FURIOSA

RNGD – Tensor Contraction Processor for Sustainable AI Computing

June Paik, Co-Founder and CEO of FuriosaAI

Key Points

01 Intro of RNGD, Sustainable AI Inference

02 HW Architecture and Chip Design

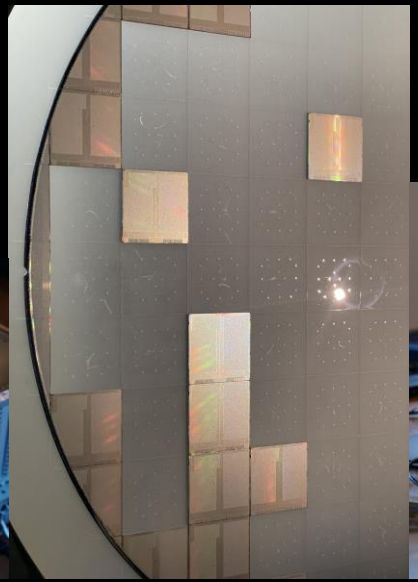
03 SW Full-stack Optimization

2017-2021

FuriosaAI founded &
Launch Gen 1 vision NPU

2021

GPT3 inspired
RNGD



2022

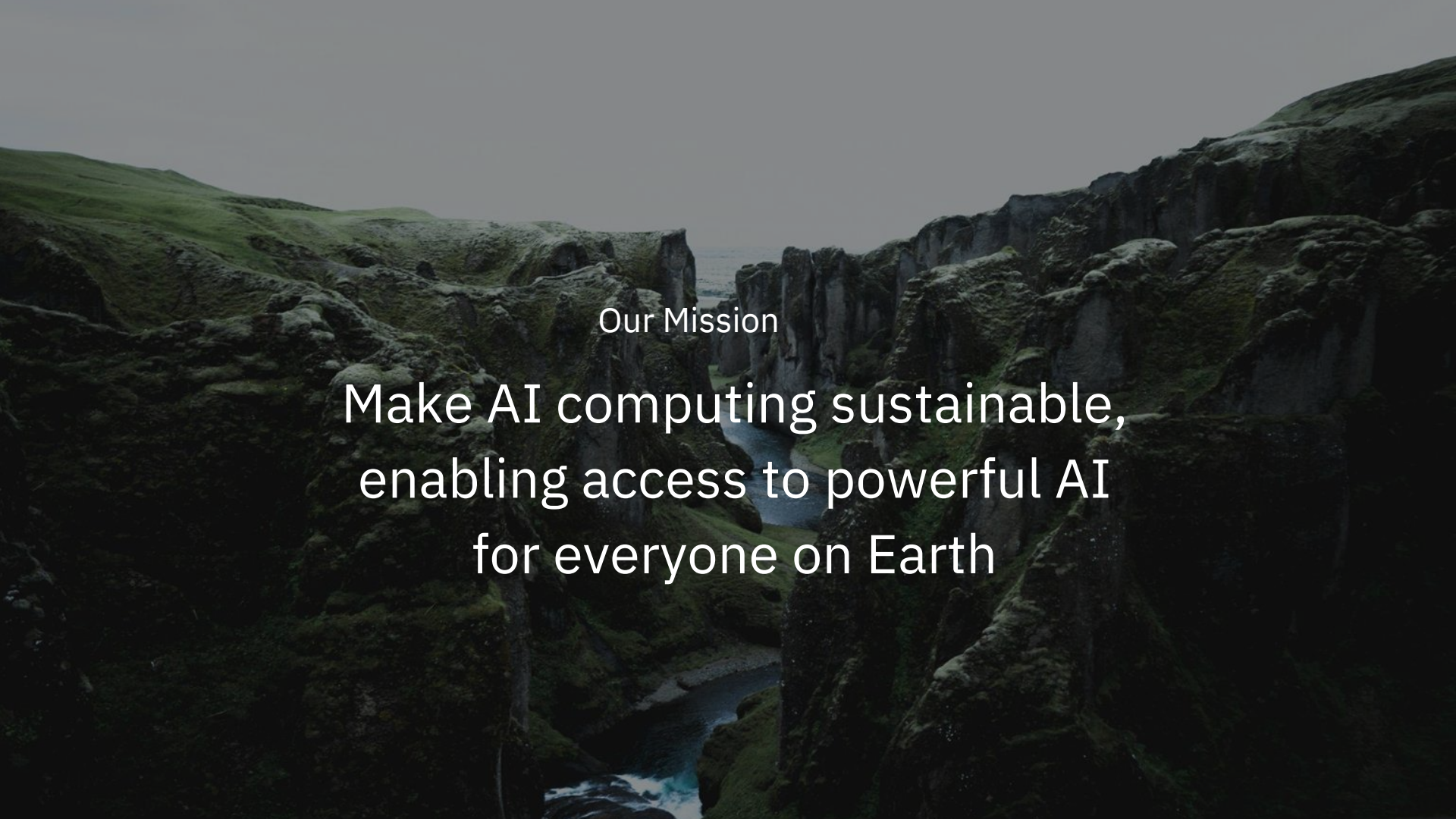
RNGD development
kick off

2024 May

RNGD raw silicon
sample arrival

2024 July

First LLM demo


Our Mission
Make AI computing sustainable,
enabling access to powerful AI
for everyone on Earth

512 TFLOPS
64 TFLOPS (FP8) x 8 processing elements

48 GB
Memory capacity

256 MB SRAM
384 TB/s on-chip bandwidth

1.5 TB/s
Memory bandwidth

150 W TDP
Targeting air-cooled datacenters

2 x HBM3
CoWoS-S

INT8 (512 TOPS), BF16 (256 TFLOPS),
INT4 (1 POPS), FP8 (512 TFLOPS)

PCIe P2P support For LLMs

Features For Cloud

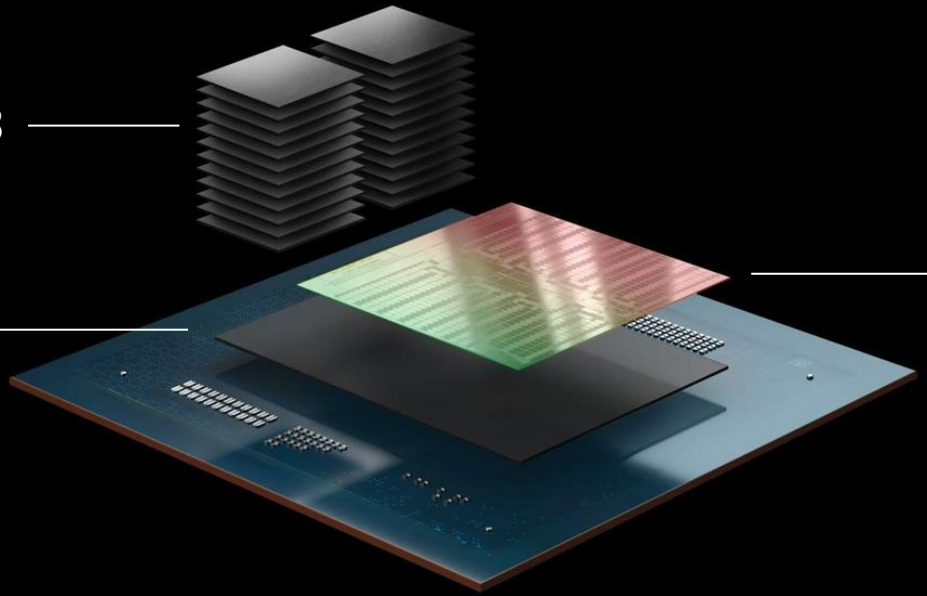
Multiple-instance support
Virtualization
Secure boot & model encryption


Delivers **high-performance LLM** workloads, while keeping the power consumption within the **150 watt range**

2 x HBM3

12-layer (24GB) x 2

Silicon Interposer
CoWoS-S



SoC

TSMC 5nm
Core CLK: 1.0 GHZ
Die Size: 653 mm²
Transistor count: 40 B

Early Performance Numbers: 60% higher perf/watt than L40S

GPT-J 6B MLPerf Benchmark Scenario (99% accuracy)

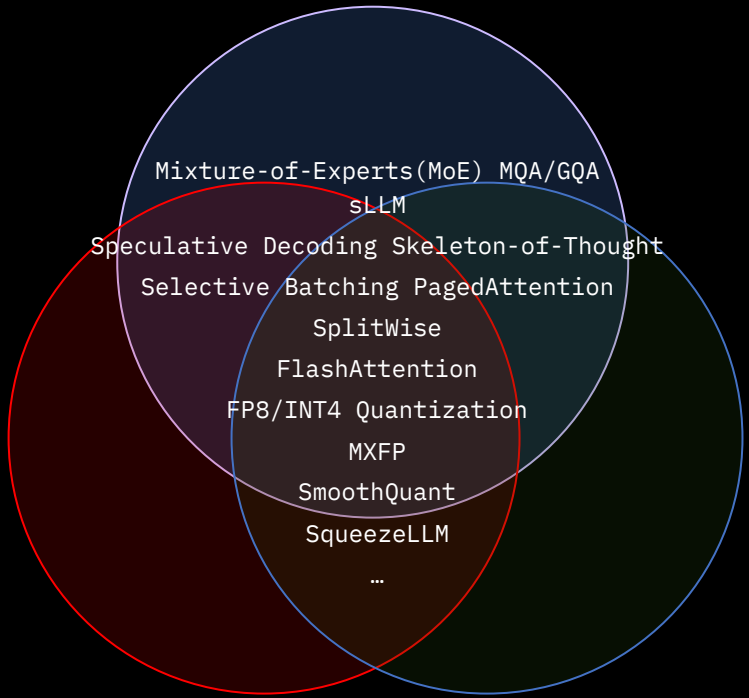
	RNGD	NVIDIA L40S	Intel Gaudi 2	Google TPU v5e
Performance (queries / sec)	11.5 (FP8)	12.3 (FP8)	10.51	2.5
Power (watt)	185	320	Unknown	Unknown
Data source	measured	measured	MLPerf 3.1	MLPerf 4.0

Disclaimer: Unverified by MLPerf

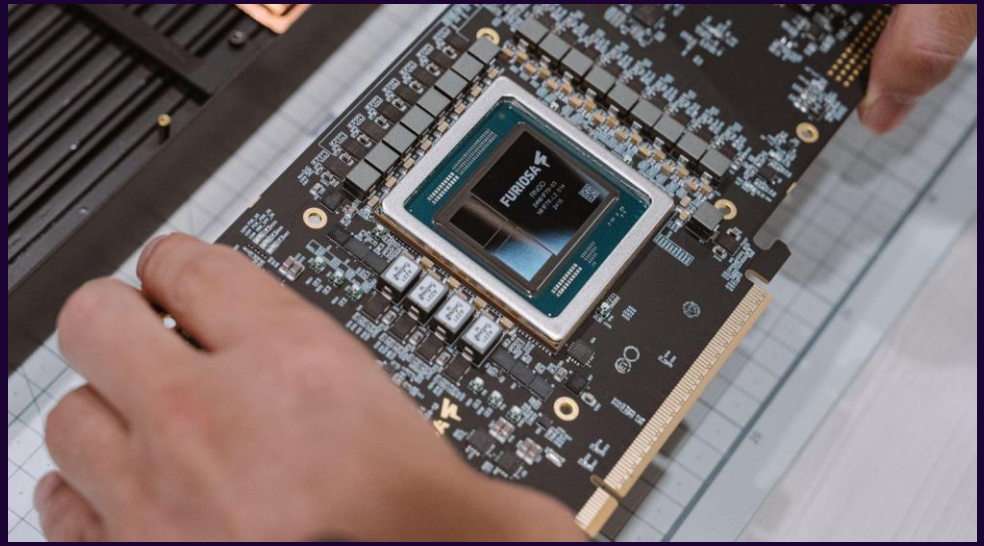
Inference Efficiency

= HW Efficiency x SW Efficiency x Algorithm Efficiency

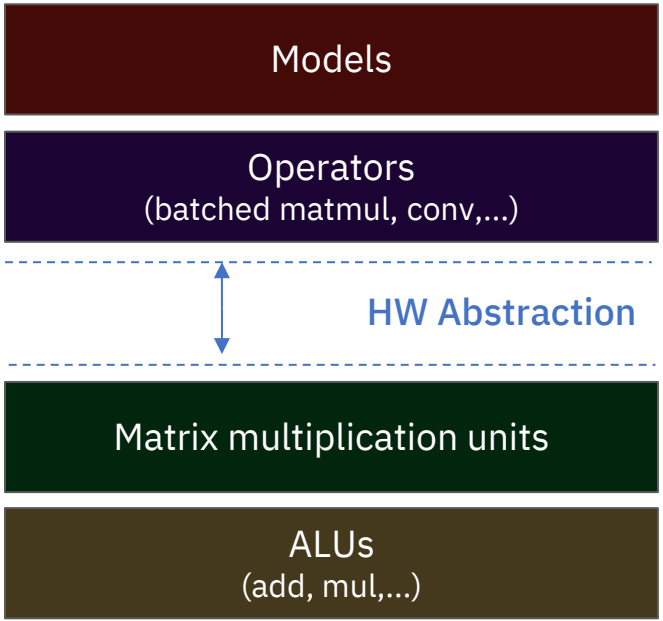
AI chip should leverage **optimizations at every level** *in addition* to HW efficiency



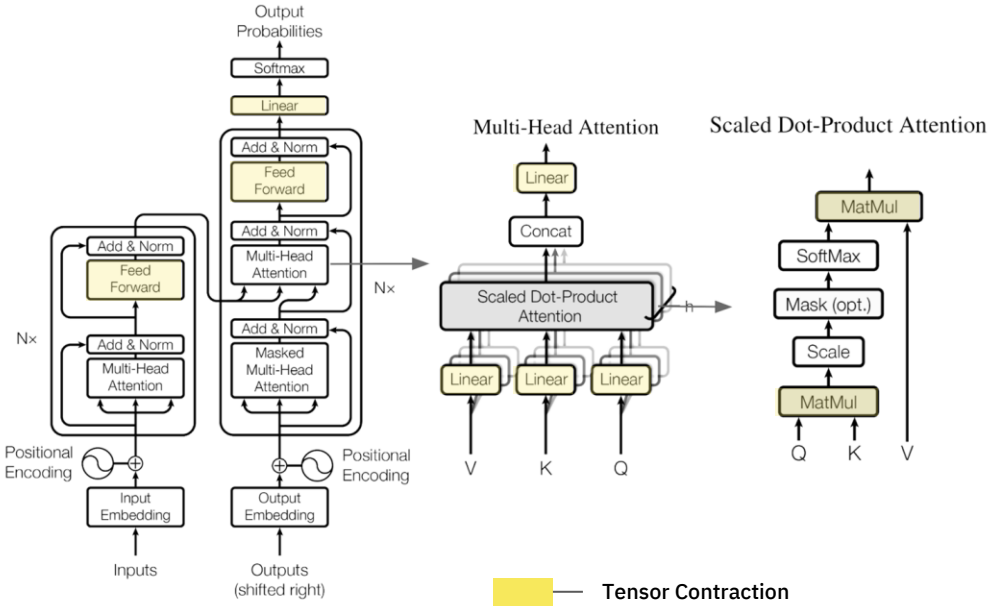
HW Architecture & Chip Design

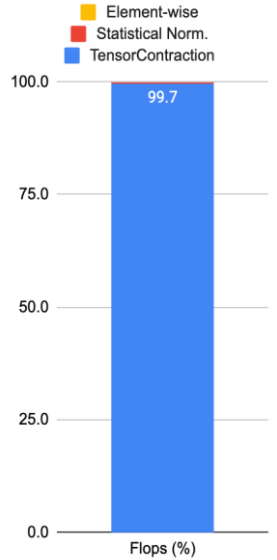


Raising the abstraction between HW and SW



Tensor Contraction, Core Computation in Deep Learning





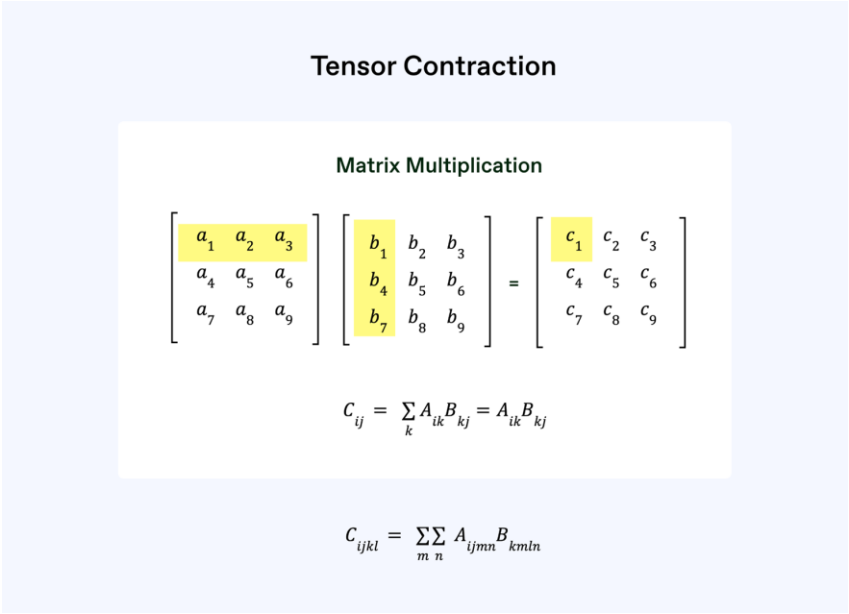
Flop analysis for BERT*

Source: Data Movement is All You Need: a case study on optimizing transformers, MLSYS'21

Tensor Contraction, *not* Matmul as a Primitive

The fundamental core computation of deep learning is **tensor contraction,** higher dimensional generalizations of matrix multiplication.

However, **most commercial deep learning accelerators incorporate fixed-size matmul** instruction as a primitive today.



TCP, Domain-Specific Architecture for Tensor Contraction

Raise level of HW/SW Interface to accelerate the whole tensor contraction as a primitive

Streamline the hardware to maximize the parallelism and data reuse

Results in **higher performance** and energy **efficiency**, while providing **flexibility** to support all deep learning models

Introducing Low-Level Einsum for our Primitive

Tensor contraction is declarative

No **explicit scheduling** for computation

No **explicit memory layout** for data

We introduce low-level einsum for our primitive

Low-level einsum =

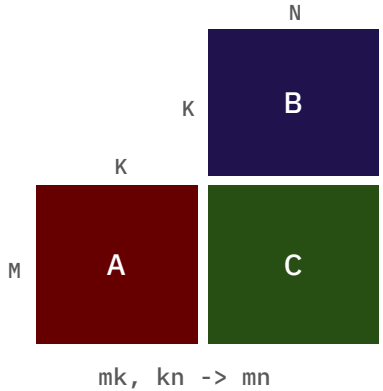
tensor contraction + **explicit memory layout** + **explicit scheduling**

$$C_{ijkl} = \sum_{m}\sum_{n} A_{ijmn} B_{kmln}$$

```
torch.einsum('ijmn,kmln->ijkl', [a, b])
```

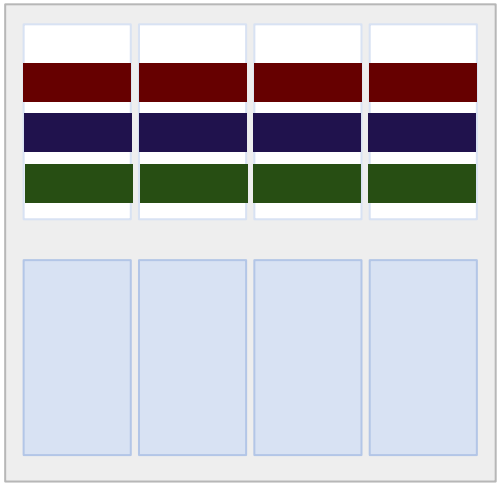
Einsum notation for tensor contraction

One Simple Example of Tensor Contraction



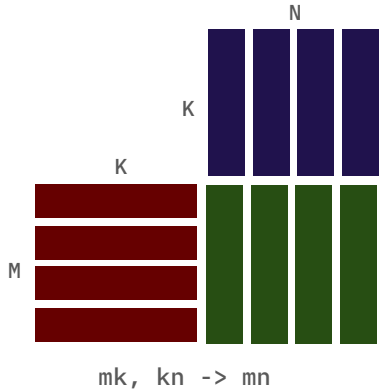
Computation

```
for (n in 0..N) {
      for (m in 0..M) {
            for (k in 0..K) {
                  C[m][n] += A[m][k] x B[k][n]
            }
       }
}
```



Chip

A Whole Tensor Contraction as a Primitive

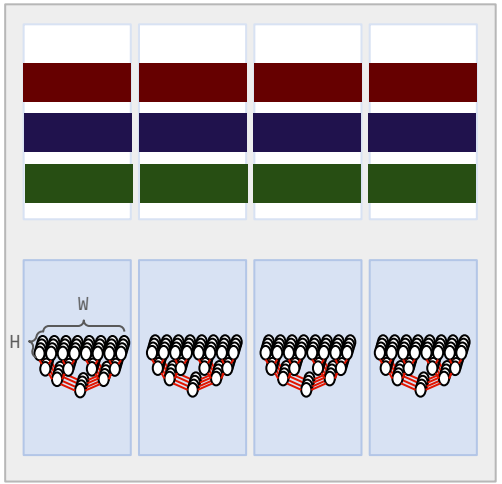



```
// spatial mapping
for (n_blk in 0..4) {

  // temporal scheduling
  for (m_blk in 0..4) {
    for (k_blk in 0..(K/W)) {
      for (m_idx in 0..(M/4)) {
        for (n_idx in 0..(N/4/H)) {

          // unit computation
          for (h in 0..H) {
            for (w in 0..W) {
              m = m_index_of(m_blk, m_idx)
              n = n_index_of(n_blk, n_idx, H)
              k = k_index_of(k_blk, W)
              C[m][n] += A[m][k]xB[k][n]

} } } } } } }
```



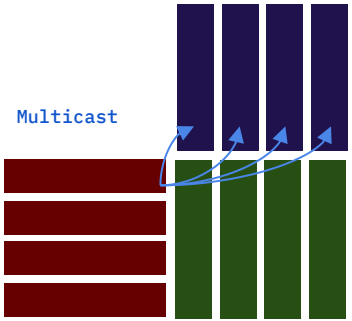
Lowered shape
memory layout of tensors

Tactic
scheduling of computation

Chip

a single primitive

A Whole Tensor Contraction as a Primitive

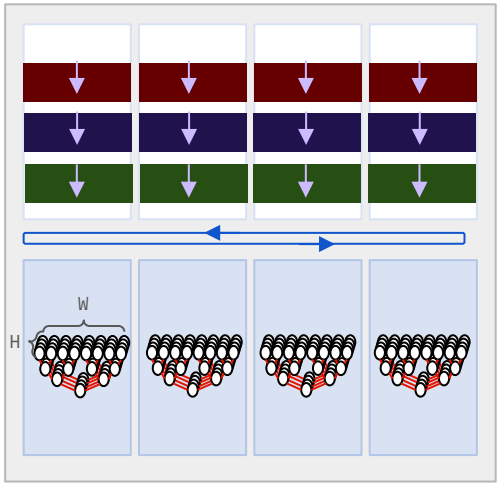



```
// spatial mapping
for (n_blk in 0..4) {

  // temporal scheduling
  for (m_blk in 0..4) {
    for (k_blk in 0..(K/W)) {
      for (m_idx in 0..(M/4)) {
        for (n_idx in 0..(N/4/H)) {

          // unit computation
          for (h in 0..H) {
            for (w in 0..W) {
              m = m_index_of(m_blk, m_idx)
              n = n_index_of(n_blk, n_idx, h)
              k = k_index_of(k_blk, w)
              C[m][n] += A[m][k]xB[k][n]

} } } } } } }
```



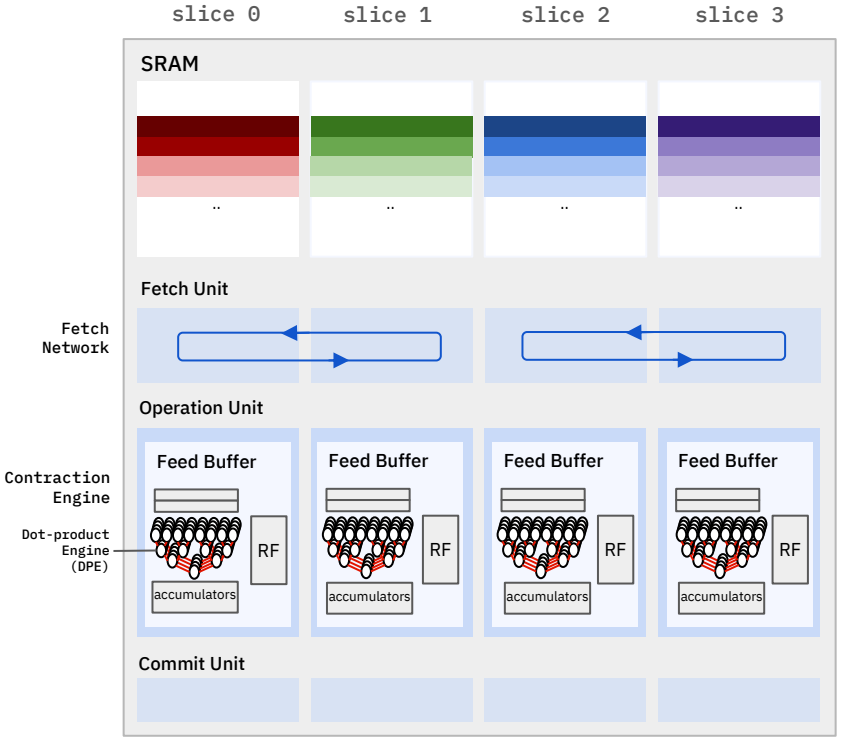
Lowered shape
memory layout of tensors

Tactic
scheduling of computation

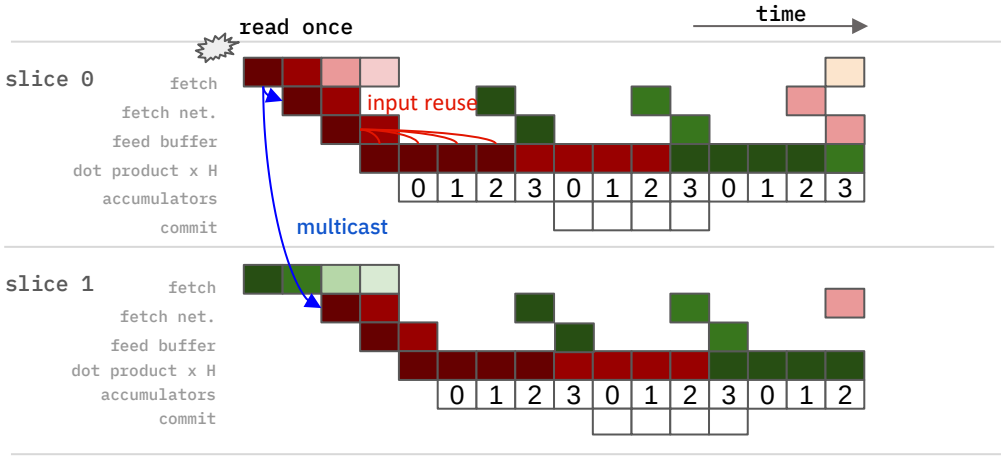
a single primitive

Chip

Spatial & Temporal Orchestration Boosts Utilization & Efficiency



The other input tensor supplied by the RF within the contraction engine will omit it from the diagram.
Other tactics can refer to the paper

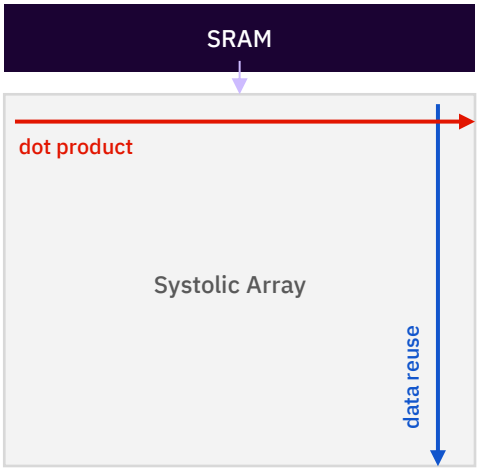


Hardware operates according to software-defined (optimizes) tactics

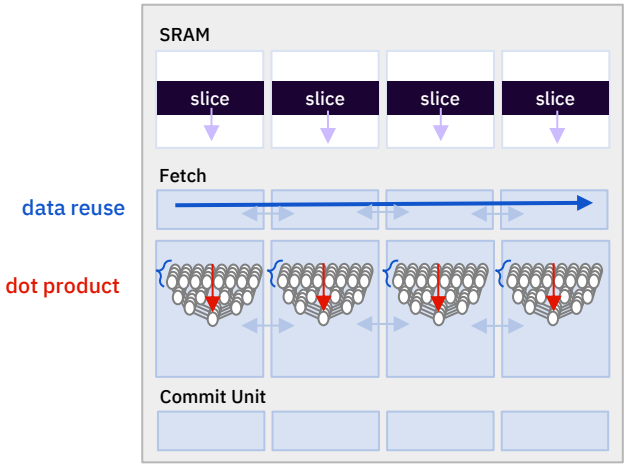
- Data read once from SRAM can be multicast and fed multiple times
- Temporal pipelining allows full utilization of spatially parallel compute units
- All data paths can be streamlined

Flexible Reconfigurability Is More Crucial in Inference

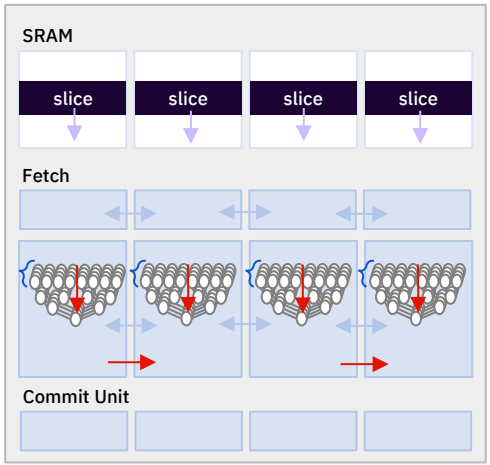
For inference, batch sizes can vary widely
Thus it is even more important to exploit parallelism and data reuse from given tensor shape



TPU 128 x 128

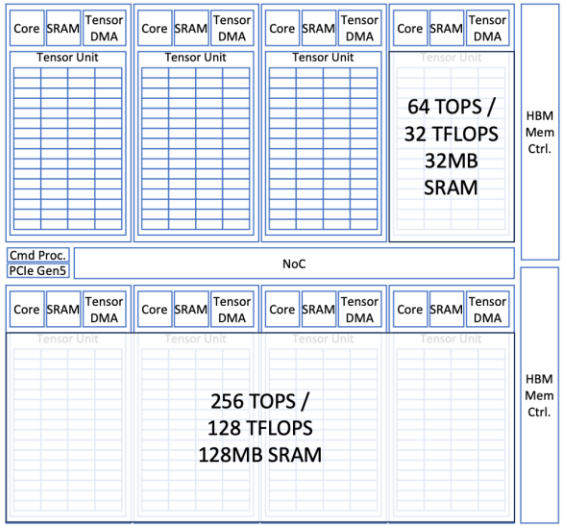


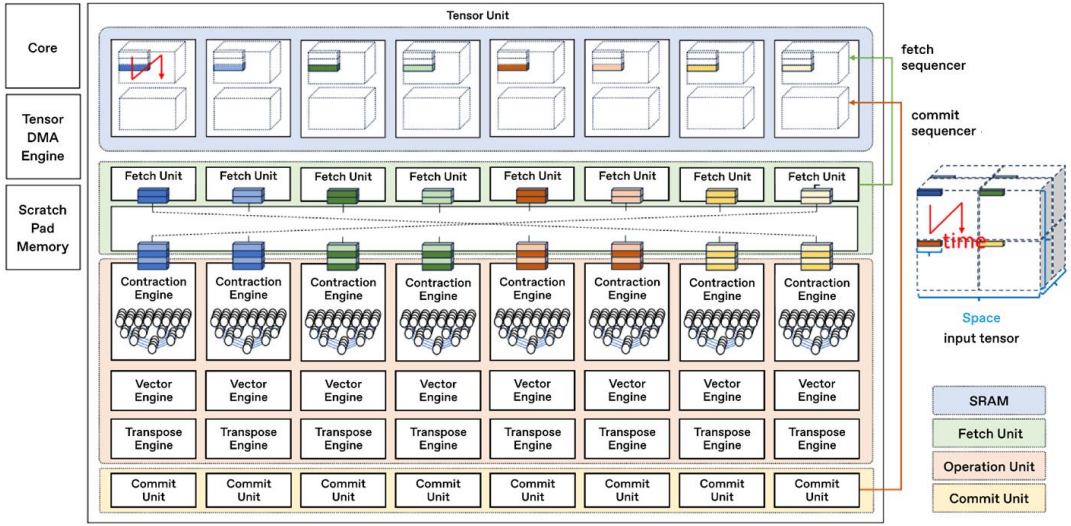
TCP W x 4H



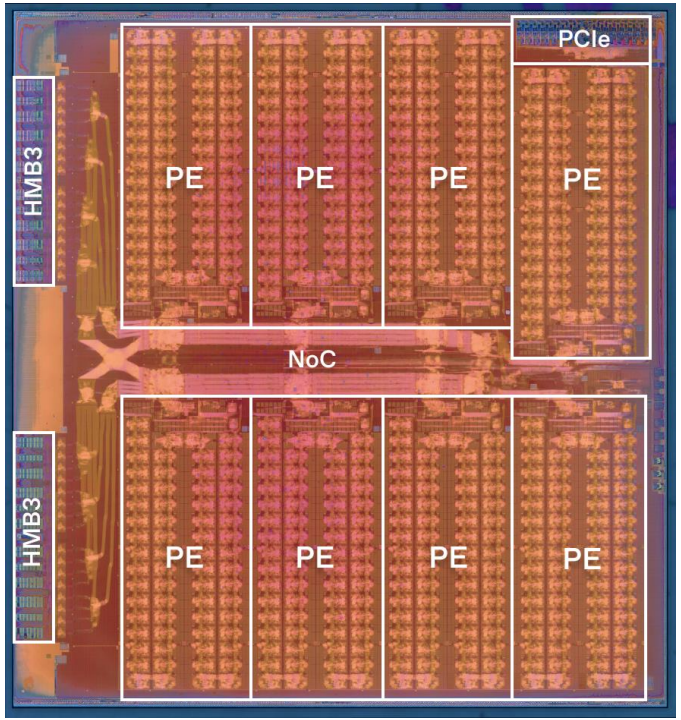
TCP (2W x H) x 2

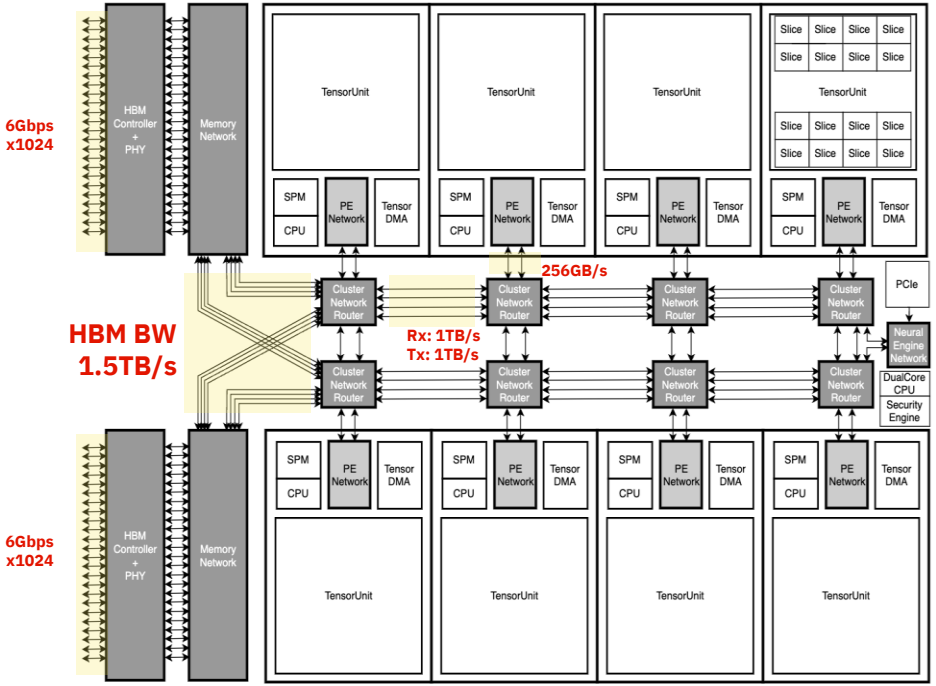
RNGD, a Tensor Contraction Processor



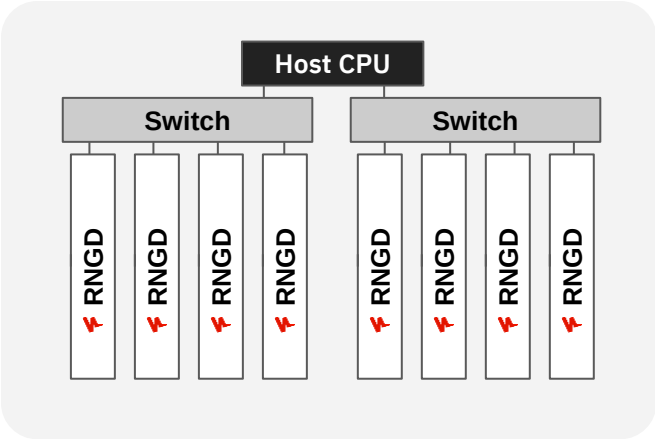


Interconnection Networks

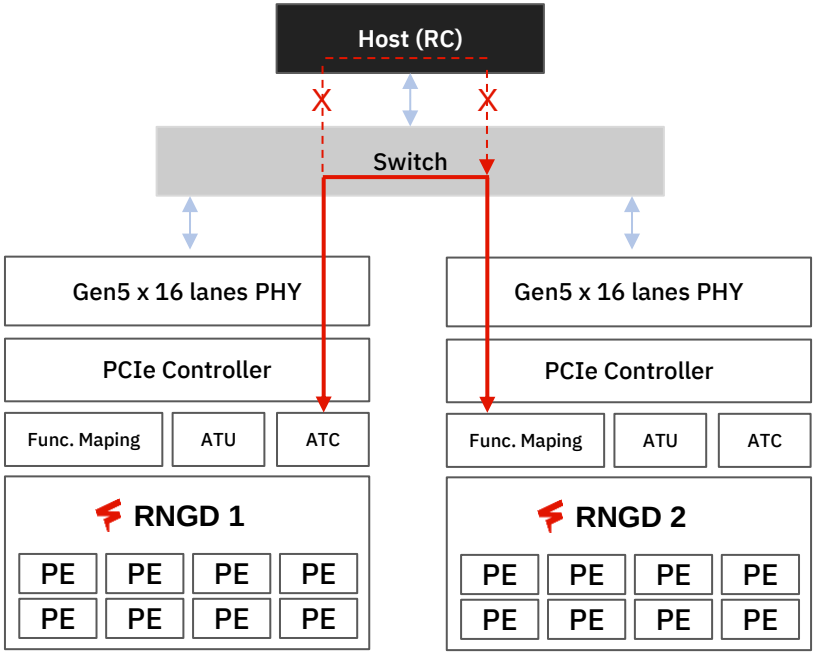




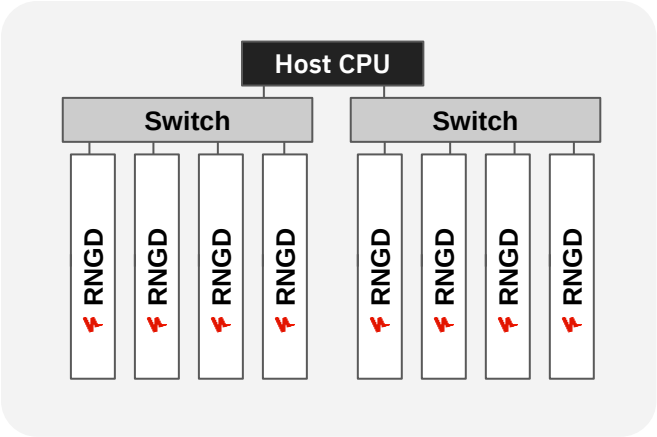
PCIe-based Chip-to-Chip Communication

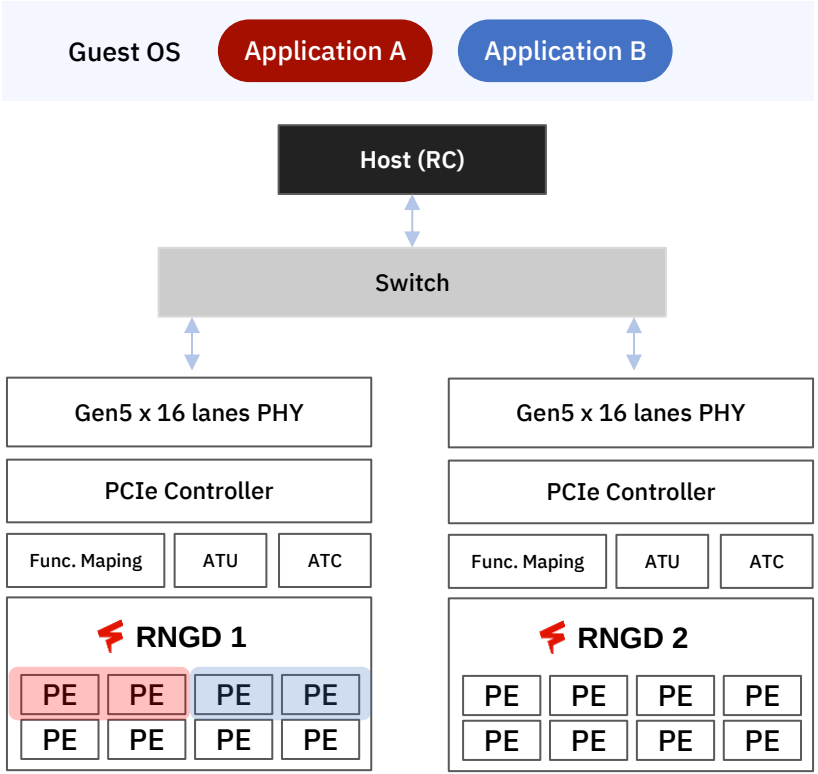


- Multi-card model support for large models
- Reduce the latency across cards via direct P2P by Address Translation Service(ATS) Feature
- No Address translation by IOMMU in Host



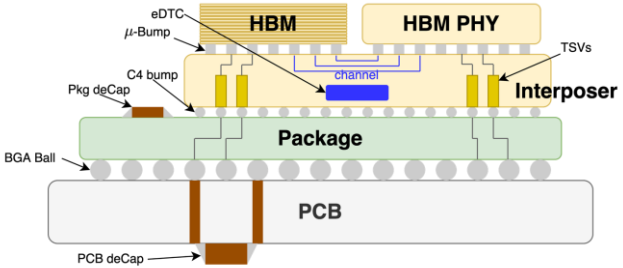
SR-IOV for Multi-Instance Support & Virtualization

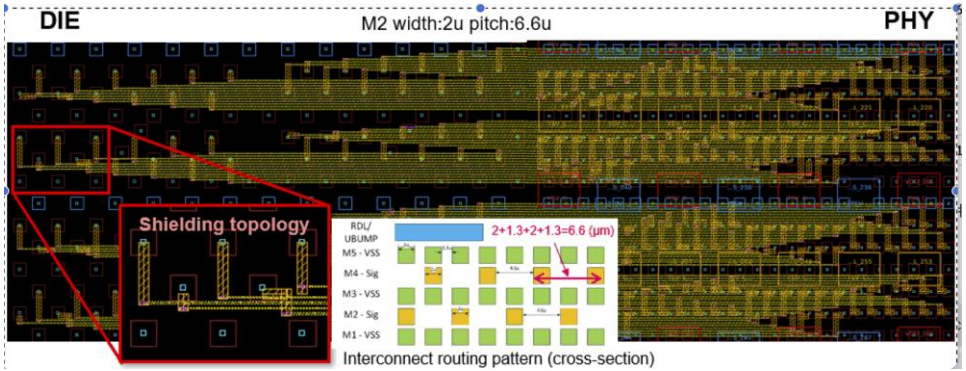




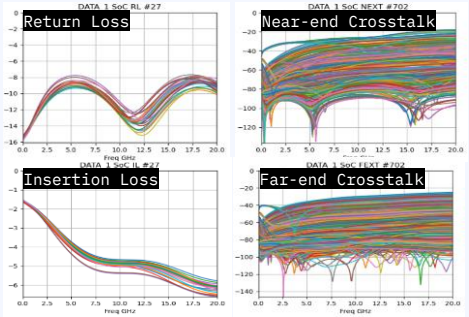
- Supports up to 8 Virtual Functions for Virtual Machines
- 1, 2 or 4 PEs can be assigned for a VM

HBM Integration Challenges: Signal and Power Integrity

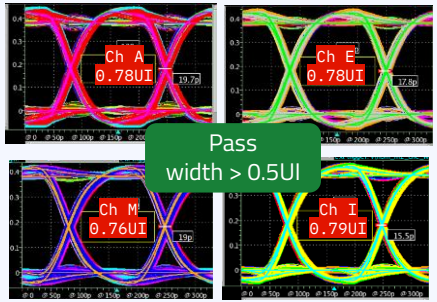




S-Parameters

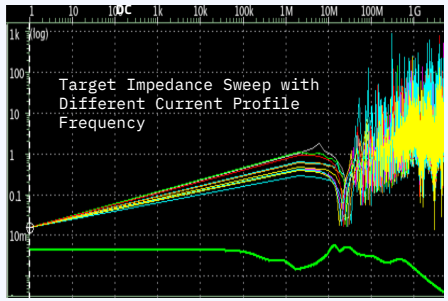


Eyediagram

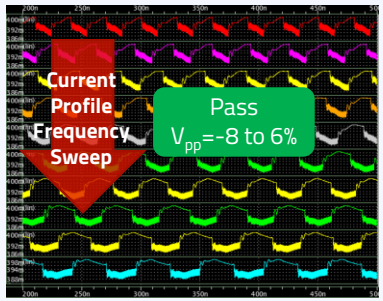


Signal Integrity

PDN Z-profile $Z_{PDN} = \frac{V_{ripple}}{I_{switch}}$

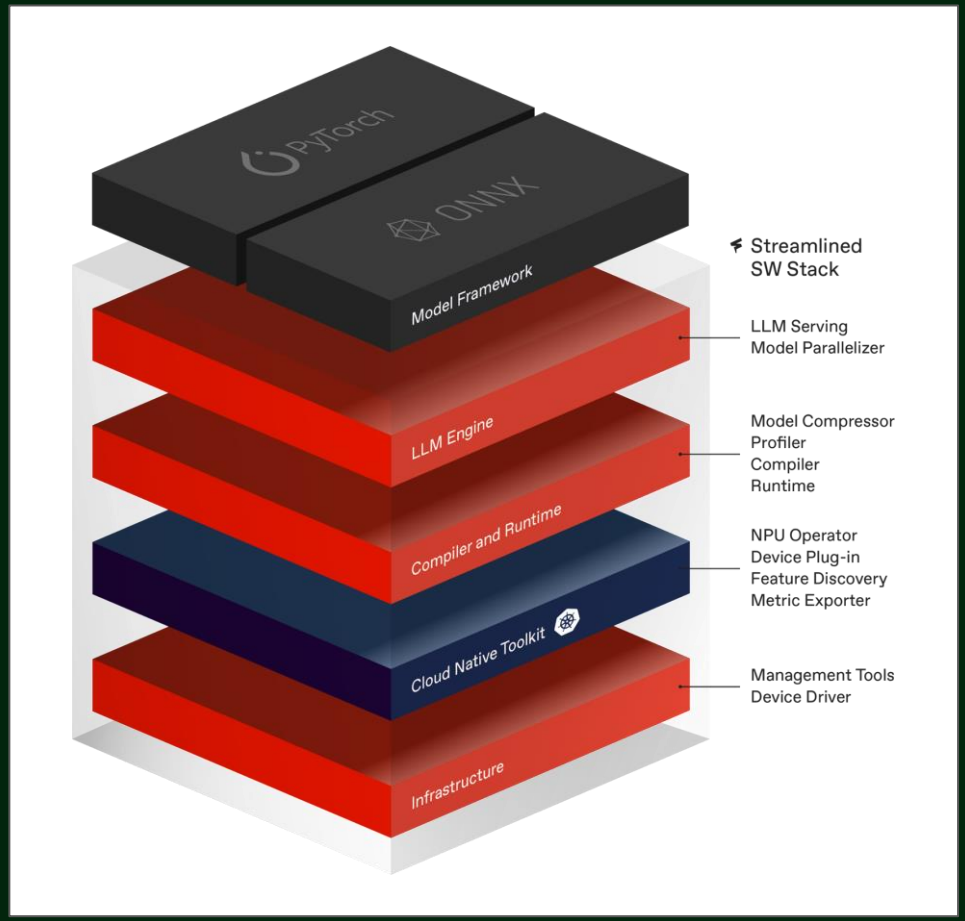


Voltage Fluctuation

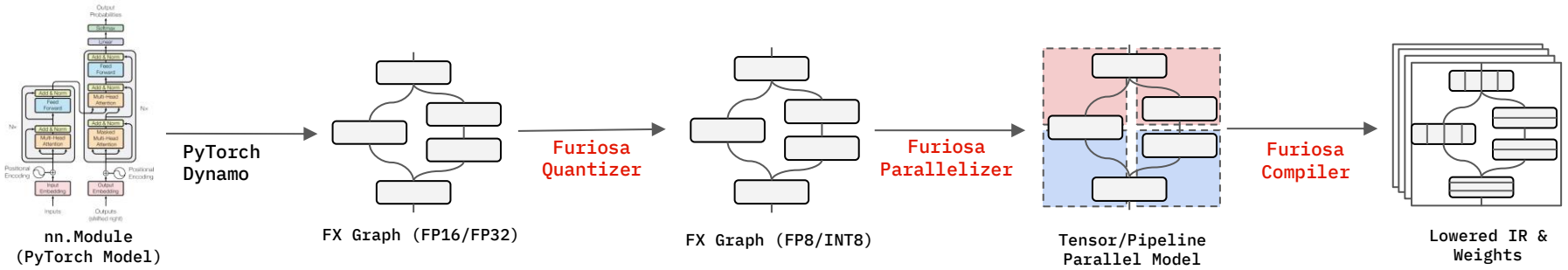


Power Integrity

Furiosa RNGD SW Stack



Furiosa LLM Stack



Furiosa LLM provides full-fledged features for LLM Serving, as following:

- PyTorch 2.0 integration
- Quantization toolkit with various quantization support (FP8, INT8, etc.)
- Tensor/Pipeline/Data parallelism over multiple cards
- Advanced DNN compiler
- State-of-the-art Serving optimization

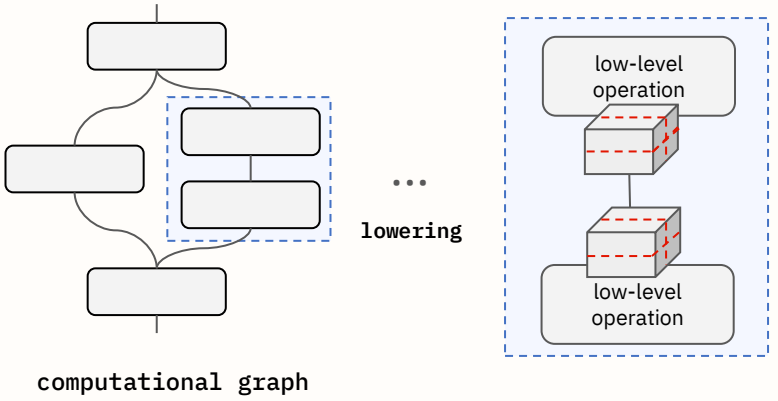
Compilation for End-to-End Model Efficiency

Compiler finds optimal tactic for given lowered shapes

- Temporal pipelining enables performance prediction by analyzing bottlenecks
- Compiler uses perf/power estimator while exploring tactic space

End-to-end automated compilation

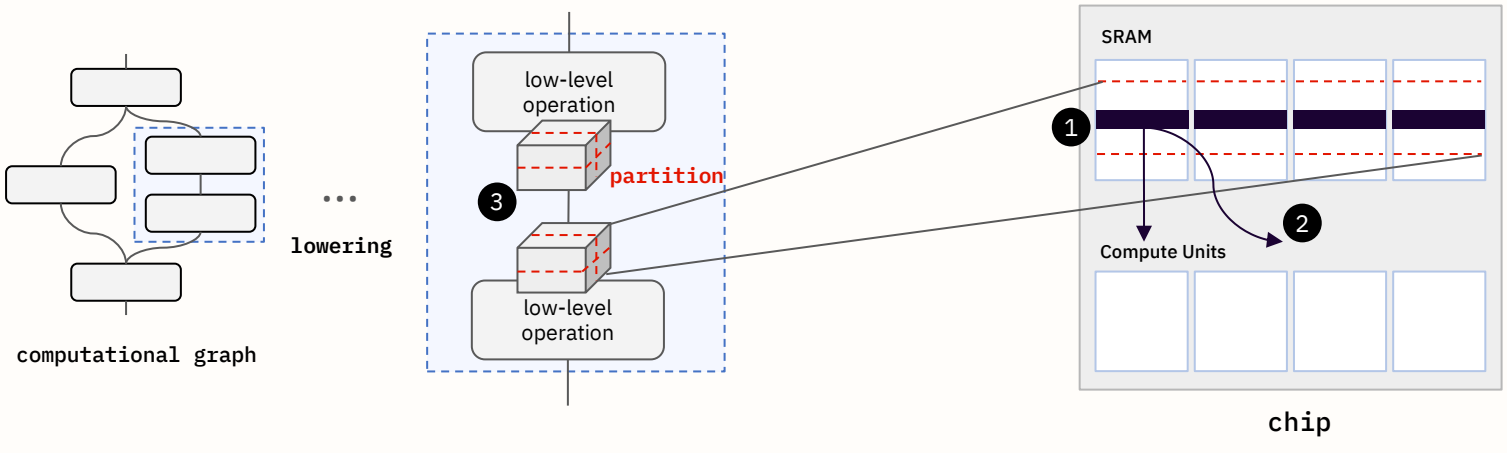
- Compiler finds optimal lowered shapes of tensors of given graph
- Optimizations are achieved through multi-level IRs including graph level optimization such as operator fusion and memory allocation split merge scheduling



Compiler Lowers Shape for Low-Level Einsum

Mapping tensors onto on-chip memory impacts performance & energy efficiency

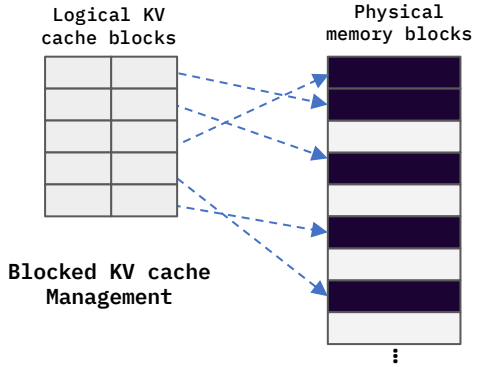
1. SRAM access performance (e.g., row/column major)
2. Data movement of input tensors to compute units for a single low-level operation
3. Data movement across operators

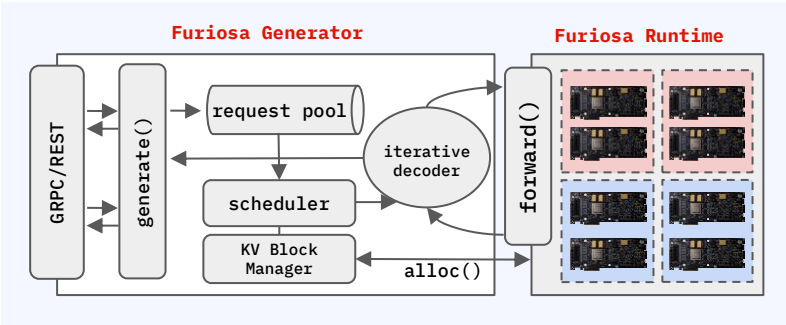


Furiosa Serving Framework

High throughput serving with SOTA optimization

- PagedAttention and blocked KV-cache management by leveraging **Furiosa Compiler and Runtime**
- Continuous batching with higher batch size, leading to high utilization and throughput for serving in production





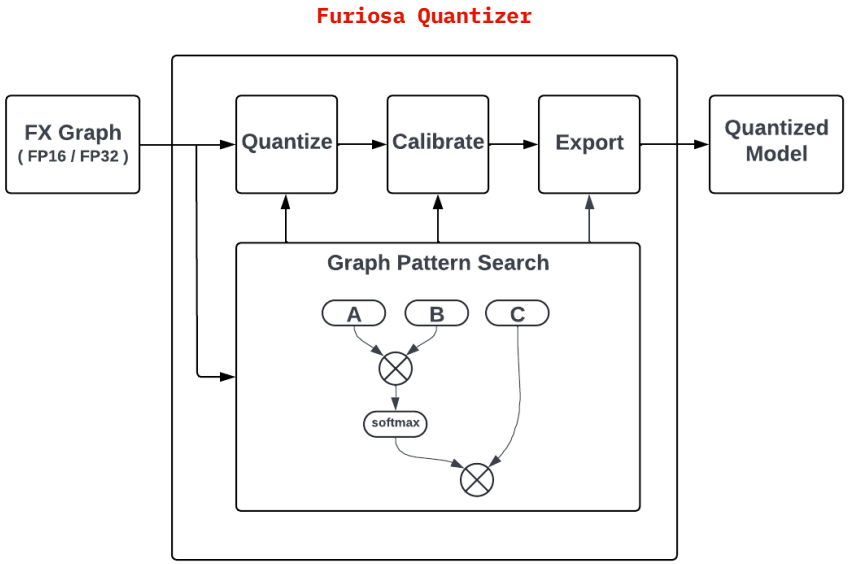
Furiosa Quantizer: Graph-Based Automation Tool

End-to-end automated quantization

- Support arbitrary customized LLM models using graph pattern search
- Analyzes transformer block starting from “softmax(AxB) x C” attention graph pattern

Various quantization schemes

- BF16 (W16A16)
- INT8 Weight-Only (W8A16)
- FP8 (W8A8)
- INT8 SmoothQuant (W8A8)
- INT4 Weight-Only (W4A16 AWQ / GPTQ)



Advanced Development Methodology

Productive language & “code”-based communication

- Rust: safe, performant and productive language for compiler & runtime
- Chisel: RTL development & simulator generation
- Python-based UVM: block-level verification

Scalable tools & infrastructure

- Kubernetes + Tekton CI on public + private cloud
- Emulation, FPGAs, QEMUs + SystemC simulation

Testcase generation & verification

- Generates all (fused) tactic kernels from (quantized) models & PyTorch 2.0 aten ops
- Randomly generated tactic kernels


RNGD, powerfully efficient and **programmable** data center AI accelerator built for the era of LLM and other generative AI models