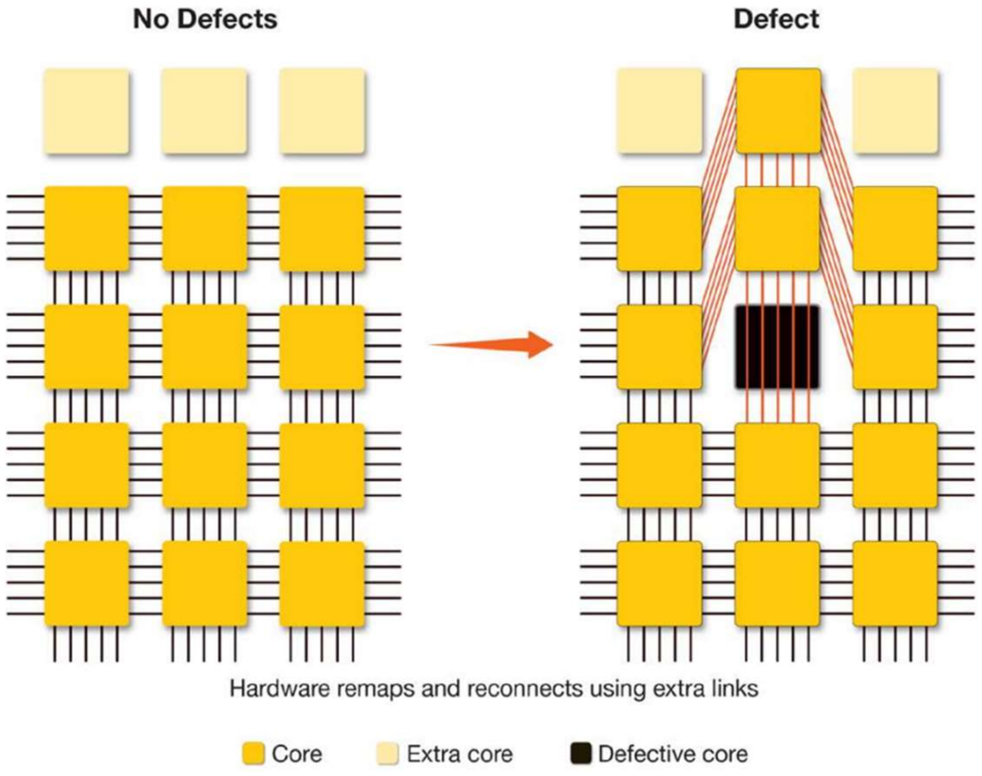
Cross-die wires

- Add wires across scribe line with TSMC
- Extend 2D mesh across die
- Same connectivity between cores and across scribe lines create a homogeneous array
- Short wires enable ultra high bandwidth with low latency



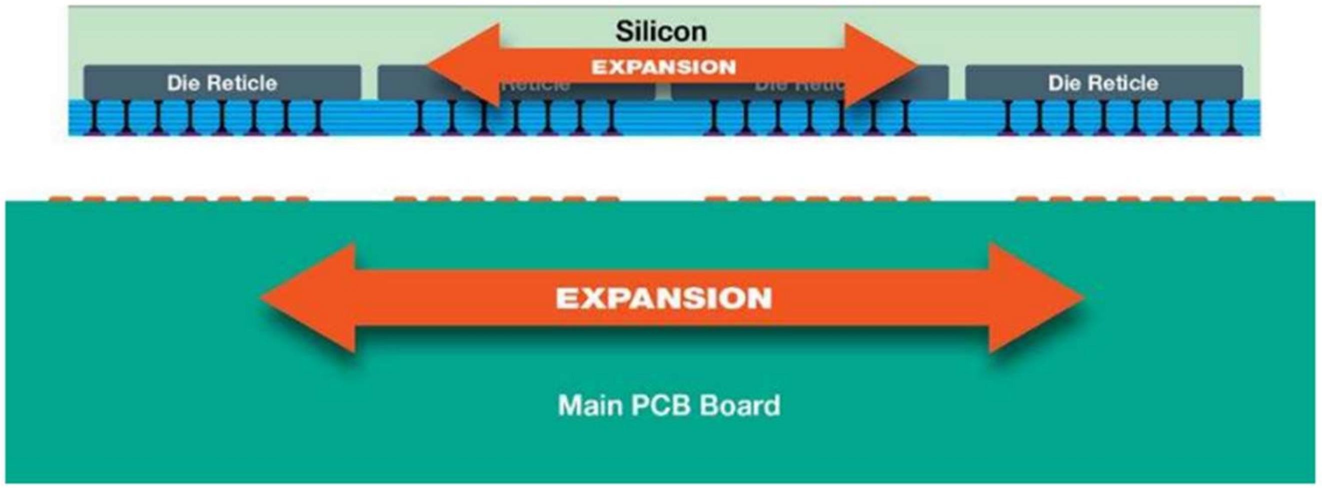
Challenges II: Yield

- Impossible to yield full wafer with zero defects
    - Silicon and process defects are inevitable even in mature process
- **Redundant cores**
    - Uniform small cores
    - Redundant cores and fabric links
    - **Redundant cores replace defective cores**
    - **Extra links** reconnect fabric to restore logical 2D mesh



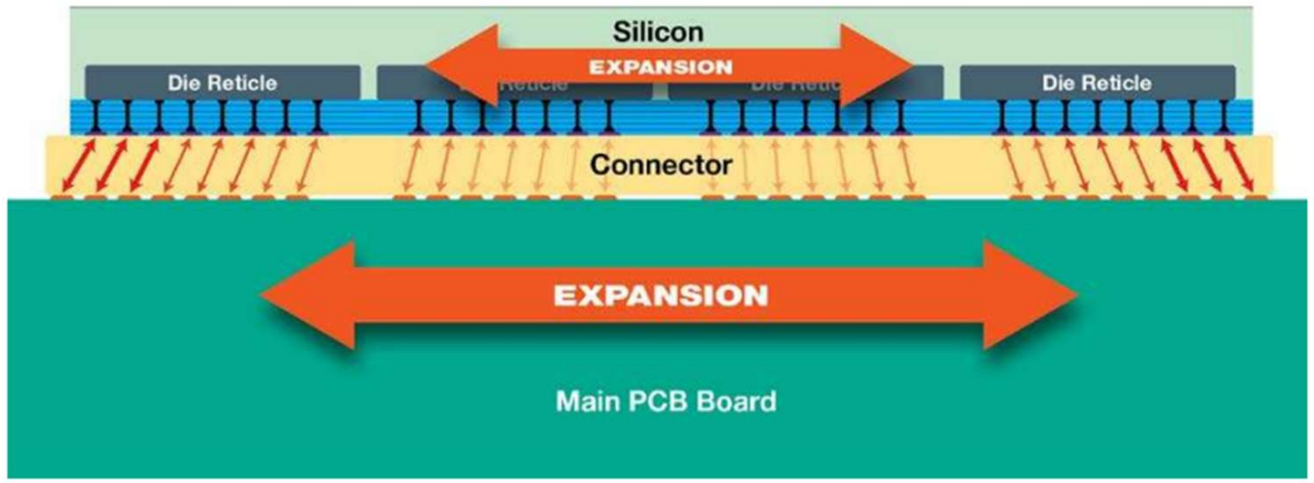
Challenge III: Thermal expansion in package

- **Silicon and PCB expand at different rates** under temperature
- Size of wafer would result in too much mechanical stress using traditional package technology



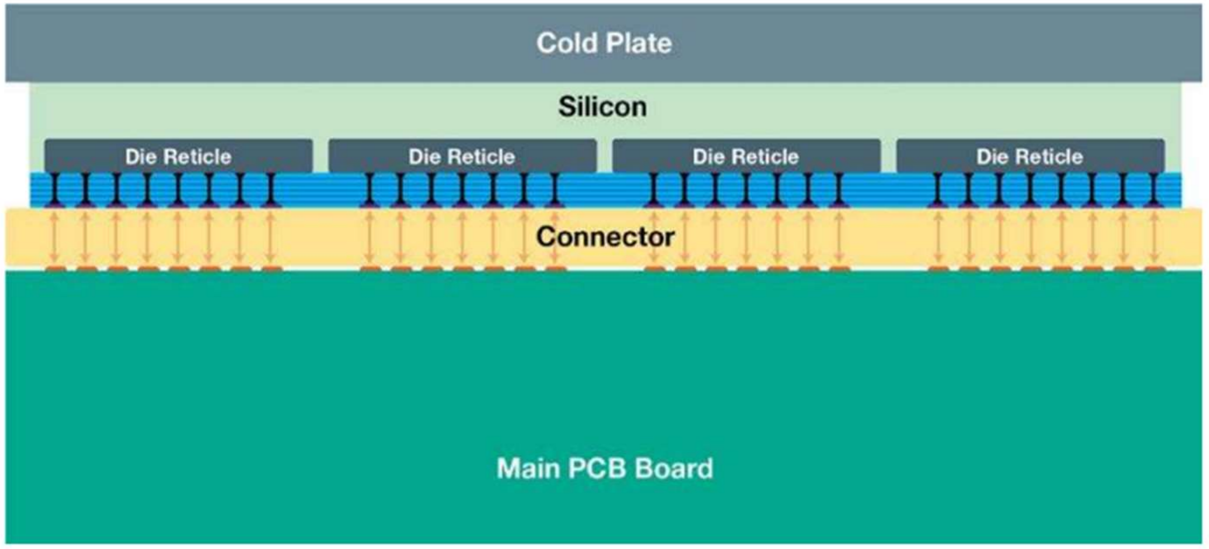
Connecting wafer to PCB

- Developed custom connector to connect wafer to PCB
- Connector absorbs the variation while maintaining connectivity



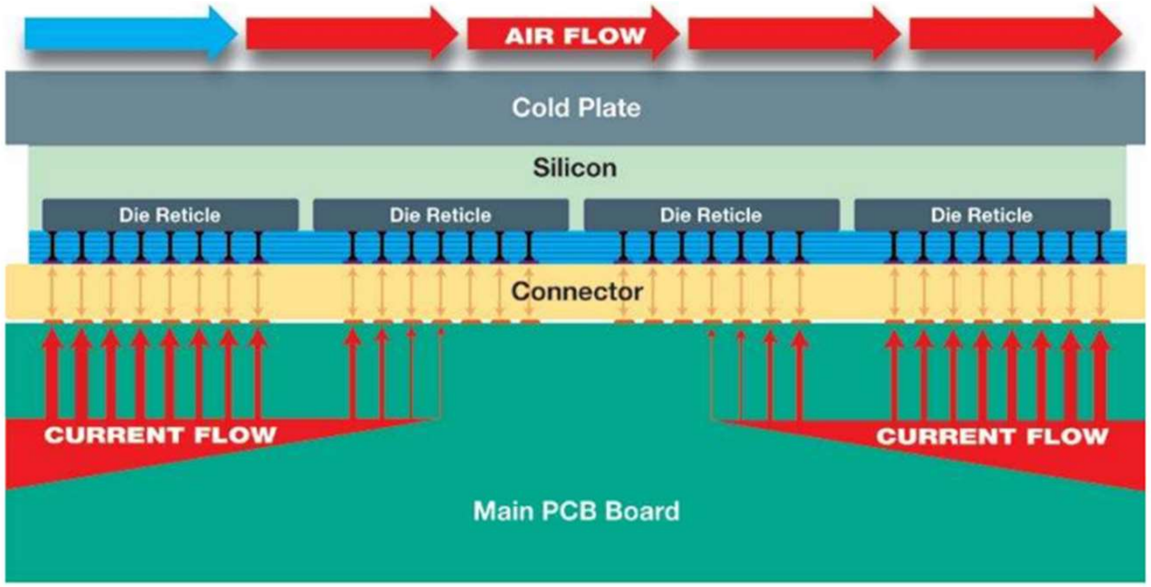
Challenge IV: Package assembly

- Package includes
  - PCB
  - Connector
  - Wafer
  - Cold plate
- All components require precise alignment
- Developed custom machines and process



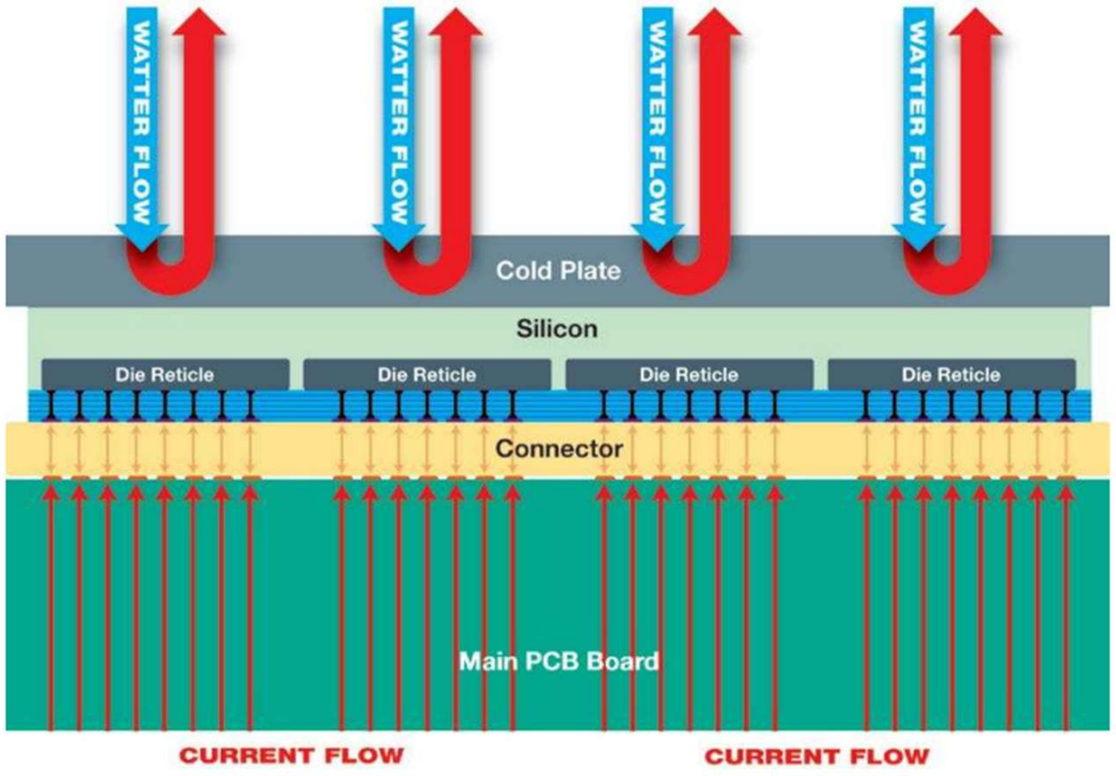
Challenge V: Power and cooling

- Concentrated high density exceeds traditional power & cooling capacities
- **Power delivery**
  - Current density too high for power plane distribution in PCB
- **Heat removal**
  - Heat density too high for direct air cooling



Using the 3rd dimension

- **Power delivery**
  - Current flow distributed in 3rd dimension perpendicular to water
- **Heat removal**
  - Water carries heat from wafer through cold plate



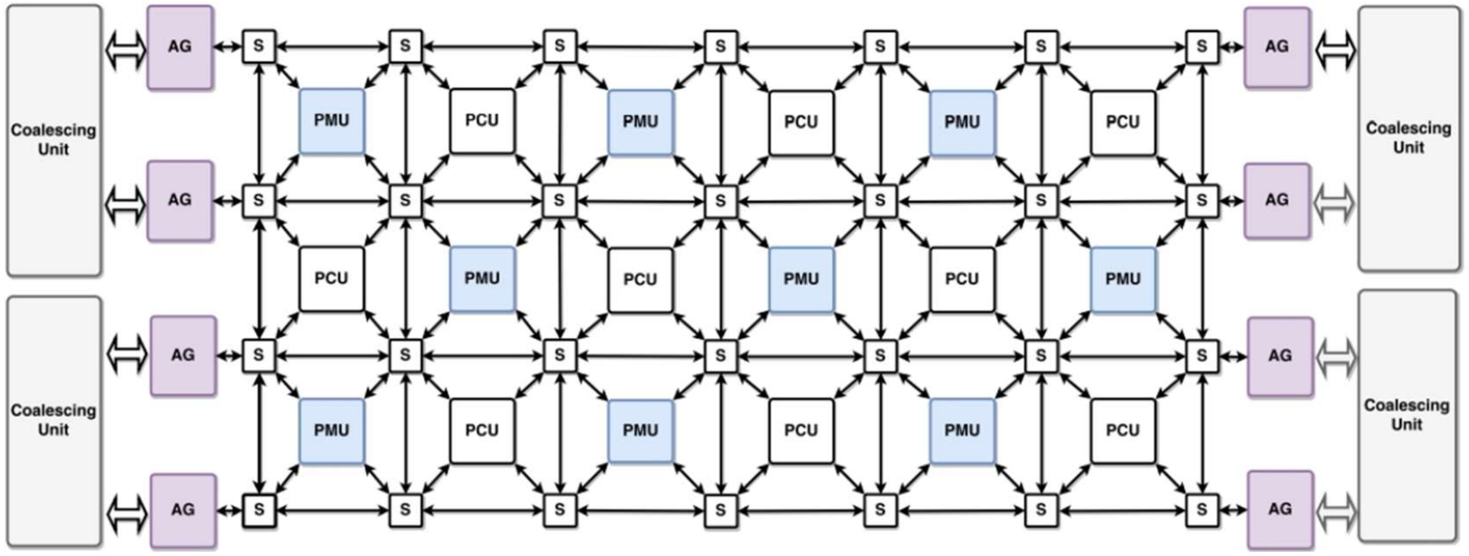
SambaNova Reconfigurable Dataflow Unit (RDU)

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

- Calculates address 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

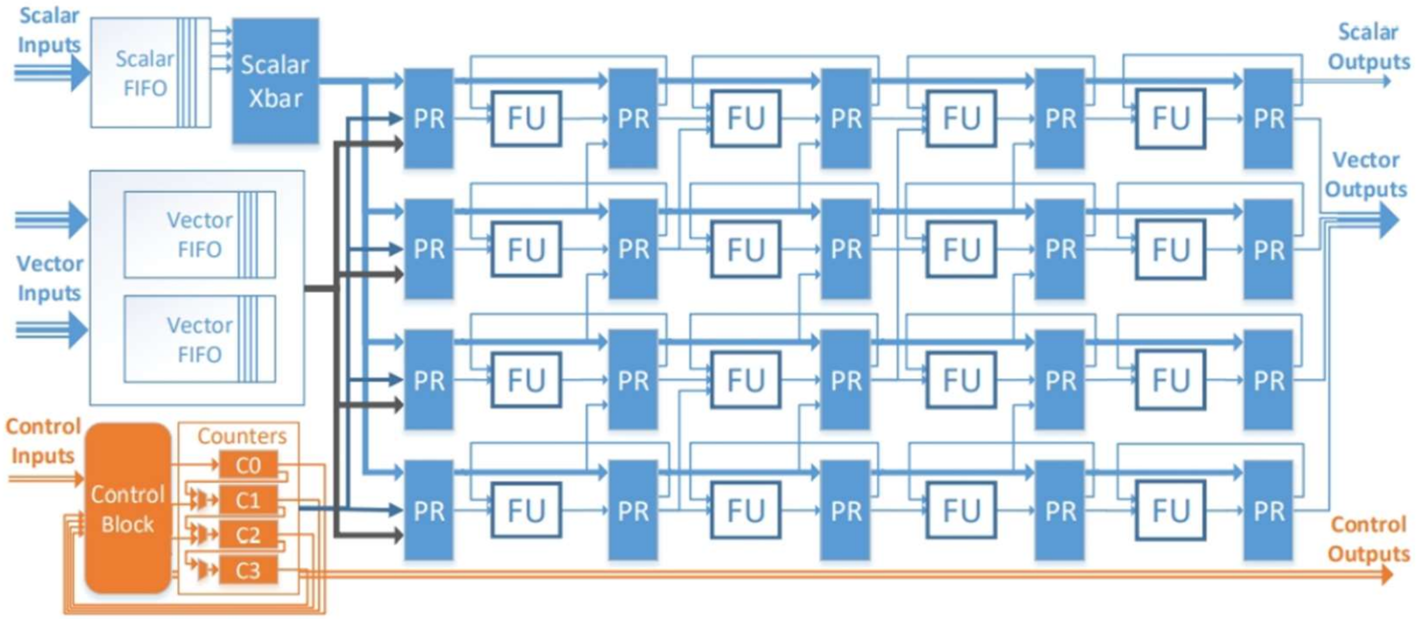


Raghu, ISCA 2017

Plasticine PCU Architecture

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

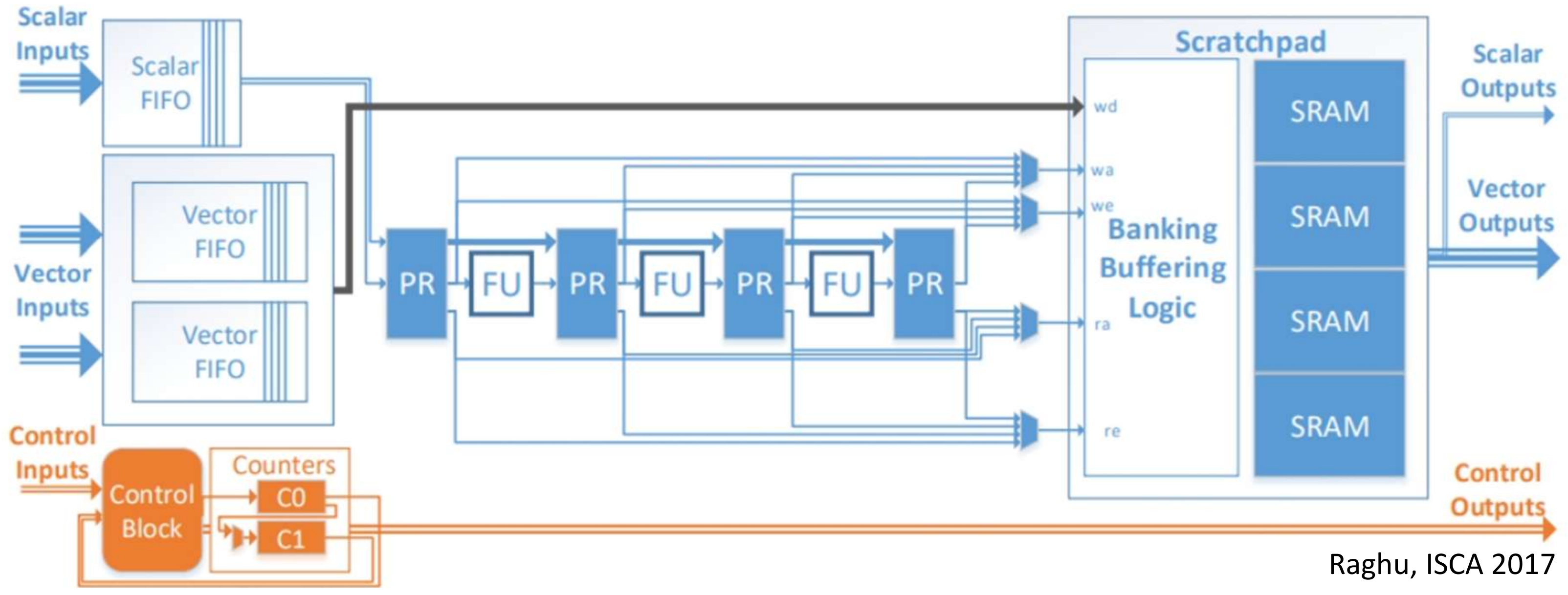
- **Pattern Compute Unit (PCU)**
  - Each stage's SIMD lane contains a FU and associated pipeline register (PR)



Raghu, ISCA 2017

Plasticine PMU Architecture

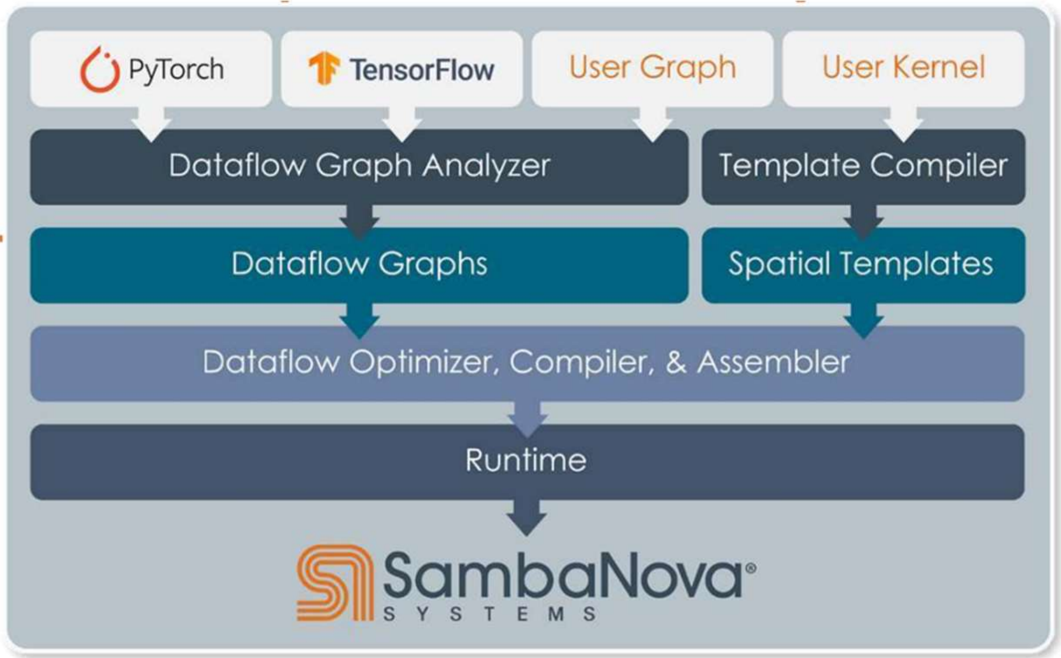
- **Pattern Memory Unit (PMU)**
  - Contains a scratchpad memory and address calculation
  - Calculates address only needs simple scalar math
  - Has simpler FUs than ones in PCUs



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

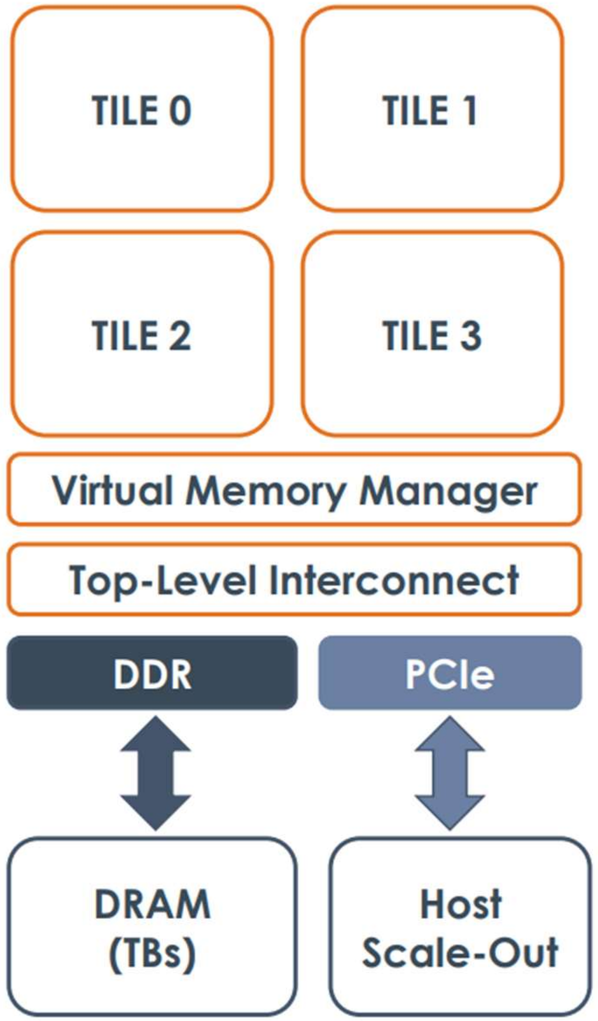


Dataflow Exploits Data Locality / Parallelism

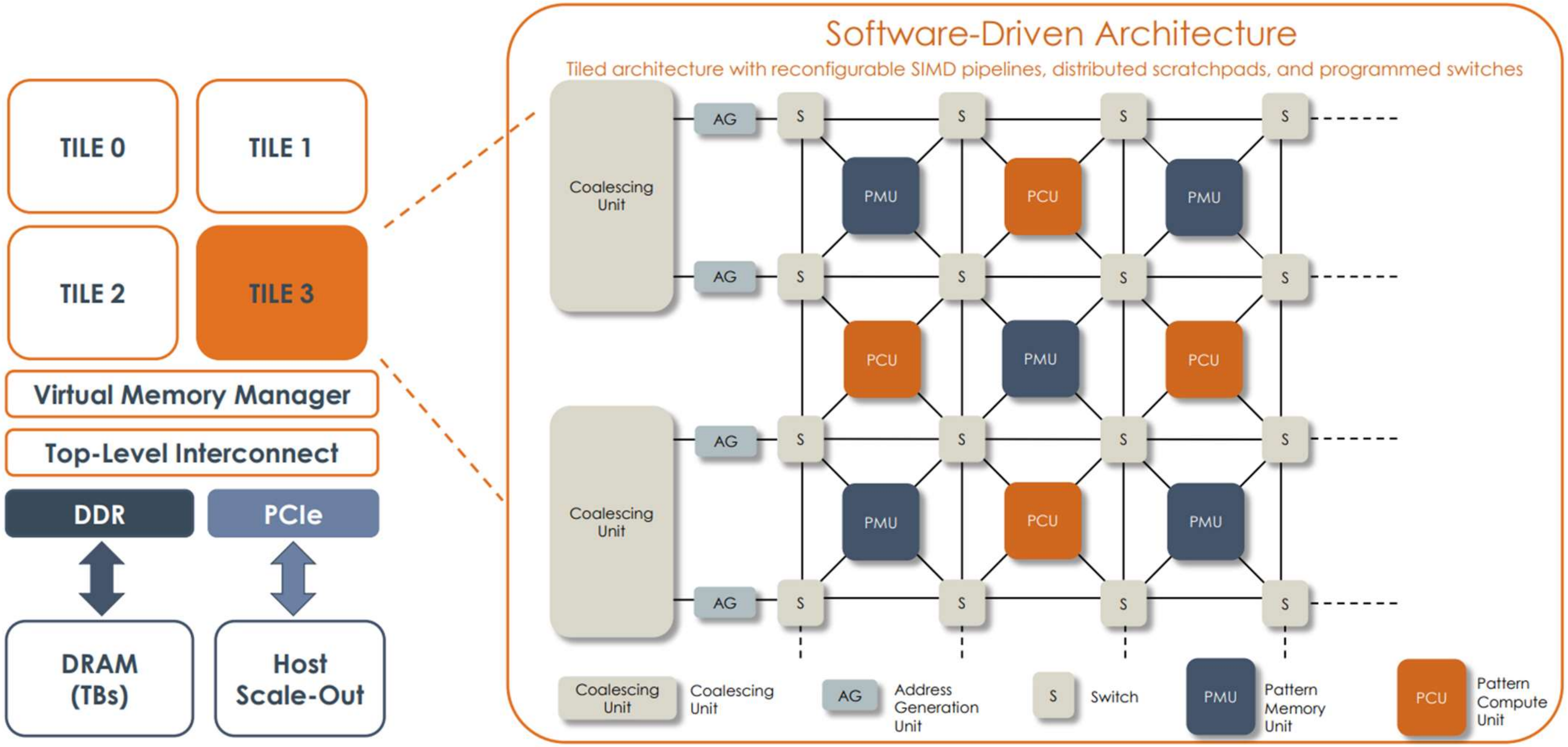
- **Software-hardware co-design architecture**
  - Dataflow captures data locality and parallelism
  - Flexible time and space scheduling to achieve higher utilization
  - Flexible memory system and interconnect to sustain high compute throughput
  - Custom dataflow pipeline

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



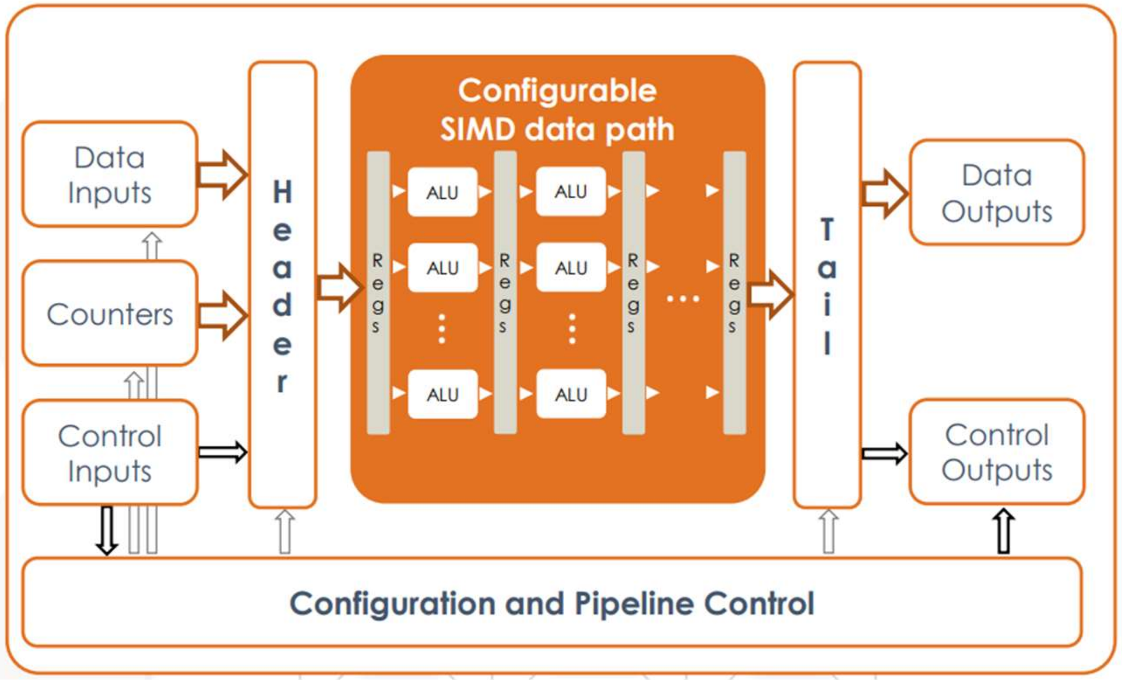
RDU Tile



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
AG
Address Generation Unit
S
Switch
PMU
Pattern Memory Unit
PCU
Pattern Compute Unit

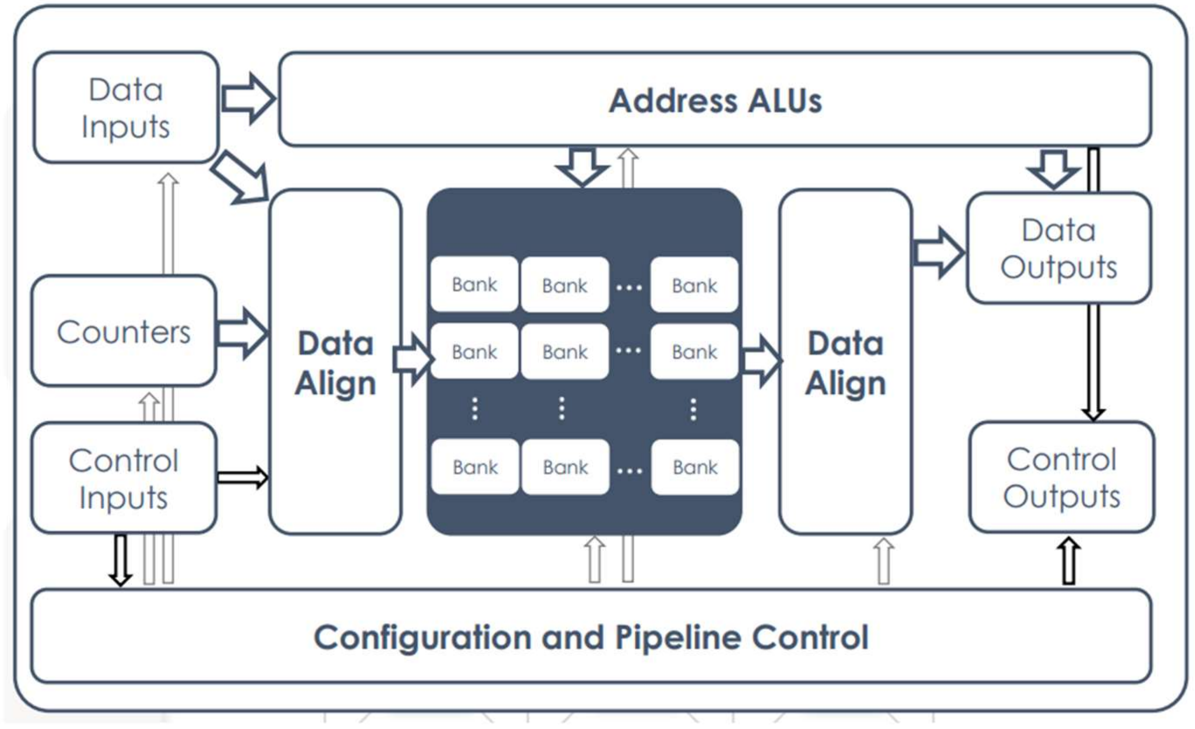
Pattern Compute Unit (PCU)

- **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



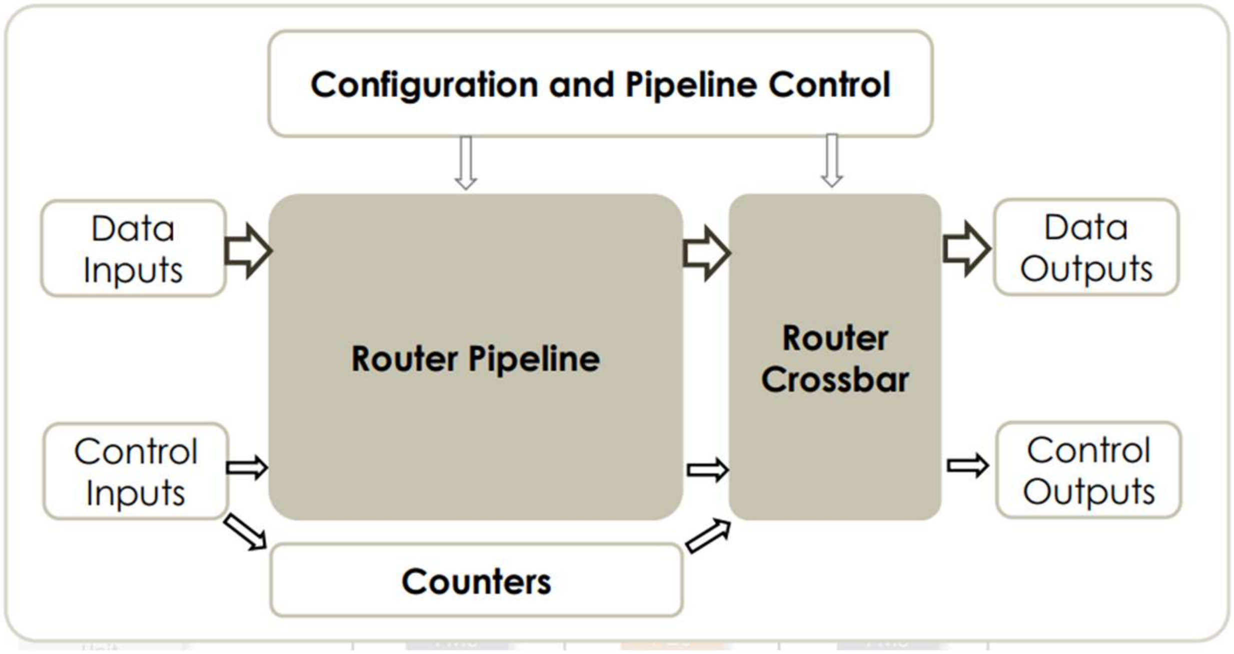
Pattern Memory Unit (PMU)

- **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



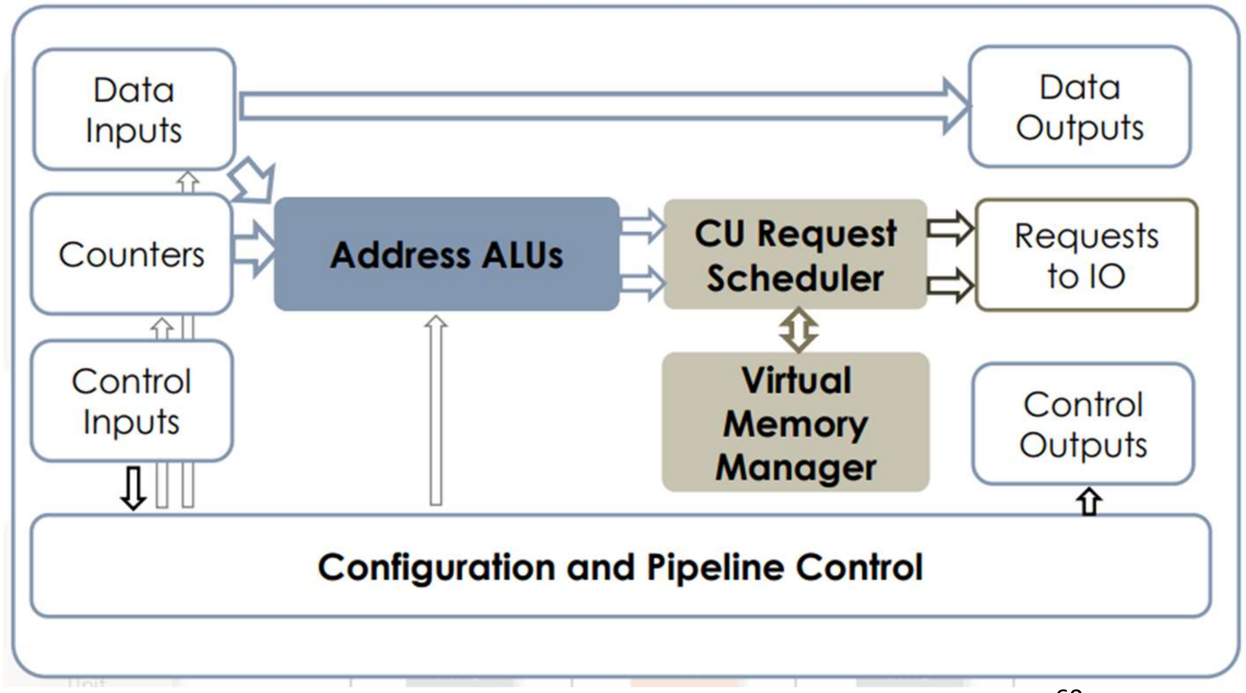
Switch and On-chip Interconnect

- **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



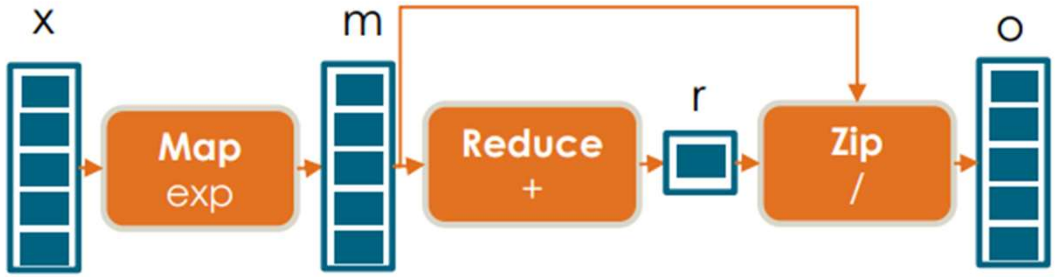
Interface to I/O Subsystem

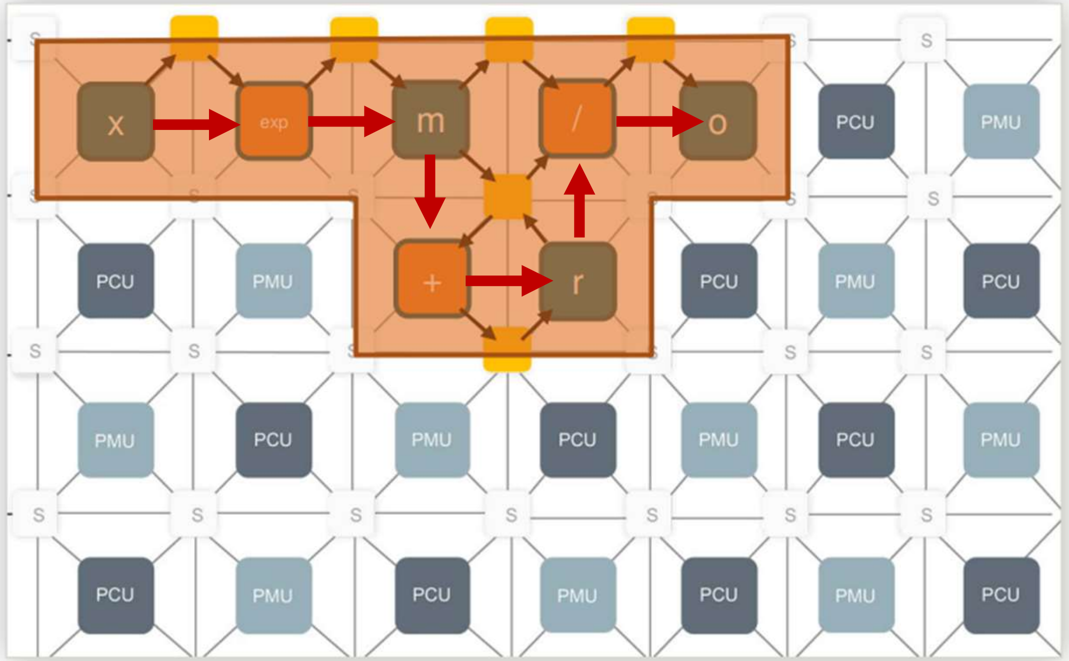
- **Address ALUs**
  - Address calculation for arbitrarily complex accesses
- **Coalescing Units**
  - Enable transparent access to memories across RDUs and host memory
- **Address space manager**
  - Programmable, variable length segments



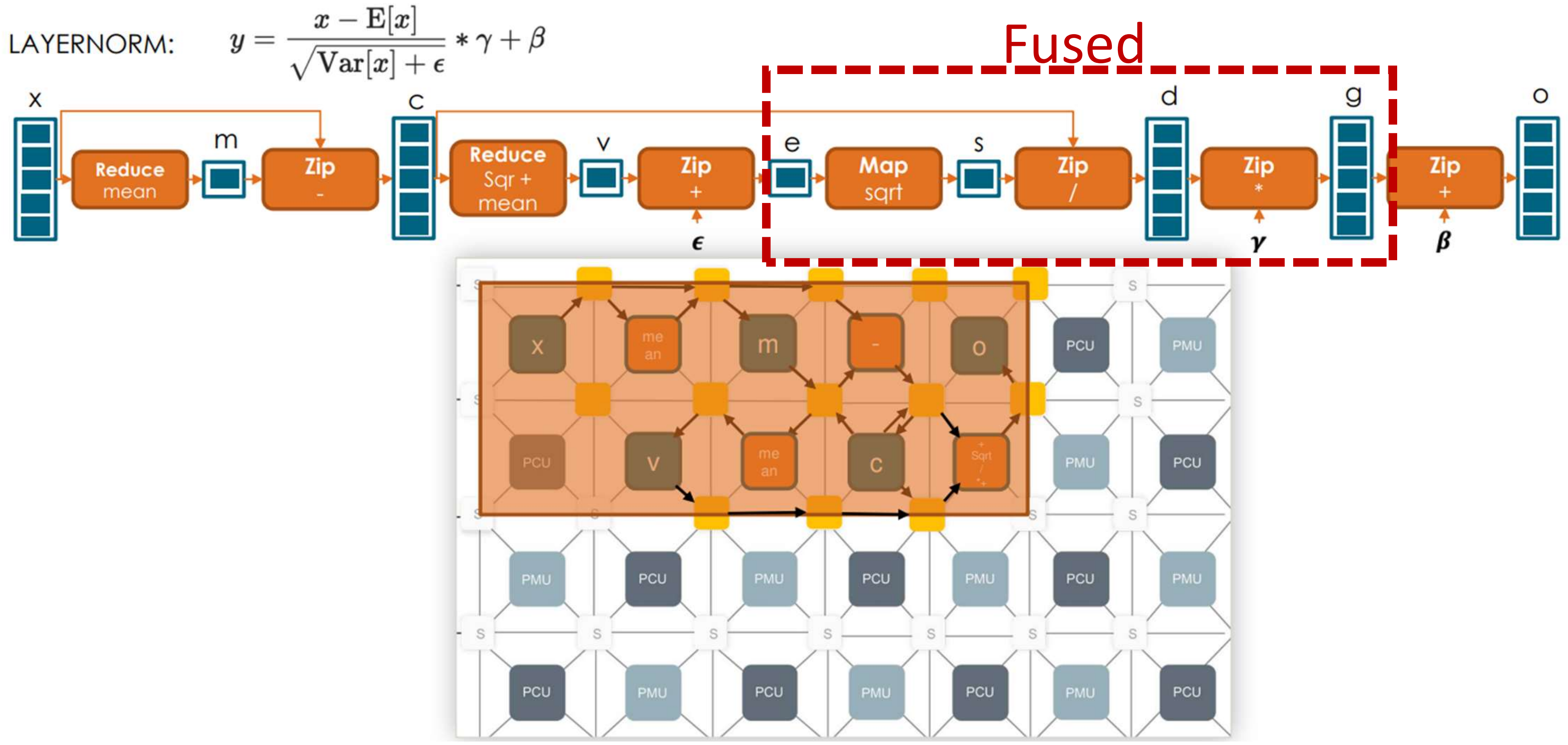
Operator Mapping (Softmax)

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



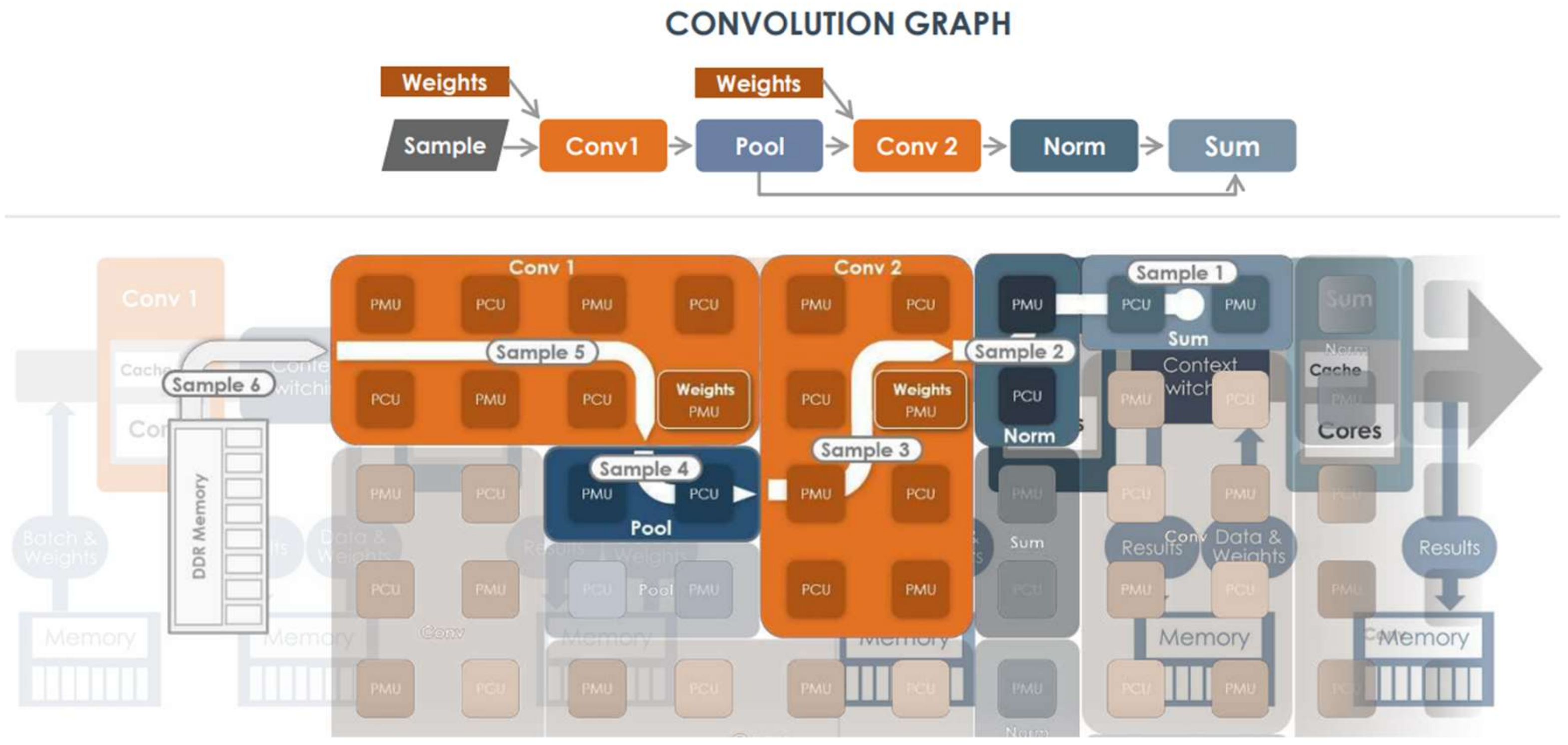


Pipelined in Space + Fused



Spatial Dataflow within an RDU

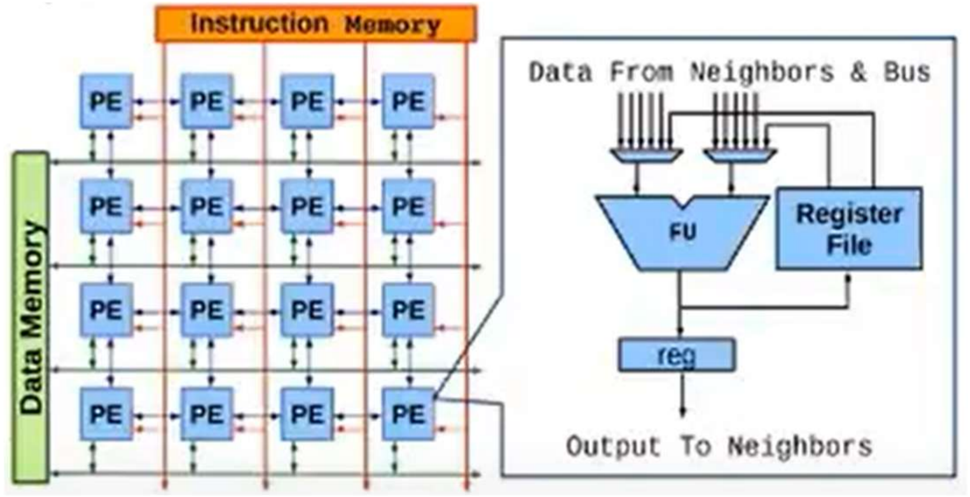
- **The dataflow removes**
  - Memory traffic and host communication overhead



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)



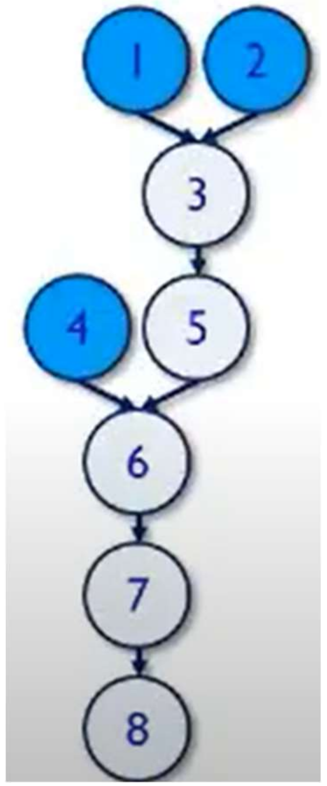
Key features of CGRA accelerators

- **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

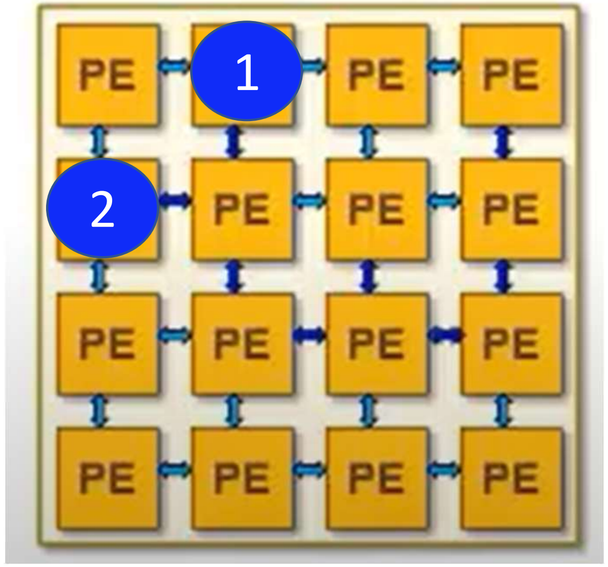
Loop:
t1 = (a[i]+b[i]-k)*c[i]
d[i] = ~t1 & 0xFFFF

Data dependency graph



Mapping data dependency graph to CGRA

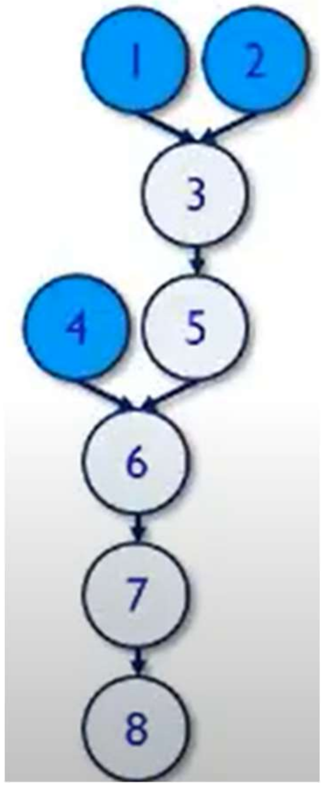
Execution time: 1



Loop execution on the CGRA

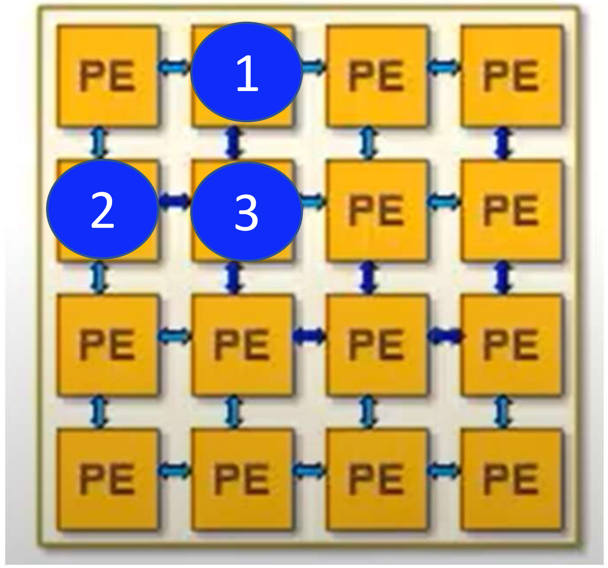
Loop:
t1 = (a[i]+b[i]-k)*c[i]
d[i] = ~t1 & 0xFFFF

Data dependency graph



Mapping data dependency graph to CGRA

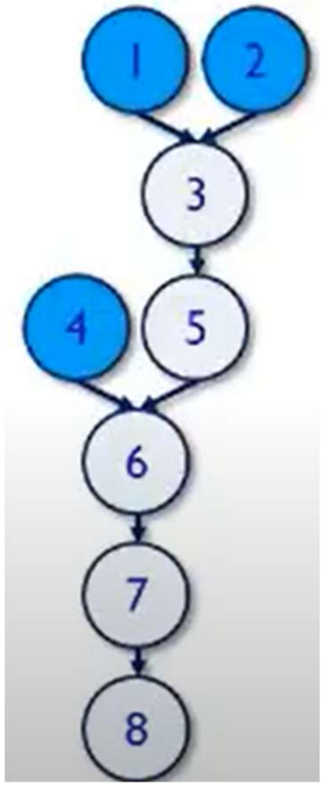
Execution time: 2



Loop execution on the CGRA

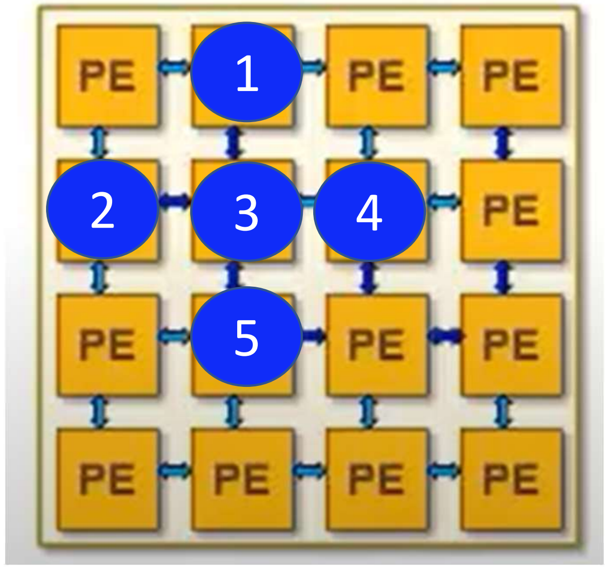
Loop:
t1 = (a[i]+b[i]-k)*c[i]
d[i] = ~t1 & 0xFFFF

Data dependency graph



Mapping data dependency graph to CGRA

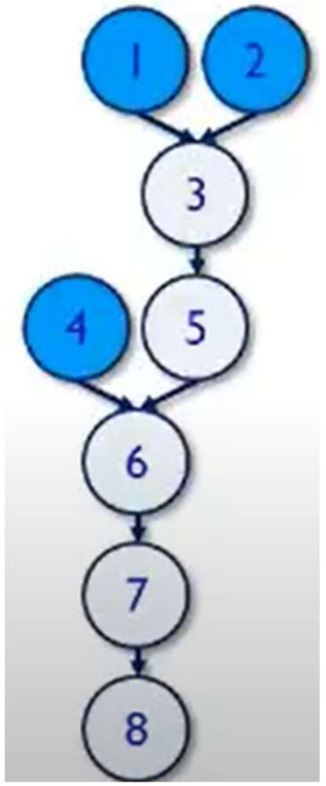
Execution time: 3



Loop execution on the CGRA

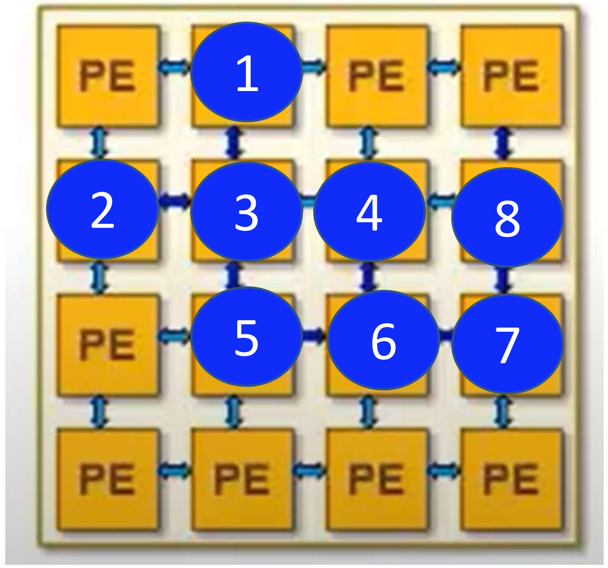
Loop:
t1 = (a[i]+b[i]-k)*c[i]
d[i] = ~t1 & 0xFFFF

Data dependency graph



Mapping data dependency graph to CGRA

Execution time: 6



Takeaway Questions

- What are 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

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