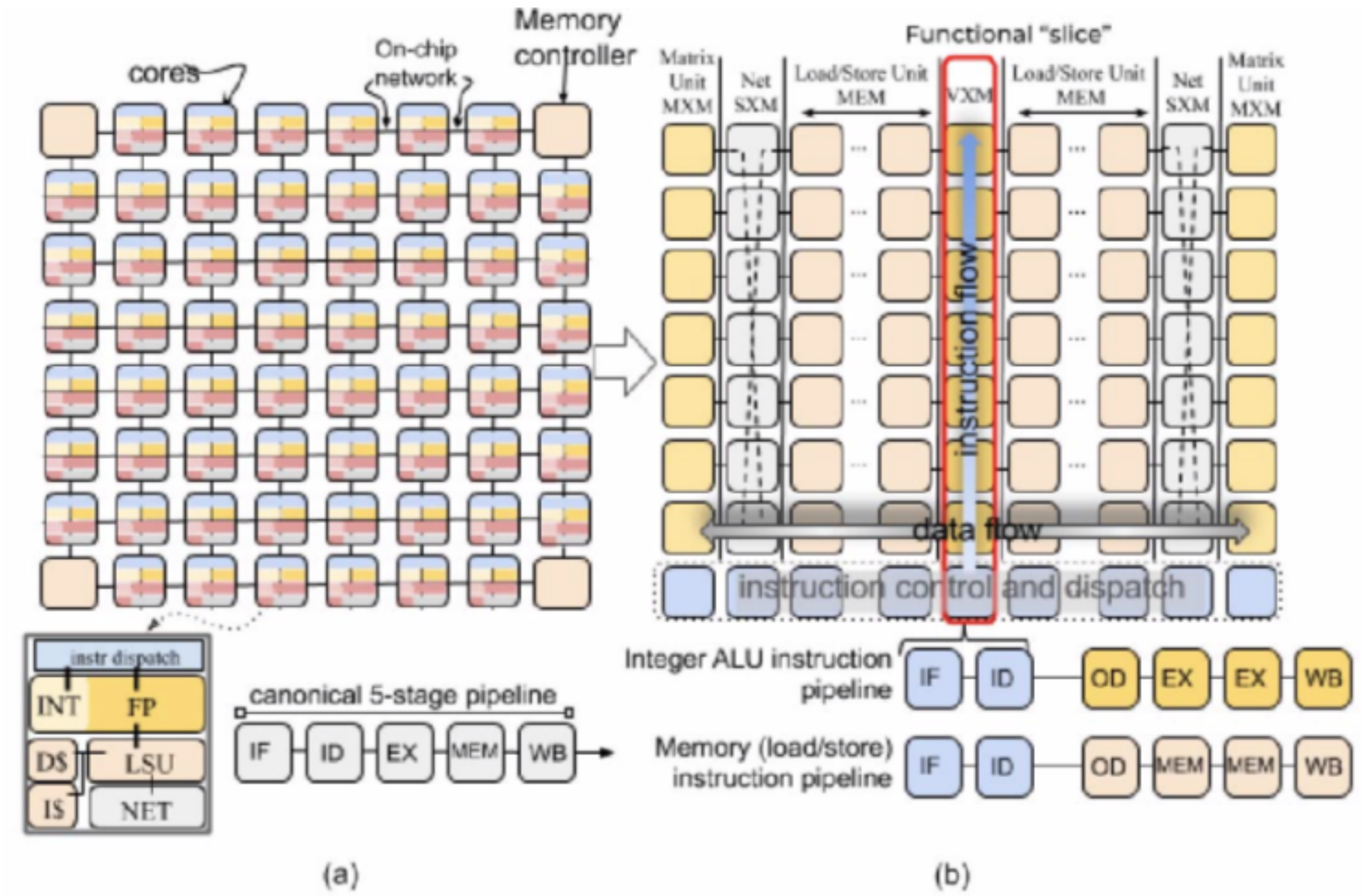
Dataflow Chips and a Compiler

Scaling Deep Learning Computation over the Inter-Core Connected Intelligence Processor with T10

管仁阳 2024.10.18

Groq



Data flow horizontally

Instruction flow vertically

230MB SRAM in total

GraphCore IPU

IPU-Tiles™

1472 independent IPU-Tiles™ each with an IPU-Core™ and In-Processor-Memory™

IPU-Core™

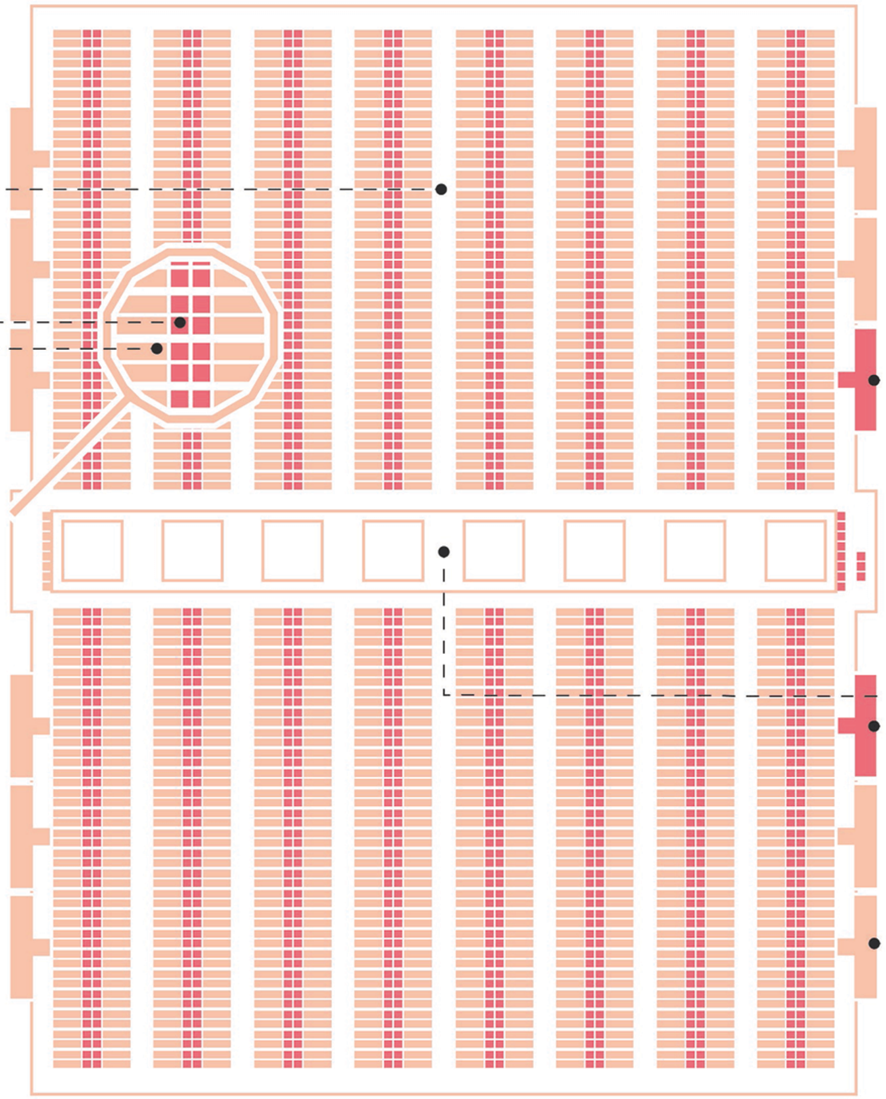
1472 independent IPU-Core™

8832 independent program threads executing in parallel

In-Processor-Memory™

900MB In-Processor-Memory™ per IPU

47.5TB/s memory bandwidth per IPU



IPU-Exchange™

8 TB/s all to all IPU-Exchange™
Non-blocking, any communication pattern

PCIe

PCI Gen4 x16
64 GB/s bidirectional bandwidth to host

IPU-Links™

10 x IPU-Links,
320GB/s chip to chip bandwidth

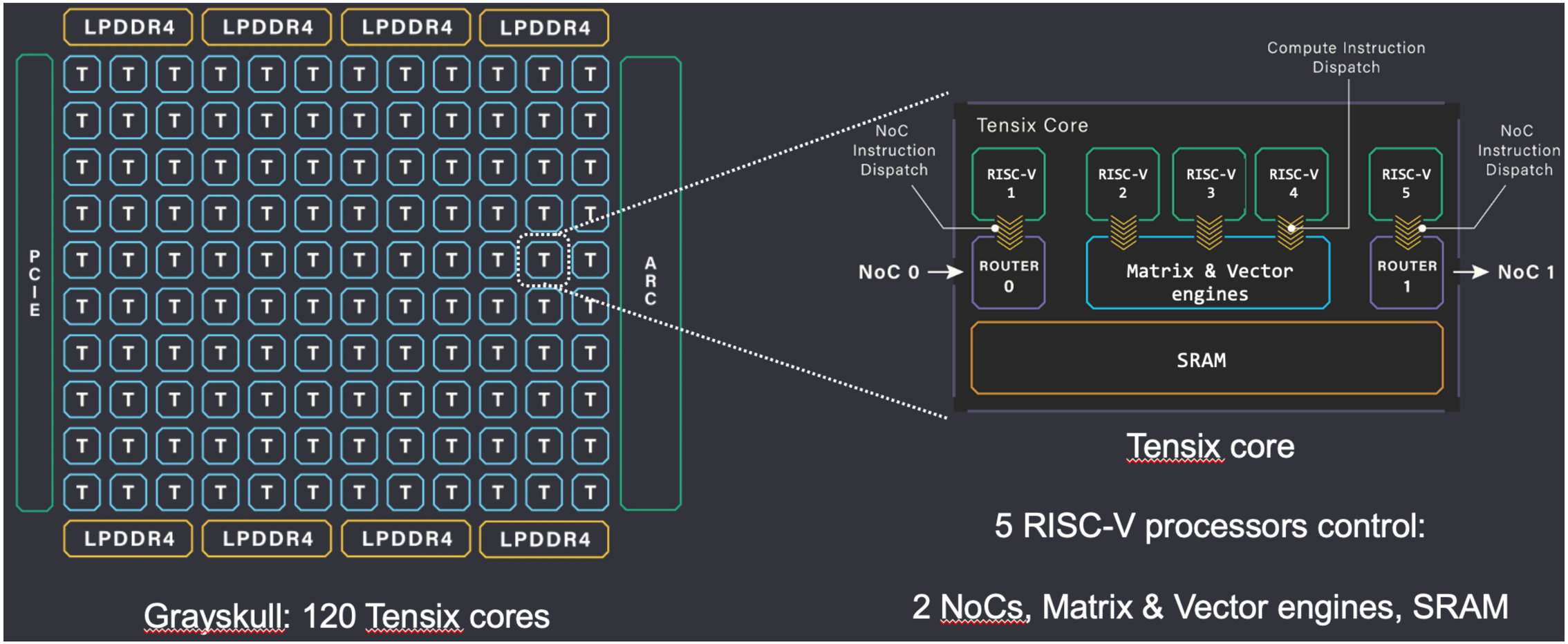
BSP communication fashion

All-to-All typology

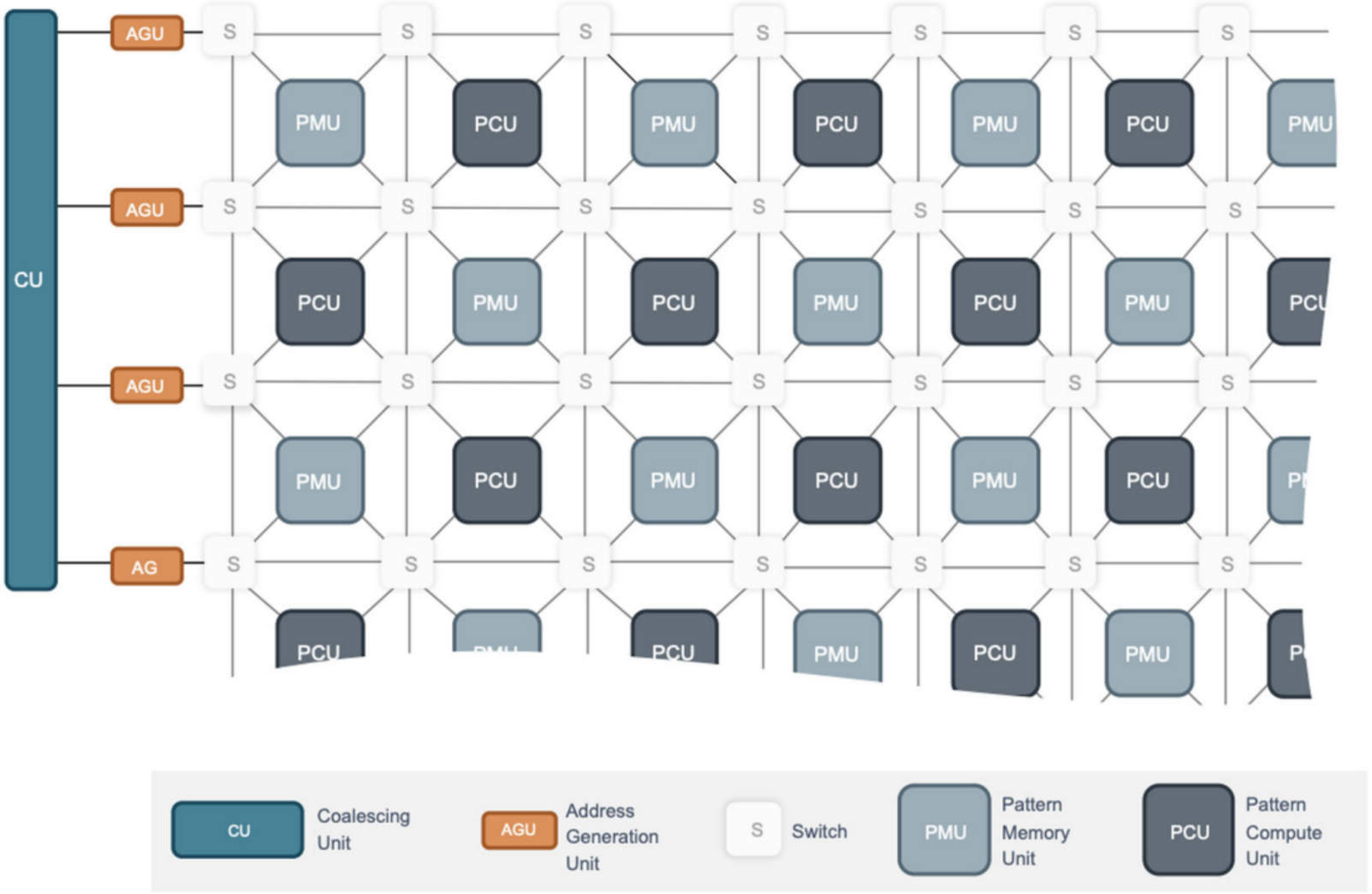
900 MB SRAM in total

Tenstorrent

192 MB SRAM in total



SambaNova

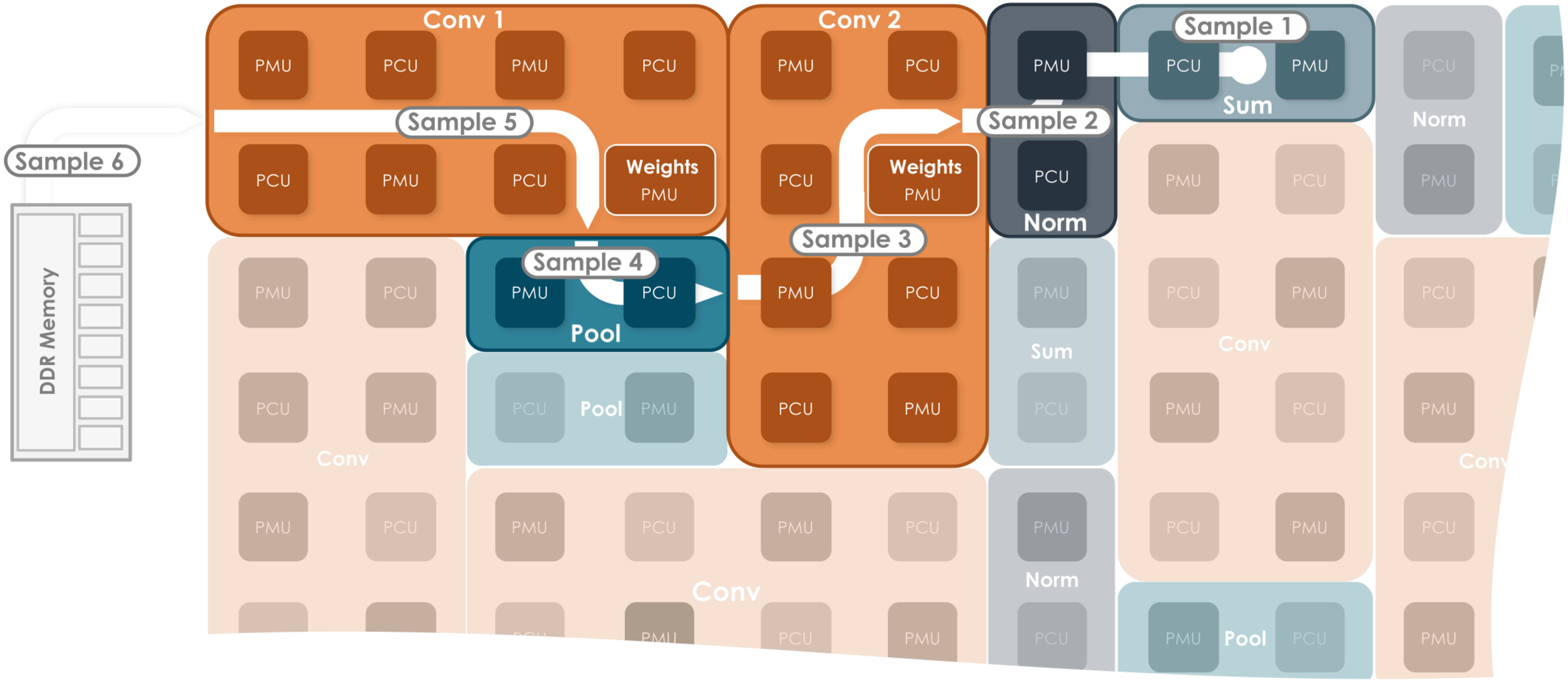


Each core is attached a SRAM

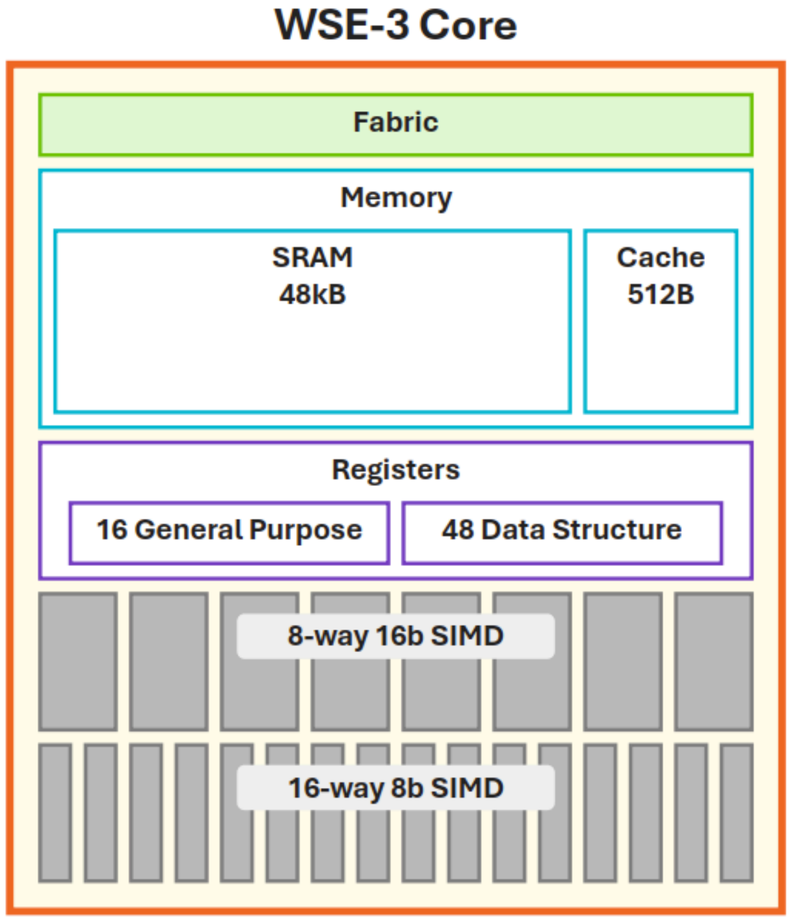
520MB SRAM in total

Mesh typology

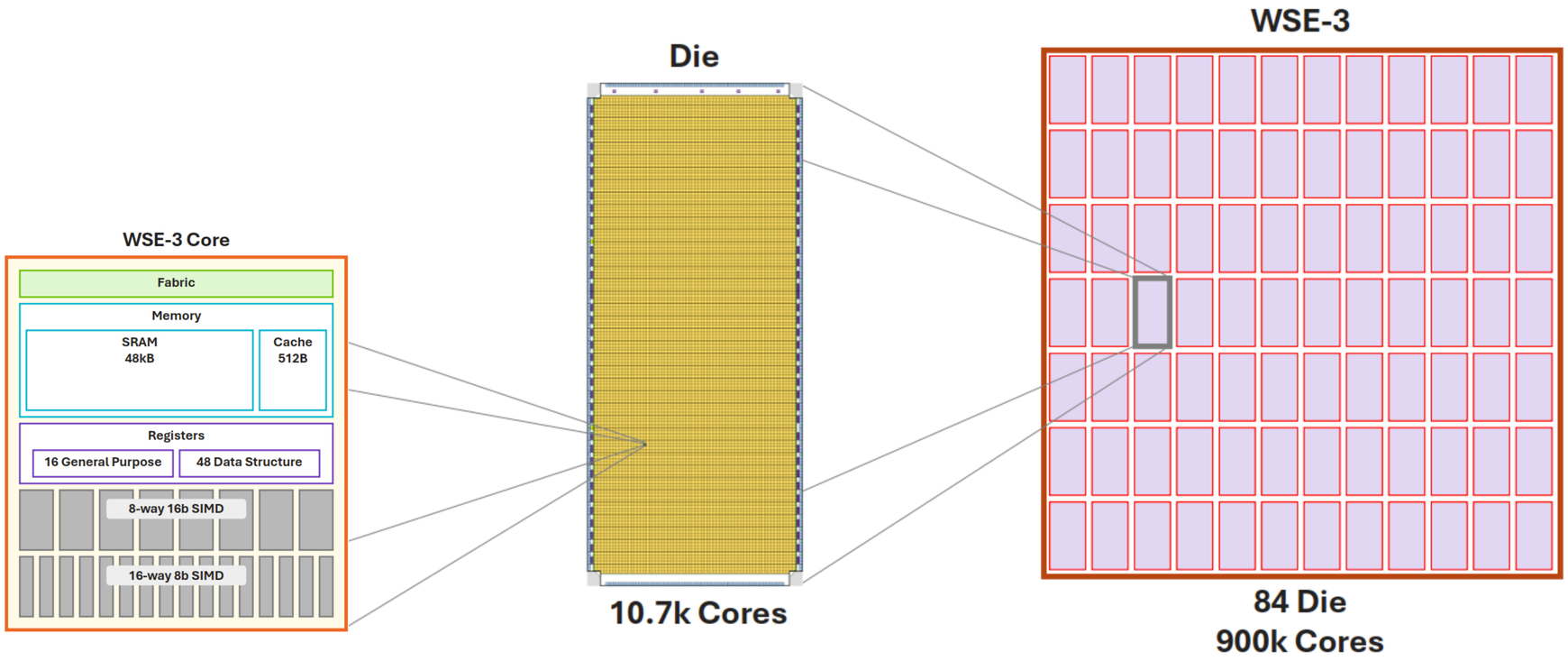
SambaNova


Sample 6
DDR Memory
Conv 1
PMU
PCU
PMU
PCU
Sample 5
PCU
PMU
PCU
Weights
PMU
Sample 4
PMU
PCU
Pool
Conv
Conv 2
Weights
PMU
Sample 3
Sample 2
PMU
PCU
Norm
Sample 1
PCU
PMU
Sum
Norm
Sum
Conv
Norm
Pool
Conv

Cerebras

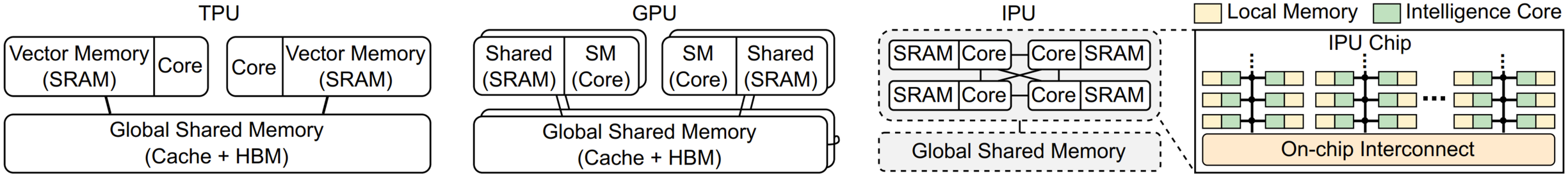

WSE-3 Core
Fabric
Memory
SRAM
48kB
Cache
512B
Registers
16 General Purpose
48 Data Structure
8-way 16b SIMD
16-way 8b SIMD

Cerebras


WSE-3 Core
Fabric
Memory
SRAM
48kB
Cache
512B
Registers
16 General Purpose
48 Data Structure
8-way 16b SIMD
16-way 8b SIMD
Die
10.7k Cores
WSE-3
84 Die
900k Cores

Trend of Many-Core architecture

T10: An AI Compiler targeting fully-connected On-Chip Cores



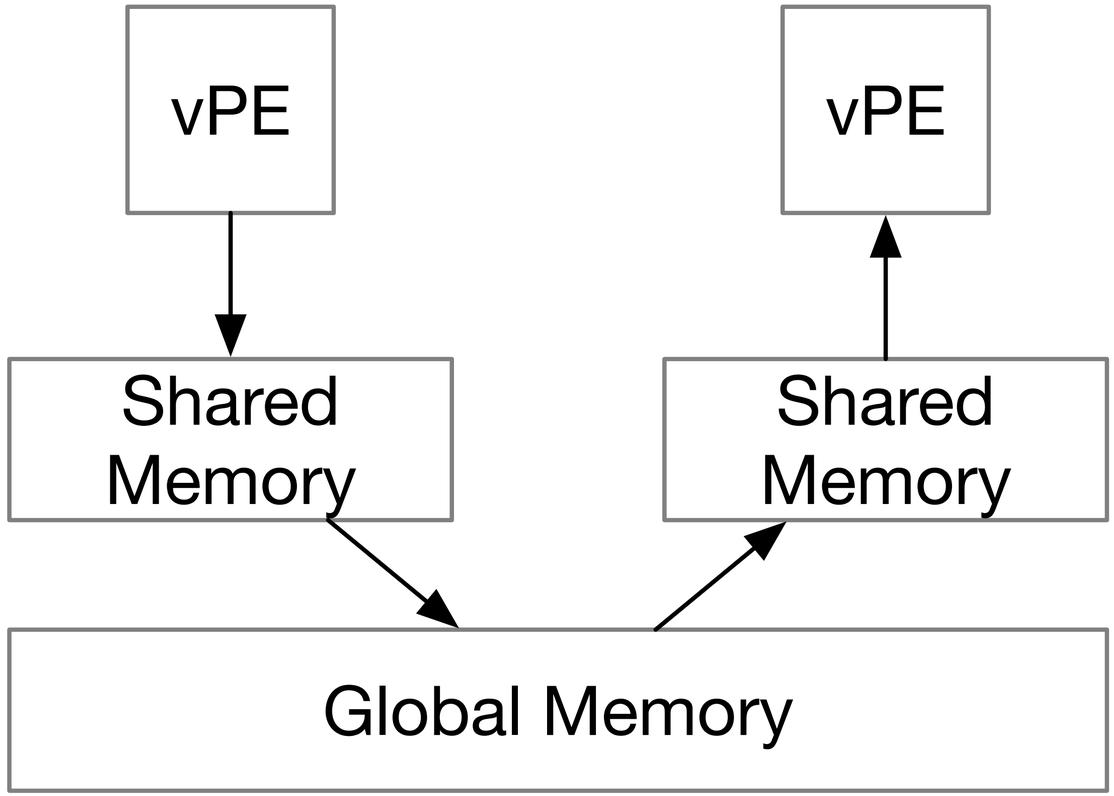
	A100 GPU [32]	IPU MK2 [24]
Local Cache (total)	20.25MB	896MB
Global Cache	40MB	N/A
Off-chip B/W	2000GB/s	8GB/s
Inter-core B/W	N/A	6GB/s per link
Number of Cores	108	1472
Total FP16 FLOPS	312TFLOPS	250TFLOPS

- All-to-all bandwidth of IPU: 8TB/s
- HBM Bandwidth: 1.94TB/s
- Pros: Greater total bandwidth.
- Cons: Complex inter-core collaboration

A new trade-off posed by inter-core connection

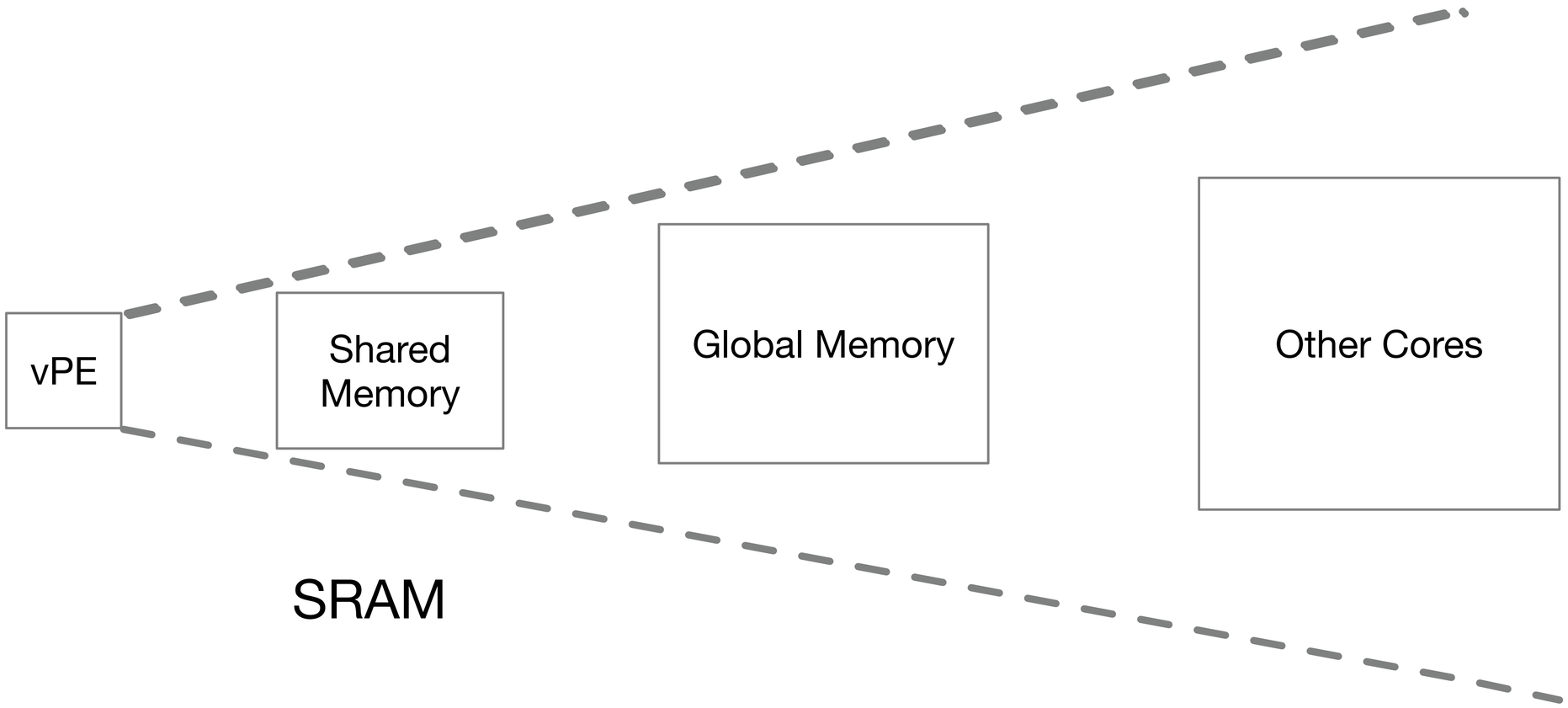
Disadvantage of existing compiler on such architecture

- Load-Compute-Store paradigm
  - Core loads/stores data from/to its own memory
  - Inter-Core communicate though shared memory



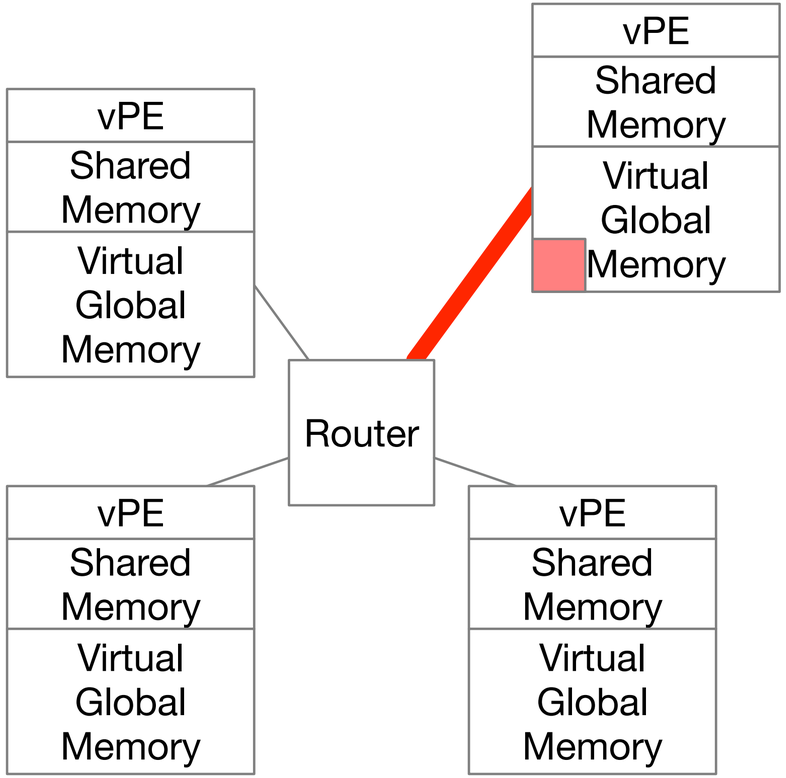
A new trade-off posed by inter-core connection

From the view of vPE

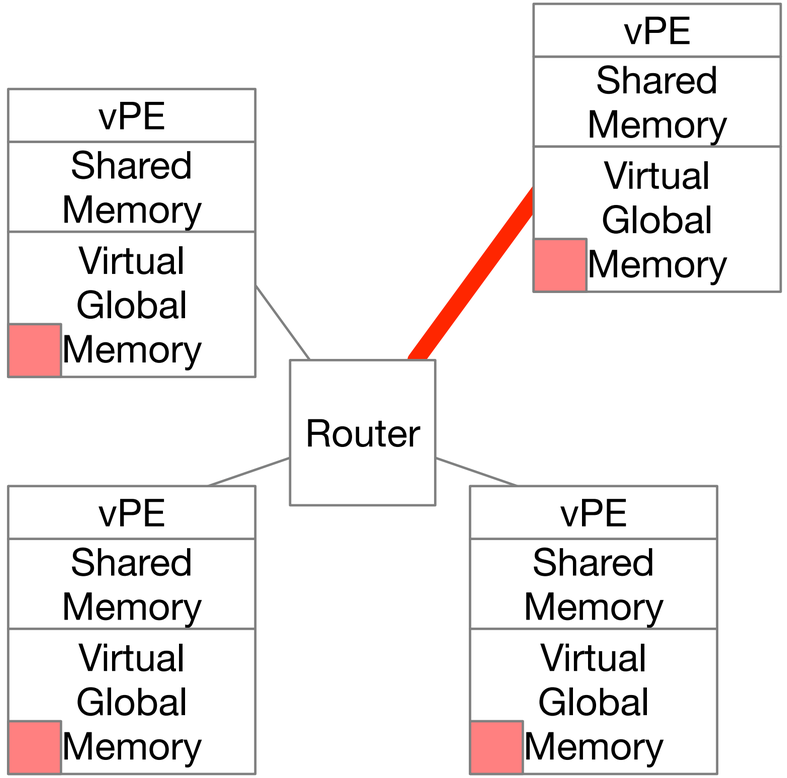


A new trade-off posed by inter-core connection

Disadvantage of existing compiler on such architecture



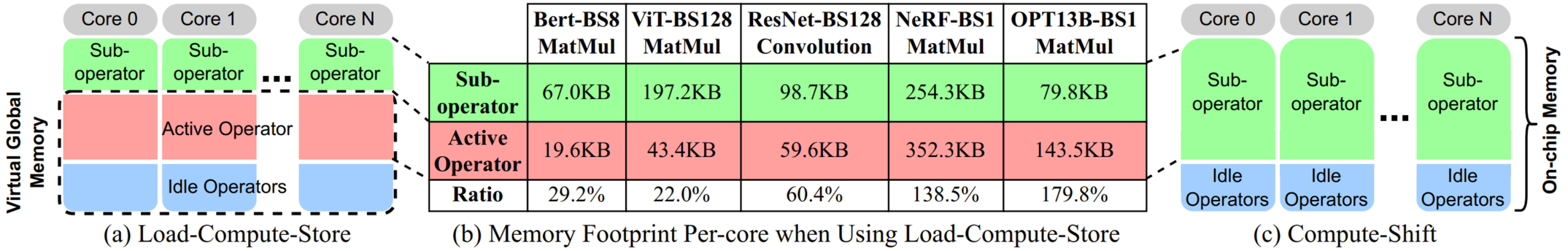
Imbalance causing inefficiency



Duplication wasting memory

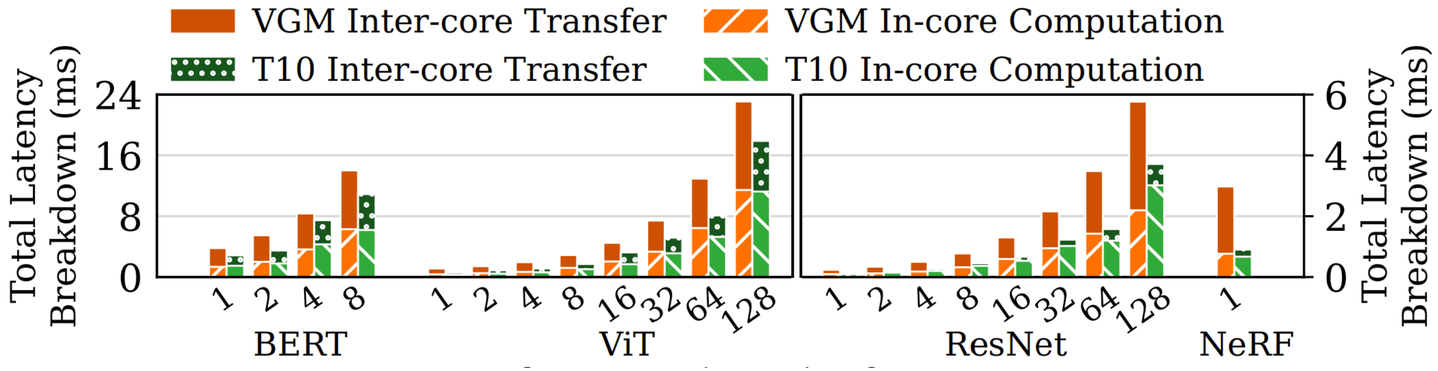
A new trade-off posed by inter-core connection

Disadvantage of existing compiler on such architecture



	Bert-BS8 MatMul	ViT-BS128 MatMul	ResNet-BS128 Convolution	NeRF-BS1 MatMul	OPT13B-BS1 MatMul
Sub-operator	67.0KB	197.2KB	98.7KB	254.3KB	79.8KB
Active Operator	19.6KB	43.4KB	59.6KB	352.3KB	143.5KB
Ratio	29.2%	22.0%	60.4%	138.5%	179.8%

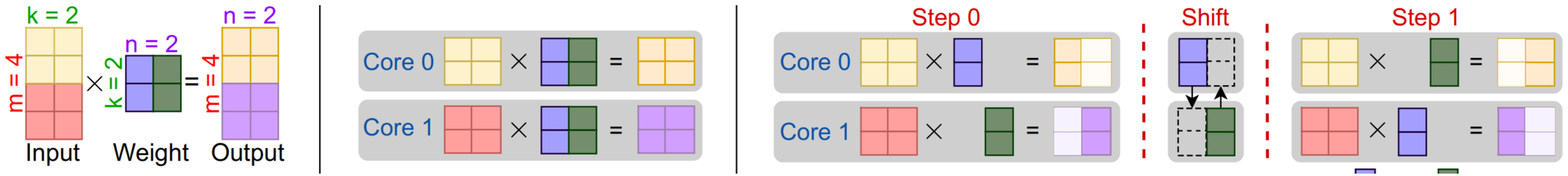
(a) Load-Compute-Store (b) Memory Footprint Per-core when Using Load-Compute-Store (c) Compute-Shift



- Emerging inter-core connected AI chips fail to compete with the traditional global shared memory-based AI accelerators due to the inefficient compiler support

A new trade-off posed by inter-core connection

Inter-Core communication and Memory Footprint



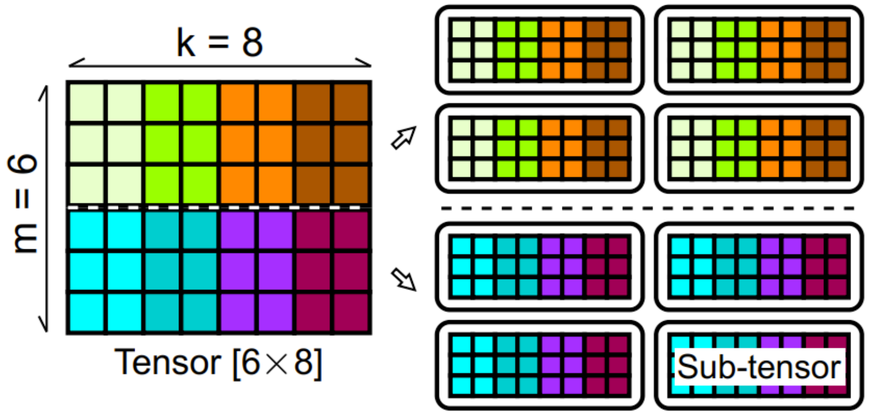
- Compute Shift Paradigm
  - Each core independently computes a sub-task with data received from its upstream neighbors and shifts the data to its downstream
  - At a time, only one core hold the sub-tensor partition, which saves memory footprint

RotatingTensor Abstraction

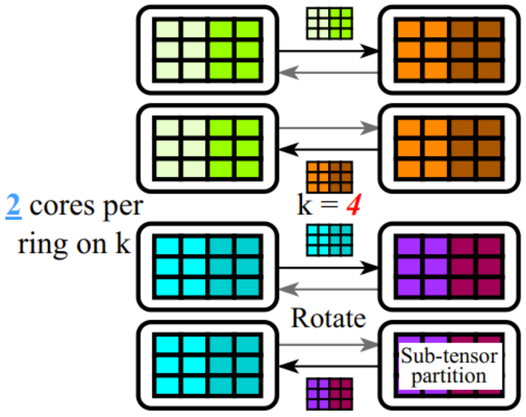
Each ring for each *duplicated* sub-tensor

```
class RotatingTensor {
        vector < size_t > shape ;
        DataType          type ;
        vector < size_t > spatial_partition_factor ;    // f_s
        vector < size_t > temporal_partition_factor ;   // f_t
        vector < size_t > rotating_pace ;               // rp
};
```

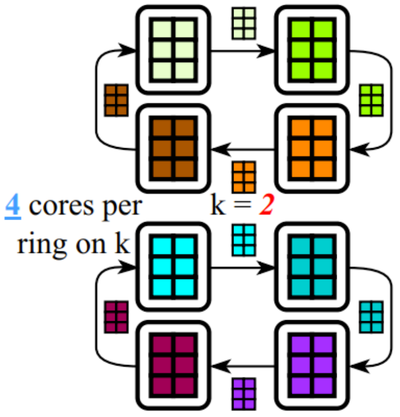
Symbol	Name	Description
f_s^X	Spatial Partition Factor	Spatially partitions a tensor X into sub-tensors.
f_t^X	Temporal Partition Factor	Temporally partitions a sub-tensor of X into sub-tensor partitions.
rp	Rotating Pace	Specifies how sub-tensor partitions are shifted among cores.
F_{op}	Operator Partition Factor	Spatially partitions an entire operator into sub-operators.



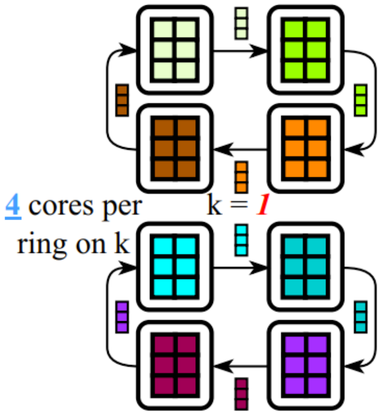
(a) Spatially partition into 2 sub-tensors with f_s=[2,1] along [m,k] across 8 cores, each sub-tensor is shared among 8/2=4 cores.



(b) Temporally partition with f_t=[1,2] on [m,k], rotate with rp=[0,4] on [m,k].



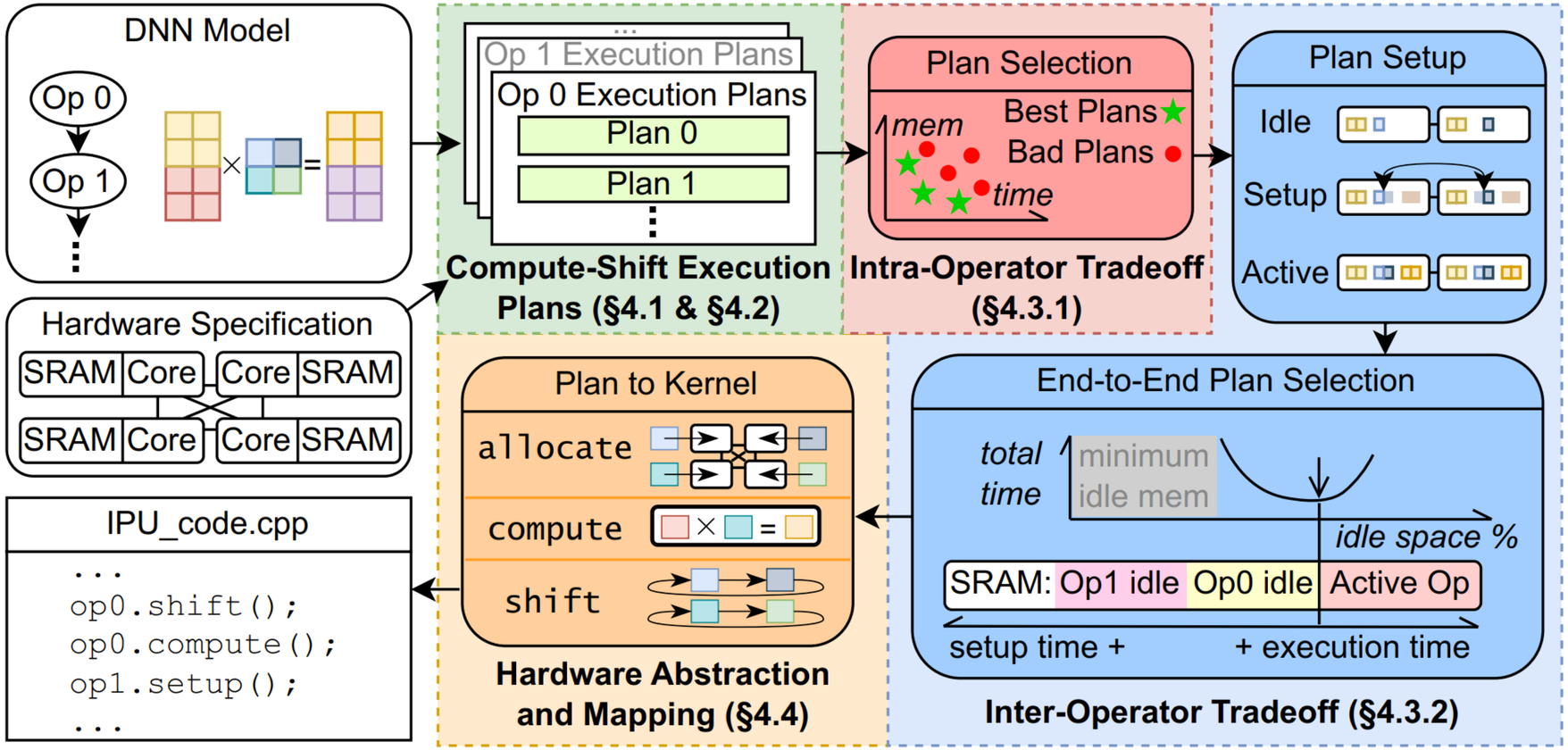
(c) f_t=[1,4] on [m,k], rp=[0,2] on [m,k].



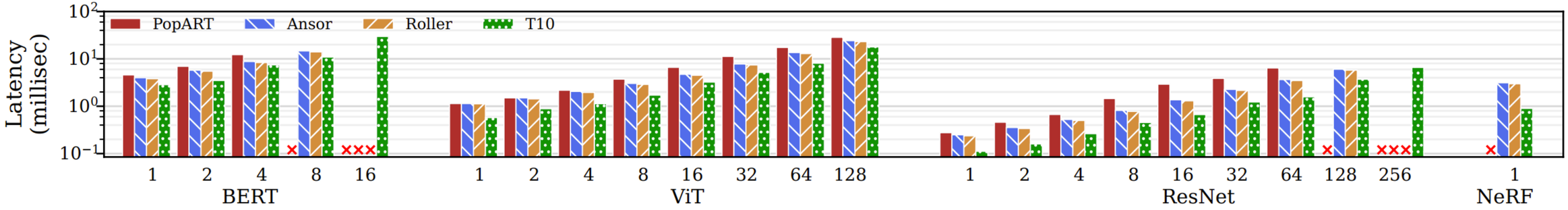
(d) f_t=[1,4] on [m,k], rp=[0,1] on [m,k].

T10 System Overview

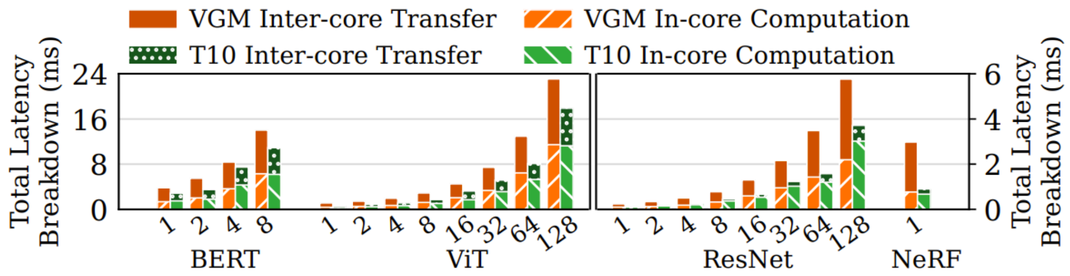
- Enumerate rTensors
- Explore search space using rule-based filtering and trained cost-model
- Code generation



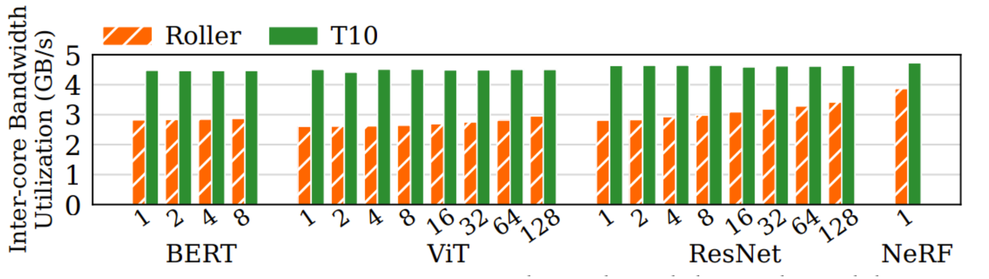
Evaluation



End-to-end evaluation



Compute/Communication breakdown



Aggregated Bandwidth

Comments

- GraphCore IPU uses All-to-ALL typology, which is rare in XPUs, to connect all the cores on chip, so placing any ring of any size to any core yields the same per-hop latency. If using mesh or torus, this solution won’t work.
- GraphCore IPU uses BSP to do communication, where all the cores do computation in one round together and then communicate in the next round. It is actually a __syncthreads in hardware. This is a natural match for the ring structure, because the ring can perform the same number of calculations and communications.
- Matrix multiplication can be regularly separated so that the sun-tensors can form a ring.
- How to compile the kernel model on inter-connected accelerators is worth thinking when any conditions doesn’t hold

Thanks!