groq™

Groq AI Workshop

ALCF AI Testbed

June 2024

Agenda - Day 2

Session	Description	Length	Speaker
Groq Compiler™ Overview	Inside look at how the compiler works to compile models for Groq, including an overview of partitioning and scheduling.	20 mins	Philip Lassen, Compiler Engineer
Groq Runtime™ Overview	Overview of the runtime, including what it is, how models are executed, and how data is transferred across the chip.	20 mins	Aviv Weinstein, Systems Software Engineer
LLMs with Groq	How Groq scales to unlock the fastest inference in the world, specifically around larger models.	20 mins	Aviv Weinstein, Systems Software Engineer
15 MINUTE BREAK 🚀			
Throughput Optimization with Groq	Walkthrough of how to compile small models with the Groq Compiler and how to optimize them for throughput.	60 mins	Christopher Culver, Software Engineer
What's Next	A talk with Igor Arsovski, our Chief Architect and Fellow, on the semiconductor space and what's next for Groq.	30 mins	Igor Arsovski, Chief Architect & Fellow

Groq™ Compiler

Philip Lassen
Compiler Engineer

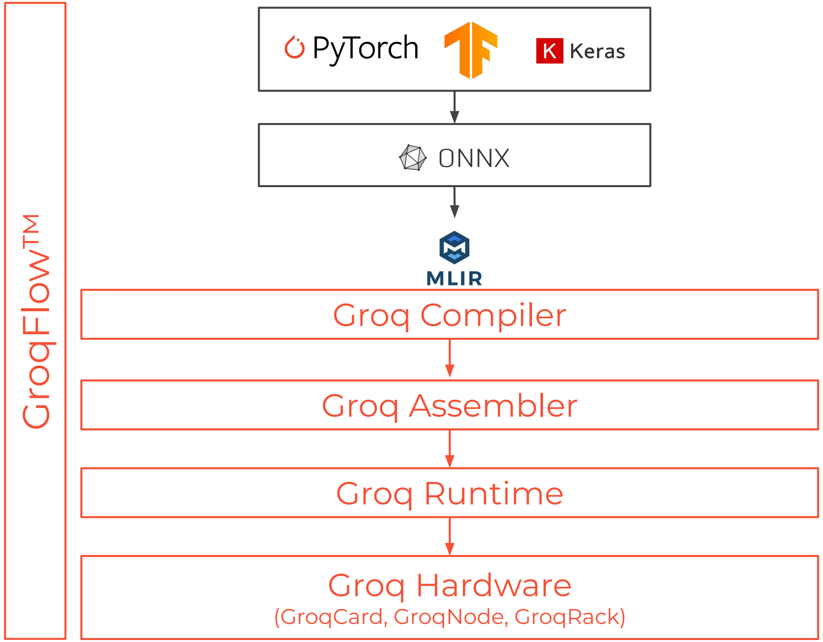
Groq™ Compiler

AGENDA

- What is the Groq Compiler
    - Groq Compiler vs GroqFlow
- Groq Compiler Overview
    - Frontend
    - Middle-end
    - Backend
        - Scheduler
        - Modes: standard vs high
- Multi-Chip
    - InterOp
    - IntraOp
    - Example: Transformers

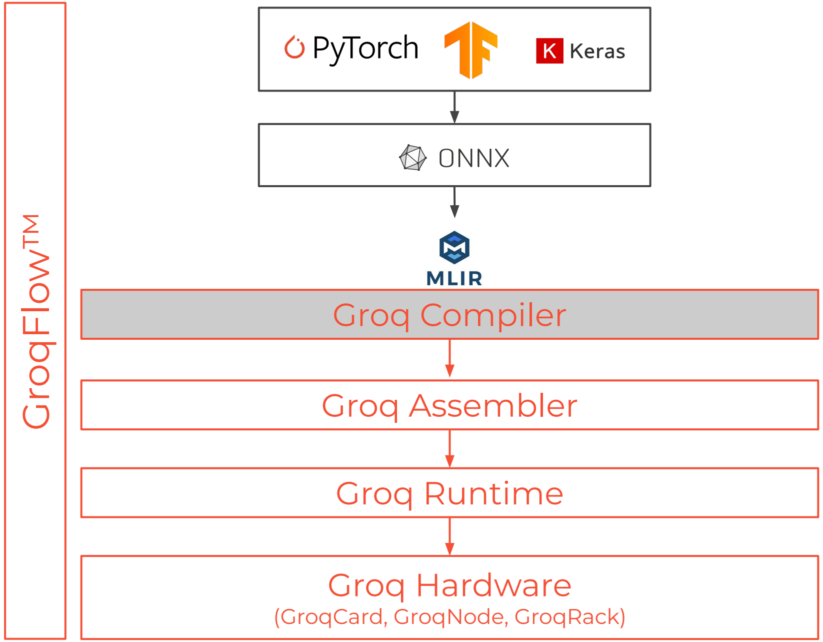
Simplified GroqFlow™ Usage Model

Groq Software to Hardware WorkFlow



Simplified GroqFlow™ Usage Model

Groq Software to Hardware WorkFlow



Input Program

```
class Model(torch.nn.Module):
    def forward(self, A, x, b):
        return torch.matmul(A, x) + b

model = Model()
torch.onnx.export(model, **inputs, “program.onnx”)
```

Compiler

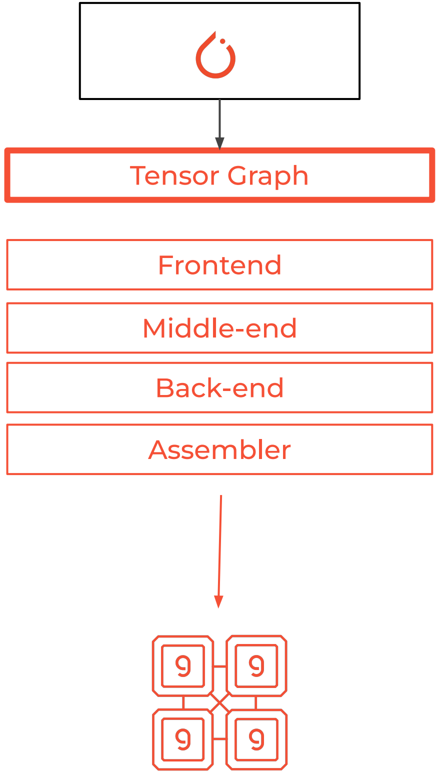
Function |	Alan Assembly Instruction
MEM	`Read` *a,s* `Write` *a,s* `Gather` *s, map* `Scatter` *s, map* `Countdown` *d* `Step` *a* `Iterations` *n*

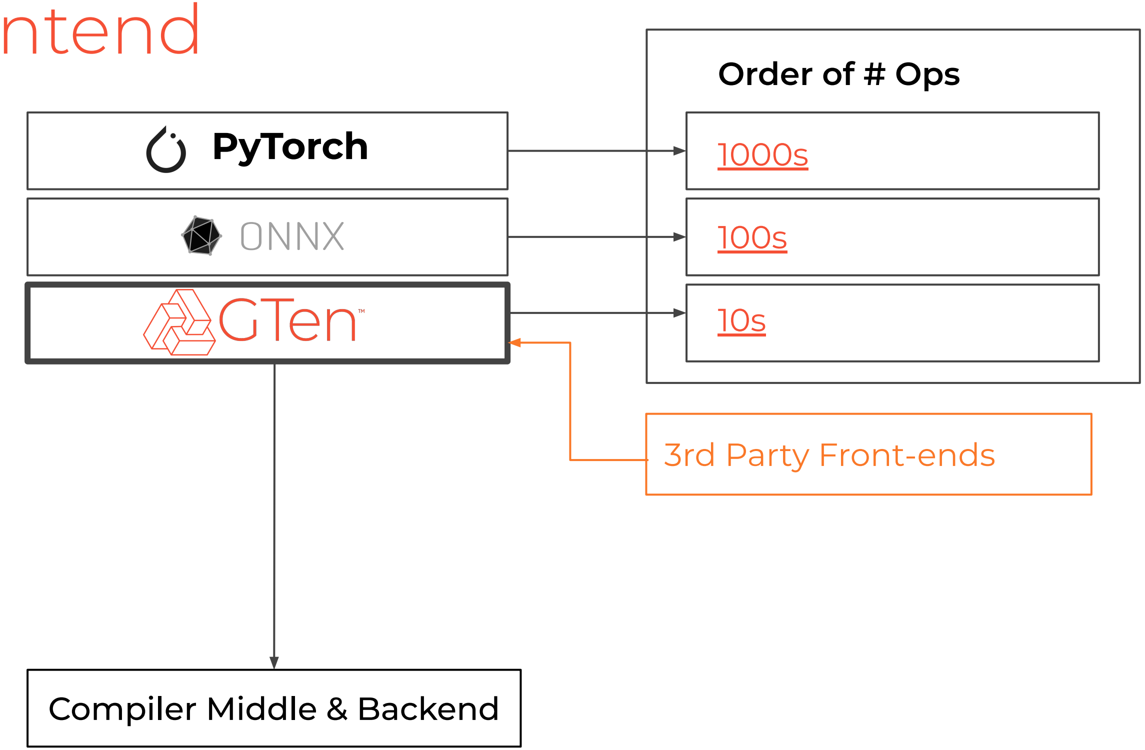
VXM	unary operation
	binary operation
	type conversions
	`Log`
	`TanH`
	`Exp`
	`RSqrt`

MXM	`LW`
	`IW`
	`ABC`
	`ACC`

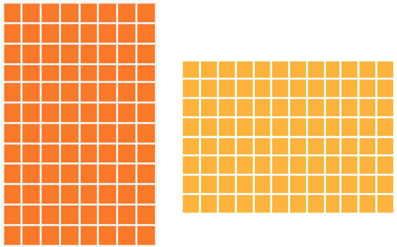
SXM	`Shift` ***up/down N*** `Permute` ***map*** `Distribute` ***map*** `Rotate` ***stream*** `Transpose` *sg16*

Groq Compiler Frontend





Middle-end



Function	Instruction
MEM	`Read` *a,s* `Write` *a,s* `Gather` *s, map* `Scatter` *s, map* `Countdown` *d* `Step` *a* `Iterations` *n*

VXM	unary operation
	binary operation
	type conversions
	`Log`
	`TanH`
	`Exp`
	`RSqrt`

MXM	`LW`
	`IW`
	`ABC`
	`ACC`

SXM	`Shift` ***up/down N*** `Permute` ***map*** `Distribute` ***map*** `Rotate` ***stream*** `Transpose` *sg16*

Scheduler

Problem:

- Schedule compute graph to minimize compute cycles

Considerations:

- Which compute cycle?
  Which Functional unit?
- What Streams? Certain streams are reserved
- Which Memory slices should we store Constants and Intermediates on?

Scheduling: Vector vs Tensor

Vector

- Schedule single vector operations at a time
- Compiler Flag = --effort=high

Tensor

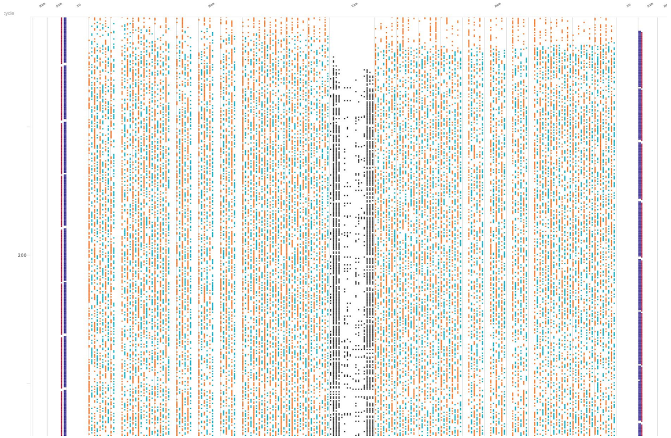
- Bulk-schedule multiple vector operations of the same type
  - So that they occupy a Functional Unit (FU) in consecutive cycles
- Compiler Flag = --effort=standard // default

	Vector	IA

`for (i = 0; i < 4; ++i)`
`C[i] = A[i] + B[i]`
	`C[0] = A[0] + B[0]`
	`C[1] = A[1] + B[1]`
	`C[2] = A[2] + B[2]`
	`C[3] = A[3] + B[3]`
		`C[0..3] = A[0..3] + B[0..3]`

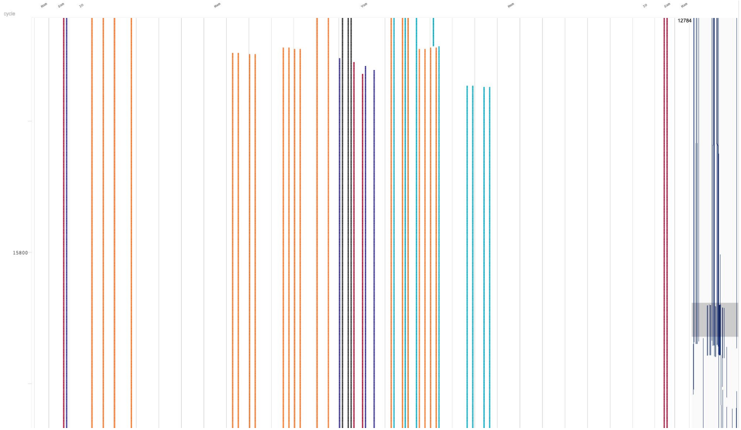
Scheduling: Vector vs Tensor

Vector



groqit(model, inputs, compiler_flags=[“--effort=high”])

Tensor

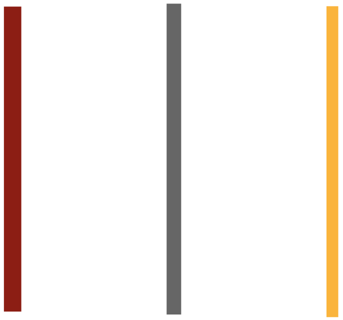


groqit(model, inputs, compiler_flags=[“--effort=standard”])

Multi-Chip

Parallelism

- 320 element SIMD units

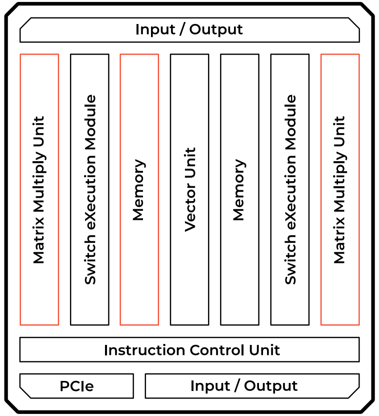


VXM
Vector-Vector
Operations

MXM
Matrix-Vector /
Matrix-Matrix
Multiply

SXM
Data
Reshapes

- Multiple Functional Units



Input / Output

Matrix Multiply Unit

Switch eXecution Module

Memory

Vector Unit

Memory

Switch eXecution Module

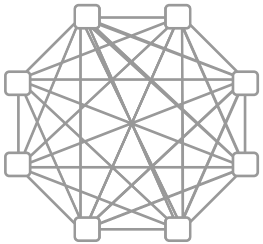
Matrix Multiply Unit

Instruction Control Unit

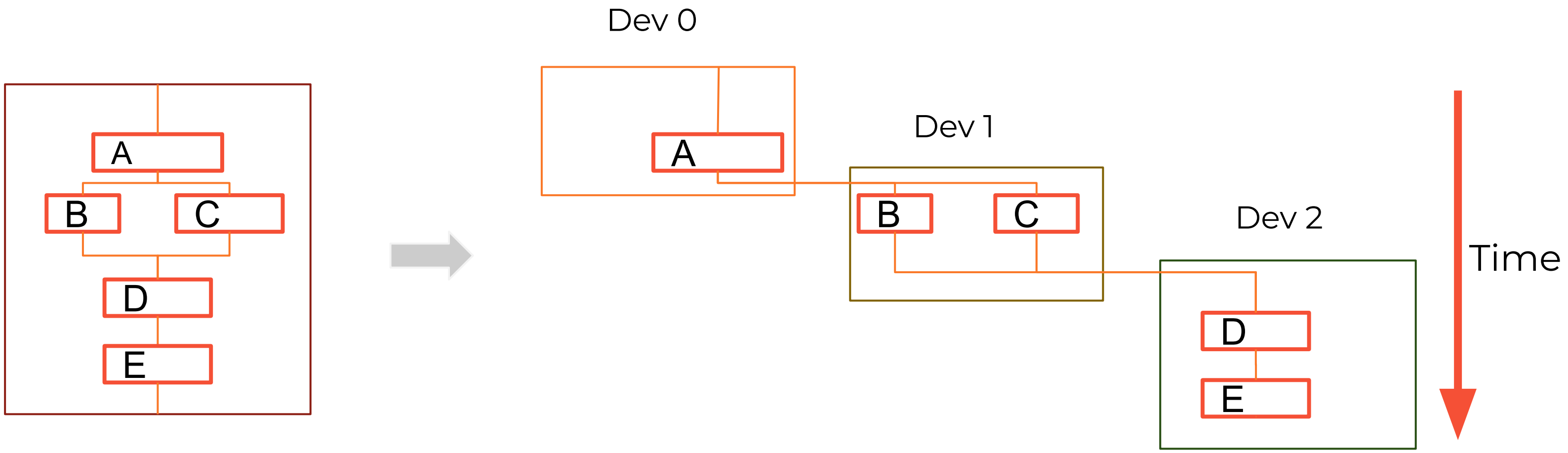
PCIe

Input / Output

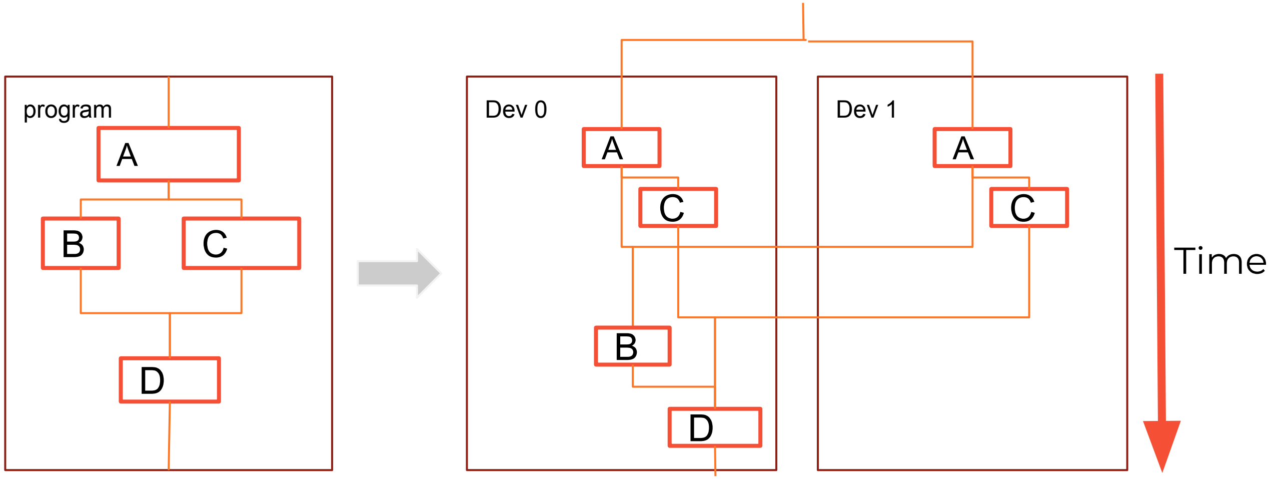
- Multiple LPUs



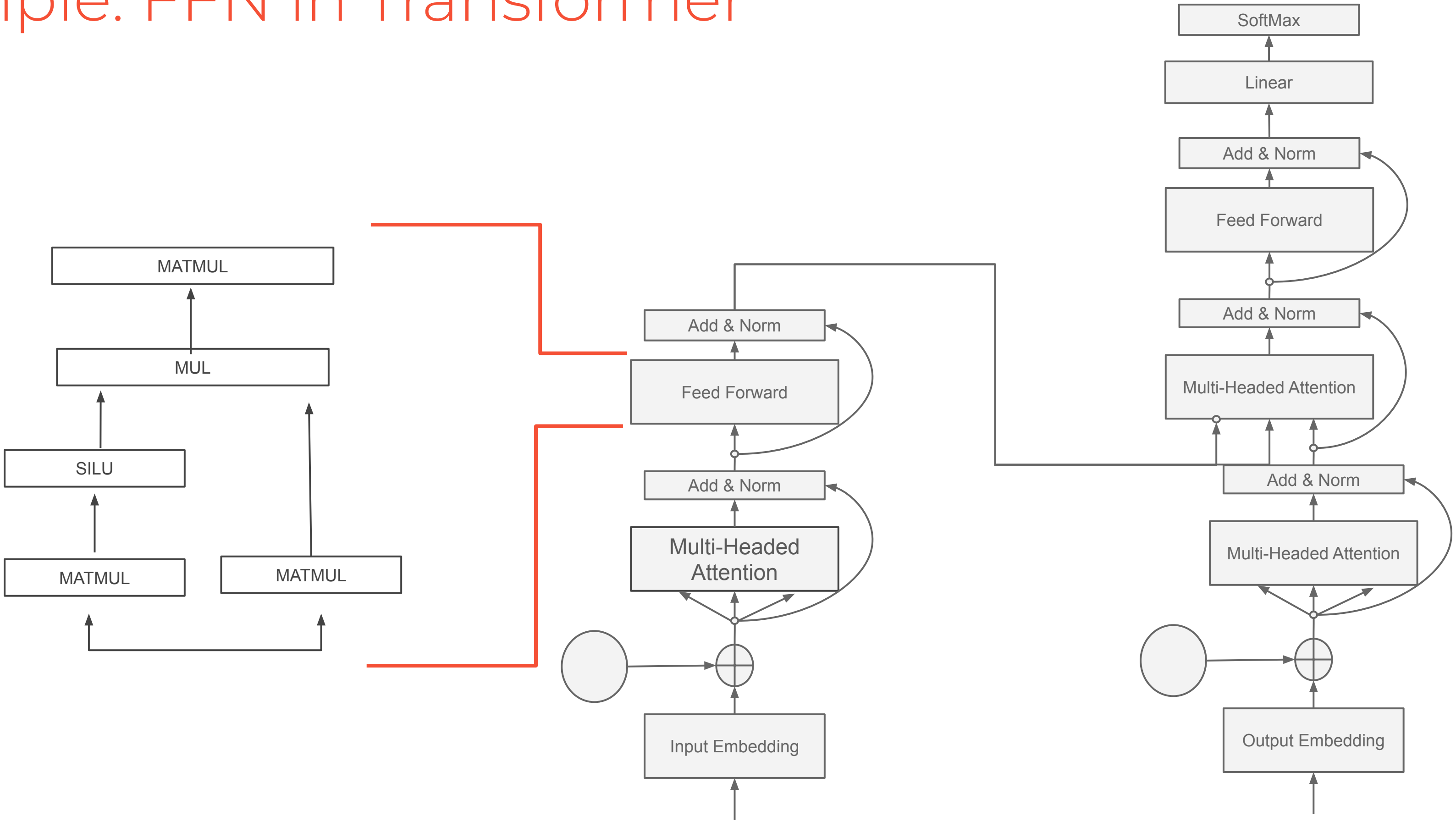
Inter Op Partitioning

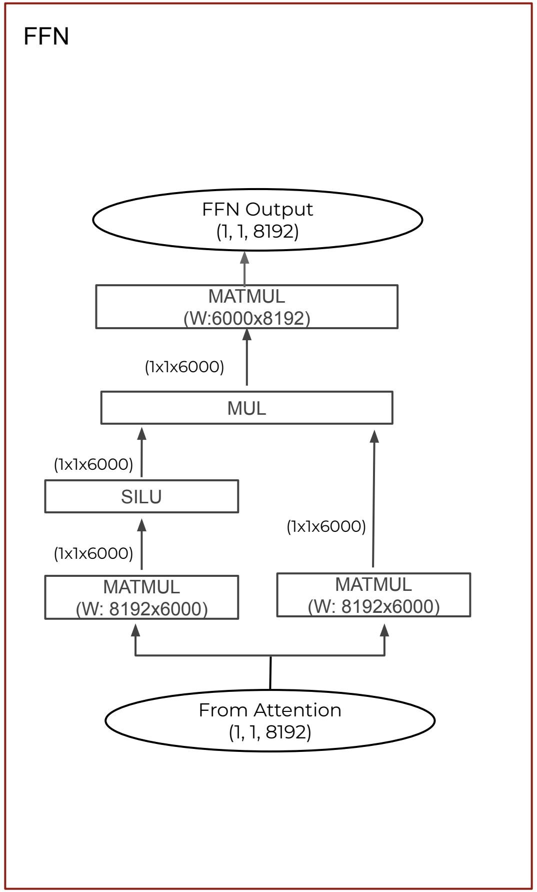


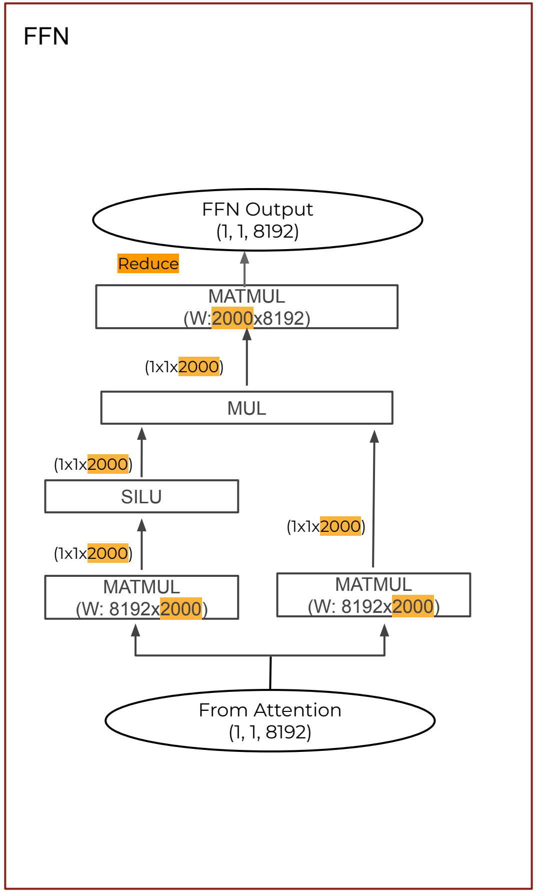
Intra Op Partitioning

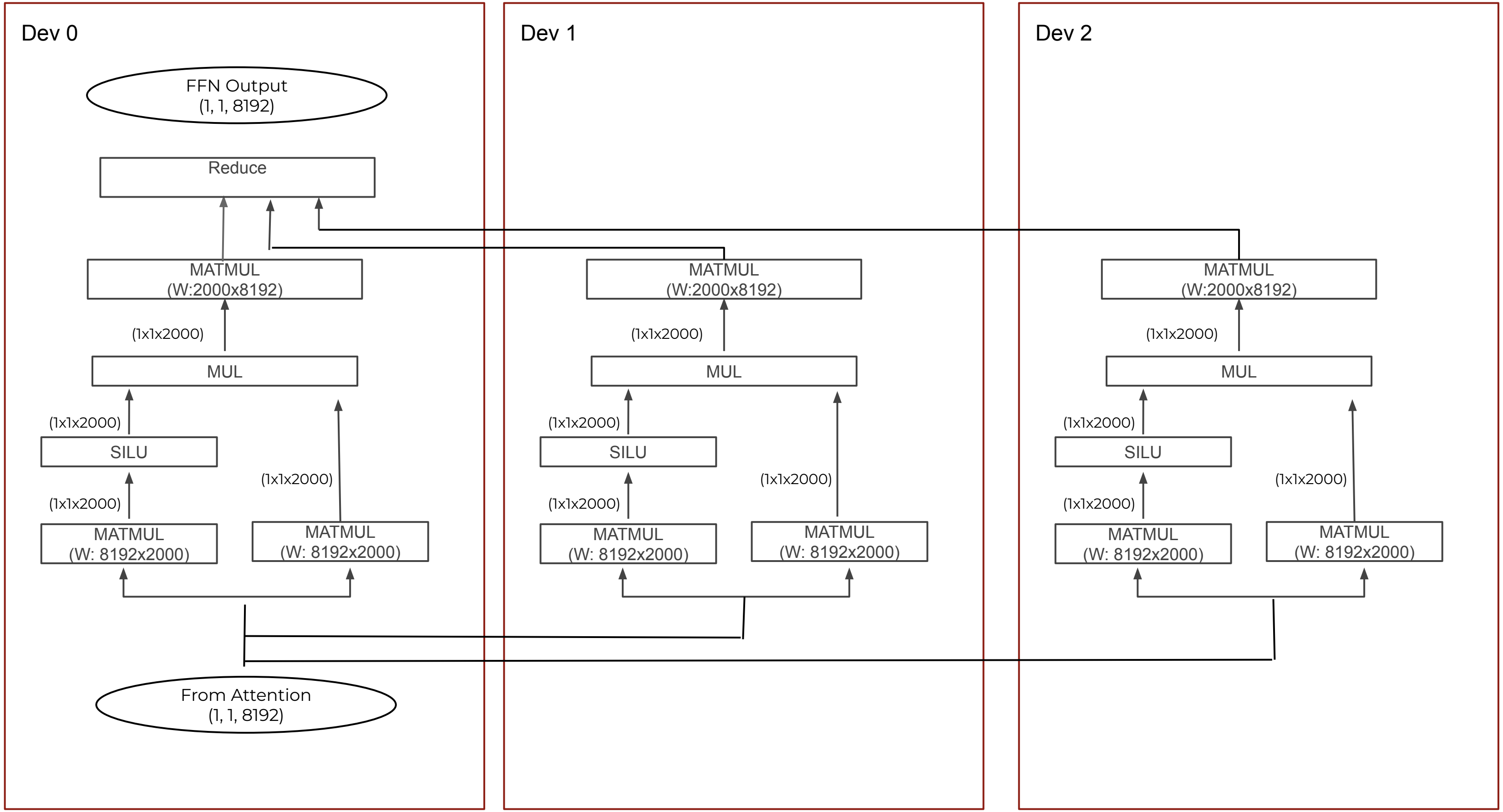


Example: FFN in Transformer

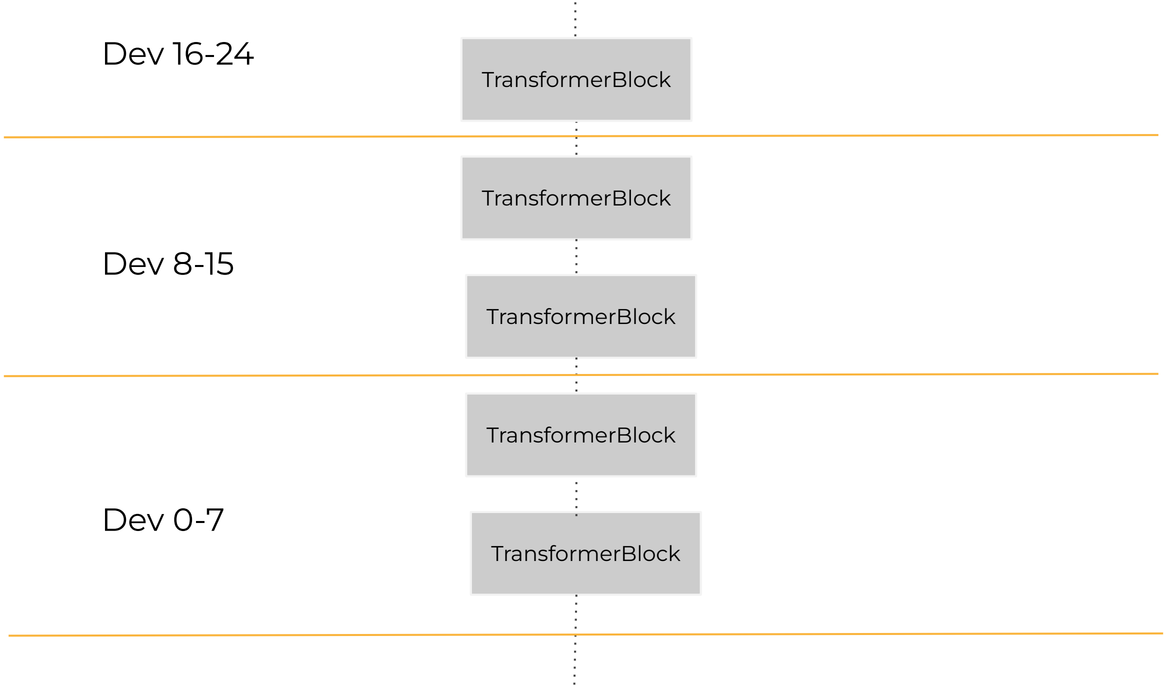



FFN
FFN Output
(1, 1, 8192)
MATMUL
(W:6000x8192)
(1x1x6000)
MUL
(1x1x6000)
SILU
(1x1x6000)
(1x1x6000)
MATMUL
(W: 8192x6000)
MATMUL
(W: 8192x6000)
From Attention
(1, 1, 8192)


FFN
FFN Output
(1, 1, 8192)
Reduce
MATMUL
(W:2000x8192)
(1x1x2000)
MUL
(1x1x2000)
SILU
(1x1x2000)
(1x1x2000)
MATMUL
(W: 8192x2000)
MATMUL
(W: 8192x2000)
From Attention
(1, 1, 8192)


Dev 0
Dev 1
Dev 2
FFN Output
(1, 1, 8192)
Reduce
MATMUL
(W:2000x8192)
(1x1x2000)
MUL
(1x1x2000)
SILU
(1x1x2000)
(1x1x2000)
MATMUL
(W: 8192x2000)
MATMUL
(W: 8192x2000)
MATMUL
(W:2000x8192)
(1x1x2000)
MUL
(1x1x2000)
SILU
(1x1x2000)
(1x1x2000)
MATMUL
(W: 8192x2000)
MATMUL
(W: 8192x2000)
MATMUL
(W:2000x8192)
(1x1x2000)
MUL
(1x1x2000)
SILU
(1x1x2000)
(1x1x2000)
MATMUL
(W: 8192x2000)
MATMUL
(W: 8192x2000)
From Attention
(1, 1, 8192)

Transformers : Inter Op Partitioning



groq™

Thank You!

plassen@groq.com

Groq Runtime

Aviv Weinstein
Systems Software Engineer

Groq Runtime

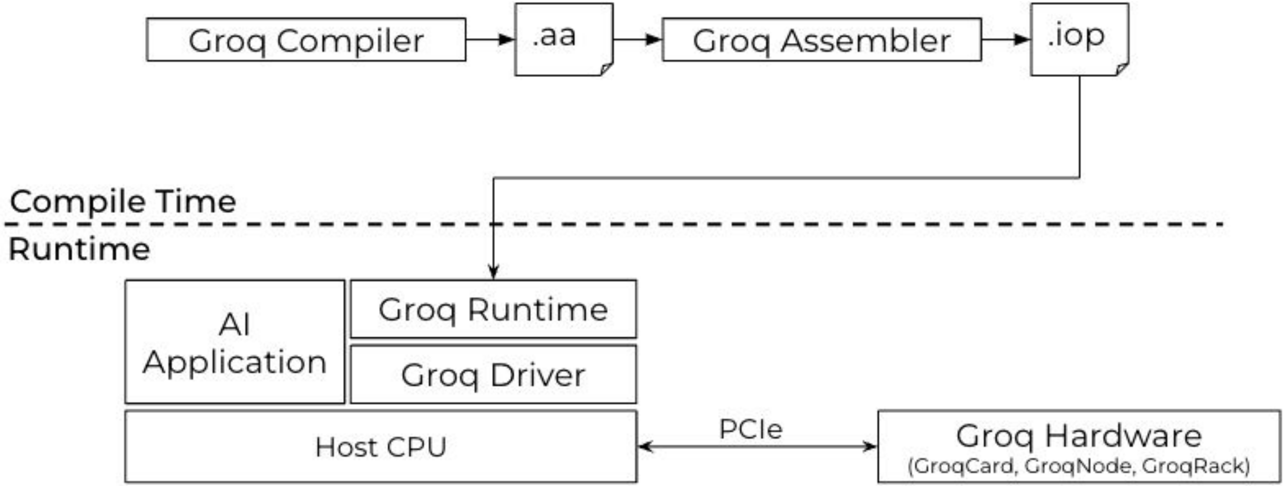
AGENDA

1. Groq Runtime HW/SW Architecture
2. Interacting with Groq Runtime as a Developer
3. Deeper Dive on Running Inferences on a GroqChip!

Groq Runtime HW/SW Architecture

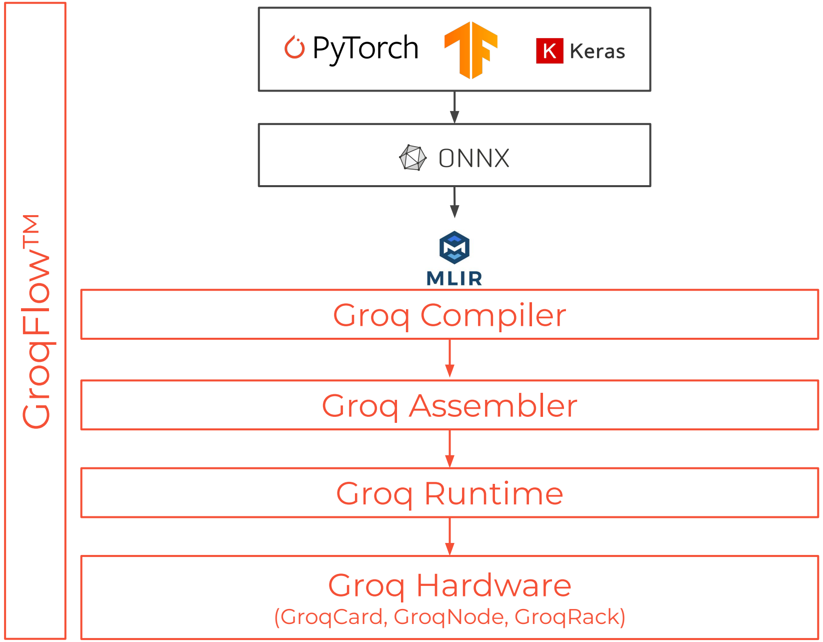
Simplified Groq Runtime Diagram

- A higher level software interface that runs on a **host CPU**.
- The runtime communicates to **Groq Hardware** using the **Groq Driver**, over a **PCIe** interface
- Deals with information inside of our compiled **.iop** files



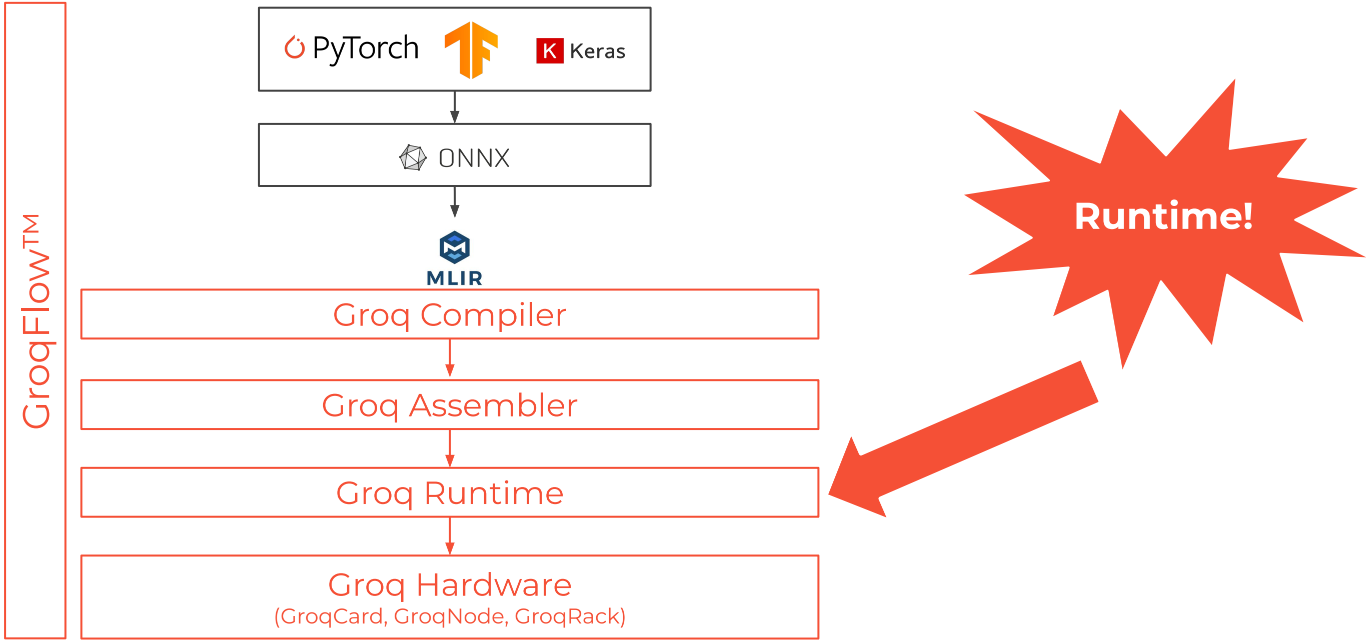
Groq Runtime HW/SW Architecture

Simplified GroqFlow Software to Hardware Diagram



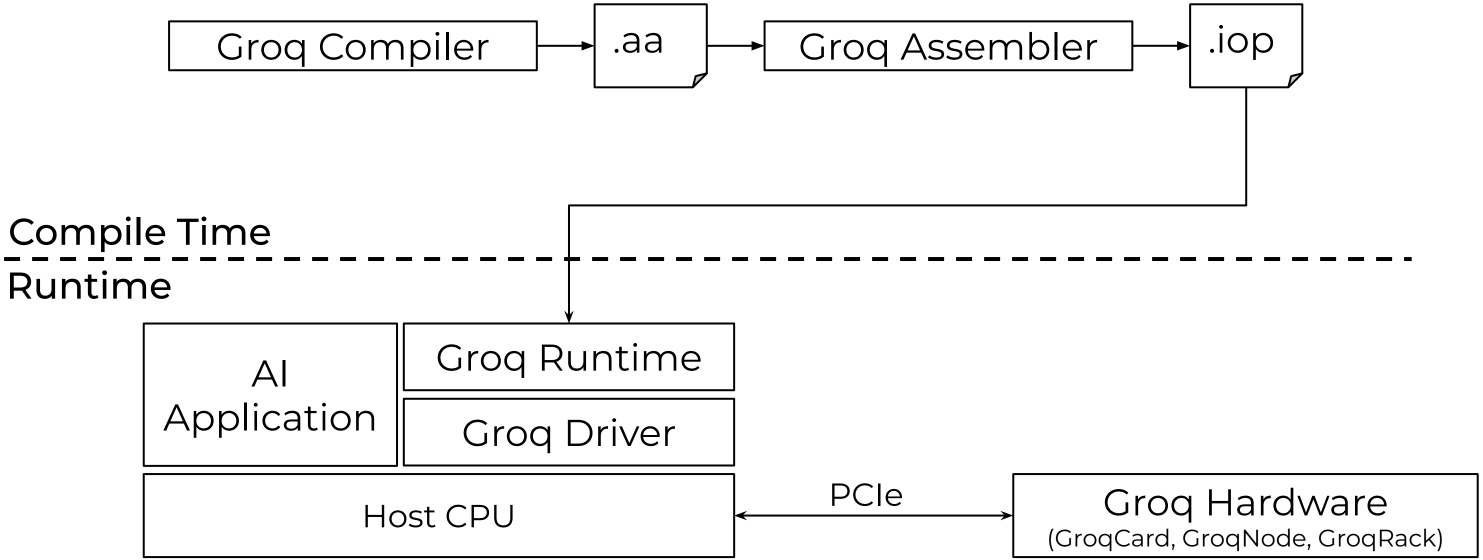
Groq Runtime HW/SW Architecture

Simplified GroqFlow Software to Hardware Diagram



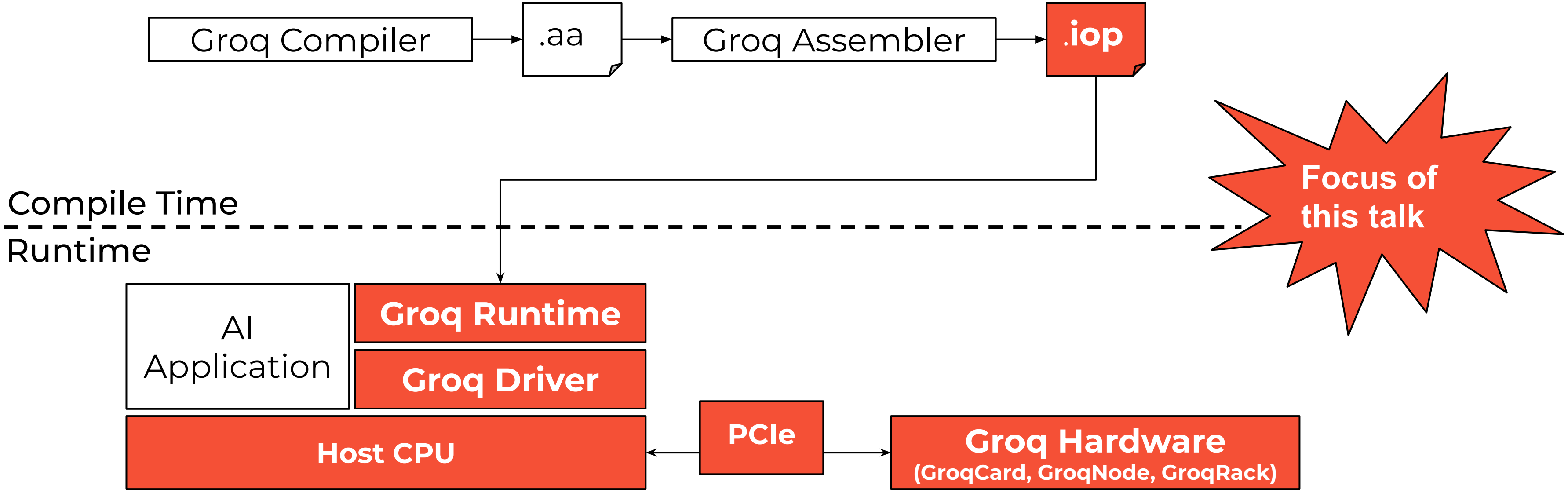
Groq Runtime HW/SW Architecture

Simplified Groq Runtime Diagram



Groq Runtime HW/SW Architecture

Simplified Groq Runtime Diagram



Groq Runtime HW/SW Architecture

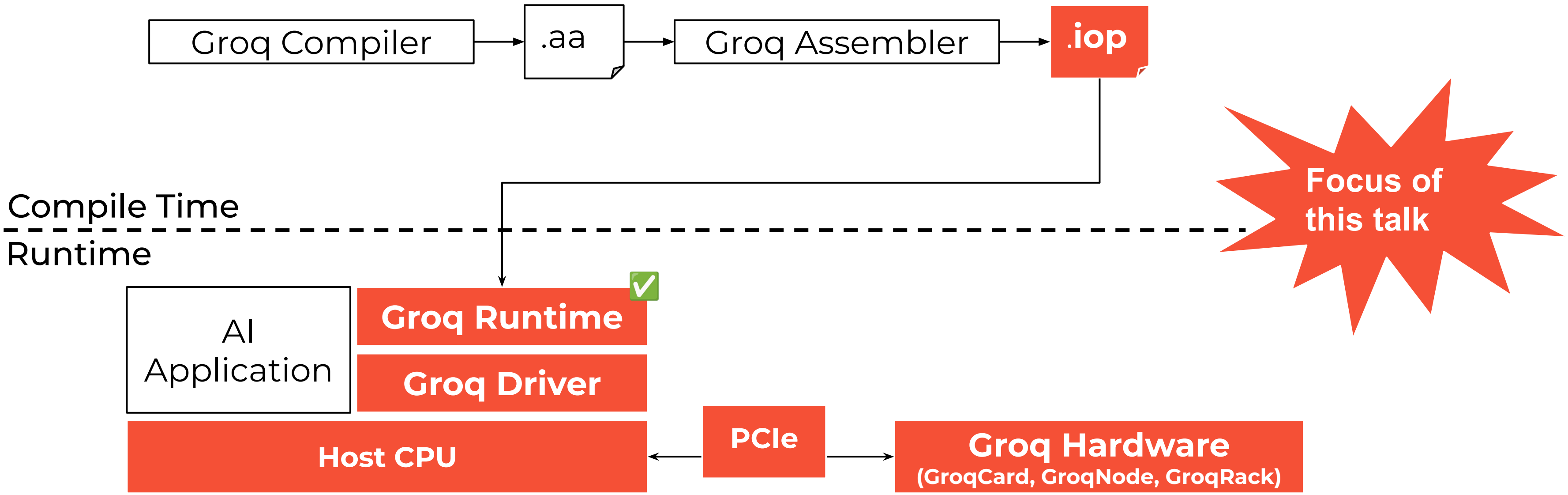
Groq Runtime

- Higher level software interface to Groq hardware
- Has an “idea” of what an .iop is and contains.
- Runtime includes code for:
  - Parsing IOP files
  - Initializing the chip
  - Allocating input and output host buffers
  - Loading and invoking programs
- C++ and Python based implementations.

Groq Runtime

Groq Runtime HW/SW Architecture

Simplified Groq Runtime Diagram


Groq Compiler

.aa

Groq Assembler

.iop

Compile Time

Runtime

AI Application

Groq Runtime ✅

Groq Driver

Host CPU

PCIe

Groq Hardware

(GroqCard, GroqNode, GroqRack)

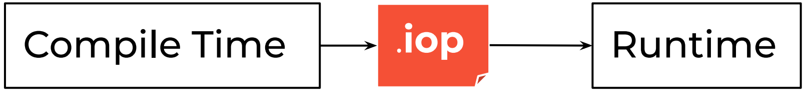
Focus of this talk

Groq Runtime HW/SW Architecture

Input/Output Package File (.iop) Format

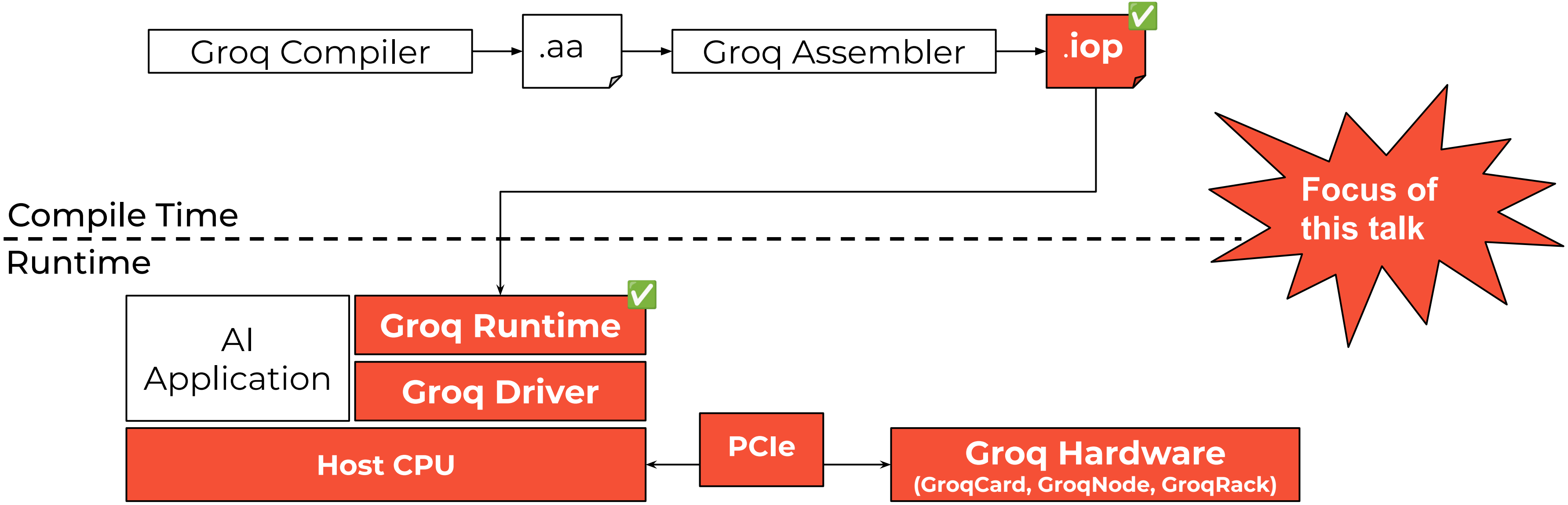
- Groq's representation of an executable for GroqChip
- Emitted by the Groq Assembler/Groq Compiler
- Protobuf container that contains information on:
  - Model instructions and weights
  - Instructions on how to load the GroqChip's SRAM.
  - Model Input/Output tensor information
  - Debug Metadata

.iop



Groq Runtime HW/SW Architecture

Simplified Groq Runtime Diagram



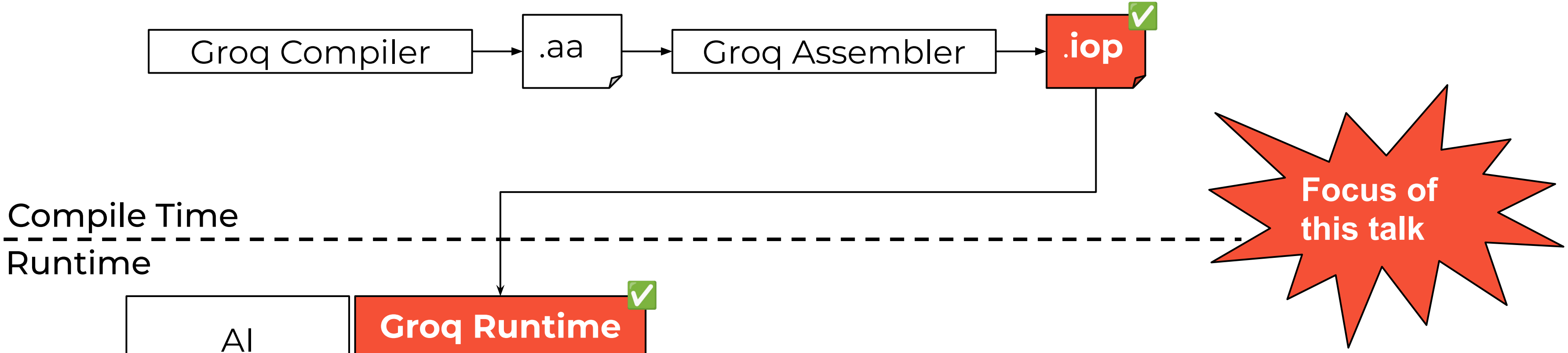
Groq Runtime HW/SW Architecture

Groq Driver

- Low-level PCIe hardware interface
  - DMA data transfers to/from GroqChip
  - CSR reads/writes
- Based on a simple Linux user-space VFIO driver
- Lowest level between how the host CPU and Groq LPU communicate with each other

Groq Runtime HW/SW Architecture

Simplified Groq Runtime Diagram



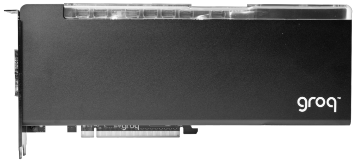
Focus of this talk

Groq Runtime HW/SW Architecture

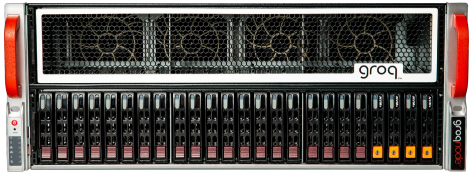
Groq Hardware

- GroqCard
  - 1 Groq LPU Chip
- GroqNode
  - 8 GroqCards per GroqNode
- GroqRack
  - 9 GroqNodes per GroqRack
  - Total of 72 GroqChips

Groq LPU™



GroqCard™

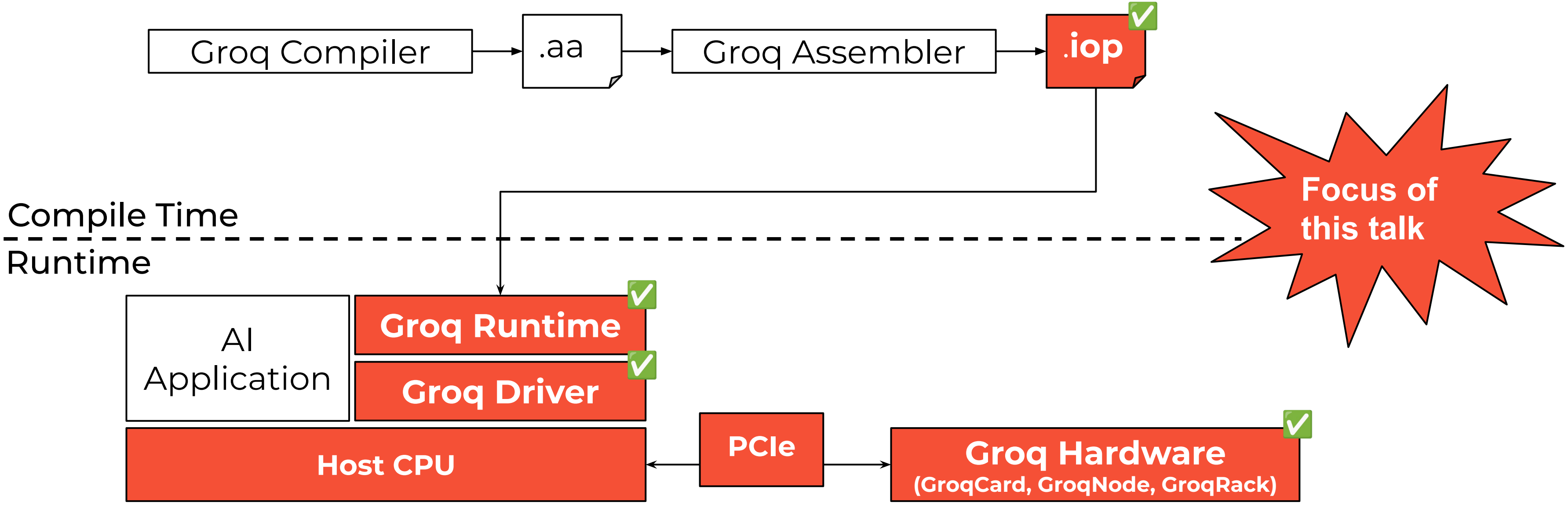


GroqNode™

GroqRack™

Groq Runtime HW/SW Architecture

Simplified Groq Runtime Diagram



Groq Runtime HW/SW Architecture

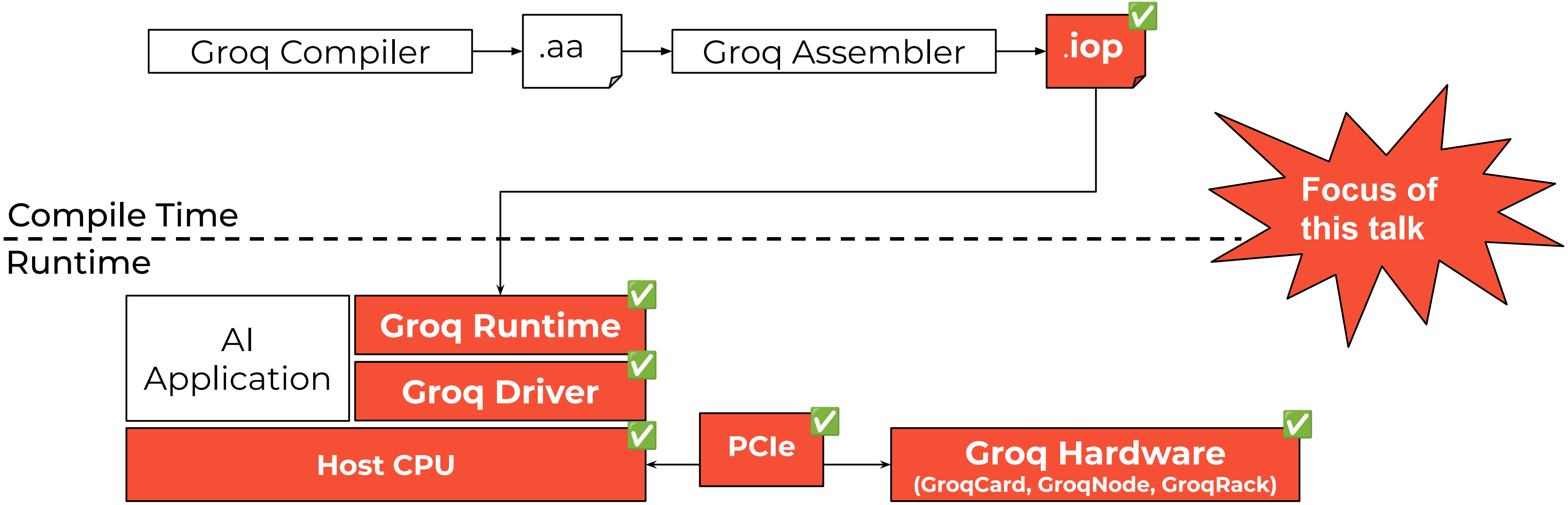
Host CPU and PCIe Connection

- Host CPU
  - x86 server CPU
- PCIe
  - Gen 4x16

Host CPU
PCIe

Groq Runtime HW/SW Architecture

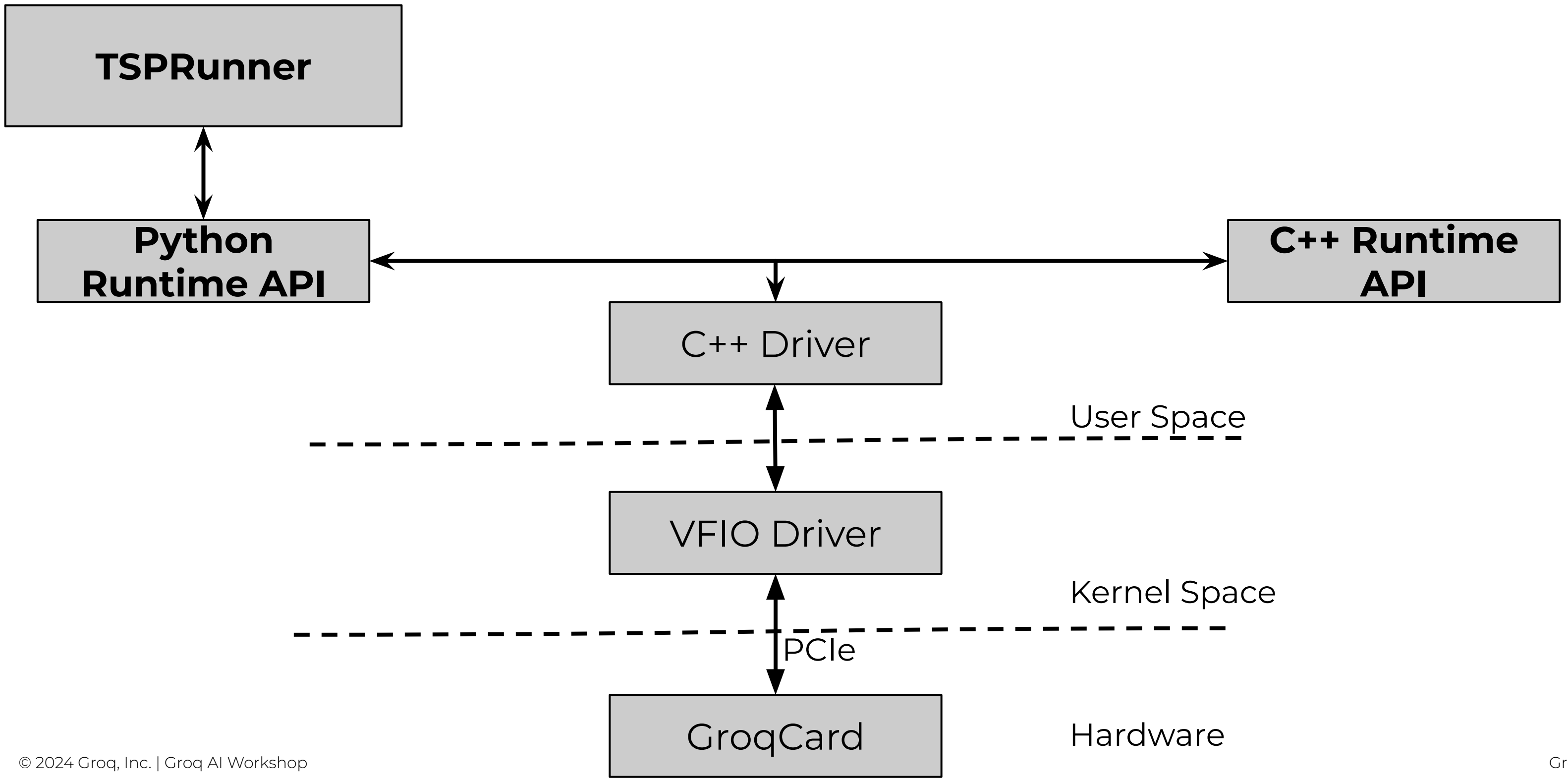
Simplified Groq Runtime Diagram



Interacting with Groq Runtime as a Developer

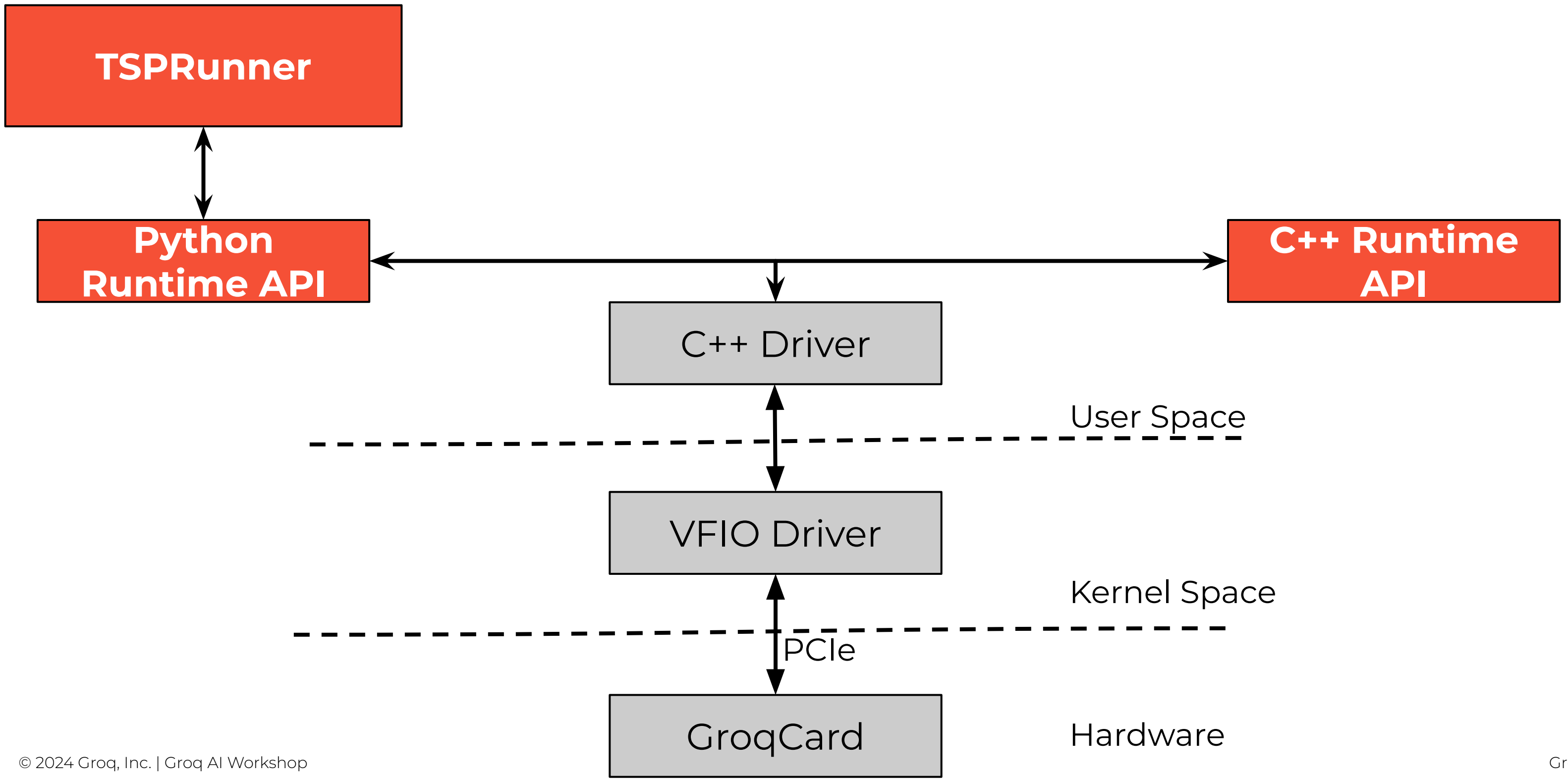
Interacting with Groq Runtime as a Developer

Groq runtimes available to developers



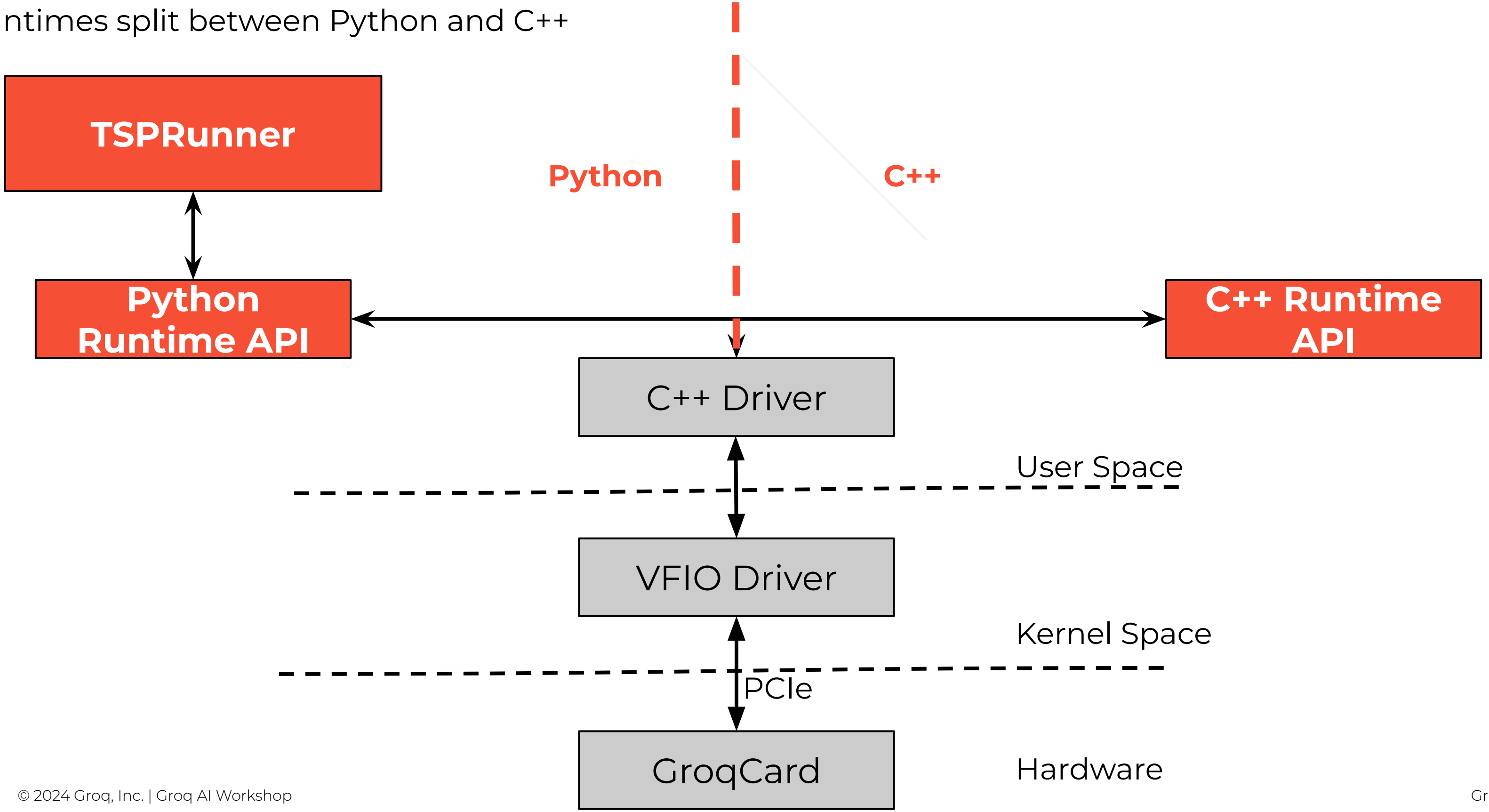
Interacting with Groq Runtime as a Developer

Groq runtimes available to developers



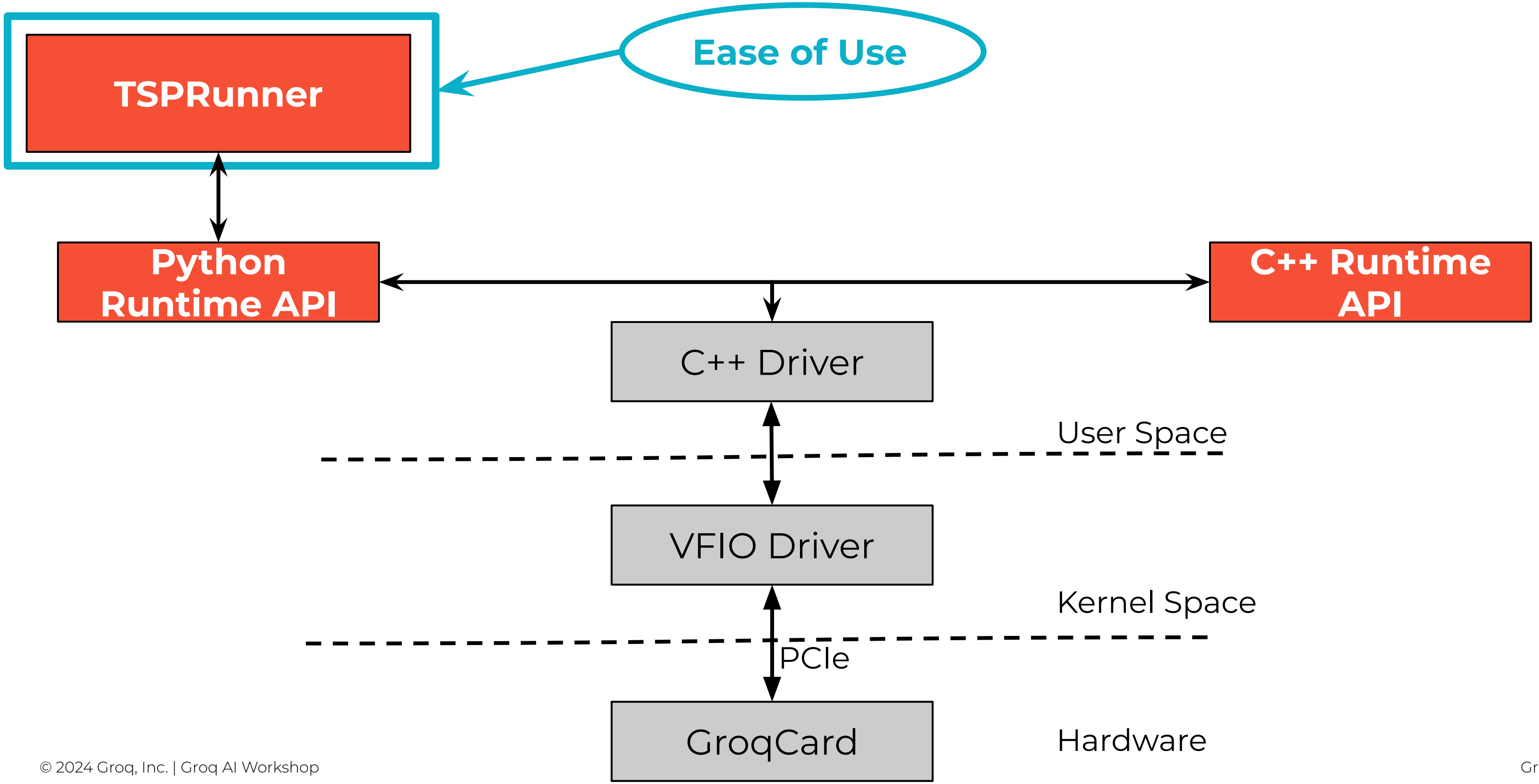
Interacting with Groq Runtime as a Developer

Groq runtimes split between Python and C++



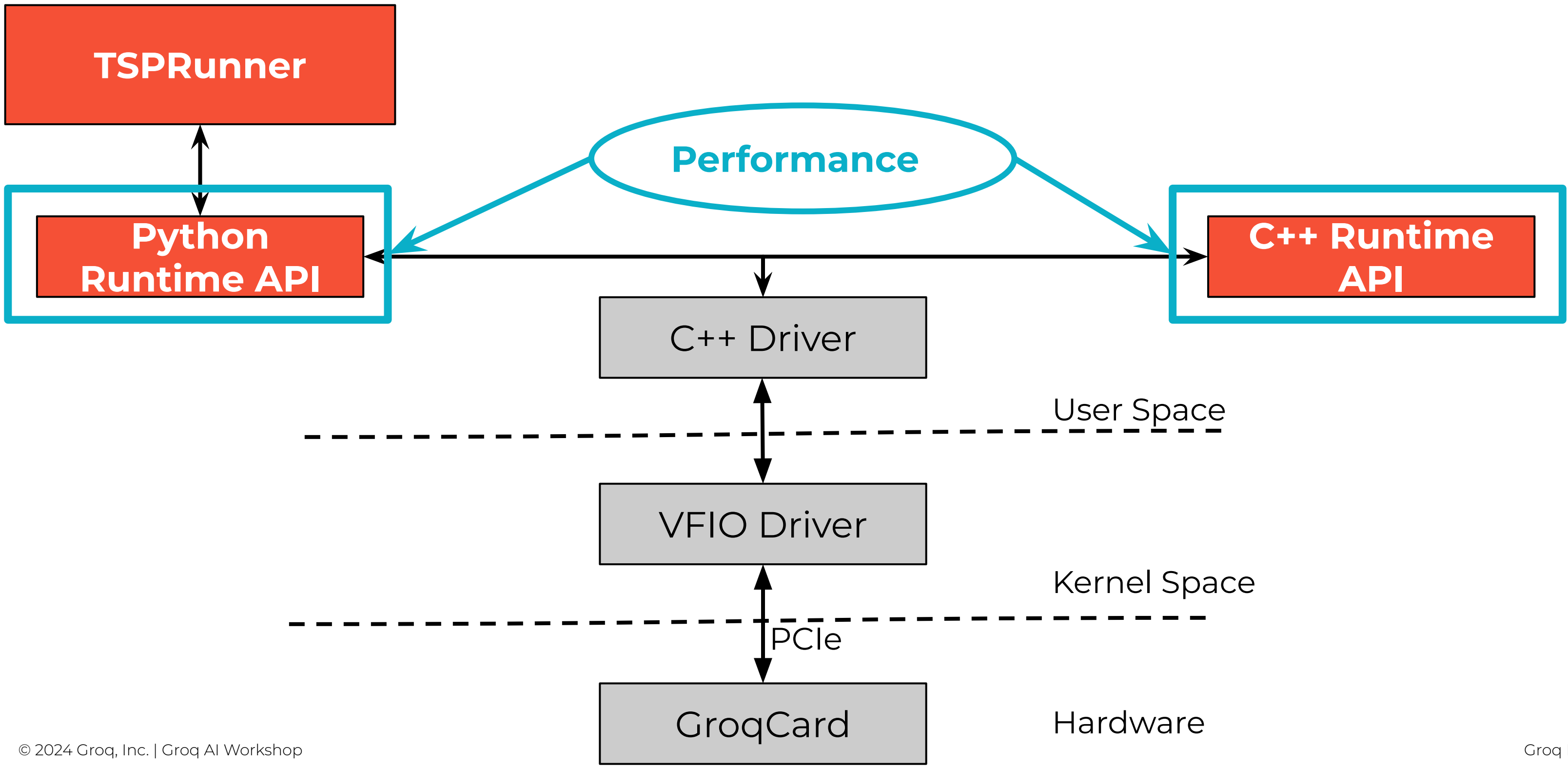
Interacting with Groq Runtime as a Developer

Ease of use oriented Groq runtimes



Interacting with Groq Runtime as a Developer

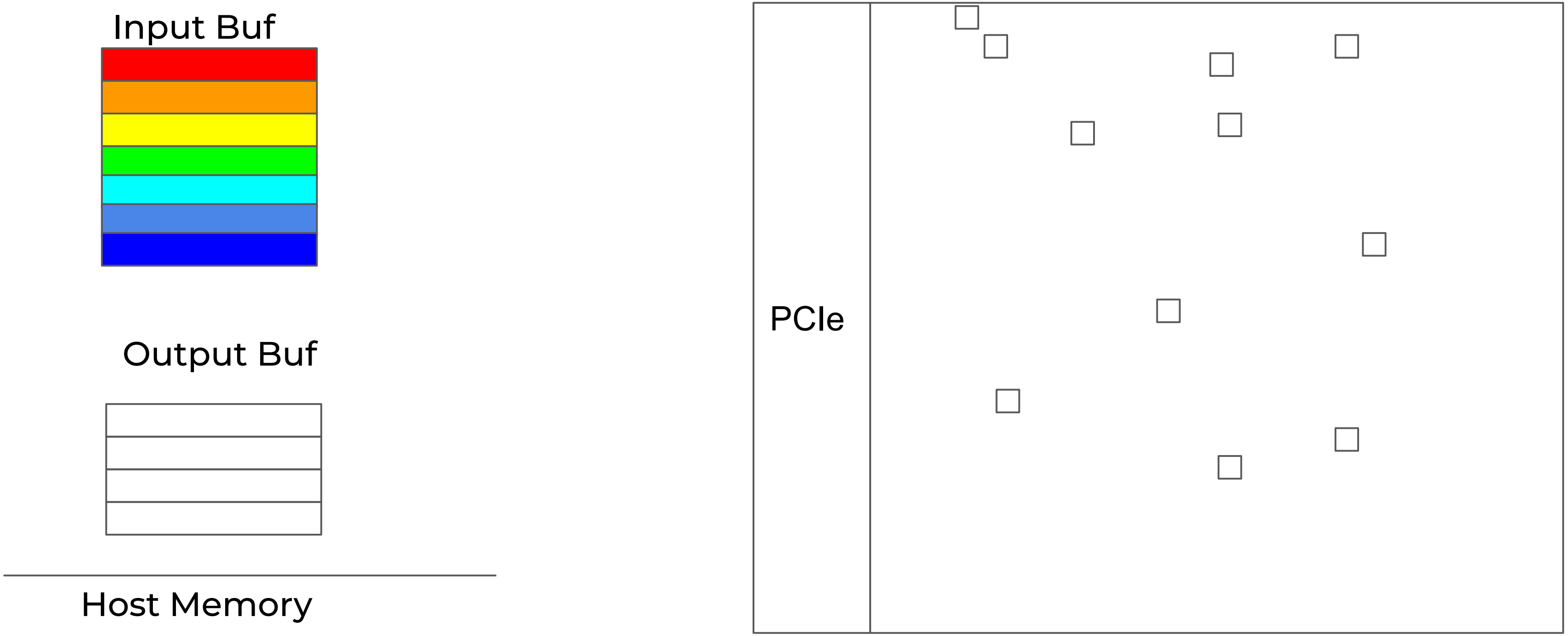
Performance oriented Groq runtimes



Deeper Dive on Running Inferences on a GroqChip!

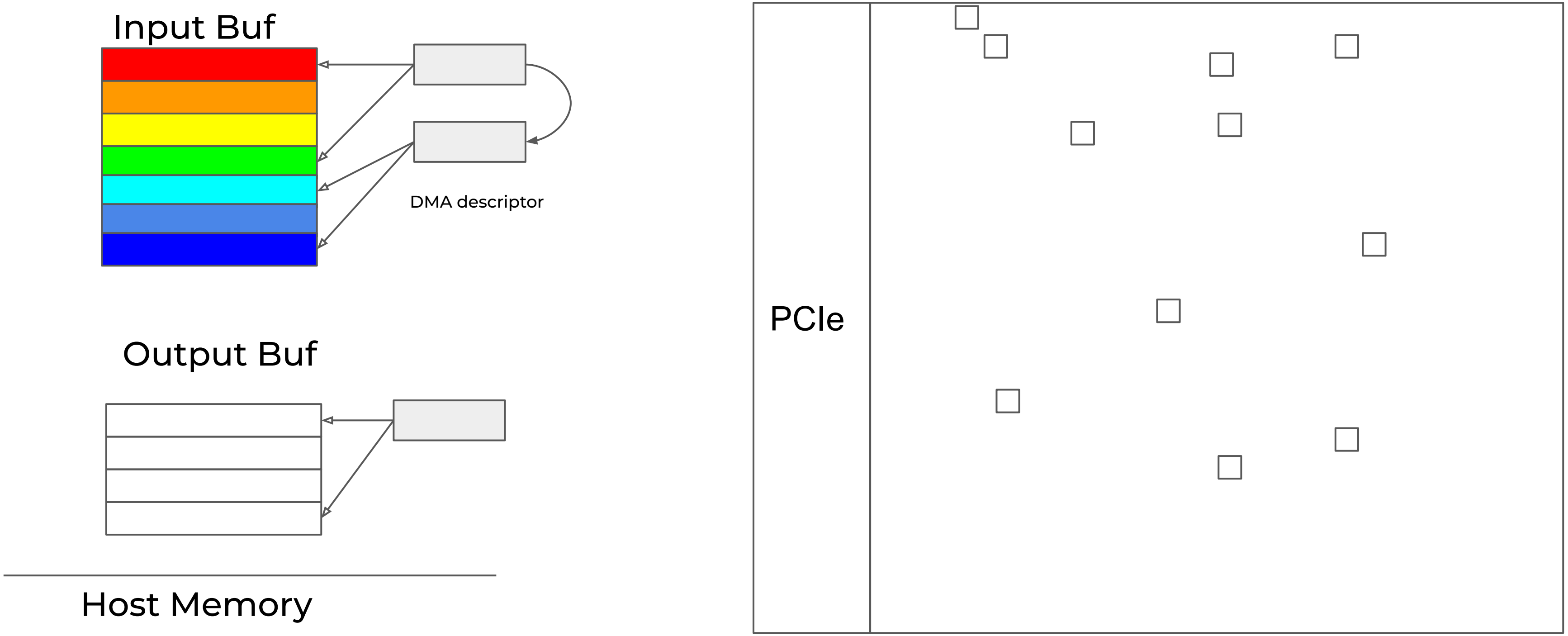
Deeper Dive on Running Inferences on a GroqChip!

Moving Data between Host CPU and Groq LPU



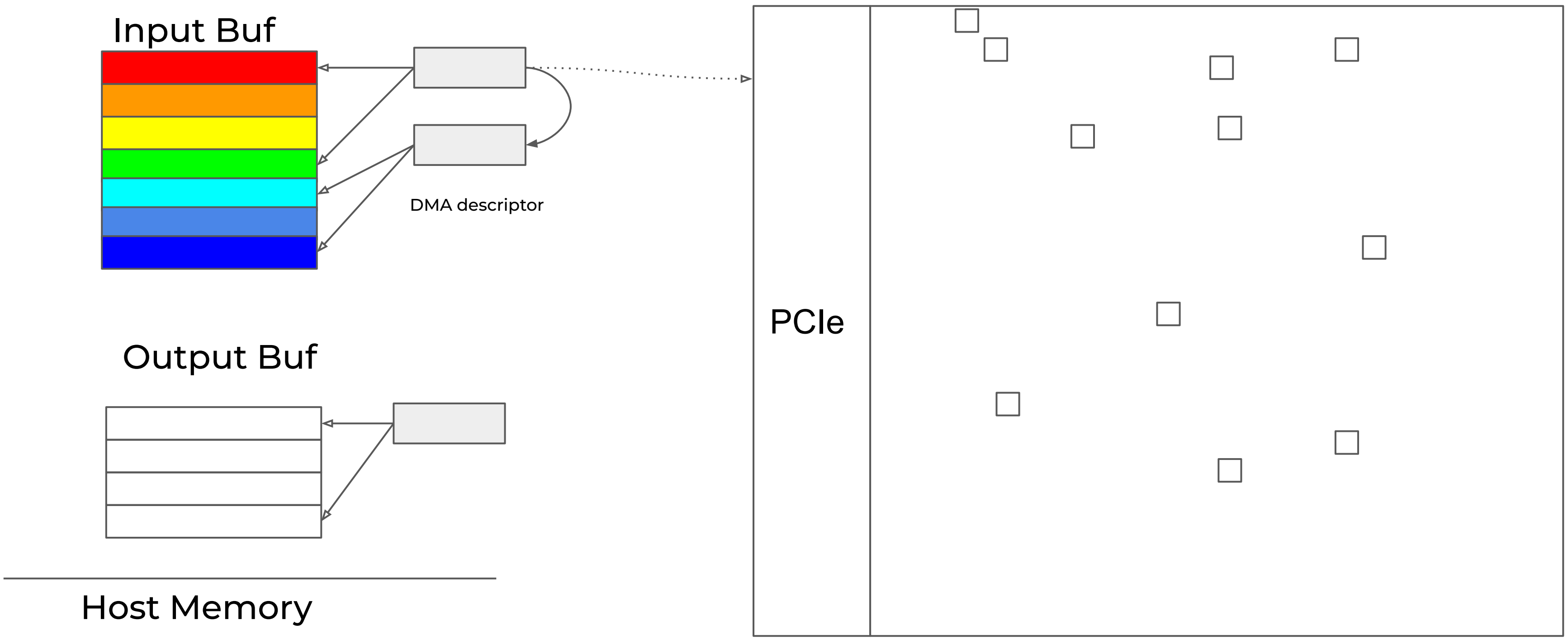
Deeper Dive on Running Inferences on a GroqChip!

DMA descriptor maps host memory buffer



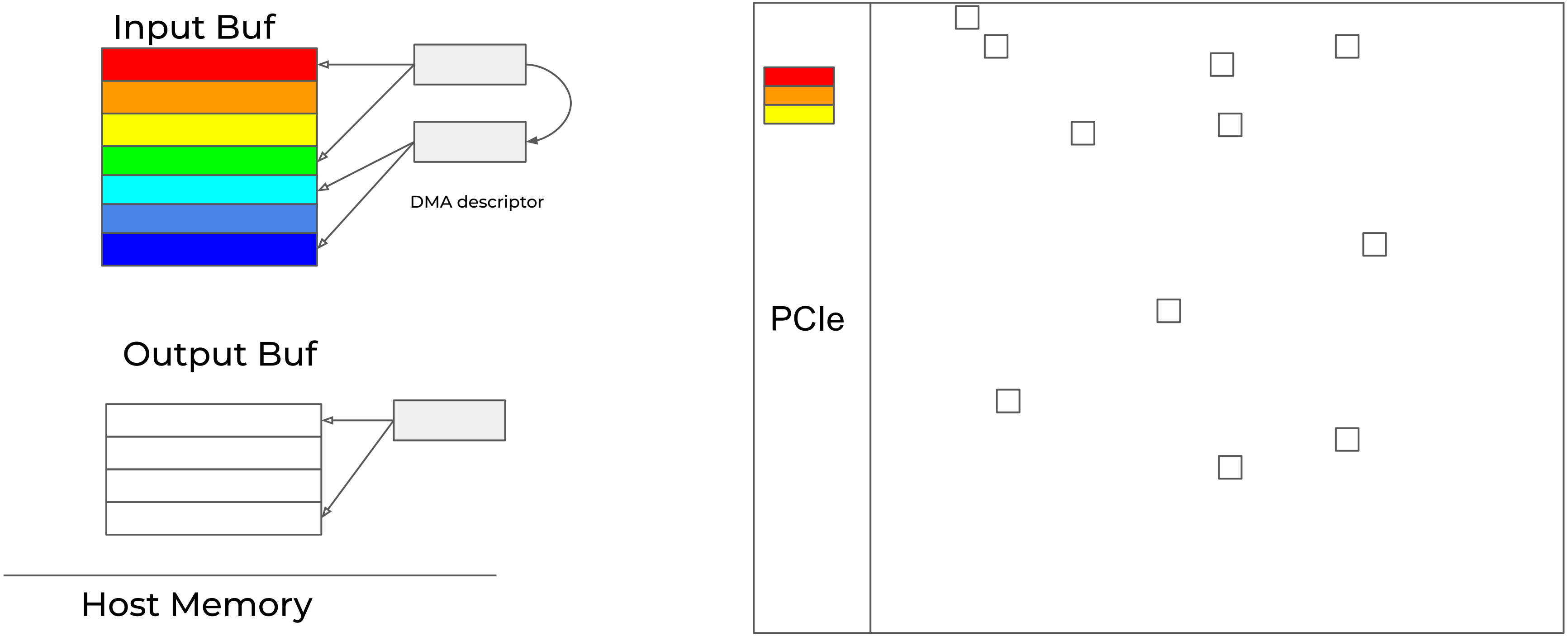
Deeper Dive on Running Inferences on a GroqChip!

Driver writes descriptor address to PCIe RX BAR



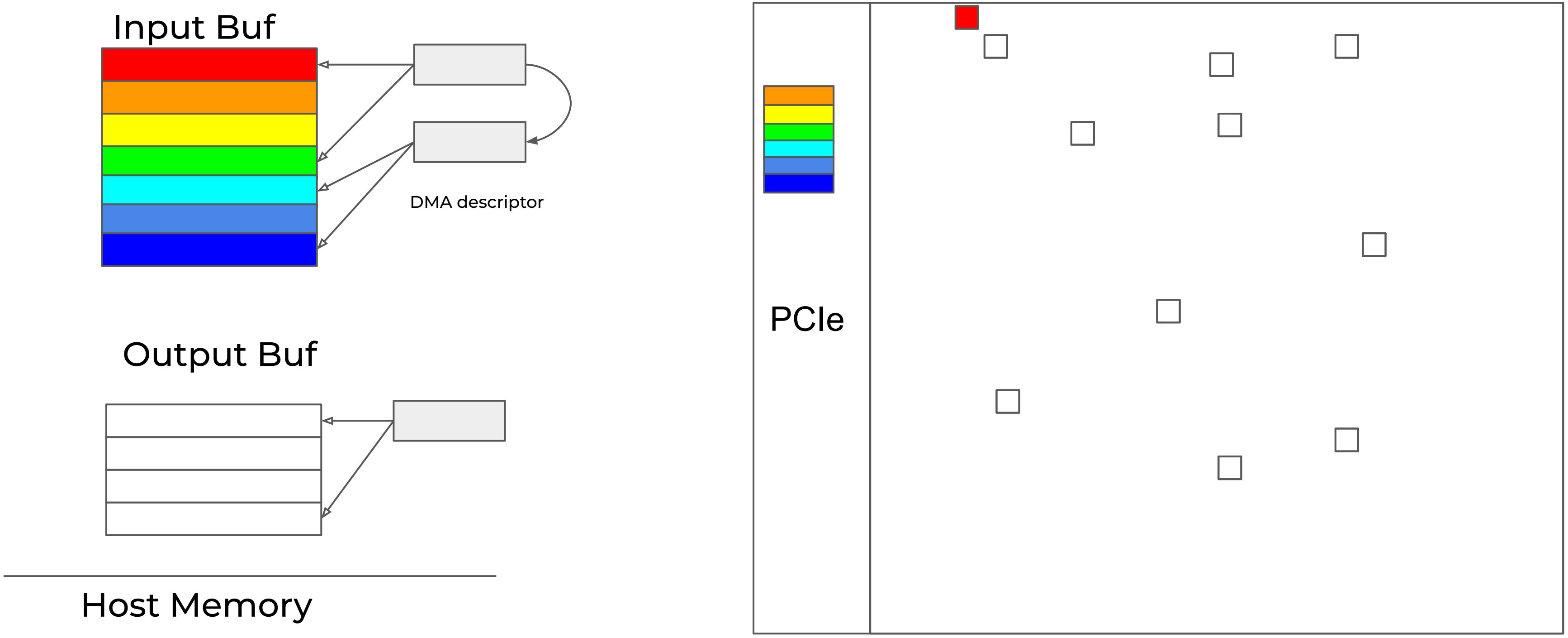
Deeper Dive on Running Inferences on a GroqChip!

PCIe block retrieves descriptor/underlying buffer data, fills FIFO



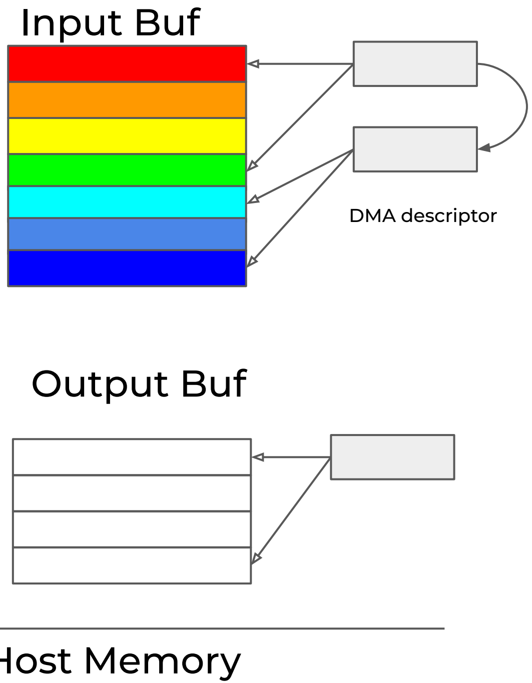
Inferences on Groq LPU

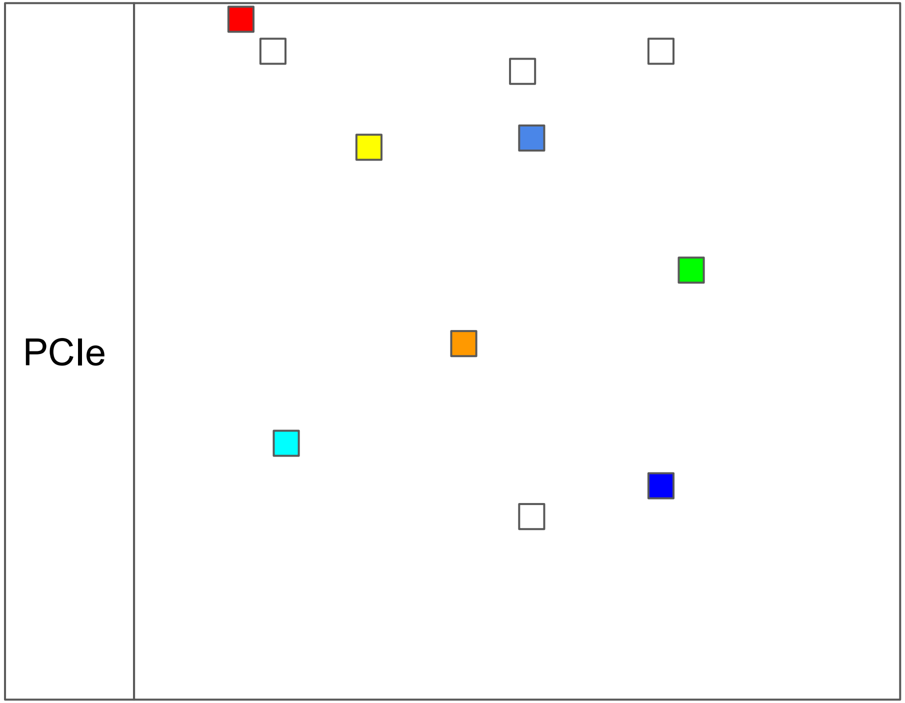
PCIe block retrieves descriptor/underlying buffer data, fills FIFO



Deeper Dive on Running Inferences on a GroqChip!

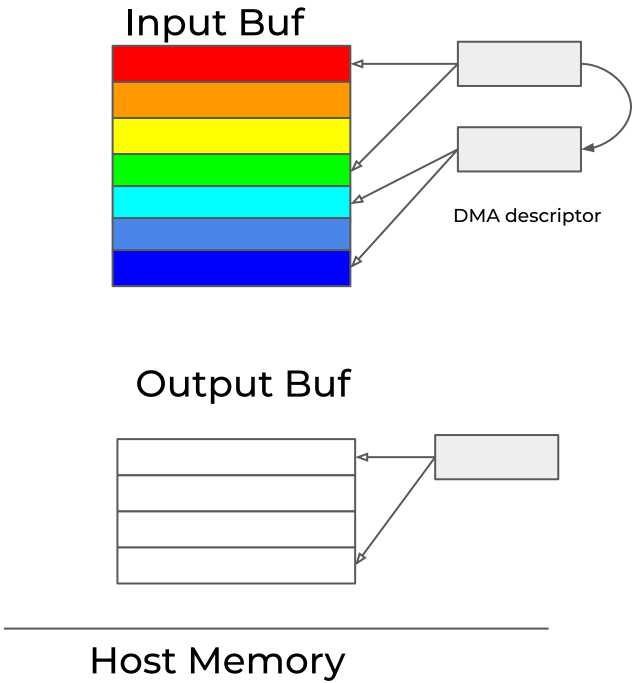
I/O harness fills all of SRAM inputs

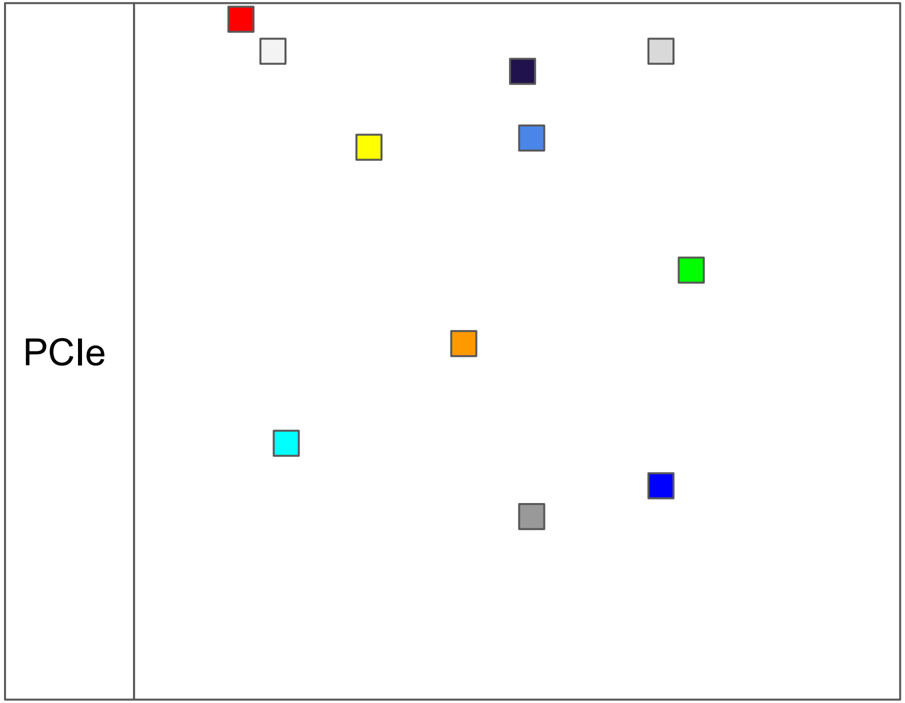




Deeper Dive on Running Inferences on a GroqChip!

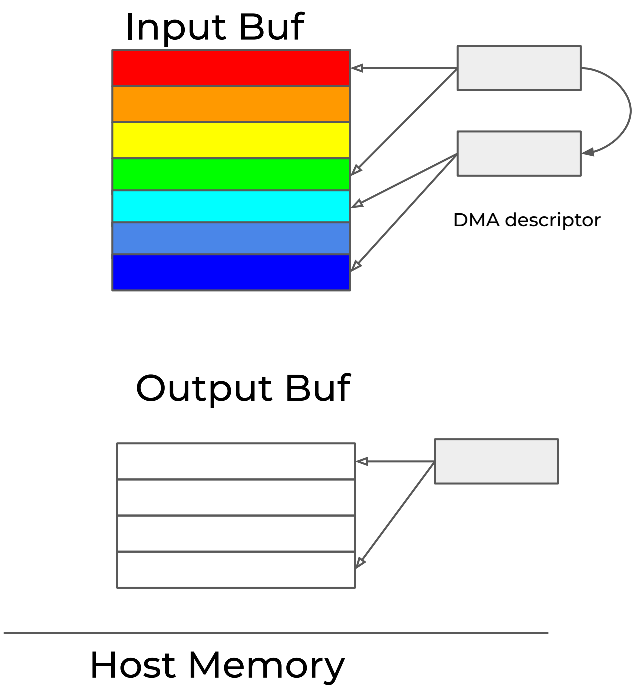
Moving Data between Host CPU and Groq LPU

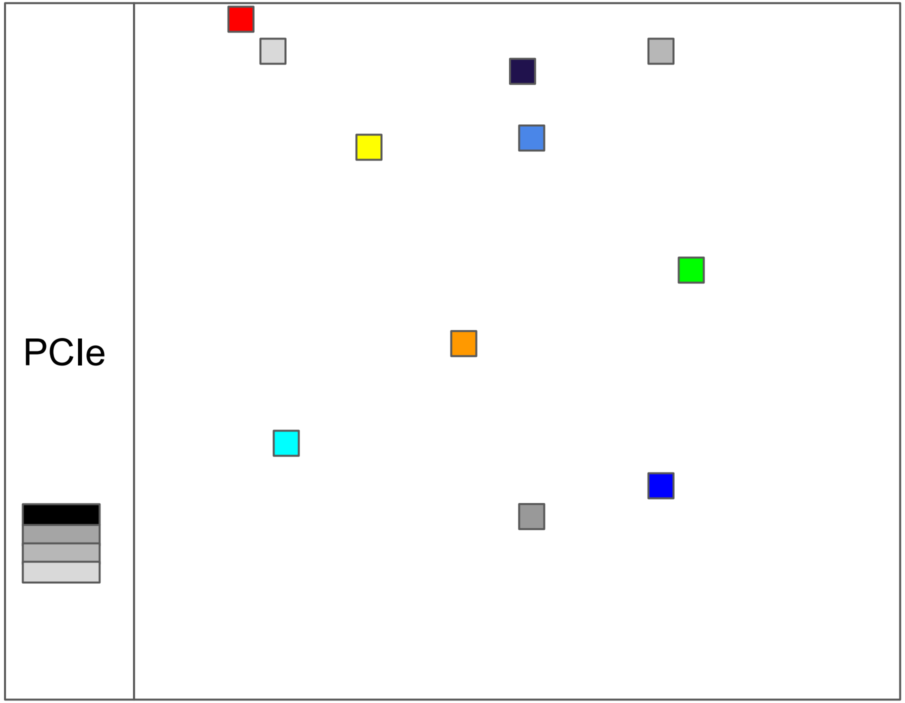




Deeper Dive on Running Inferences on a GroqChip!

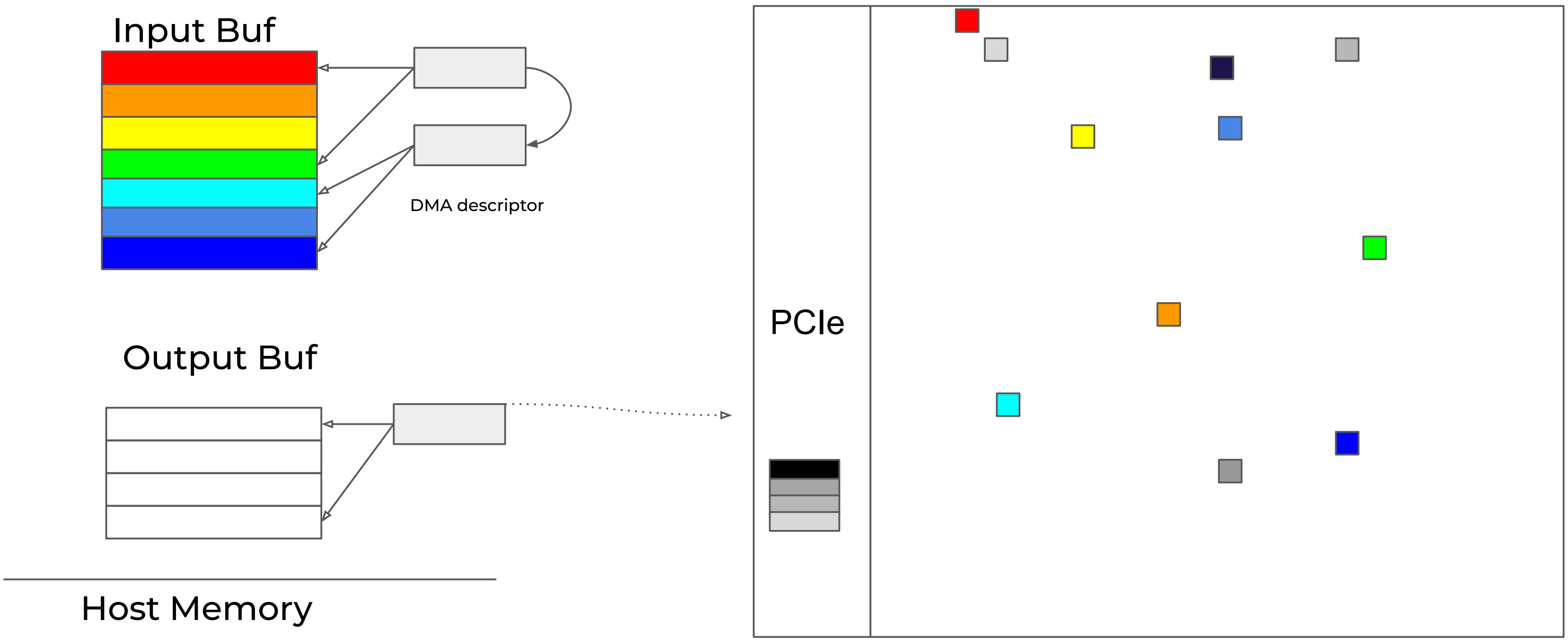
Initiate core compute and PCIe TX ICU reads vectors from SRAM and pushes to FIFO





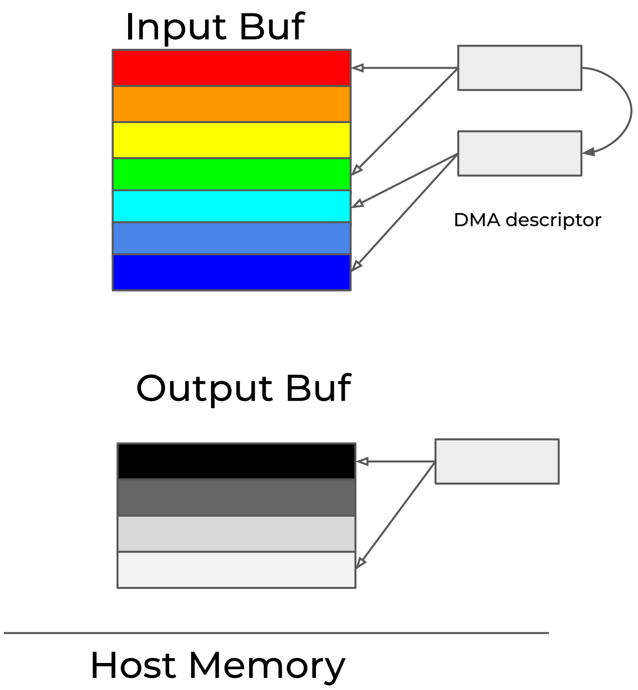
Deeper Dive on Running Inferences on a GroqChip!

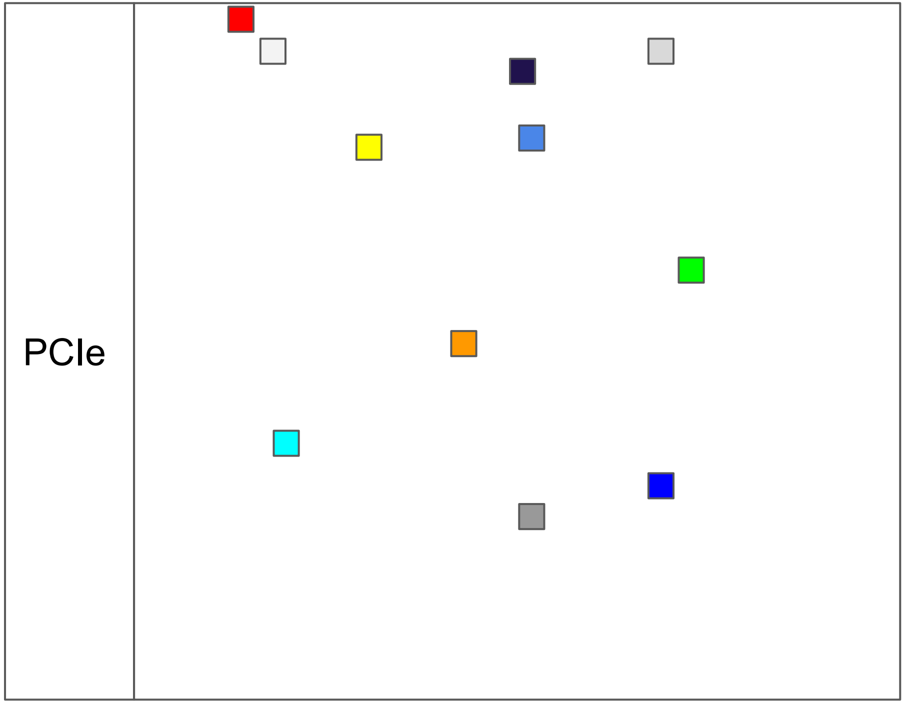
Driver writes descriptor address to PCIe TX BAR



Deeper Dive on Running Inferences on a GroqChip!

PCIe block drains FIFO, writes results back to host memory





LLMs with Groq

Aviv Weinstein
Systems Software Engineer

LLMs: The next Revolution in Computing

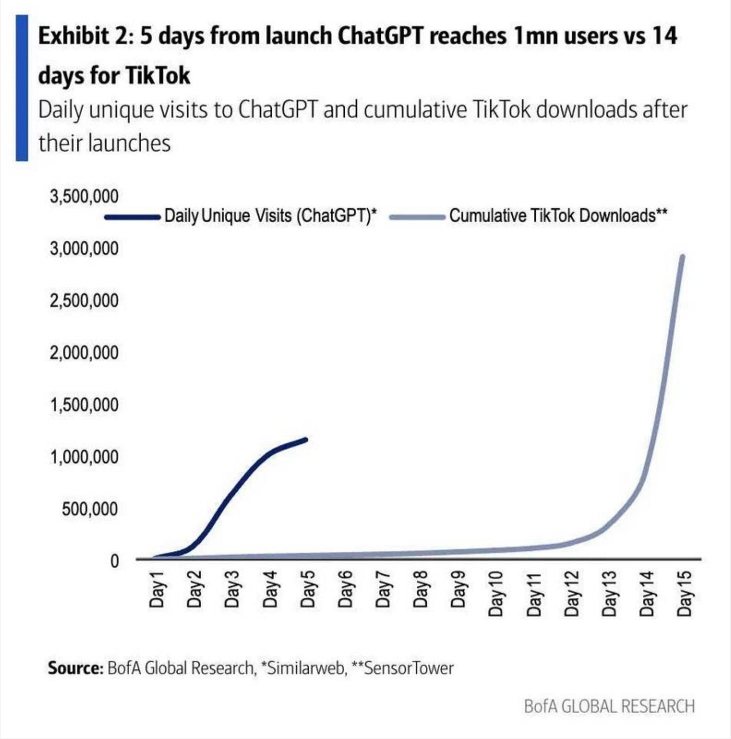
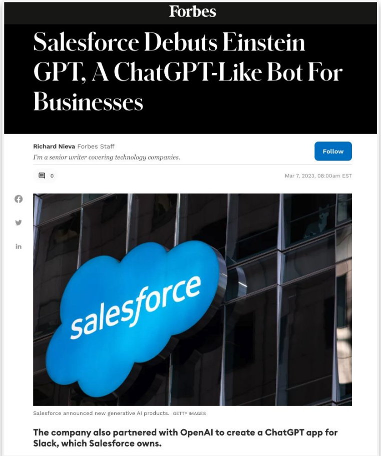


Exhibit 2: 5 days from launch ChatGPT reaches 1mn users vs 14 days for TikTok

Daily unique visits to ChatGPT and cumulative TikTok downloads after their launches

Source: BofA Global Research, *Similarweb, **SensorTower

BofA GLOBAL RESEARCH



Forbes

Salesforce Debuts Einstein GPT, A ChatGPT-Like Bot For Businesses

Richard Nieva Forbes Staff

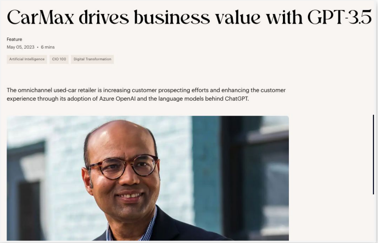
I'm a senior writer covering technology companies.

Mar 7, 2023, 08:00am EST

Salesforce announced new generative AI products. GETTY IMAGES

The company also partnered with OpenAI to create a ChatGPT app for Slack, which Salesforce owns.

Source: forbes.com

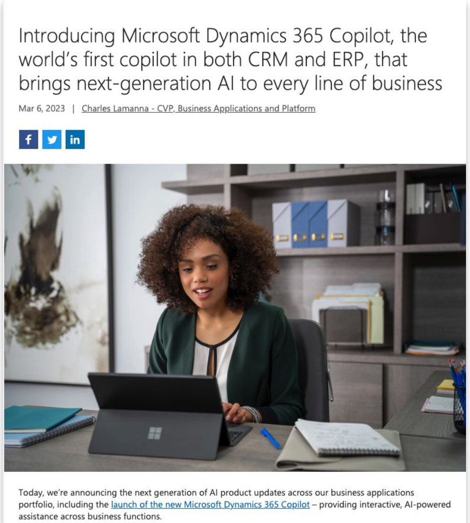


CarMax drives business value with GPT-3.5

Feature

May 05, 2023 • 6 mins

The omnichannel used-car retailer is increasing customer prospecting efforts and enhancing the customer experience through its adoption of Azure OpenAI and the language models behind ChatGPT.



Introducing Microsoft Dynamics 365 Copilot, the world's first copilot in both CRM and ERP, that brings next-generation AI to every line of business

Mar 6, 2023 | Charles Lamanna - CVP, Business Applications and Platform

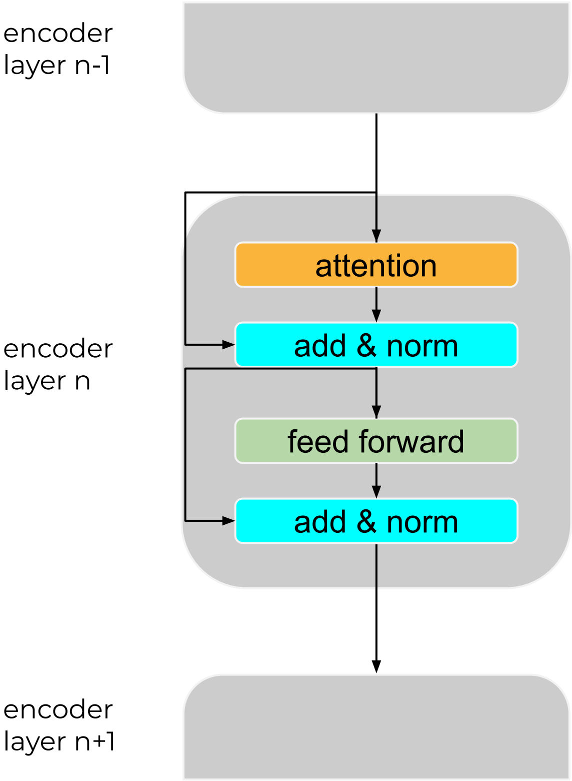
Today, we're announcing the next generation of AI product updates across our business applications portfolio, including the launch of the new Microsoft Dynamics 365 Copilot – providing interactive, AI-powered assistance across business functions.

Source: blogs.microsoft.com

LLMs in Science

- Rapid development of LLMs and related technologies
- Groq offers fastest LLMs to date
- LLMs are applicable for a wide range of scientific applications
- Two possible approaches:
  - Use general LLMs to assist scientific method
  - Specialised LLMs encode sequences from chemistry, biology or physics

Why Groq LPUs are suitable for running LLMs



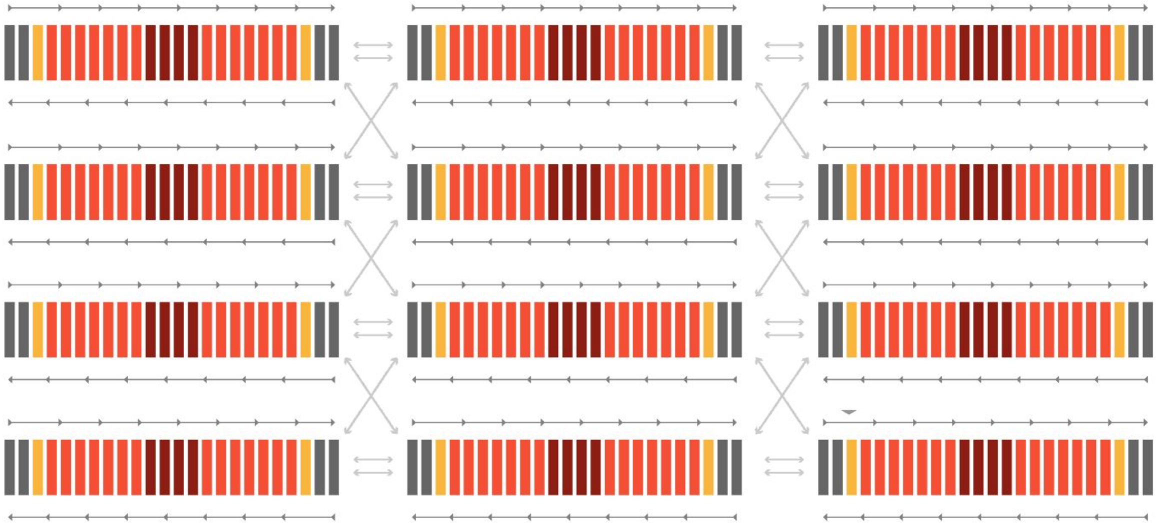
- The large matrix multiplication operations are effectively mapped to MXM
- Running LLMs is a serial problem - it requires generating the first 99 tokens before the 100th one (auto-regressive behaviour). This requires a lot of weights loading which is accelerated by LPU’s high SRAM bandwidth.

Groq LPUs connect to form one large Assembly Line

No switches required to connect LPUs

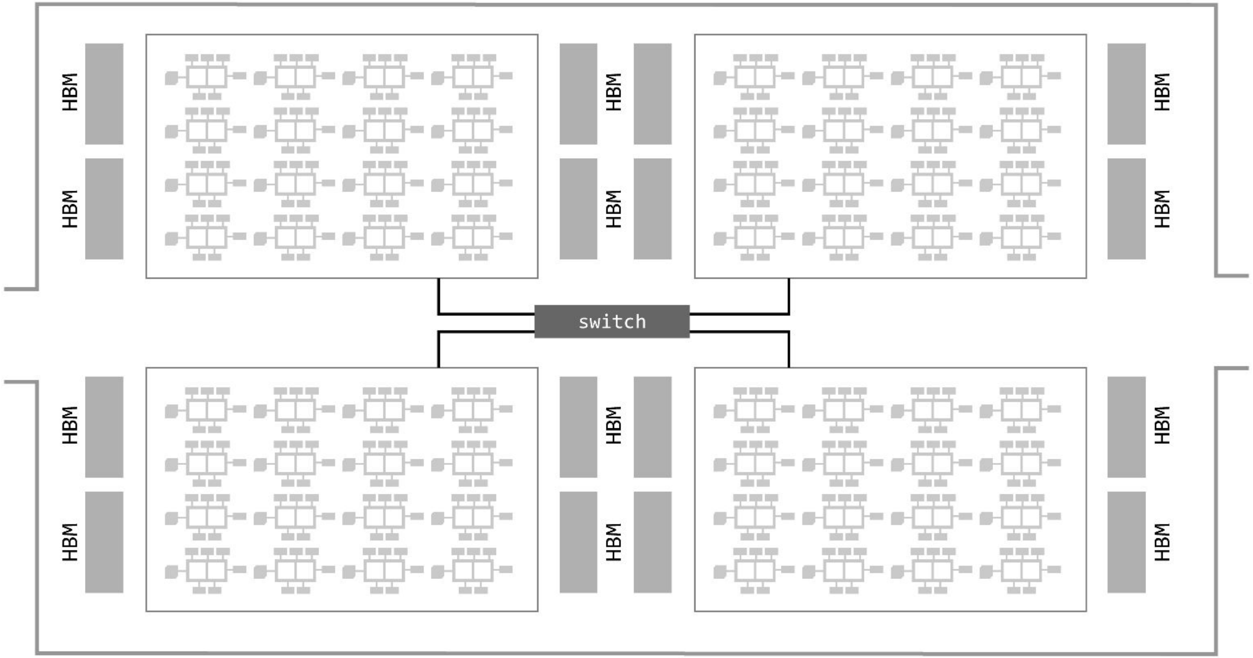
C2C between LPUs act like conveyor belts between them

Statically scheduled networking - no congestion, even under heavy load



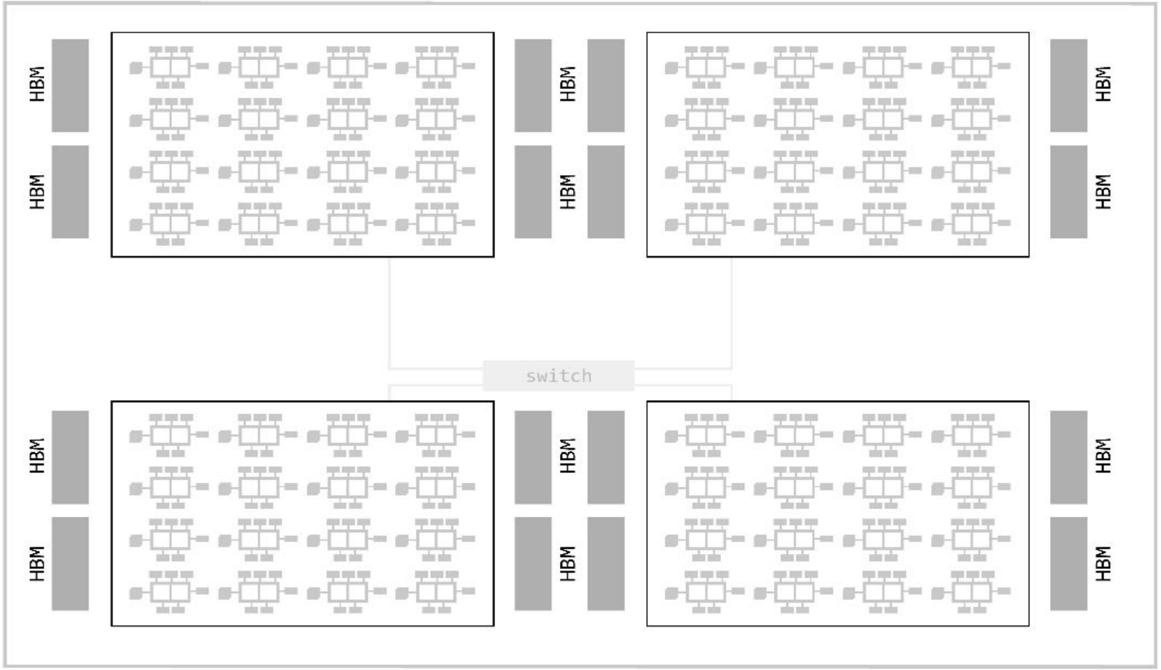
How is this different from a GPU?

GPU system - collection of GPUs and switches



GPUs scale largely in time, some in space (clustering)

For large compute volume, GPUs iterate over multiple partitions of model code, weights etc in time



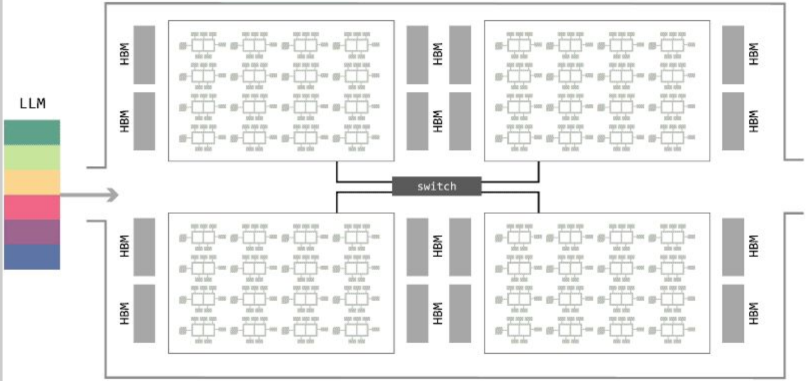
Groq LPU scales largely in space as an assembly line

For large compute volume, LPUs partition model code across multiple LPUs to form an assembly line

GPU

GPU cores and systems are “hub-and-spoke” architecture

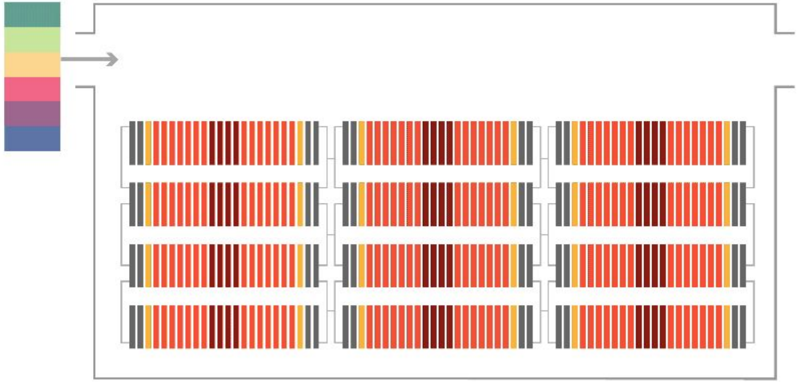
- GPUs scale largely in time, some in space (clustering)-GPUs iterate over multiple partitions of model code, weights, etc. in time
- Uses expensive & supply constrained components - HBMs, interposer, switches
- Inference performance (token/s/GPU & token/s/user) is limited by HBM BW
- Out-of-the-box compiler has poor HW utilization - requires hand-coded kernels



LPU

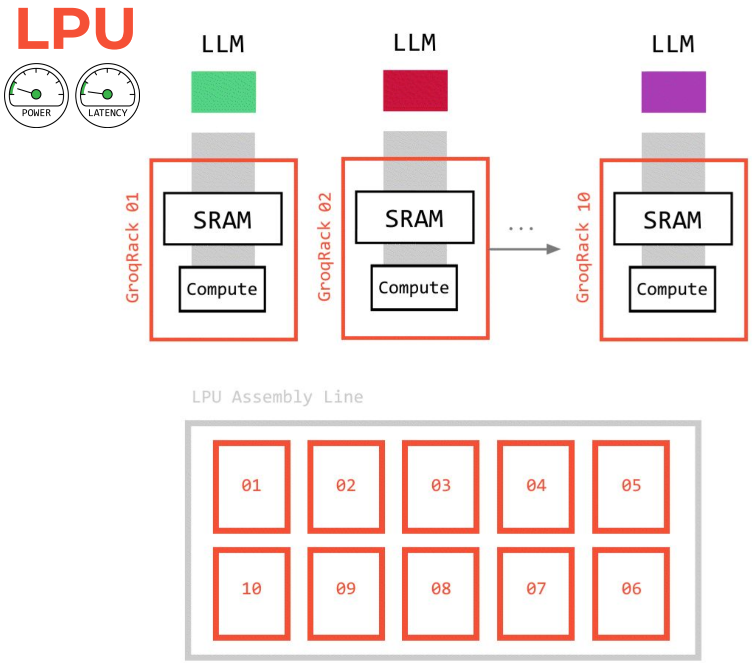
LPU and systems are programmable assembly line architecture

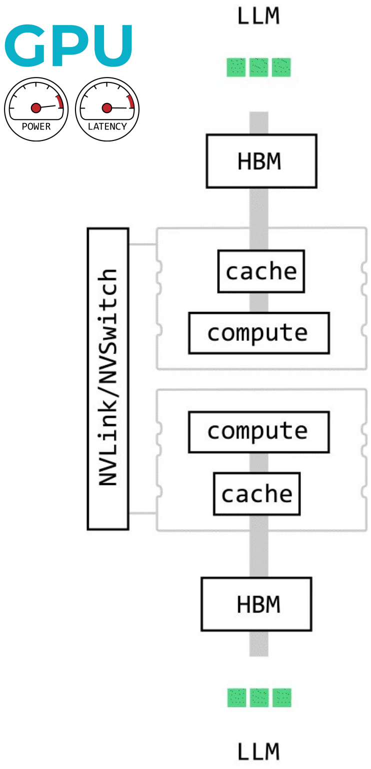
- Groq LPU scales largely in space as an assembly line - model code, weights etc are mapped across multiple chips
- Efficient assembly line arch is deterministic at clock granularity, uses high BW SRAM for fast compute - no HBMs, interposer etc
- Efficient assembly line enables high token/s & token/s/user at lower $ & watts
- Out-of-the-box compiler has good HW utilization



Welcome to the token factory

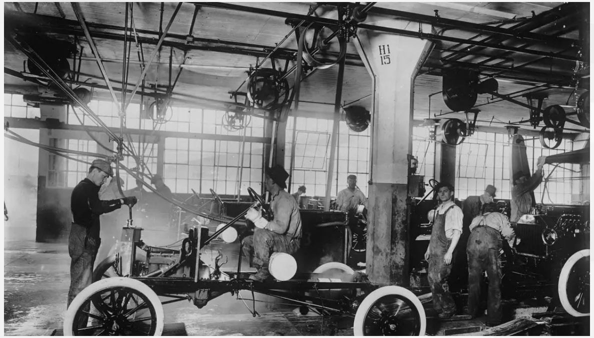
LPUs tightly connected to form a highly efficient assembly line v/s multiple independent small GPU shops for same capacity





Groq LPU™ Inference Engine

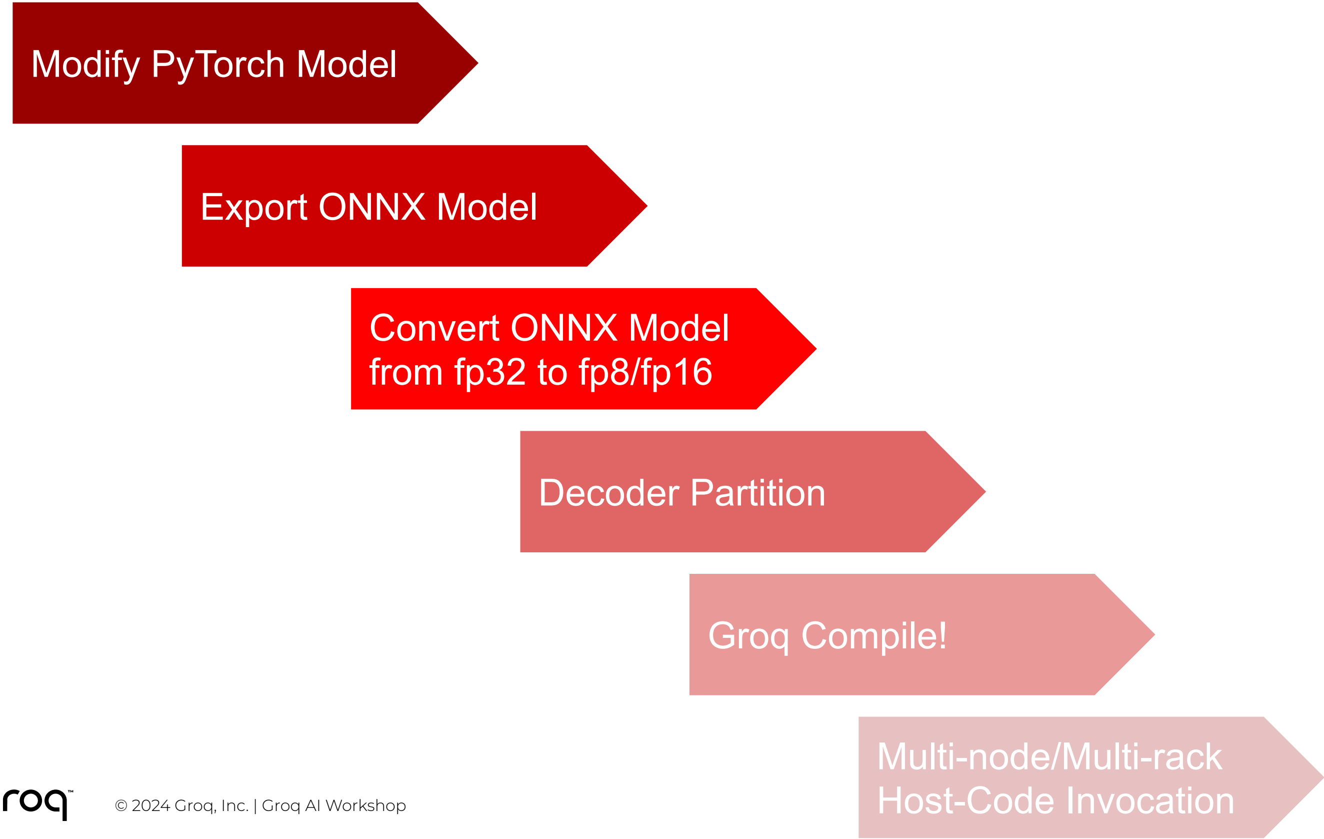
Scale by Design



You can build a car in one location **but** in volume it's quicker and cheaper to use an assembly line

You can compute an inference with a GPU that has a lot of external memory **but** at scale it's quicker and cheaper to use an LPU Inference Engine

General Groq LLM Development Flow



Modify PyTorch Model

Export ONNX Model

Convert ONNX Model from fp32 to fp8/fp16

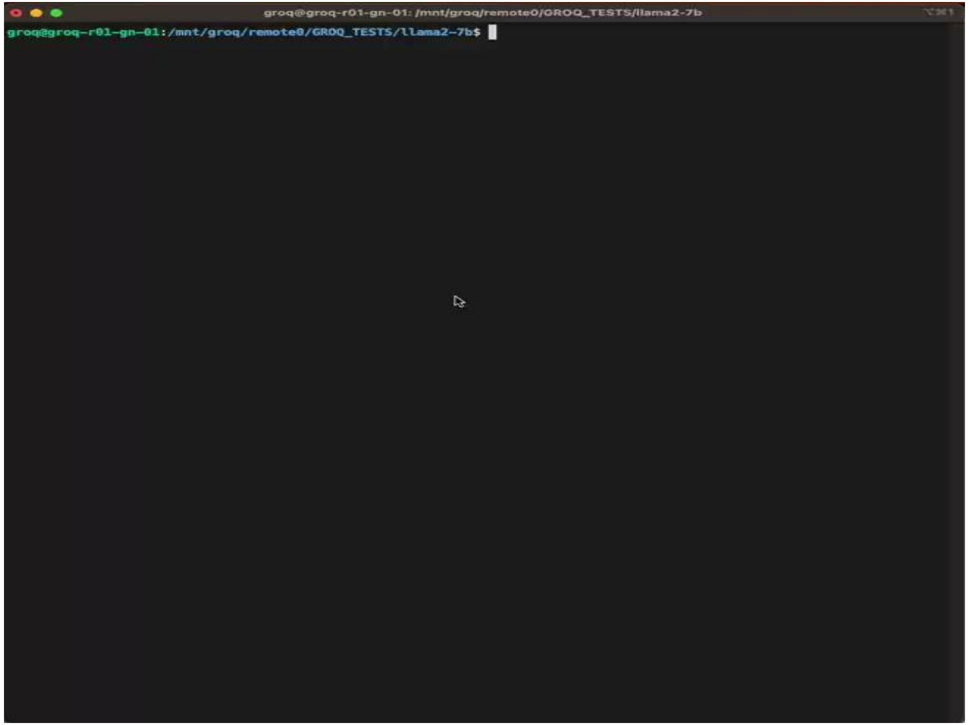
Decoder Partition

Groq Compile!

Multi-node/Multi-rack Host-Code Invocation

Llama-2 7B/2048

Optimized and available for the single GroqRack deployed at the ALCF AI Testbed



```
groq@groq-r01-gn-01:/mnt/groq/remote0/GROQ_TESTS/llama2-7b$
```

groq™

Thank You!

aweinstein@groq.com

Throughput Optimization with Groq™

Christopher Culver
Software Engineer

Throughput Optimization with Groq™

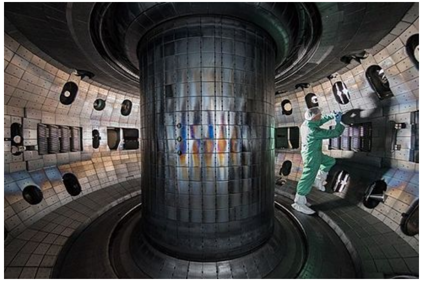
AGENDA

1. Latency, throughput and scaling
2. Compiling small models for throughput
3. Case studies with molecular transformers

Latency vs Throughput

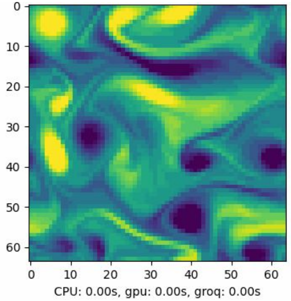
Latency critical applications

Advanced photon source, data processing off detector

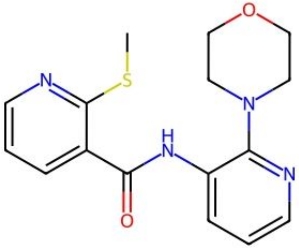


Tokamak fusion reactor, real time control

Throughput critical applications



Computational fluid dynamics

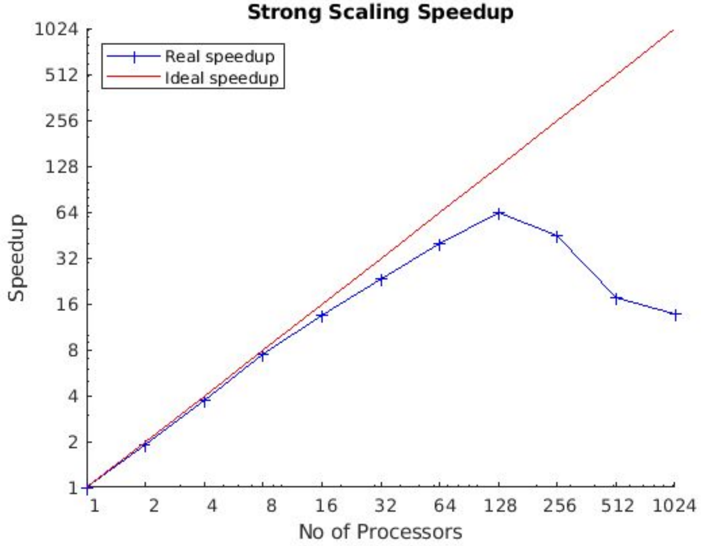


Predicting molecular properties

Scaling

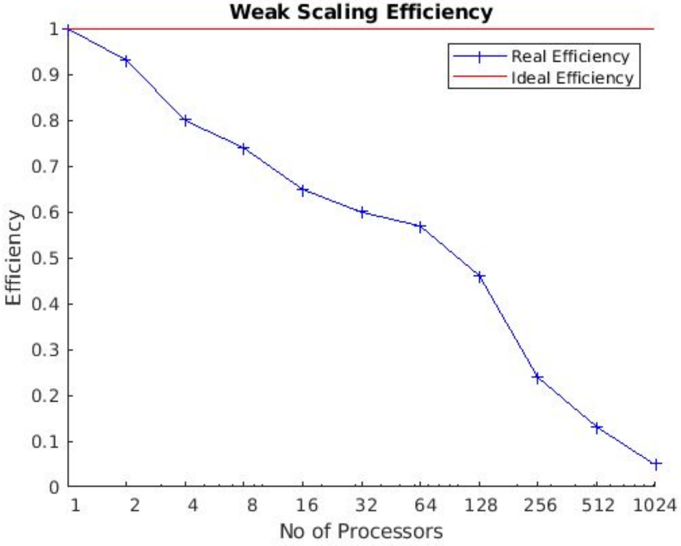
Strong Scaling

- Fixed problem size
- Increasing number of compute resources



Weak Scaling

- Increase problem size with number of compute resources



Important: these assume a fixed implementation

Image Credit: HPC Wiki

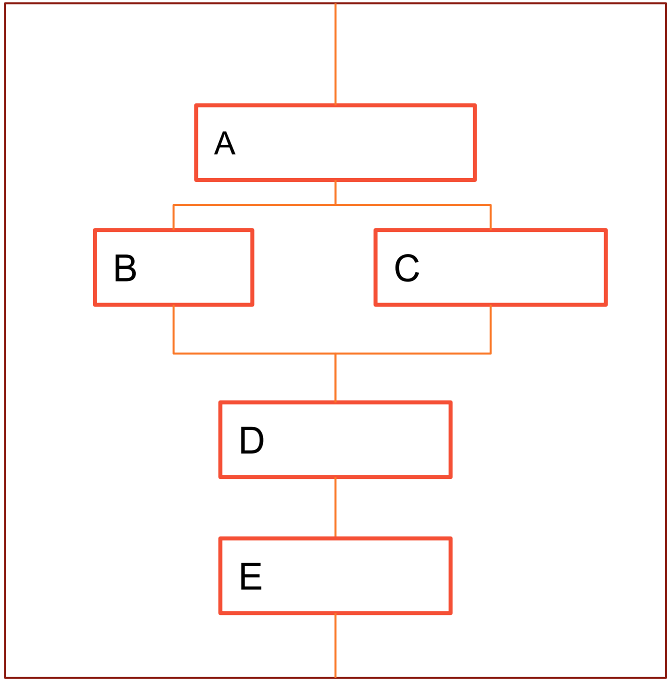
Scaling Neural Networks with Groq Compiler

Optimize inferences per second (IPS) of model

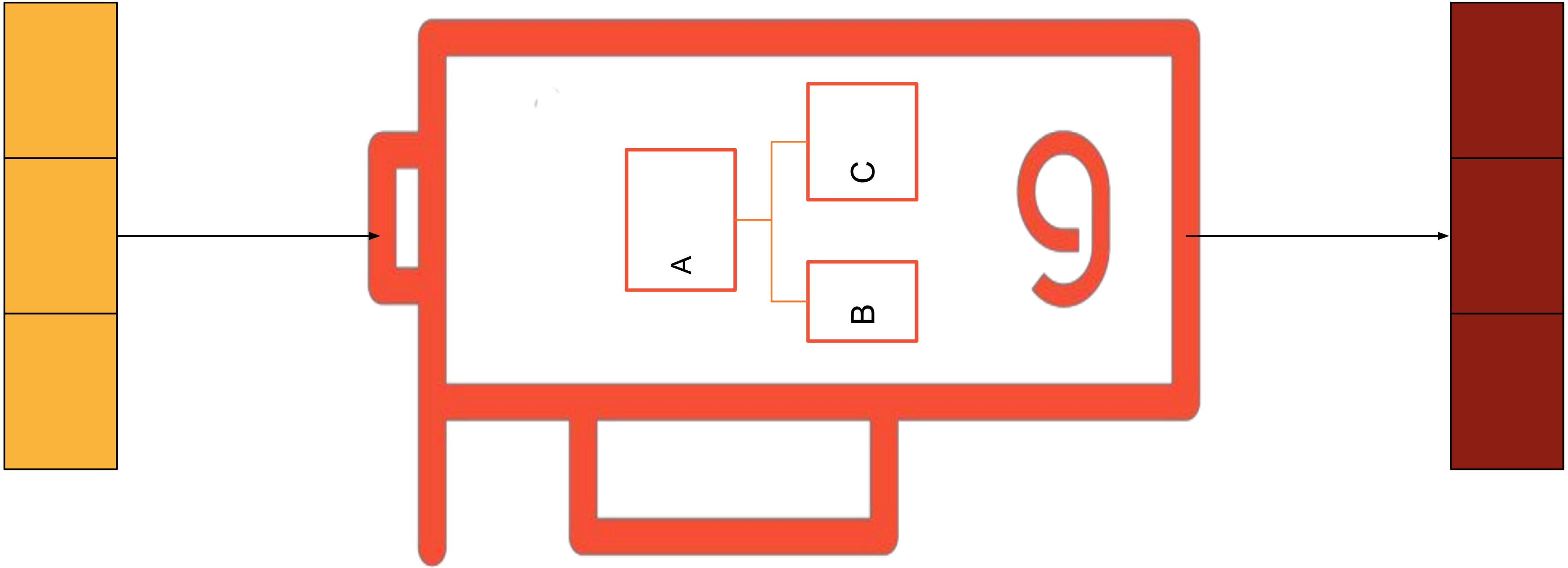
- Batch size
- Number of cards

Parallelization strategies

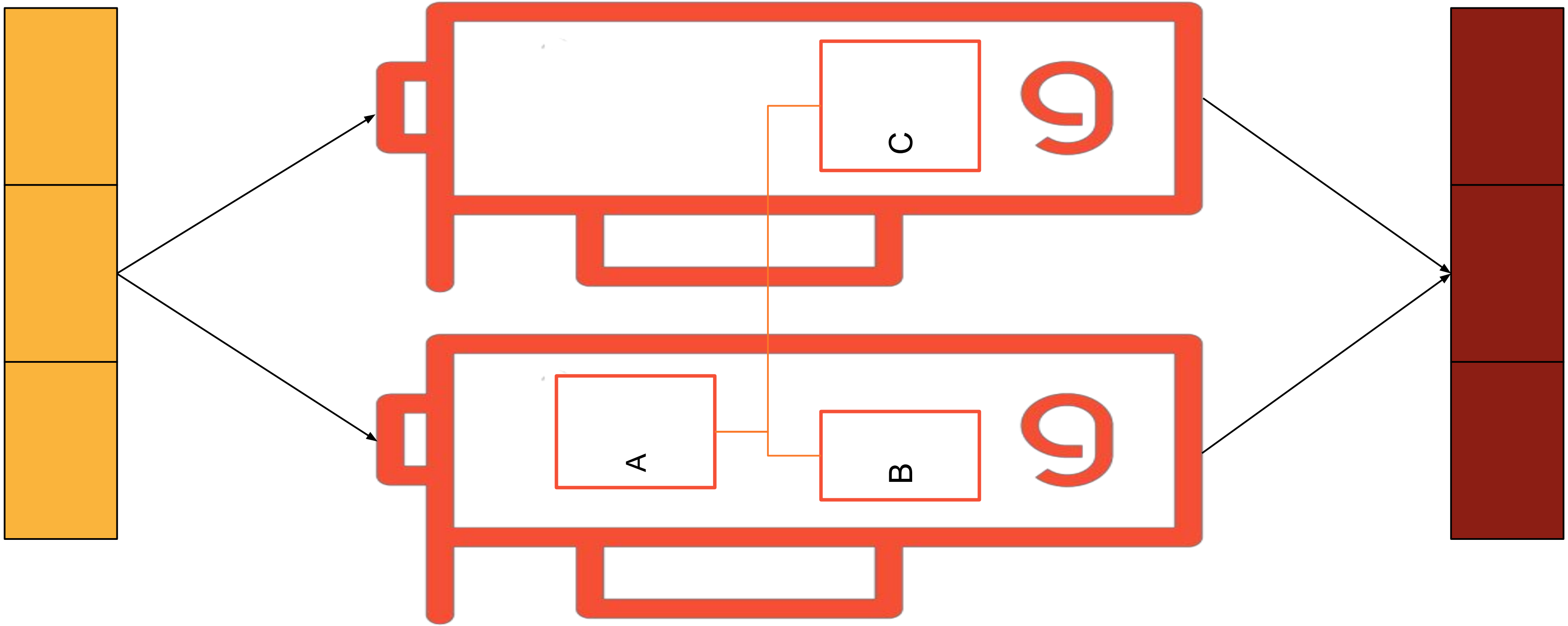
- Inter op
- Intra op



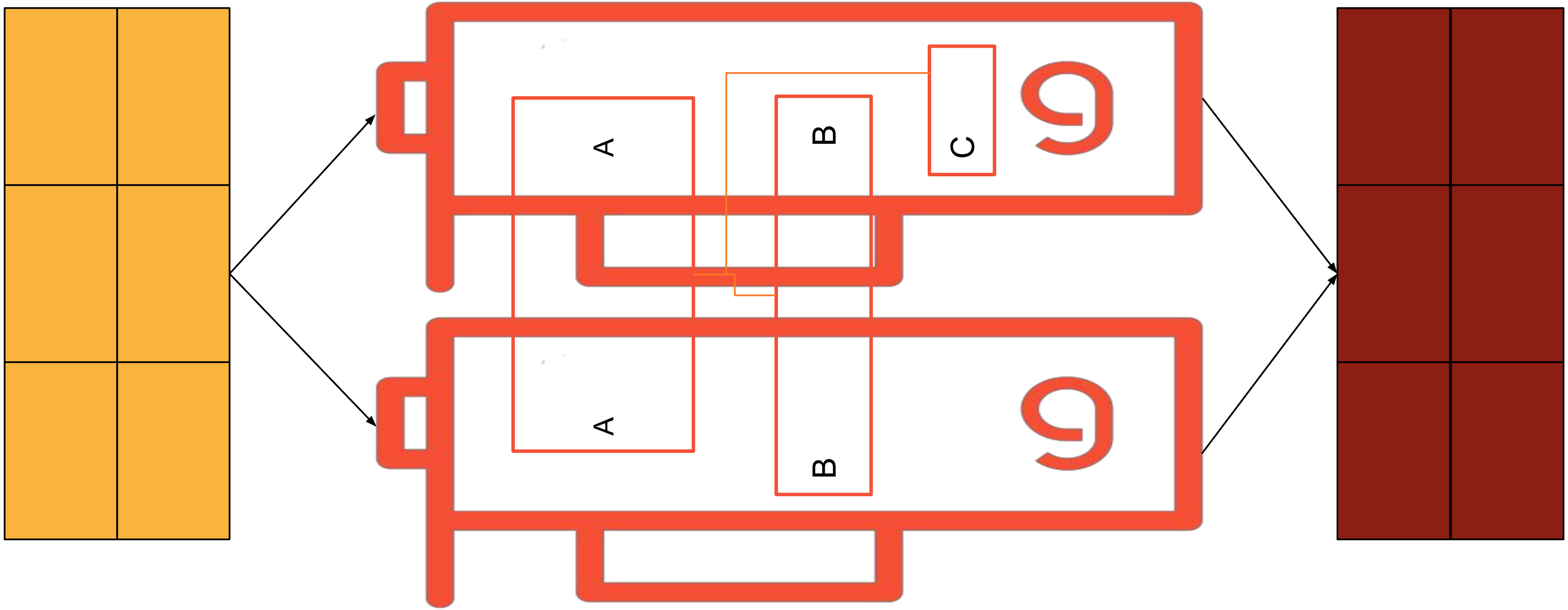
Single chip all ops on the card



Compiler will find the best parallelization

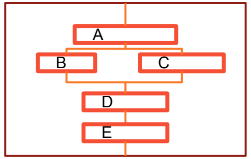


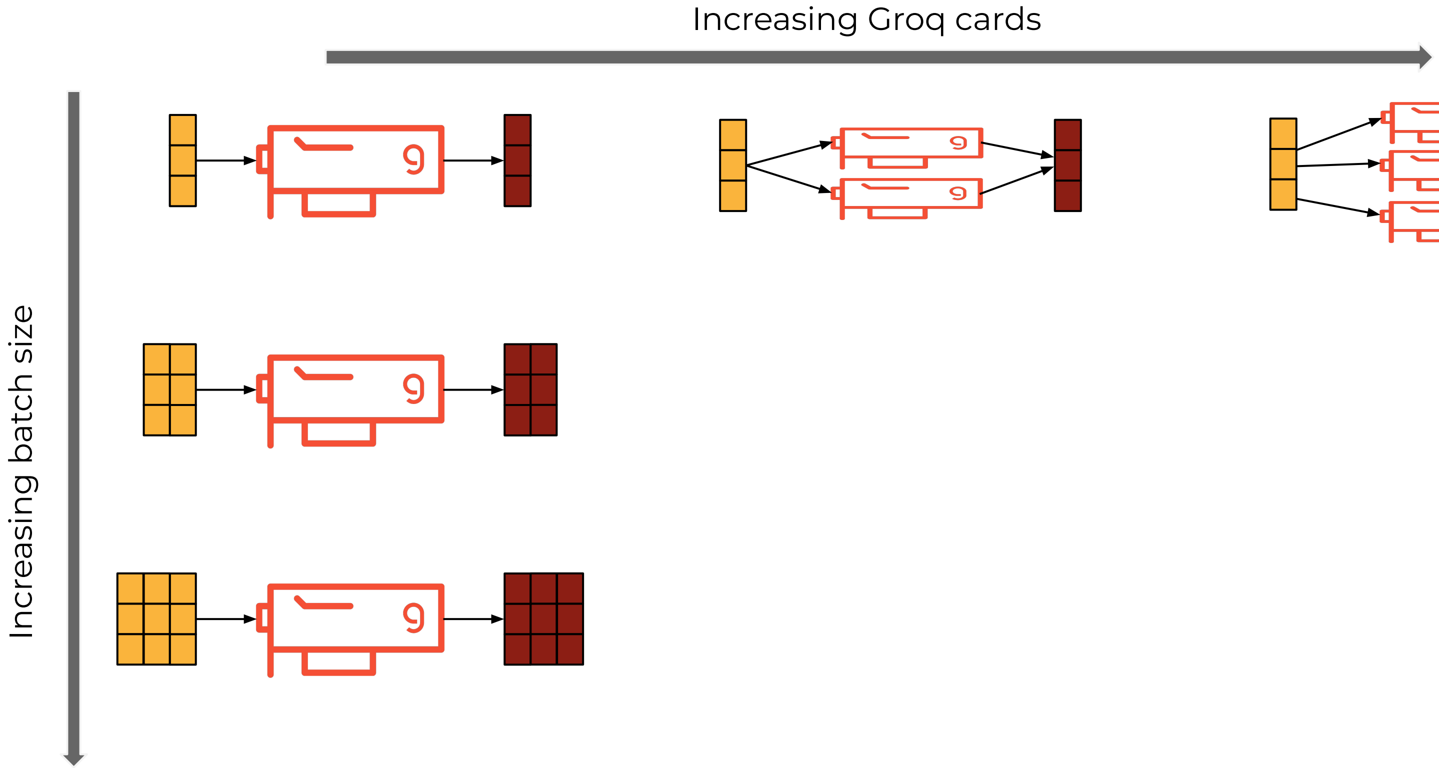
But may change with different batch, num cards



Compiler experiments

Fixed model architecture:





Setup

- N=total LPUs you can allocate (1 node = 8 chips, 1 rack = 72 chips)
- B=batch size
- G=LPUs compiled for
- L=latency

$$\mathrm{IPS} = \frac{N}{G}\frac{B\,[\mathrm{inferences}]}{L\,[s]}$$

Compiler Flags

Compiler topology

```
        --multichip=TOPOLOGY_STRING
```

Within a node

```
        DF_A14_N_CHIP       N in 1,2,4,8
```

Within a rack

```
        RT09_A14_N_CHIP     N in 16,24,32,40,48,56,64,72
```

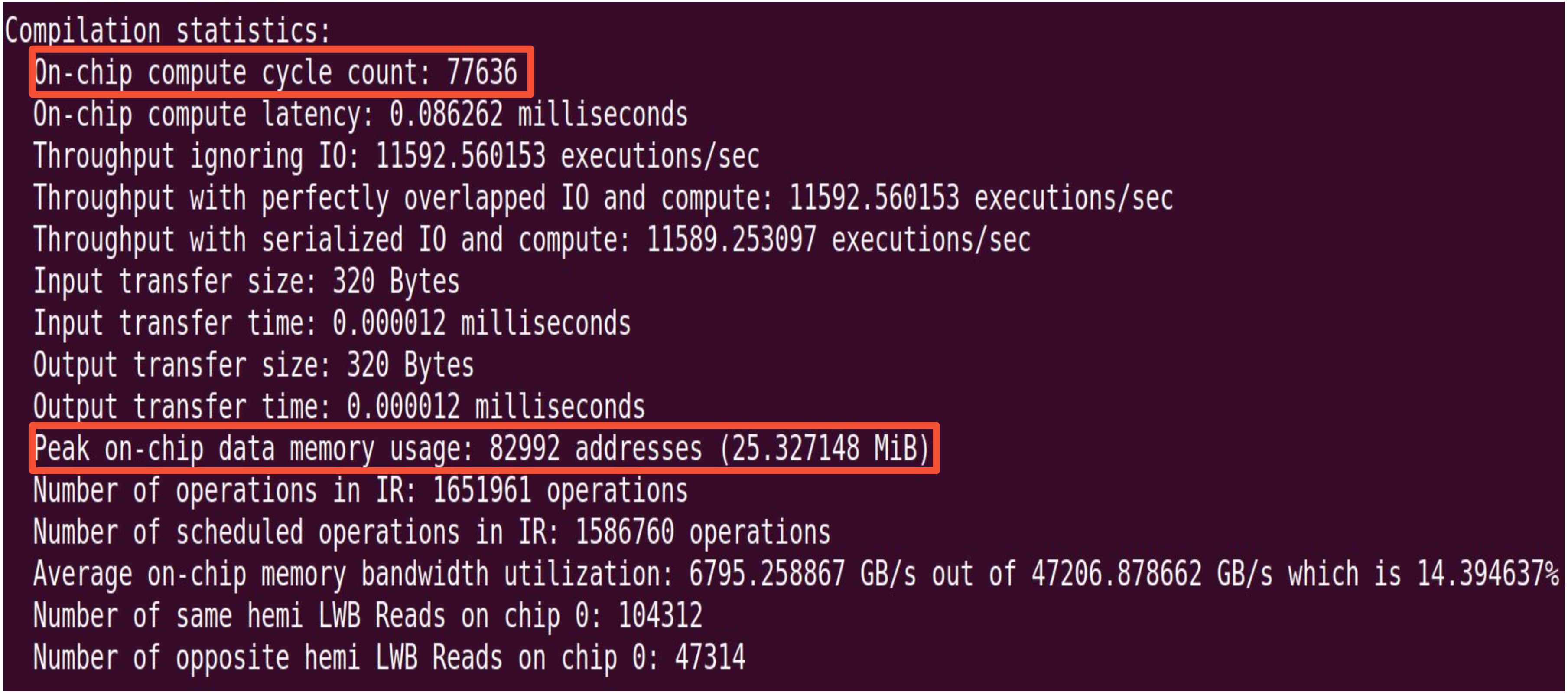
Performance Statistics

```
--power-analysis
--save-stats
```

Assembler flag

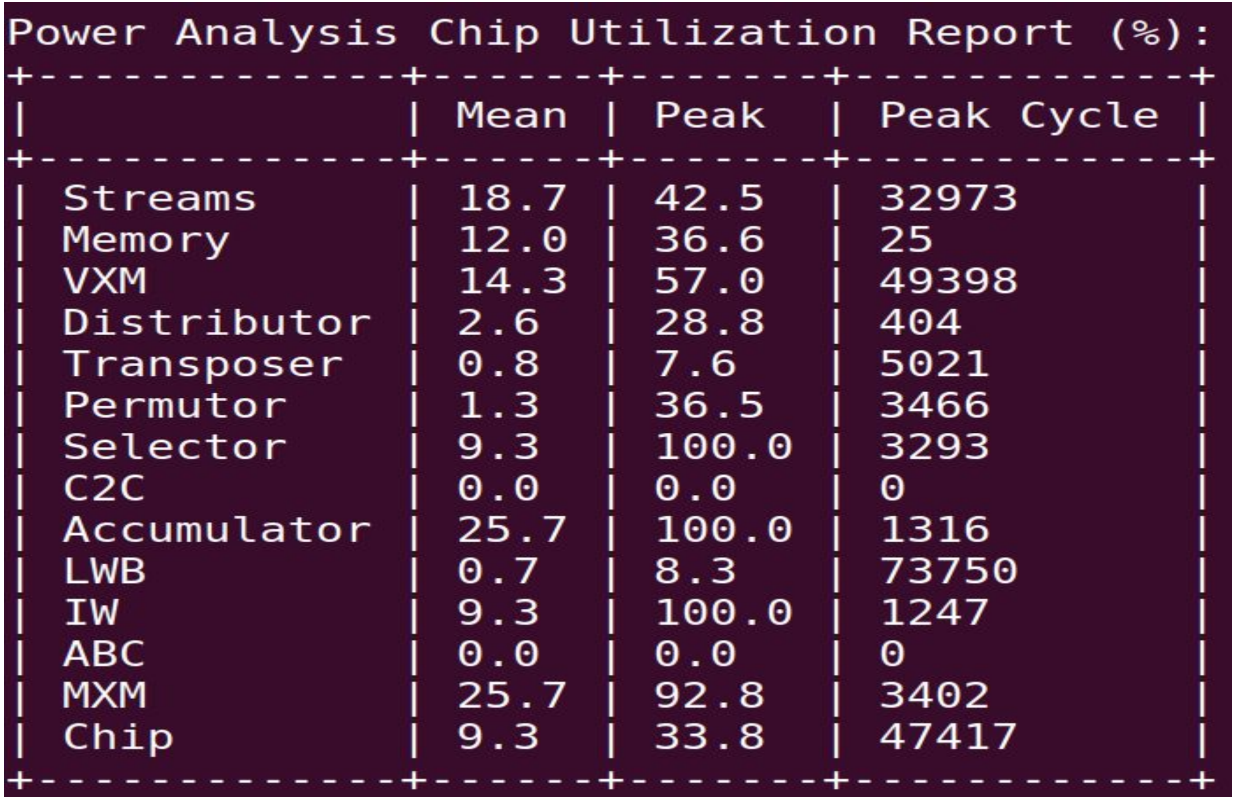
```
        --topology=TOPOLOGY_STRING
```

Compiler Output



```
Compilation statistics:
  On-chip compute cycle count: 77636
  On-chip compute latency: 0.086262 milliseconds
  Throughput ignoring IO: 11592.560153 executions/sec
  Throughput with perfectly overlapped IO and compute: 11592.560153 executions/sec
  Throughput with serialized IO and compute: 11589.253097 executions/sec
  Input transfer size: 320 Bytes
  Input transfer time: 0.000012 milliseconds
  Output transfer size: 320 Bytes
  Output transfer time: 0.000012 milliseconds
  Peak on-chip data memory usage: 82992 addresses (25.327148 MiB)
  Number of operations in IR: 1651961 operations
  Number of scheduled operations in IR: 1586760 operations
  Average on-chip memory bandwidth utilization: 6795.258867 GB/s out of 47206.878662 GB/s which is 14.394637%
  Number of same hemi LWB Reads on chip 0: 104312
  Number of opposite hemi LWB Reads on chip 0: 47314
```

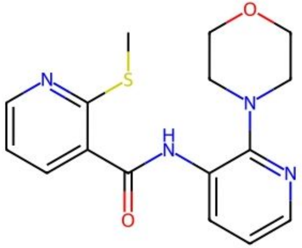
Compiler Output



```
Power Analysis Chip Utilization Report (%):
+-------------+------+-------+------------+
|             | Mean | Peak  | Peak Cycle |
+-------------+------+-------+------------+
| Streams     | 18.7 | 42.5  | 32973      |
| Memory      | 12.0 | 36.6  | 25         |
| VXM         | 14.3 | 57.0  | 49398      |
| Distributor | 2.6  | 28.8  | 404        |
| Transposer  | 0.8  | 7.6   | 5021       |
| Permutor    | 1.3  | 36.5  | 3466       |
| Selector    | 9.3  | 100.0 | 3293       |
| C2C         | 0.0  | 0.0   | 0          |
| Accumulator | 25.7 | 100.0 | 1316       |
| LWB         | 0.7  | 8.3   | 73750      |
| IW          | 9.3  | 100.0 | 1247       |
| ABC         | 0.0  | 0.0   | 0          |
| MXM         | 25.7 | 92.8  | 3402       |
| Chip        | 9.3  | 33.8  | 47417      |
+-------------+------+-------+------------+
```

Molecular Language

Molecules are composed of atoms held together by chemical bonds, physically 3D objects



Neural network architectures

- GNN/Message passing
- Convolutions

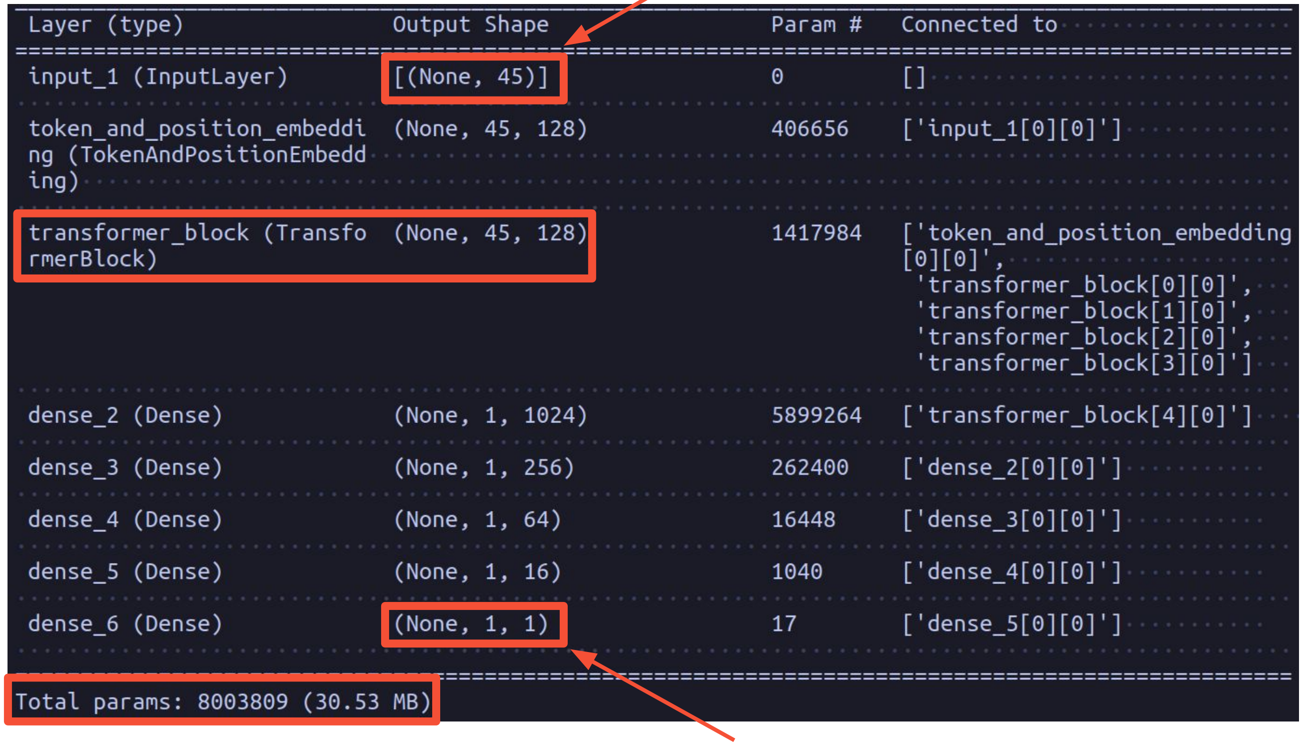
Molecules can be represented with sequence of characters

SMILES representation

- Ethanol CH_3CH_2OH →CCO
- Benzene C_6H_6
  - C1=CC=CC=C1 (double bonds)
  - c1ccccc1 (ring structure)

Use transformers!

SMILES Transformer

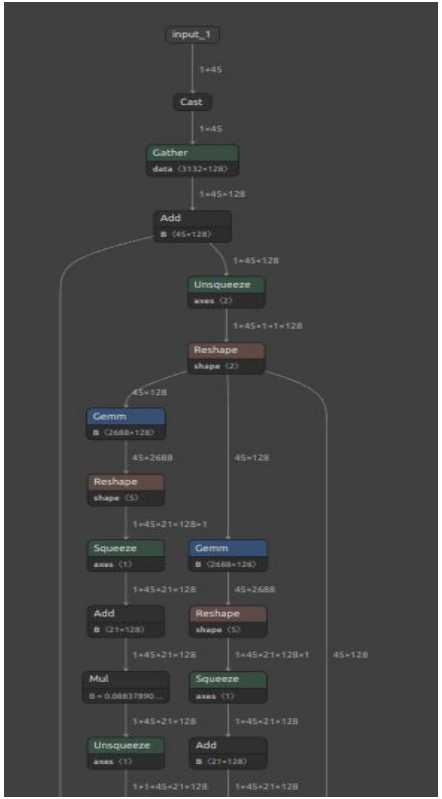


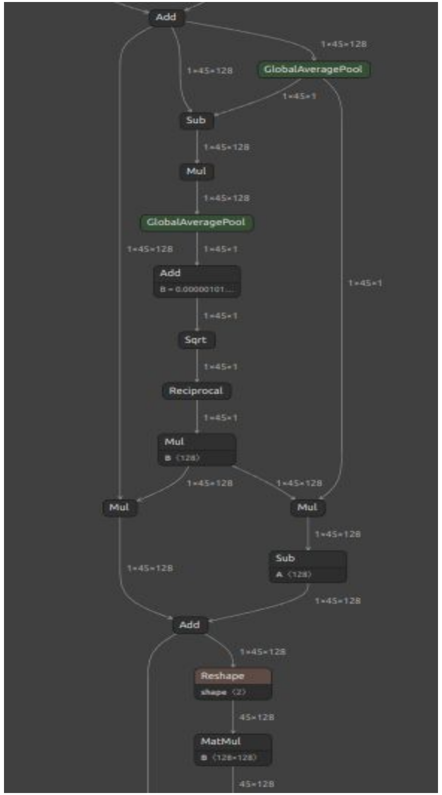
Input: batch_size, tokens

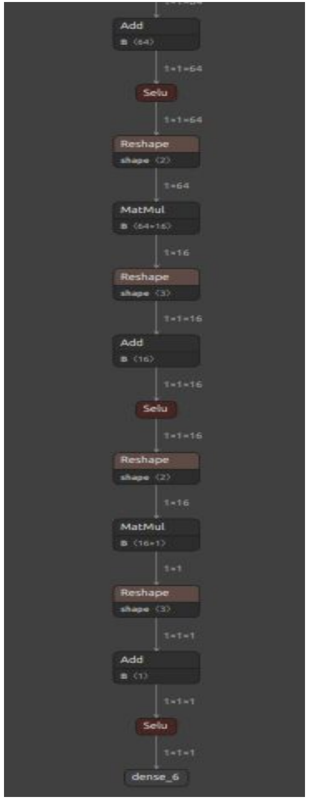
```
 Layer (type)                Output Shape                 Param #   Connected to
==================================================================================================
 input_1 (InputLayer)        [(None, 45)]                 0         []

 token_and_position_embeddi  (None, 45, 128)              406656    ['input_1[0][0]']
 ng (TokenAndPositionEmbedd
 ing)

 transformer_block (Transfo  (None, 45, 128)              1417984   ['token_and_position_embedding
 rmerBlock)                                                         [0][0]',
                                                                     'transformer_block[0][0]',
                                                                     'transformer_block[1][0]',
                                                                     'transformer_block[2][0]',
                                                                     'transformer_block[3][0]']

 dense_2 (Dense)             (None, 1, 1024)              5899264   ['transformer_block[4][0]']

 dense_3 (Dense)             (None, 1, 256)               262400    ['dense_2[0][0]']

 dense_4 (Dense)             (None, 1, 64)                16448     ['dense_3[0][0]']

 dense_5 (Dense)             (None, 1, 16)                1040      ['dense_4[0][0]']

 dense_6 (Dense)             (None, 1, 1)                 17        ['dense_5[0][0]']

==================================================================================================
Total params: 8003809 (30.53 MB)
```

Molecular property prediction

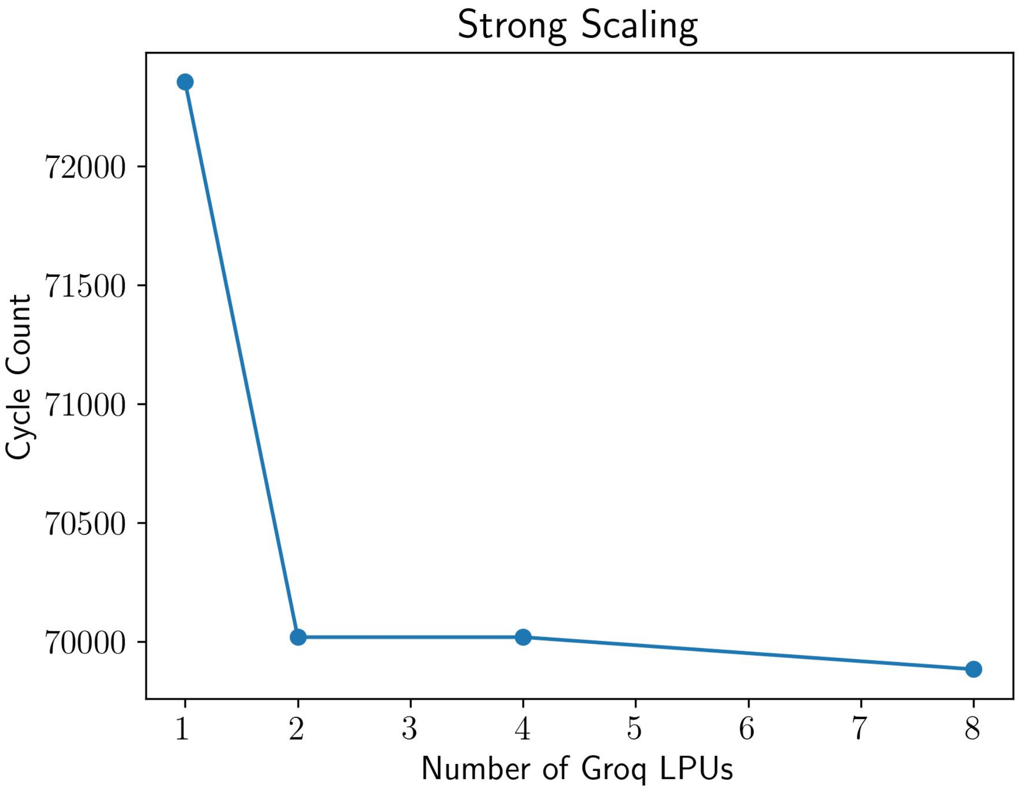
SMILES Transformer





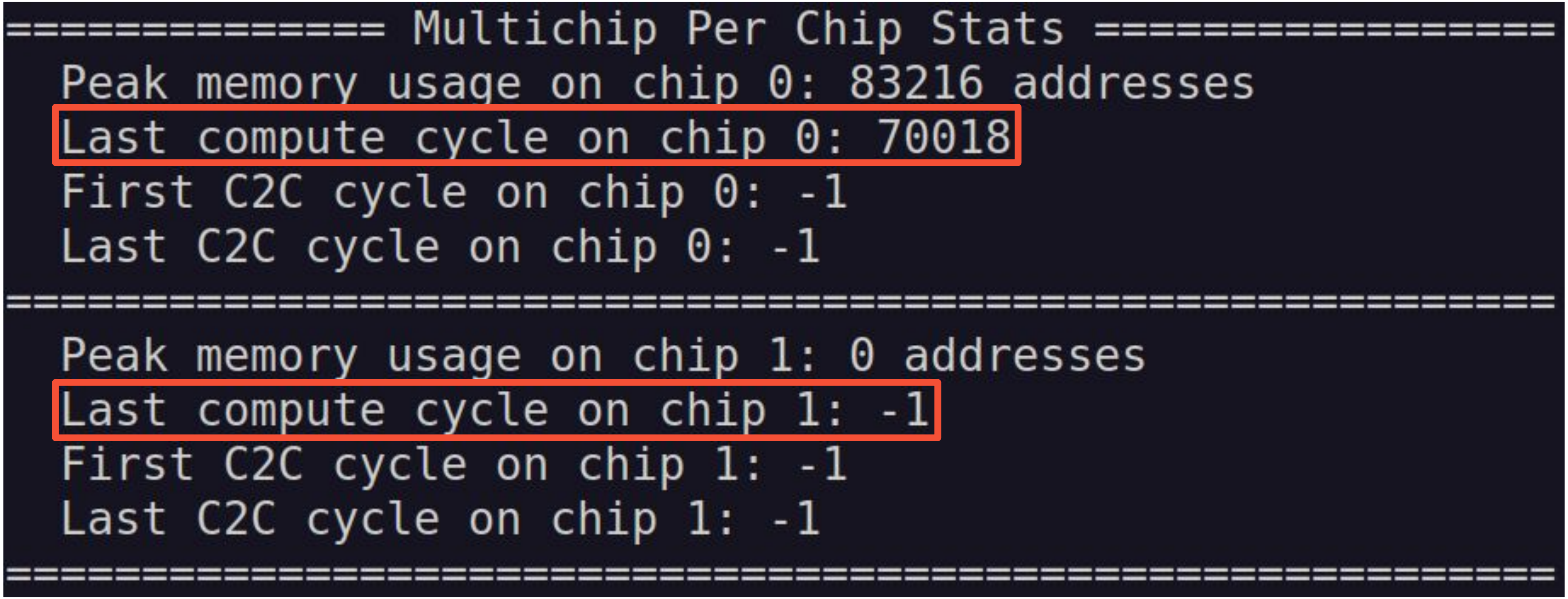


SMILES Transformer



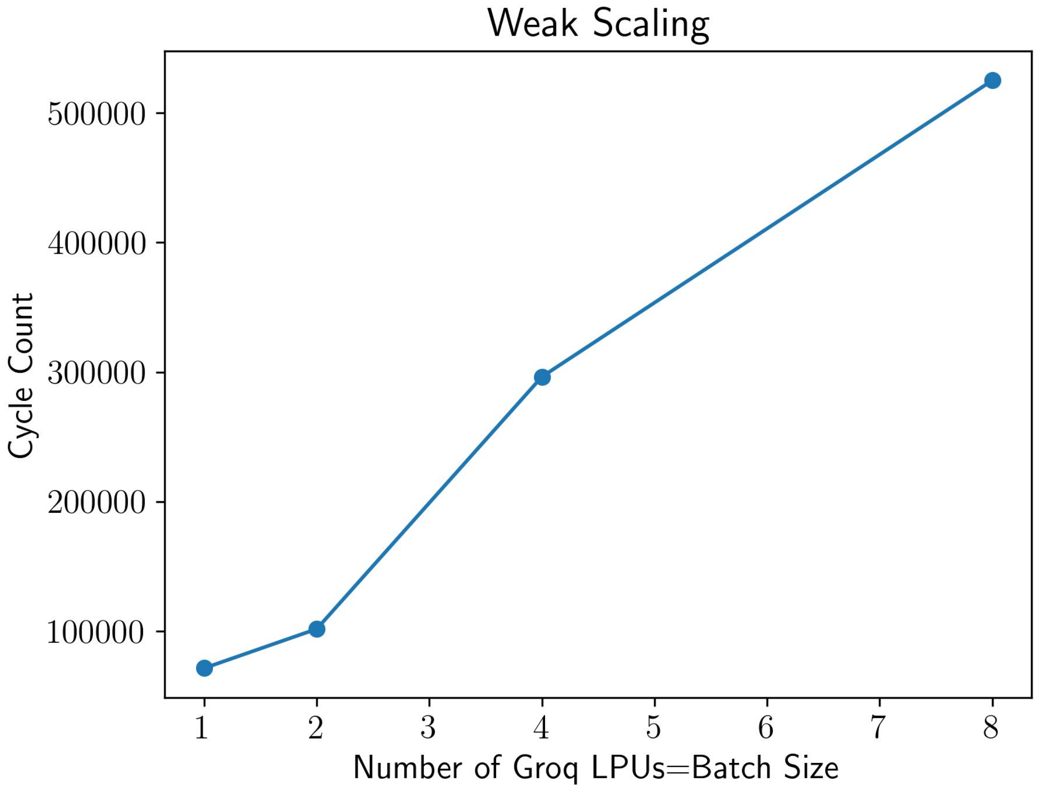
SMILES Transformer

batch=1, nchips=2 compile

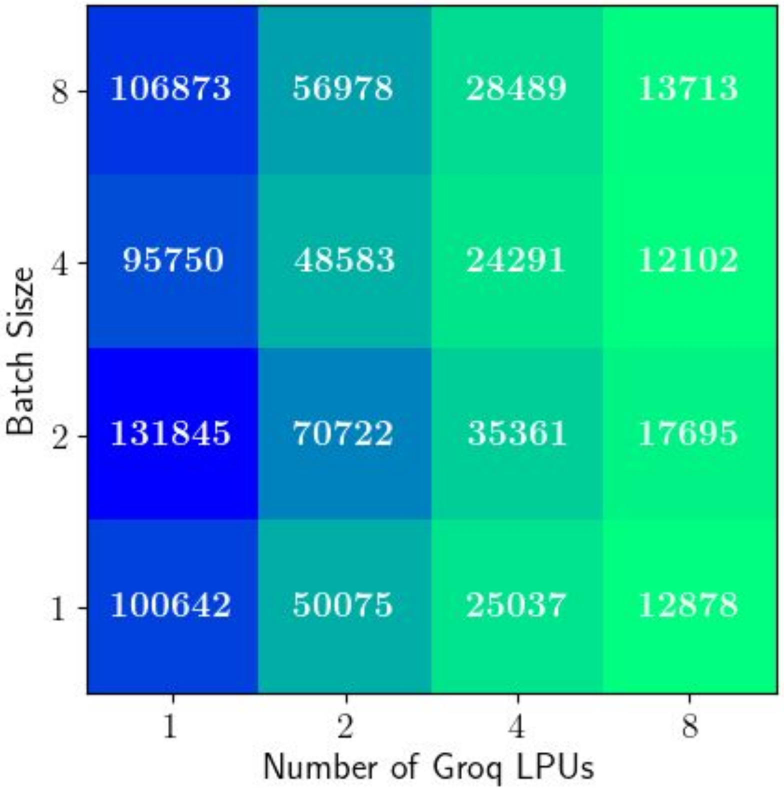


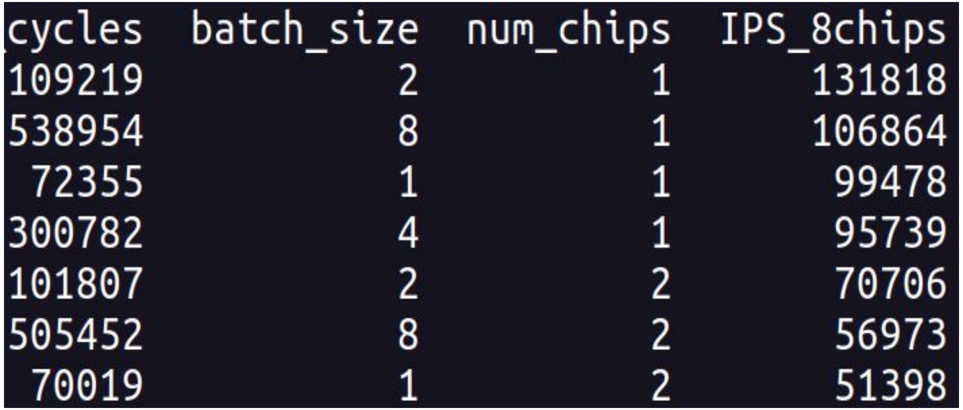
```
============== Multichip Per Chip Stats =================
  Peak memory usage on chip 0: 83216 addresses
  Last compute cycle on chip 0: 70018
  First C2C cycle on chip 0: -1
  Last C2C cycle on chip 0: -1
=========================================================
  Peak memory usage on chip 1: 0 addresses
  Last compute cycle on chip 1: -1
  First C2C cycle on chip 1: -1
  Last C2C cycle on chip 1: -1
=========================================================
```

SMILES Transformer



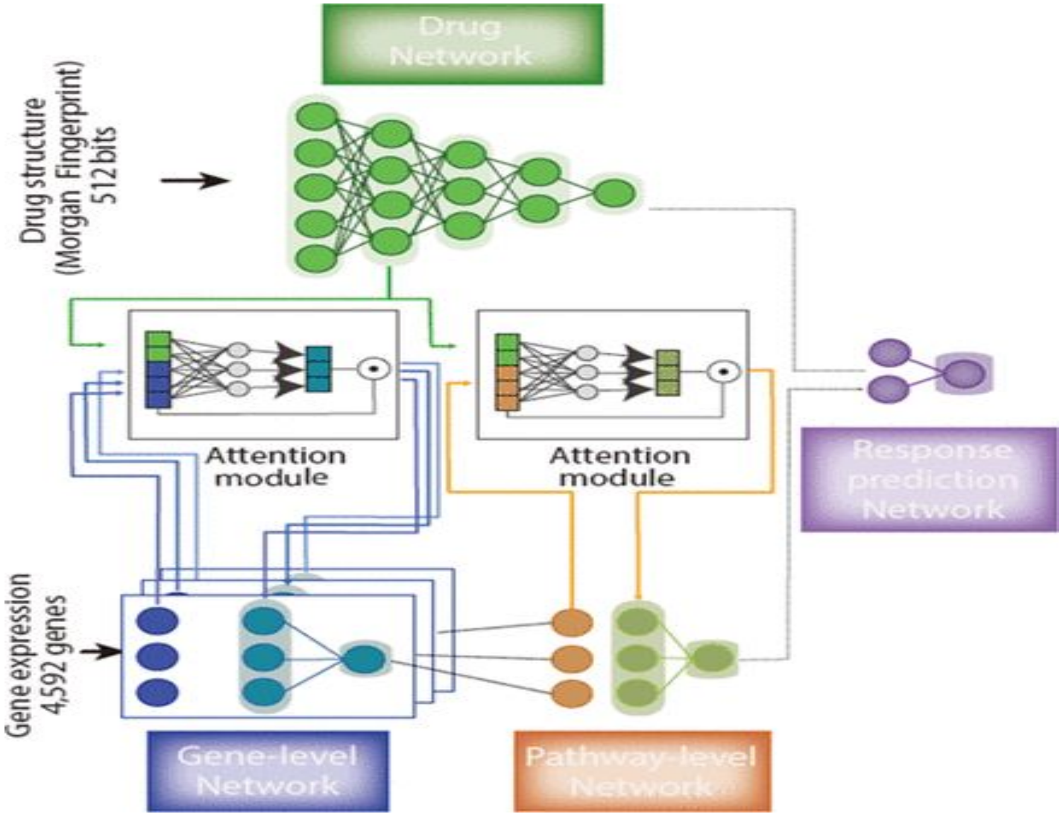
Maximum IPS





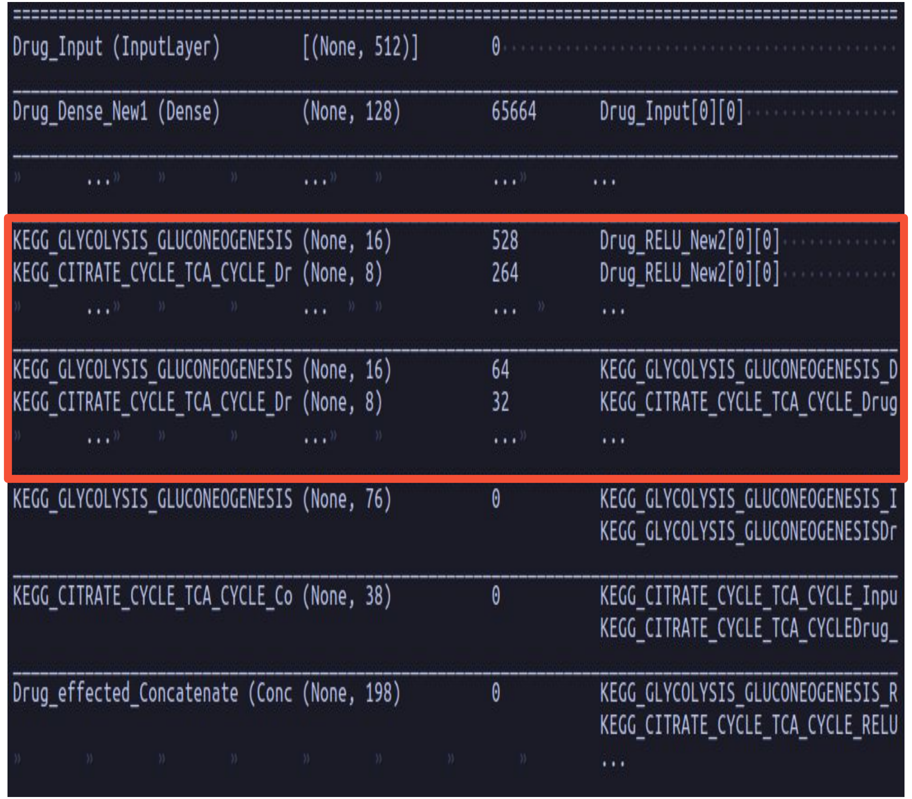
```
cycles  batch_size  num_chips  IPS_8chips
109219           2          1      131818
538954           8          1      106864
 72355           1          1       99478
300782           4          1       95739
101807           2          2       70706
505452           8          2       56973
 70019           1          2       51398
```

HiDRA

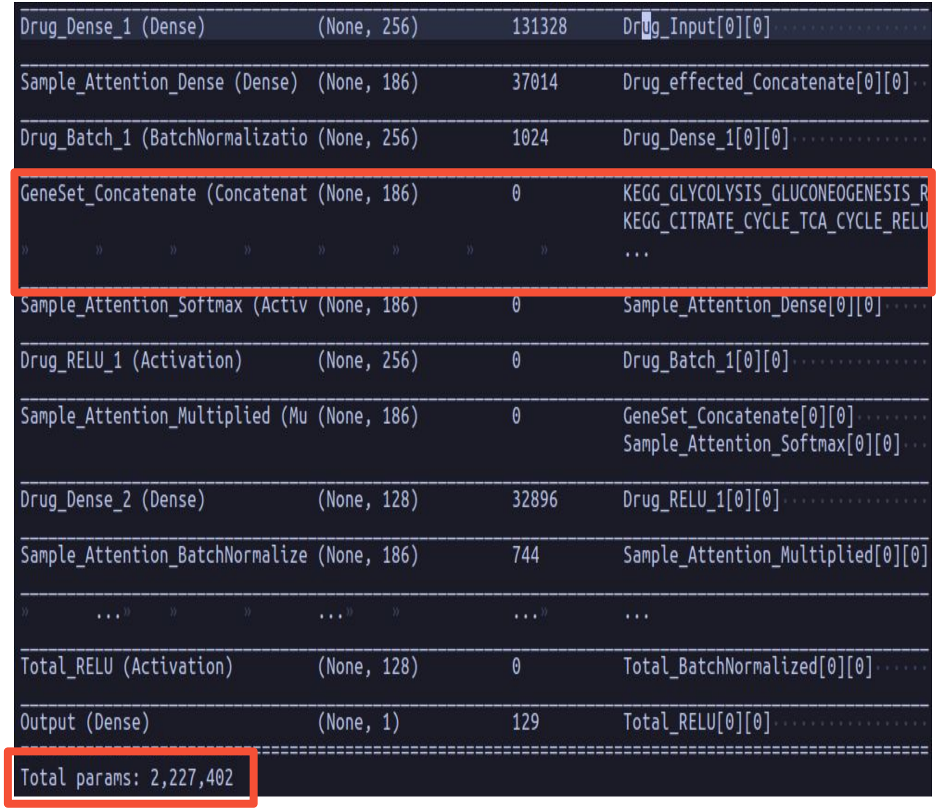


J. Chem. Inf. Model. 2021, 61, 8, 3858−3867

HiDRA

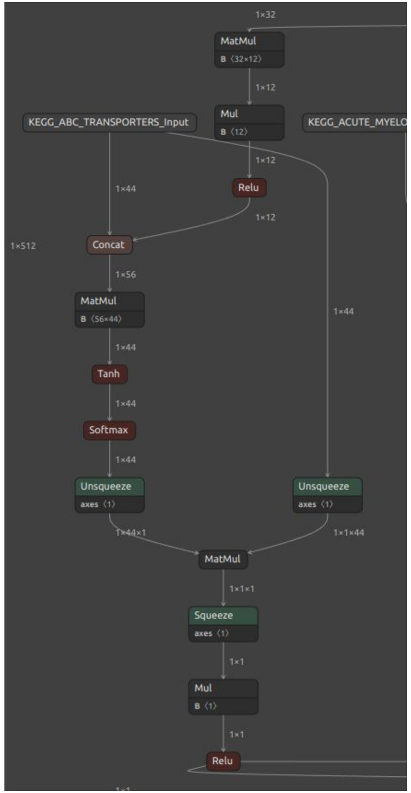


```
==================================================================================
Drug_Input (InputLayer)         [(None, 512)]        0

__________________________________________________________________________________
Drug_Dense_New1 (Dense)         (None, 128)          65664       Drug_Input[0][0]

__________________________________________________________________________________
        ...                         ...                  ...         ...

__________________________________________________________________________________
KEGG_GLYCOLYSIS_GLUCONEOGENESIS (None, 16)           528         Drug_RELU_New2[0][0]
KEGG_CITRATE_CYCLE_TCA_CYCLE_Dr (None, 8)            264         Drug_RELU_New2[0][0]
        ...                         ...                  ...         ...

__________________________________________________________________________________
KEGG_GLYCOLYSIS_GLUCONEOGENESIS (None, 16)           64          KEGG_GLYCOLYSIS_GLUCONEOGENESIS_D
KEGG_CITRATE_CYCLE_TCA_CYCLE_Dr (None, 8)            32          KEGG_CITRATE_CYCLE_TCA_CYCLE_Drug
        ...                         ...                  ...         ...

__________________________________________________________________________________
KEGG_GLYCOLYSIS_GLUCONEOGENESIS (None, 76)           0           KEGG_GLYCOLYSIS_GLUCONEOGENESIS_I
                                                                 KEGG_GLYCOLYSIS_GLUCONEOGENESISDr

__________________________________________________________________________________
KEGG_CITRATE_CYCLE_TCA_CYCLE_Co (None, 38)           0           KEGG_CITRATE_CYCLE_TCA_CYCLE_Inpu
                                                                 KEGG_CITRATE_CYCLE_TCA_CYCLEDrug_

__________________________________________________________________________________
Drug_effected_Concatenate (Conc (None, 198)          0           KEGG_GLYCOLYSIS_GLUCONEOGENESIS_R
                                                                 KEGG_CITRATE_CYCLE_TCA_CYCLE_RELU
                                                                 ...
```

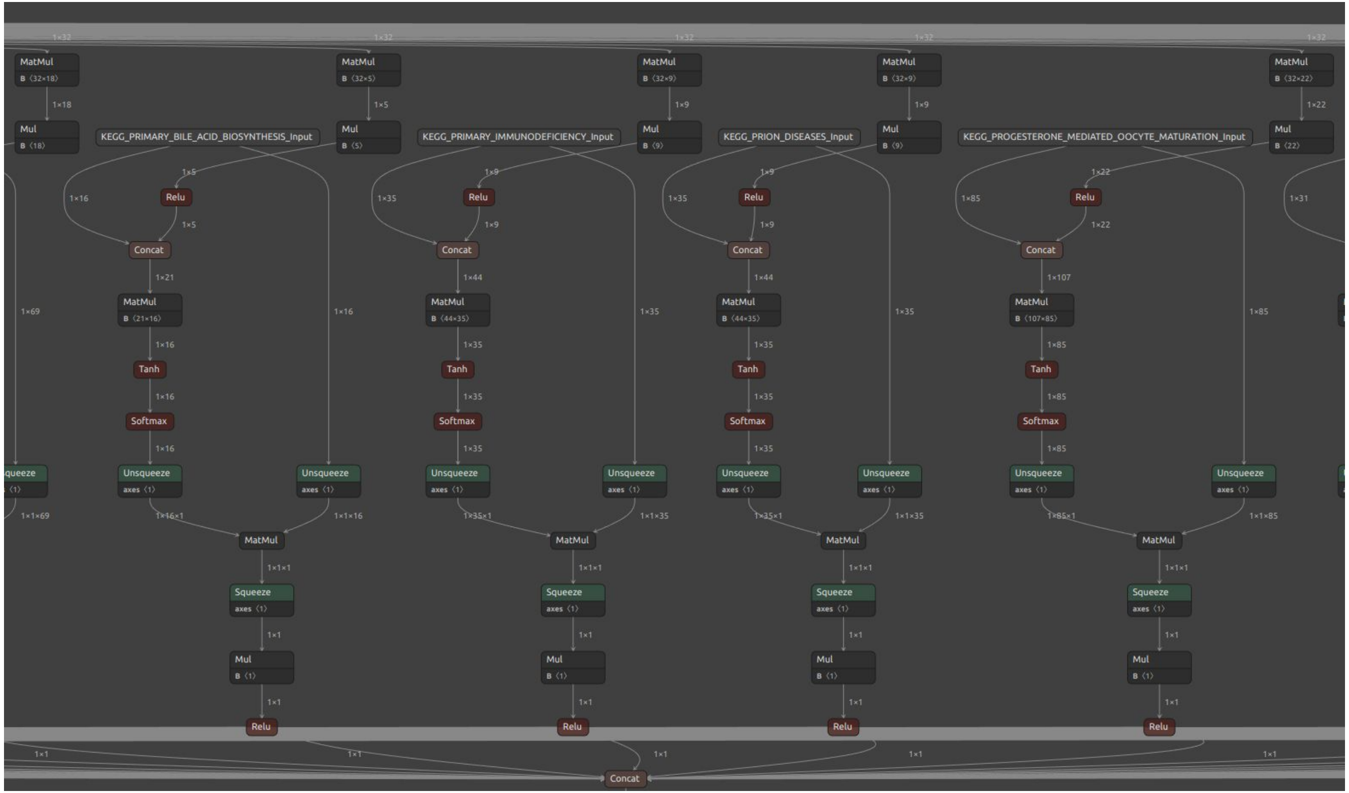


```
__________________________________________________________________________________
Drug_Dense_1 (Dense)            (None, 256)          131328      Drug_Input[0][0]

__________________________________________________________________________________
Sample_Attention_Dense (Dense)  (None, 186)          37014       Drug_effected_Concatenate[0][0]

__________________________________________________________________________________
Drug_Batch_1 (BatchNormalizatio (None, 256)          1024        Drug_Dense_1[0][0]

__________________________________________________________________________________
GeneSet_Concatenate (Concatenat (None, 186)          0           KEGG_GLYCOLYSIS_GLUCONEOGENESIS_R
                                                                 KEGG_CITRATE_CYCLE_TCA_CYCLE_RELU
                                                                 ...

__________________________________________________________________________________
Sample_Attention_Softmax (Activ (None, 186)          0           Sample_Attention_Dense[0][0]

__________________________________________________________________________________
Drug_RELU_1 (Activation)        (None, 256)          0           Drug_Batch_1[0][0]

__________________________________________________________________________________
Sample_Attention_Multiplied (Mu (None, 186)          0           GeneSet_Concatenate[0][0]
                                                                 Sample_Attention_Softmax[0][0]

__________________________________________________________________________________
Drug_Dense_2 (Dense)            (None, 128)          32896       Drug_RELU_1[0][0]

__________________________________________________________________________________
Sample_Attention_BatchNormalize (None, 186)          744         Sample_Attention_Multiplied[0][0]

__________________________________________________________________________________
        ...                         ...                  ...         ...

__________________________________________________________________________________
Total_RELU (Activation)         (None, 128)          0           Total_BatchNormalized[0][0]

__________________________________________________________________________________
Output (Dense)                  (None, 1)            129         Total_RELU[0][0]
==================================================================================
Total params: 2,227,402
```

HiDRA



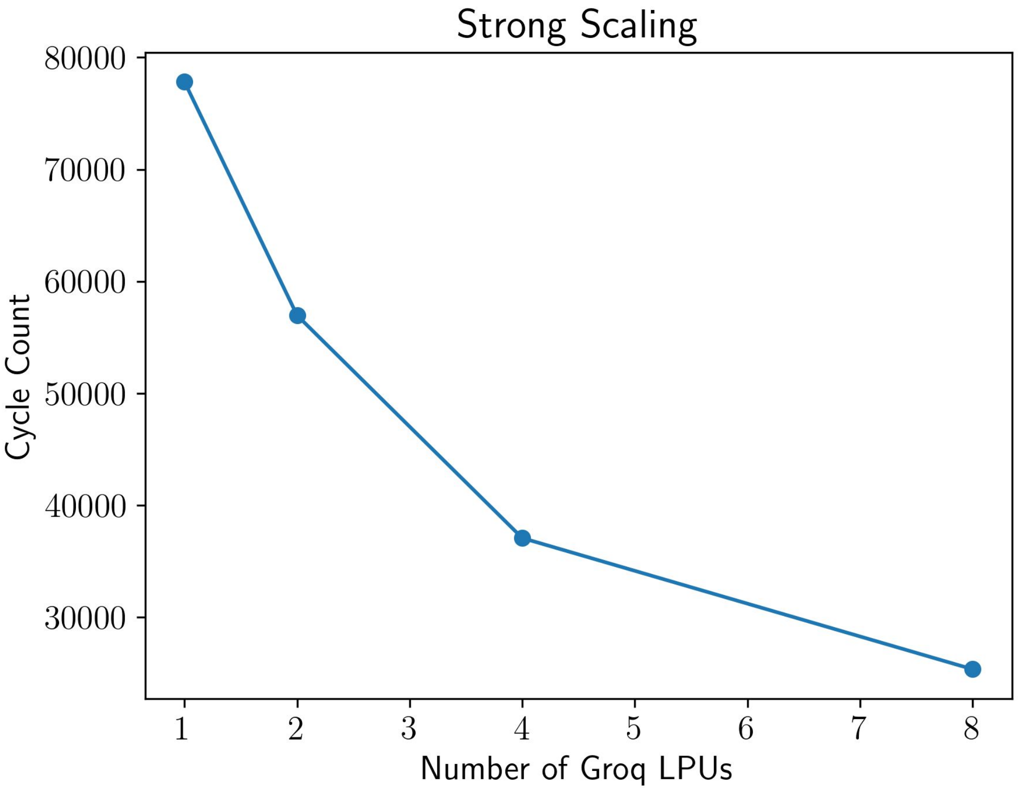
1×32
MatMul
B ⟨32×12⟩
1×12
Mul
B ⟨12⟩
KEGG_ABC_TRANSPORTERS_Input
KEGG_ACUTE_MYELO
1×12
Relu
1×44
1×12
1×512
Concat
1×56
MatMul
B ⟨56×44⟩
1×44
Tanh
1×44
Softmax
1×44
1×44
Unsqueeze
axes ⟨1⟩
Unsqueeze
axes ⟨1⟩
1×44×1
1×1×44
MatMul
1×1×1
Squeeze
axes ⟨1⟩
1×1
Mul
B ⟨1⟩
1×1
Relu



1×32
MatMul
B ⟨32×18⟩
1×18
Mul
B ⟨18⟩
KEGG_PRIMARY_BILE_ACID_BIOSYNTHESIS_Input
MatMul
B ⟨32×5⟩
1×5
Mul
B ⟨5⟩
KEGG_PRIMARY_IMMUNODEFICIENCY_Input
MatMul
B ⟨32×9⟩
1×9
Mul
B ⟨9⟩
KEGG_PRION_DISEASES_Input
MatMul
B ⟨32×9⟩
1×9
Mul
B ⟨9⟩
KEGG_PROGESTERONE_MEDIATED_OOCYTE_MATURATION_Input
MatMul
B ⟨32×22⟩
1×22
Mul
B ⟨22⟩
1×16
1×5
Relu
1×5
1×35
1×9
Relu
1×9
1×35
1×9
Relu
1×9
1×85
1×22
Relu
1×22
1×31
Concat
Concat
Concat
Concat
1×69
1×21
MatMul
B ⟨21×16⟩
1×16
1×44
MatMul
B ⟨44×35⟩
1×35
1×35
1×44
MatMul
B ⟨44×35⟩
1×35
1×35
1×107
MatMul
B ⟨107×85⟩
1×85
1×85
Tanh
Softmax
Unsqueeze
axes ⟨1⟩
1×1×69
1×16×1
1×1×16
1×35×1
1×1×35
1×35×1
1×1×35
1×85×1
1×1×85
MatMul
1×1×1
Squeeze
axes ⟨1⟩
1×1
Mul
B ⟨1⟩
1×1
Relu
1×1
Concat

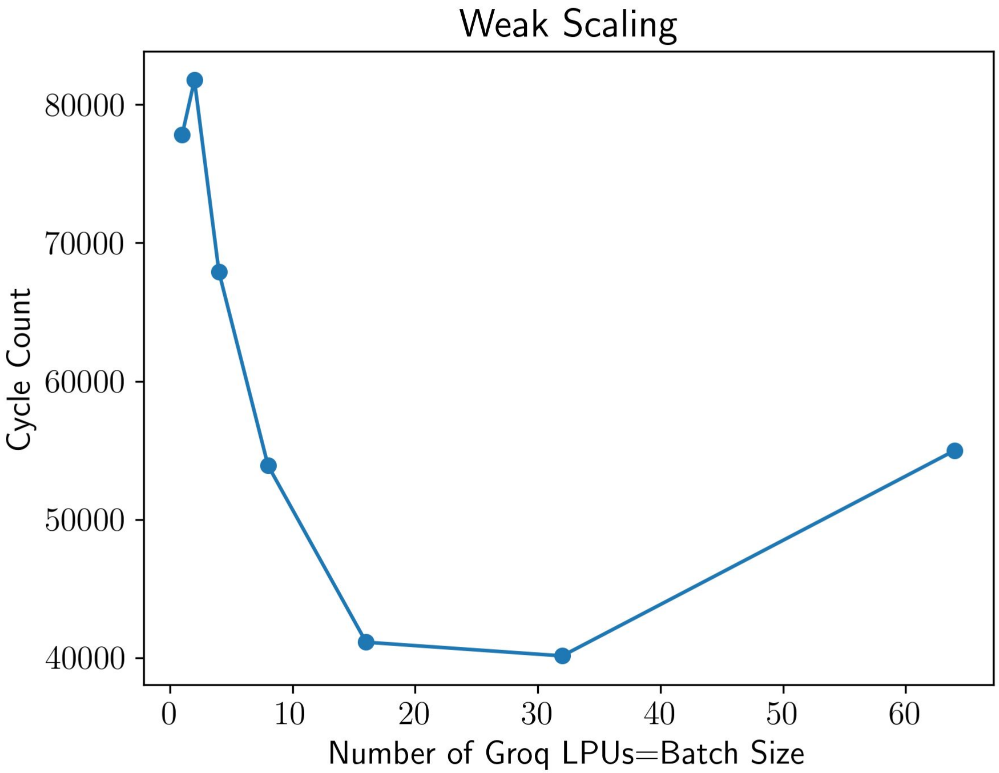
HiDRA

v0.10.x SDK



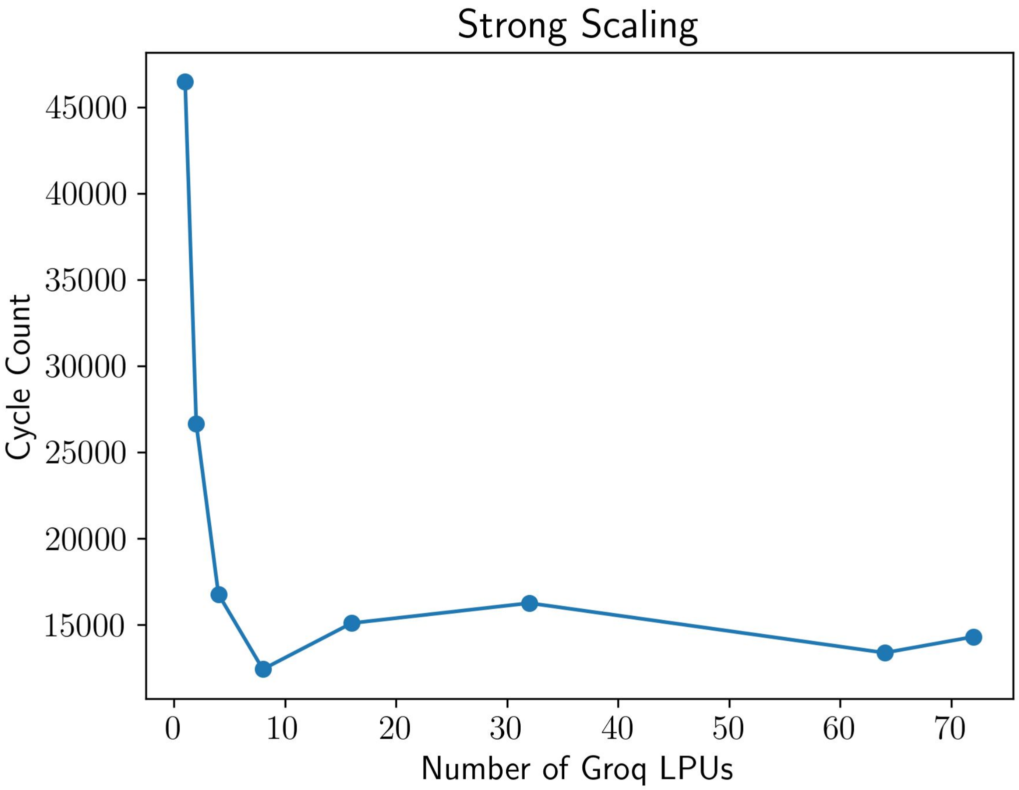
HiDRA

v0.10.x SDK



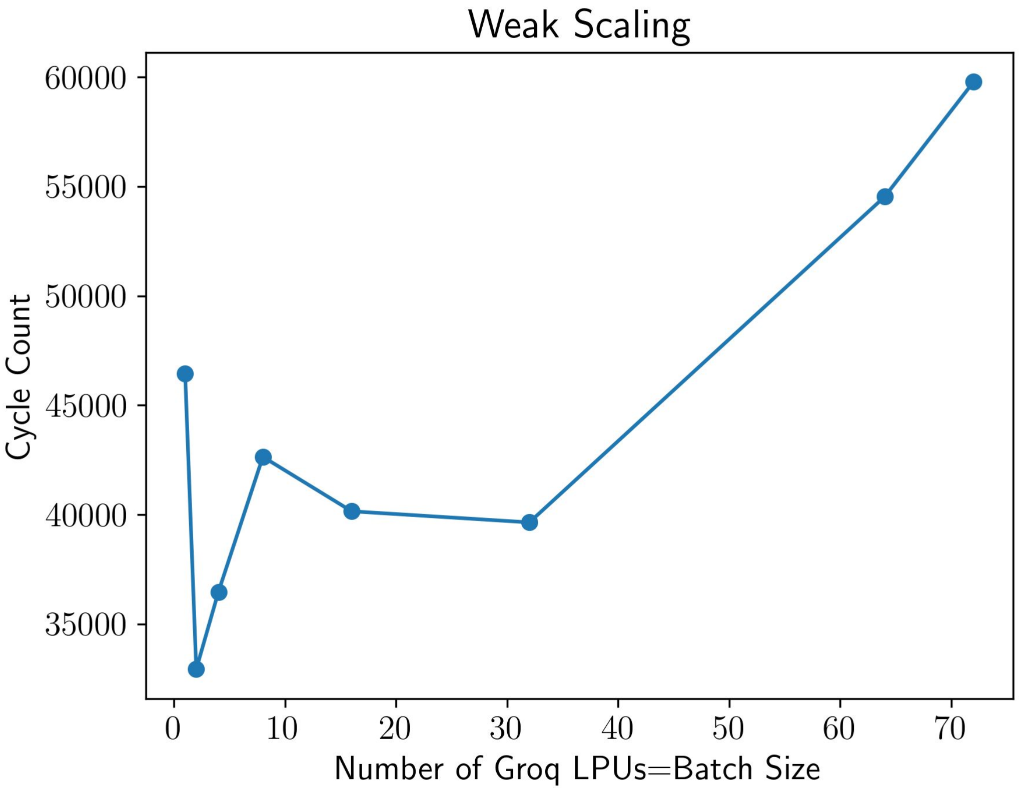
HiDRA

v0.11 SDK



HiDRA

v0.11 SDK



Drug Response Prediction Models

with the IMPROVE project

Argonne NATIONAL LABORATORY

Graph

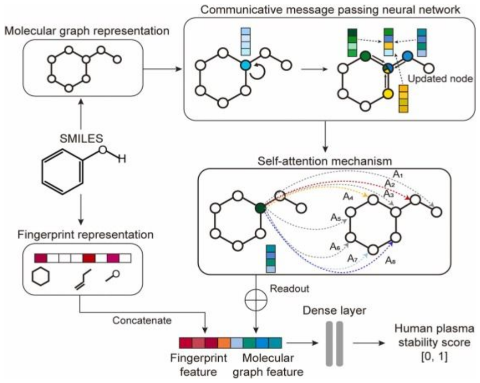
- Atoms are nodes
- Bonds are edges
- Physically motivated representation

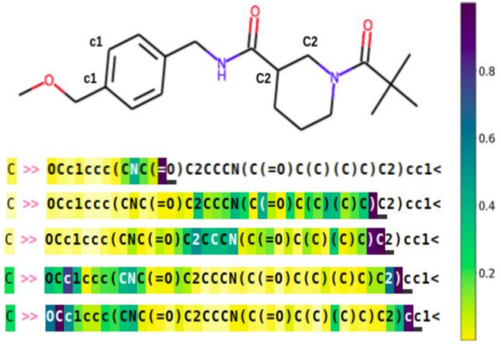
Attention

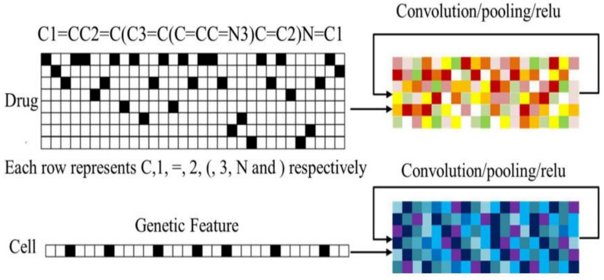
- Atoms and bonds as a string
- Encoding of molecule and its properties

Convolutions

- One hot encoding of atoms and bonds
- Feature pooling


Communicative message passing neural network
Molecular graph representation
Updated node
SMILES
H
Self-attention mechanism
A₁
A₂
A₃
A₄
A₅
A₆
A₇
A₈
A₉
Fingerprint representation
Readout
Dense layer
Concatenate
Human plasma stability score [0, 1]
Fingerprint feature
Molecular graph feature

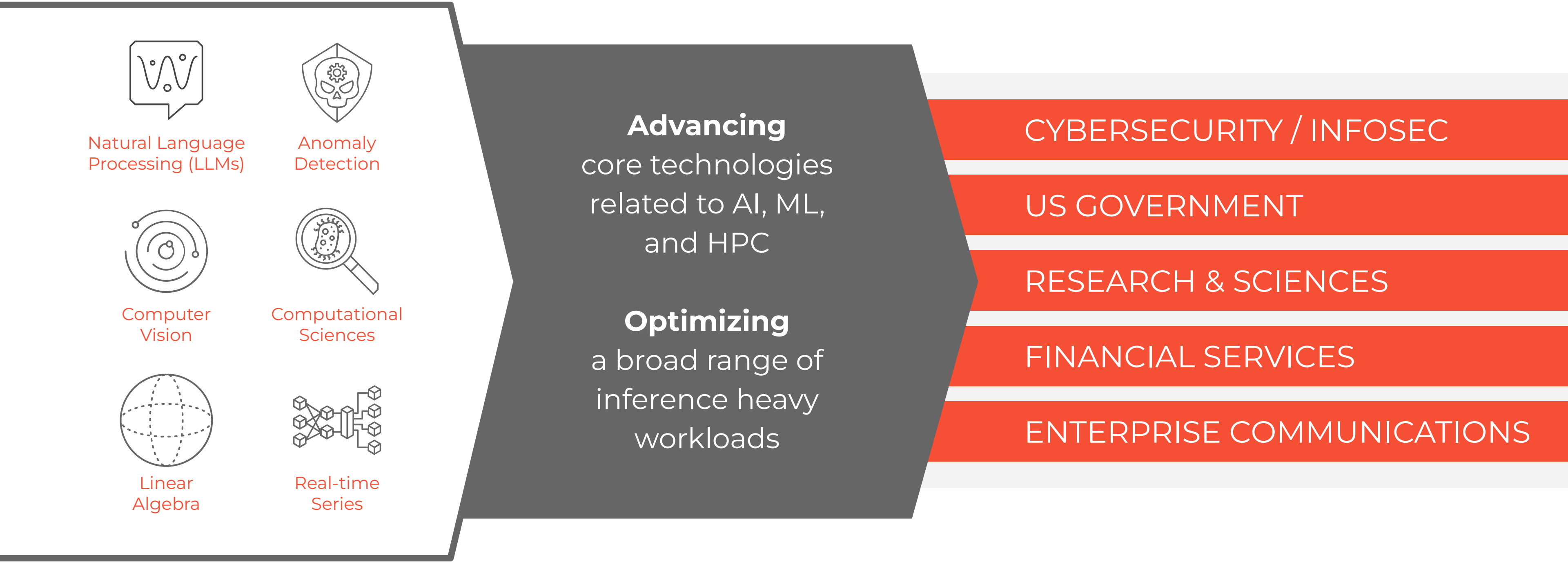

c1
c1
C2
C2
O
O
N
H
N
C >> OCc1ccc(CNC(=O)C2CCCN(C(=O)C(C)(C)C)C2)cc1<
C >> OCc1ccc(CNC(=O)C2CCCN(C(=O)C(C)(C)C)C2)cc1<
C >> OCc1ccc(CNC(=O)C2CCCN(C(=O)C(C)(C)C)C2)cc1<
C >> OCc1ccc(CNC(=O)C2CCCN(C(=O)C(C)(C)C)C2)cc1<
C >> OCc1ccc(CNC(=O)C2CCCN(C(=O)C(C)(C)C)C2)cc1<
0.8
0.6
0.4
0.2


Convolution/pooling/relu
C1=CC2=C(C3=C(C=CC=N3)C=C2)N=C1
Drug
Each row represents C,1, =, 2, (, 3, N and) respectively
Convolution/pooling/relu
Genetic Feature
Cell

Up to 10,000 predictions/s on Groq!

Image Credit:
Left: Jang, W.D. et al, PredPS: Attention-based graph neural network for predicting stability of compounds in human plasma.
Center: Bagal, V. et al, MolGPT: Molecular Generation Using a Transformer-Decoder Model
Right: Liu, P. et al, Improving prediction of phenotypic drug response on cancer cell lines using deep convolutional network

Groq and Scientific Applications



Natural Language Processing (LLMs)

Anomaly Detection

Computer Vision

Computational Sciences

Linear Algebra

Real-time Series

Advancing core technologies related to AI, ML, and HPC

Optimizing a broad range of inference heavy workloads

CYBERSECURITY / INFOSEC

US GOVERNMENT

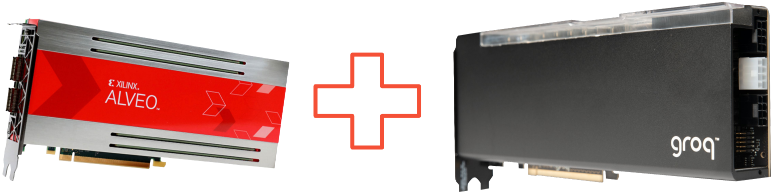
RESEARCH & SCIENCES

FINANCIAL SERVICES

ENTERPRISE COMMUNICATIONS

X-Ray Detector Signal Processing

- Next generation of x-ray sources will be over 100 times brighter
- 1 Tbps bandwidth off the detector chip
  - 16-bit resolution 256x256 image
- Processing this data enables
  - Faster time to observation
  - Focus on rare event
- Codesign with FPGAs for real-time access to detector data, avoid slow PCIE transfers



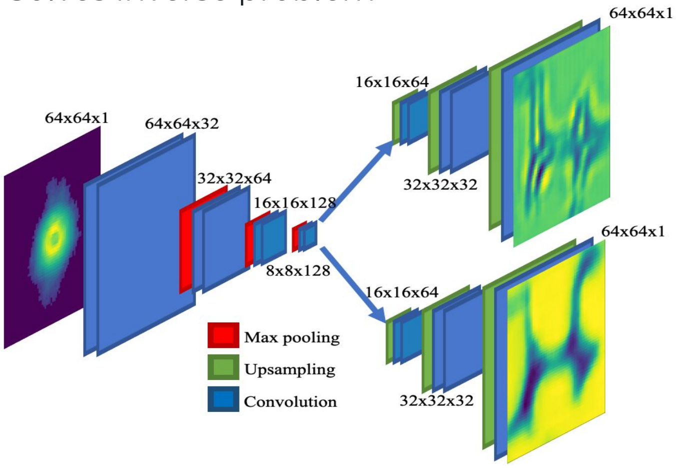
Advanced Photon Source at Argonne

X-Ray Physics Compression Models

Models to sift desired data from noisy X-ray diffraction signals

PtychoNN: Deep learning of ptychographic imaging

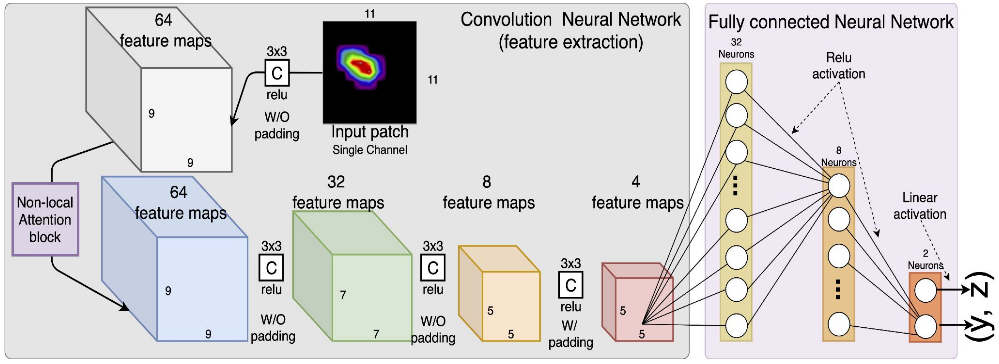
- Two-headed encoder-decoder network
- Predict amplitude & phase of incoming photon
- Solves inverse problem



Cherukara et al. https://pubs.aip.org/aip/apl/article/117/4/044103/39570/AI-enabled-high-resolution-scanning-coherent https://github.com/mcherukara/PtychoNN?tab=readme-ov-file

BraggNN: Bragg peak finding

- Convolutional neural network
- Predicts peak position
- Traditional algorithms take weeks of HPC



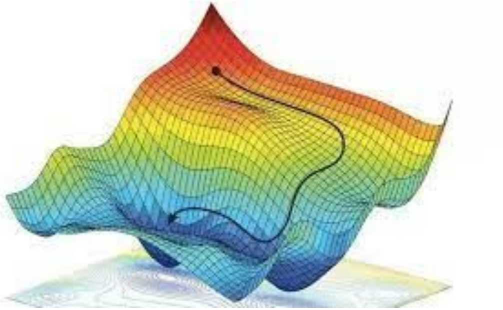
Zhengchun et al. https://arxiv.org/abs/2008.08198 https://github.com/lzhengchun/BraggNN?tab=readme-ov-fil

Intro to QUBO

What is QUBO?

QUBO - **Quadratic** Unconstrained **Binary** Optimization

- Mathematical framework for solving optimization problems
- Involves **binary** variables
- Quadratic objective function expresses problem's objectives and constraints



$$y = -5x_1 - 3x_2 - 8x_3 - 6x_4 + 4x_1x_2 + 8x_1x_3 + 2x_1x_3 + 10x_3x_4$$

$$\begin{pmatrix} x_1 & x_2 & x_3 & x_4 \end{pmatrix} \times \begin{pmatrix} -5 & 4 & 8 & 0 \\ 0 & -3 & 2 & 0 \\ 0 & 0 & -8 & 10 \\ 0 & 0 & 0 & -6 \end{pmatrix} \times \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}$$

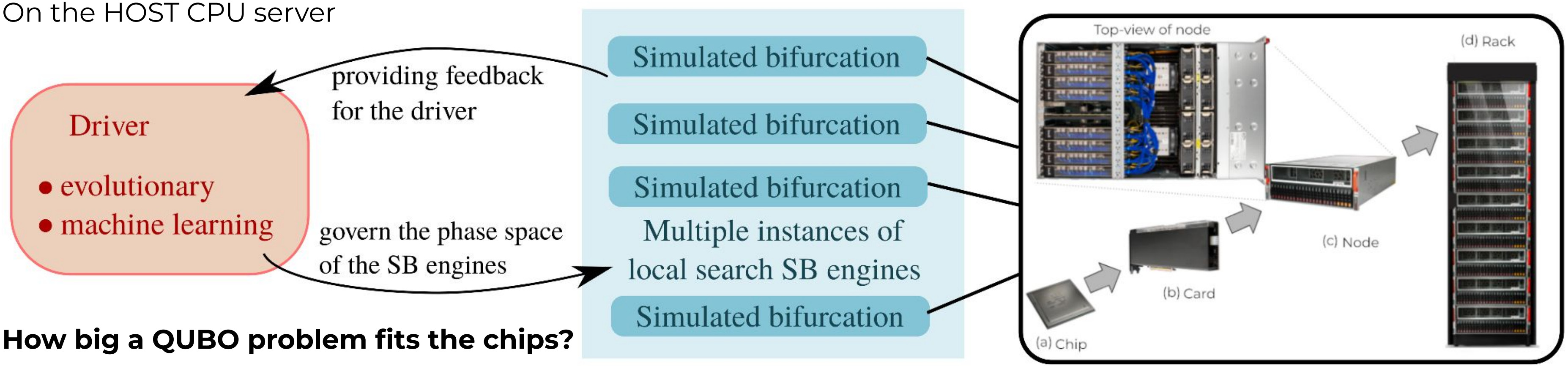
Goal is to **minimize y = x^T • Q • x** where x is a vector of binary decision variables and **Q is a square matrix of constants**

- It's common to assume that the **Q** matrix is **symmetric or** in **upper/lower triangular** form which can be assumed without loss of generality

The concept of the SB solver

On the HOST CPU server

On the LPU (+ CPU oracle)



Driver

- evolutionary
- machine learning

providing feedback for the driver

govern the phase space of the SB engines

Simulated bifurcation

Simulated bifurcation

Simulated bifurcation

Multiple instances of local search SB engines

Simulated bifurcation

How big a QUBO problem fits the chips?

Single LPU:	9K x 9K
LPU node:	25K x 25K
LPU rack:	72K x 72K
10 LPU racks:	225K x 225K

Use cases:

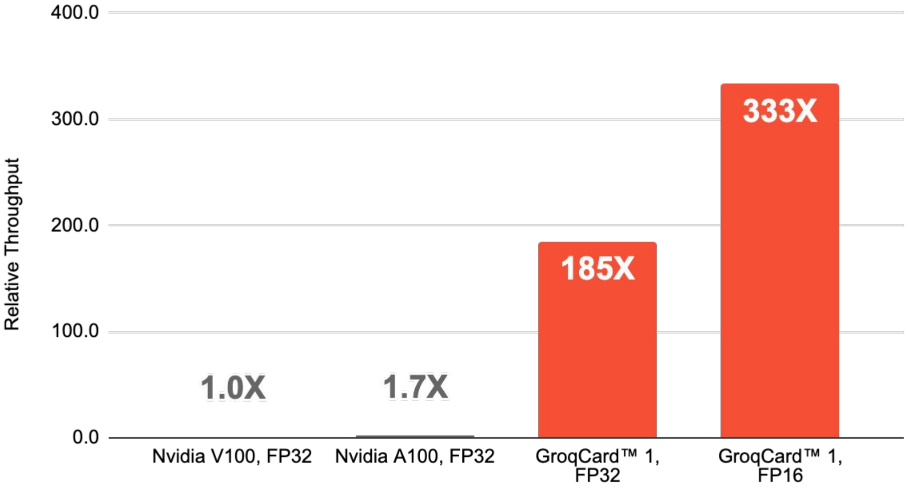
- Portfolio optimization
- Traffic routes

Accelerating Drug Discovery

Performance enables pharma / bio human innovation

Argonne NATIONAL LABORATORY

CANDIDATE TESTING THROUGHPUT



RELATIVE PERFORMANCE

Higher is Better.

Baselined to Nvidia V100, FP32

Groq Advantages

Accuracy with lower precision

Large on-chip memory

GroqCard 1 delivers >300x better throughput for drug discovery vs existing GPU-based competitor reducing the time-to-solution from days to minutes[1]

[1] As measured at the host across 31 unique model inferences, at batch size 512, with 12,800 batches per run.

Cyber security

Publicly disclosed customer & partners

Argonne NATIONAL LABORATORY

U.S. ARMY

ENTANGLEMENT

OAK RIDGE National Laboratory

iqt IN-Q-TEL™

OneNano

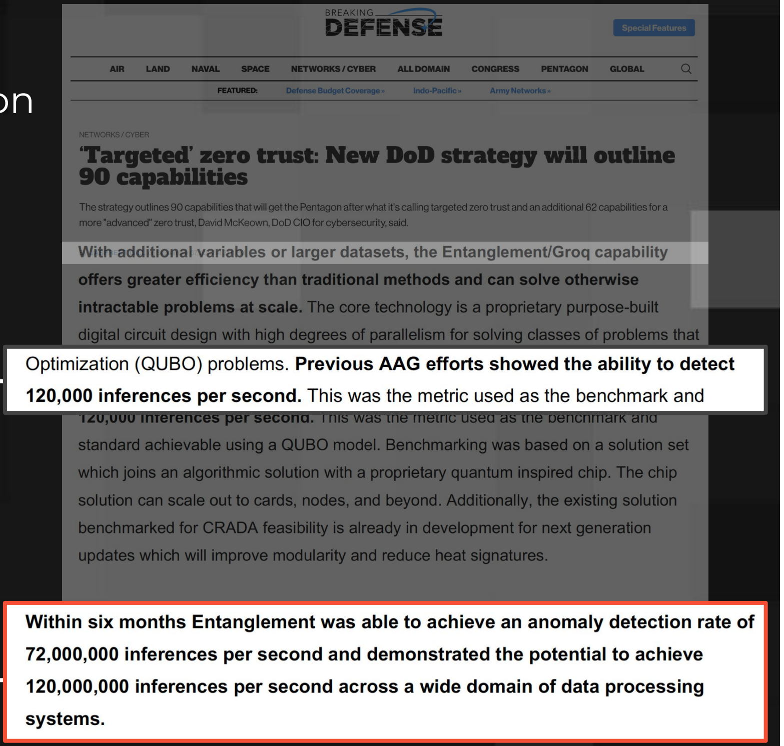
BittWare a molex company

Groq is also currently working with (non-publicly disclosed) **customers from the following markets**:

- Enterprise Web Communications
- Large-scale Banking Provider
- Automotive Manufacturer
- Hyperscalers

Excerpts
US Army Validation Report Summary

>600X



BREAKING DEFENSE

Special Features

AIR LAND NAVAL SPACE NETWORKS / CYBER ALL DOMAIN CONGRESS PENTAGON GLOBAL

FEATURED: Defense Budget Coverage » Indo-Pacific » Army Networks »

NETWORKS / CYBER

'Targeted' zero trust: New DoD strategy will outline 90 capabilities

The strategy outlines 90 capabilities that will get the Pentagon after what it's calling targeted zero trust and an additional 62 capabilities for a more "advanced" zero trust, David McKeown, DoD CIO for cybersecurity, said.

With additional variables or larger datasets, the Entanglement/Groq capability **offers greater efficiency than traditional methods and can solve otherwise intractable problems at scale.** The core technology is a proprietary purpose-built digital circuit design with high degrees of parallelism for solving classes of problems that

Optimization (QUBO) problems. **Previous AAG efforts showed the ability to detect 120,000 inferences per second.** This was the metric used as the benchmark and

120,000 inferences per second. This was the metric used as the benchmark and

standard achievable using a QUBO model. Benchmarking was based on a solution set which joins an algorithmic solution with a proprietary quantum inspired chip. The chip solution can scale out to cards, nodes, and beyond. Additionally, the existing solution benchmarked for CRADA feasibility is already in development for next generation updates which will improve modularity and reduce heat signatures.

Within six months Entanglement was able to achieve an anomaly detection rate of 72,000,000 inferences per second and demonstrated the potential to achieve 120,000,000 inferences per second across a wide domain of data processing systems.

XTX Acceleration

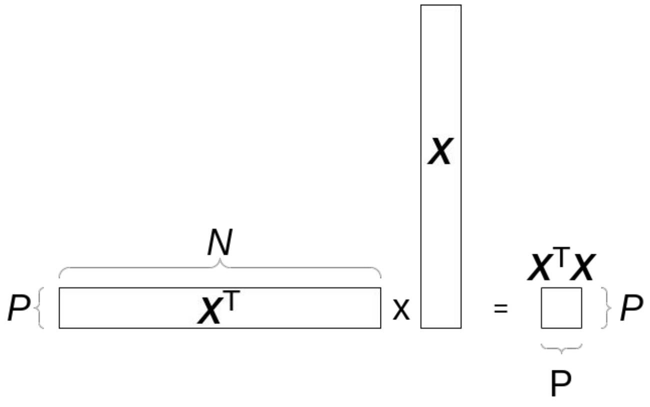
Build fast applications from tall and skinny matrix operations

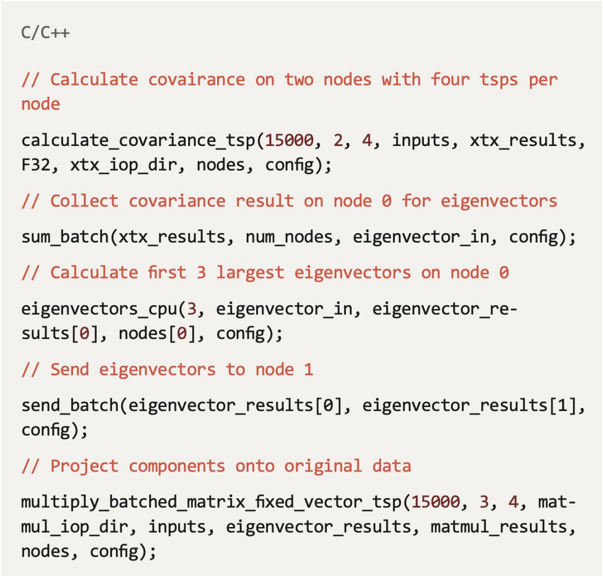
Library to build large scale physics and data-science applications:

- Express applications as multiplication of tall and skinny matrix to give large performance boost
- Typical matrix sizes (PxN):10k x 1B to 100k x 10B
- API to easily compose applications out of modular, high performance building blocks which run on GroqChip processors or CPUs
- API supports scaling from a single GroqChip to multiple racks

Application areas:

- Finance: correlation
- Physics: quantum error mitigation
- Data science: principal component analysis, multi-linear regression





C/C++

```
// Calculate covairance on two nodes with four tsps per node

calculate_covariance_tsp(15000, 2, 4, inputs, xtx_results, F32, xtx_iop_dir, nodes, config);

// Collect covariance result on node 0 for eigenvectors

sum_batch(xtx_results, num_nodes, eigenvector_in, config);

// Calculate first 3 largest eigenvectors on node 0

eigenvectors_cpu(3, eigenvector_in, eigenvector_results[0], nodes[0], config);

// Send eigenvectors to node 1

send_batch(eigenvector_results[0], eigenvector_results[1], config);

// Project components onto original data

multiply_batched_matrix_fixed_vector_tsp(15000, 3, 4, matmul_iop_dir, inputs, eigenvector_results, matmul_results, nodes, config);
```

Throughput Optimization with Groq™

Recap

1. Model architectures dictate parallelization strategies
2. Deterministic hardware means compiler experiments tell best configuration

groq™

Thank You!

cculver@groq.com

What is next?

What is next?

AGENDA

1. Systems Roadmap and Capability
2. Chip Determinism unlocks LPU Superpower
3. More Moore Scaling Benefits of Determinism

© 2024 Groq, Inc. | Groq AI Workshop

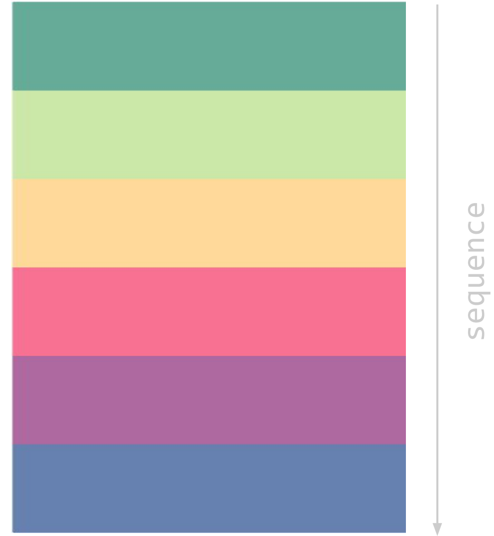
LLMs are

Getting Larger

This poses a problem for GPUs

- **Accuracy improves** as model sizes are increasing
- **Memory bound** and need fast memory access
- **Need inference** to be done in reasonable time **and** at reasonable cost

Large Language Model



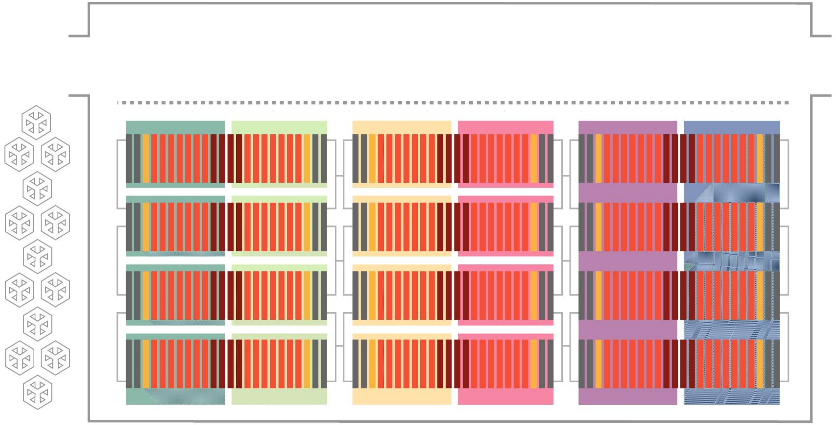
GPUs scale largely in time, some in space (clustering)

For large compute volume, GPUs iterate over multiple partitions of model code, weights etc in time

- Most of GPU time/energy spent paging weights & KV cache in/out of **HBM**
- **Highly Inefficient** → **High Cost / Token**
  - ✖ Low HBM bandwidth 1/100X of on-chip SRAM
  - ✖ High HBM access latency (300ns-1300ns)
  - ✖ High HBM access power (4-6 pJ/bit for R/W)
  - ✖ Need high-batch size to saturate compute (100s-1000s)
  - ✖ Poor GPU-to-GPU collectives with asynchronous communication (through switches) and high batch sizes results in high latency
- **Expensive BOM & Supply Concerns** with HBM, exotic packaging, network switches, etc.

Groq LPU scales largely in space as an assembly line

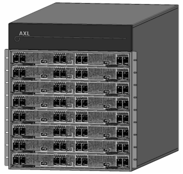
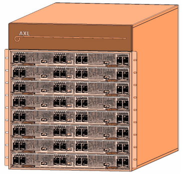
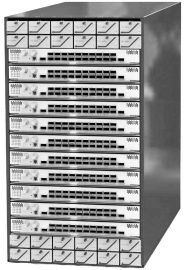
For large compute volume, LPUs partition model code across multiple chips to form an assembly line



- Break through the memory wall by computing on weights & KV cache from **SRAM**
- **Highly efficient → Lowest Cost / Token**
  - ✔ 100X higher bandwidth than HBM
  - ✔ Lowest SRAM access latency (<5 ns)
  - ✔ Lowest SRAM power (0.3 pJ/bit for R/W)
  - ✔ Saturate compute at low batch sizes
  - ✔ Efficient LPU-to-LPU collectives due to deterministic communication and low batch size
- Single chip module, no HBMs, no expensive external switches → abundant supply

Same Software

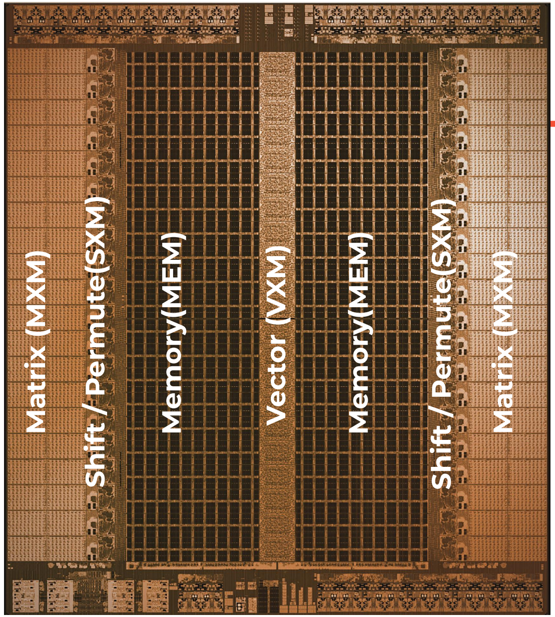
Compiles Across All Platforms

				
Silicon Generation	1: □	1: □	2: □	2: ⊞
LPU™ Accelerators Per Chassis	8 x V1-LPU™	32 x V1-LPU™	32 x V2-LPU™	336 x V2-LPU™
Single Core Cluster	264 x LPU™ (4 Racks)	4,128 x LPU (33 Racks)	40,960 x LPU (320 Racks)	680,064 x LPU (675 Racks)

* Enough SRAM to fit 250+ Llama-2 70B models

GROQ Enables

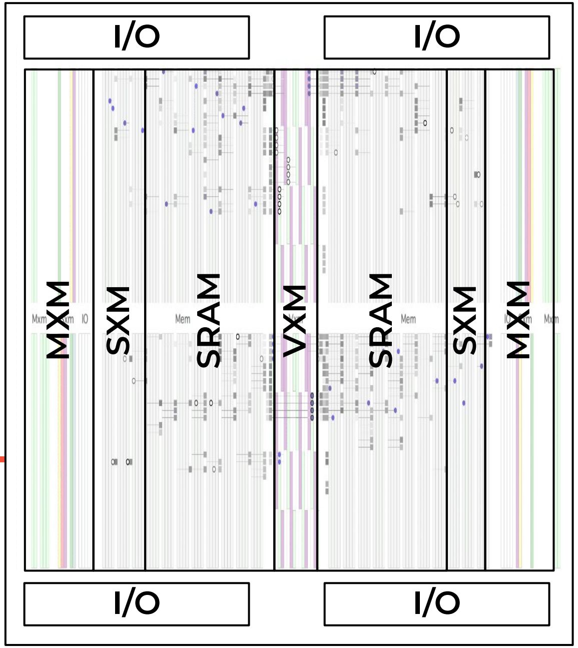
Software & Hardware Co-optimization



GROQ® COMPILER ENABLES

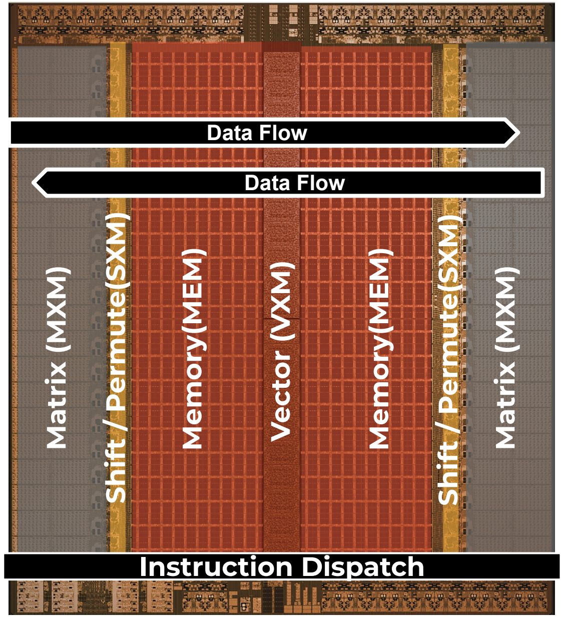
Hardware & Software

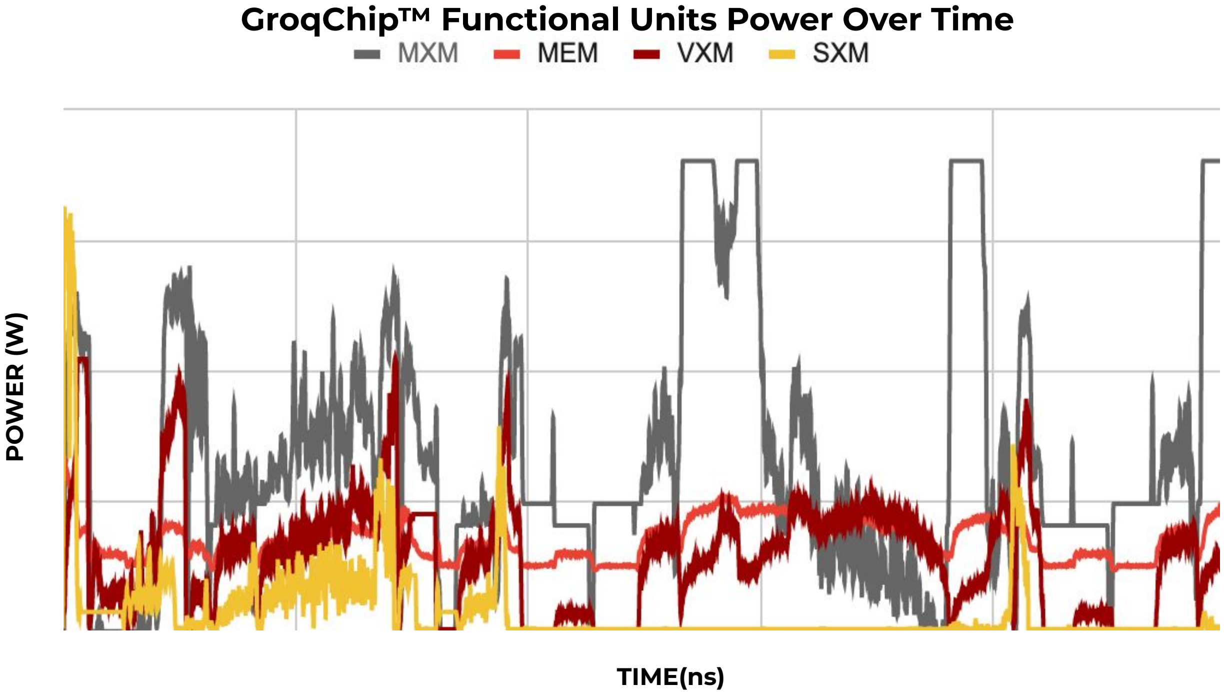
Co-optimization



GROQ® COMPILER

Enables Performance, Power, Ldi/dt, & Thermal Profiling

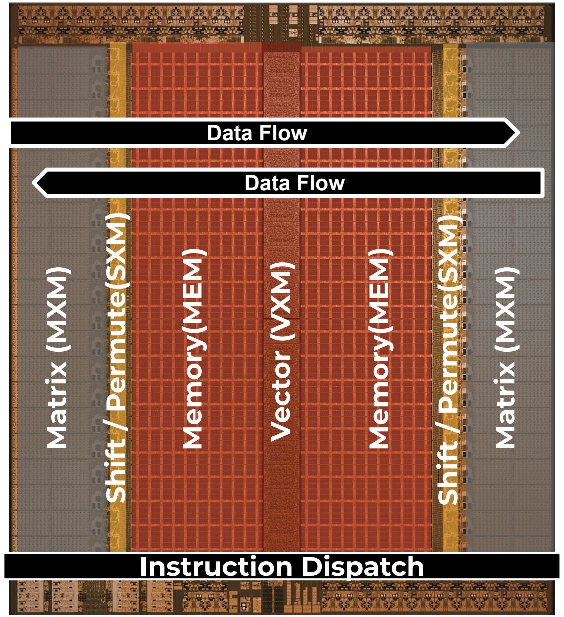


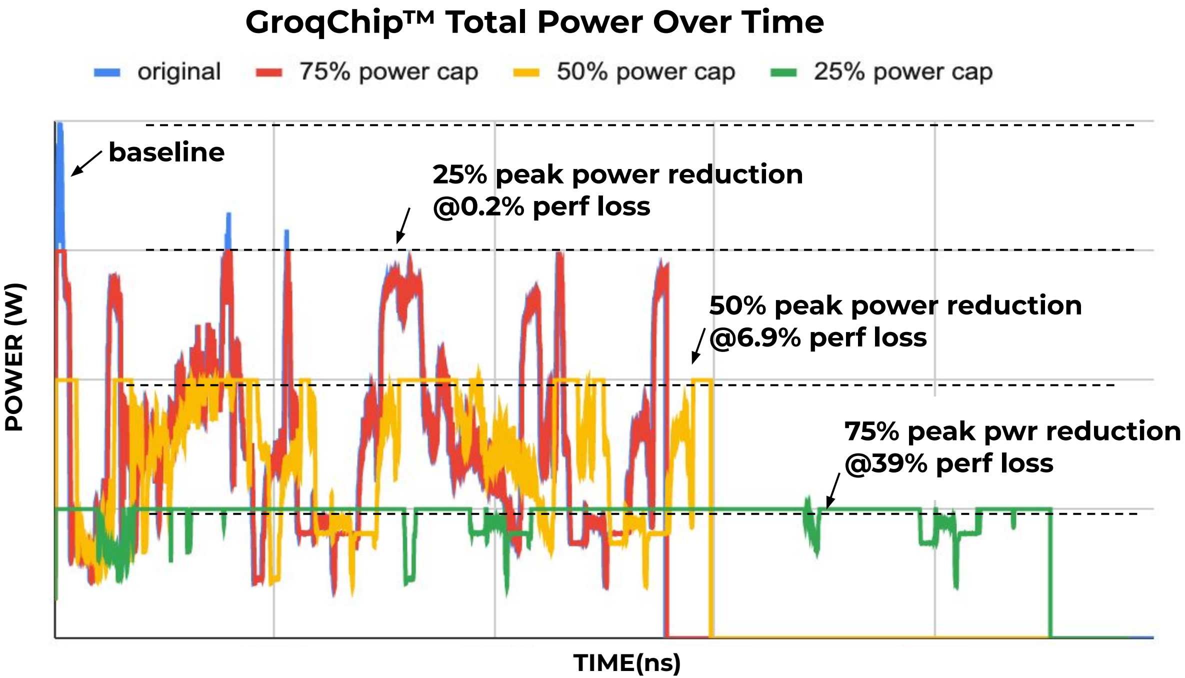


Groq Compiler can profile 100% deterministic power, temp, di/dt down to a “ns”

GROQ® COMPILER

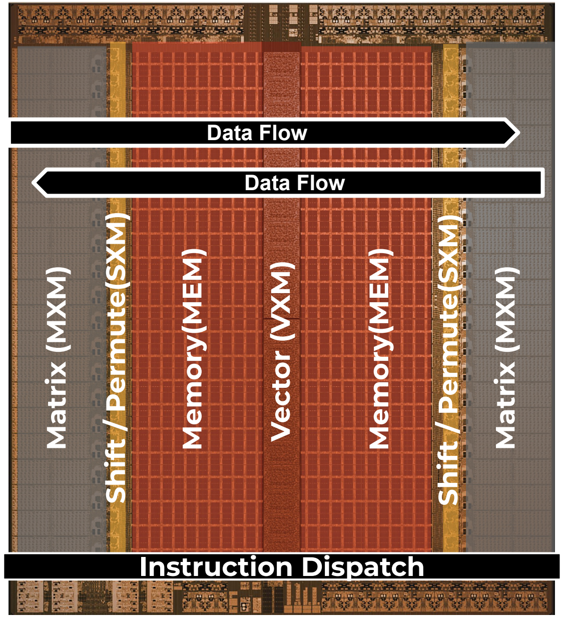
Enables Performance, Power, Ldi/dt, & Thermal Control

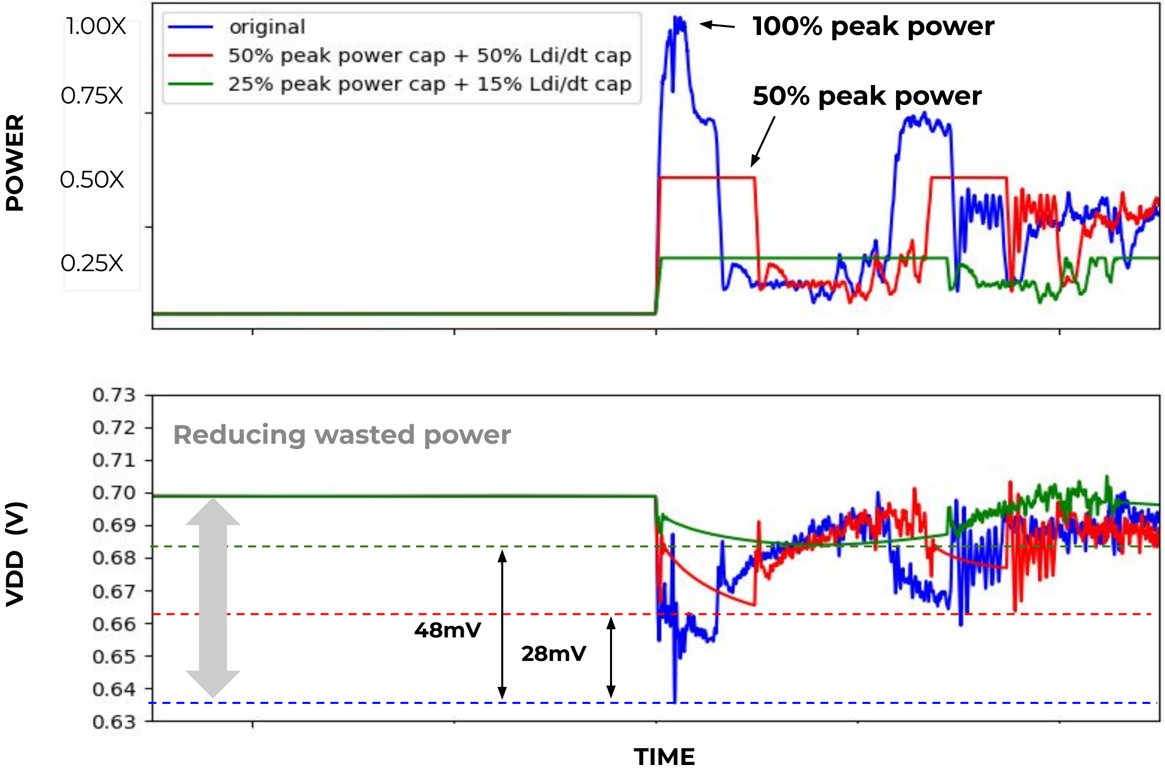




Groq Compiler controls LPU power, temp, di/dt down to a “ns” - key for reliability & compute density (2D/3DIC)

GROQ COMPILER ENABLES
Ldi/dt Control



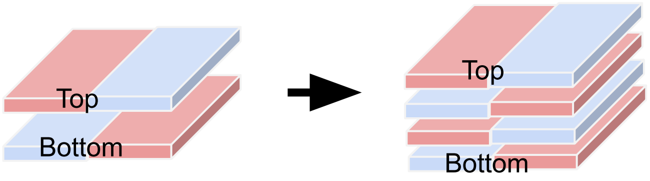


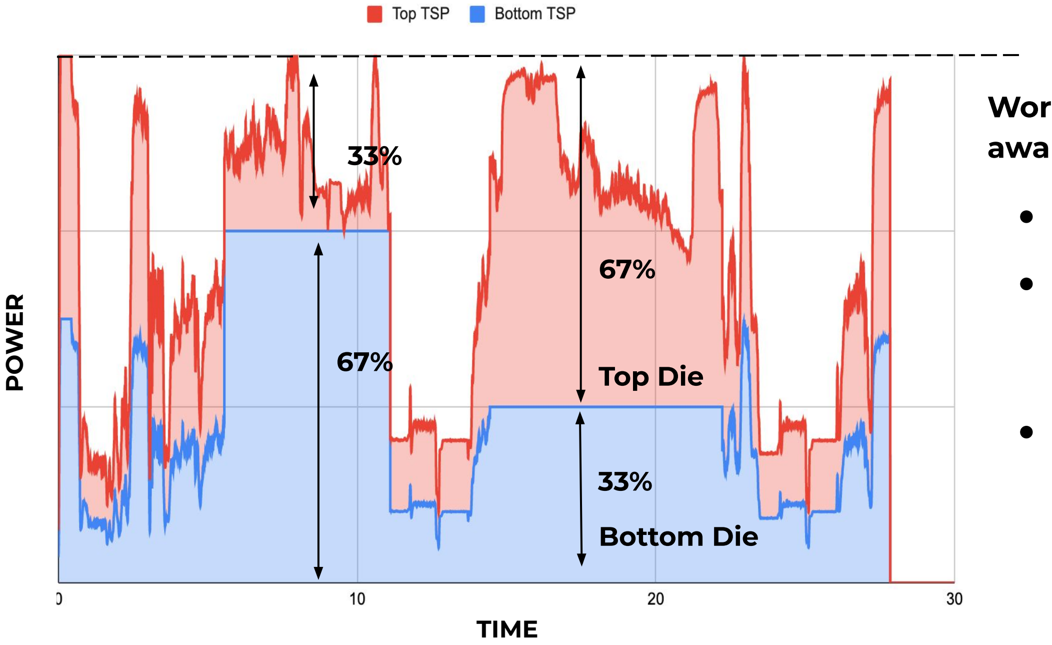
Groq Compiler optimizes Ldi/dt in 2D/3D module space/time

GROQ™ COMPILER ENABLES

Thermal Optimization for 3D Logic-on-Logic Stacking

Deterministic Functional Units Scheduling Allows Complementary Power Consumption across two or more dies in a 3DIC

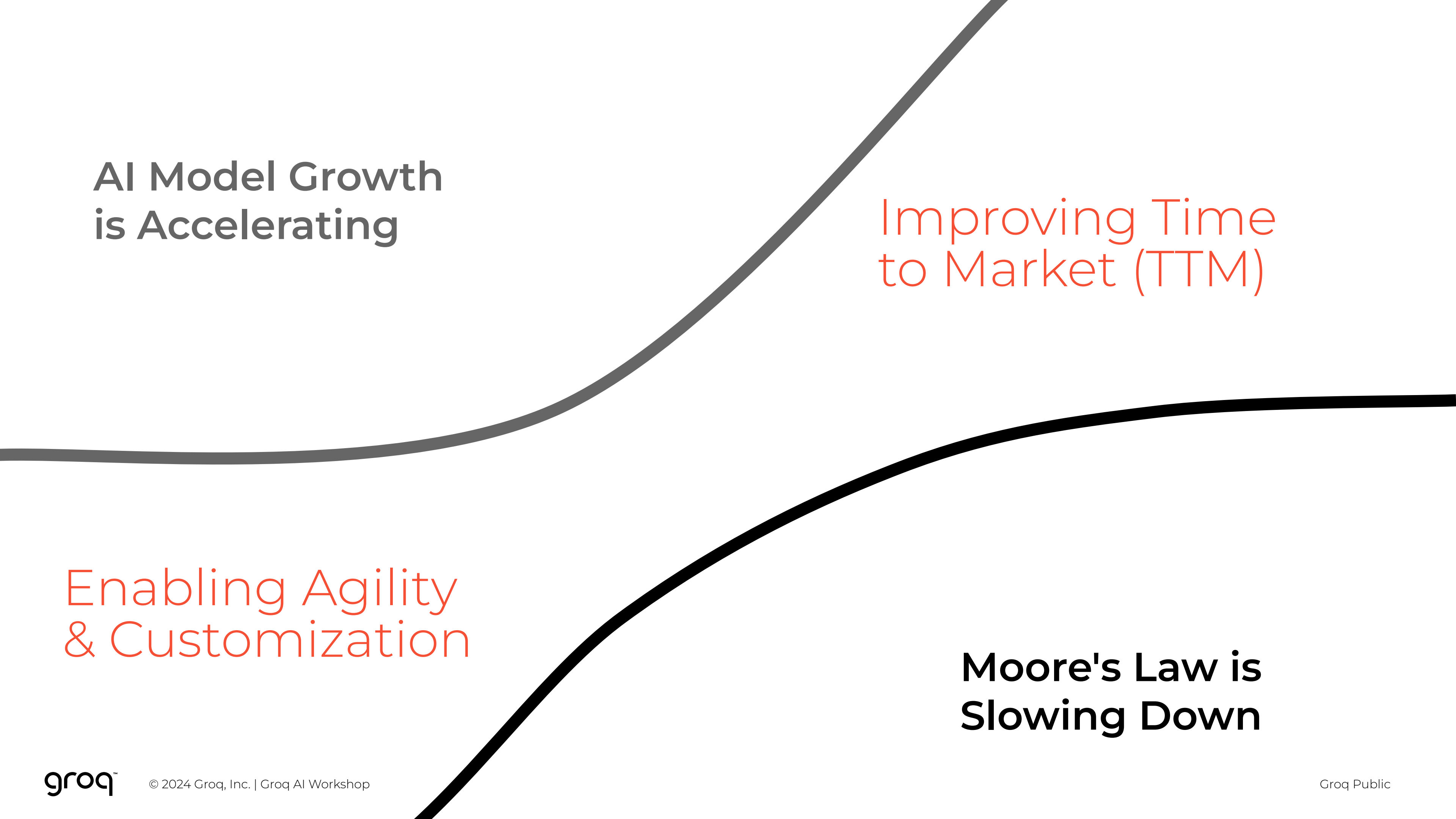




Workload scheduled across functional units with awareness of location and thermal impact

- Multiple 3DIC share the same thermal envelope.
- Each chip can allocate a power budget from the total budget pool while maintaining thermal envelope
- PVT monitors used for calibration before deployment, and act as guardrails if the compiler mis-predicts power consumption after deployment

Groq Compiler optimizes Thermals in 2D/3D module space/time



AI Model Growth is Accelerating

Improving Time to Market (TTM)

Enabling Agility & Customization

Moore's Law is Slowing Down

SCALABLE

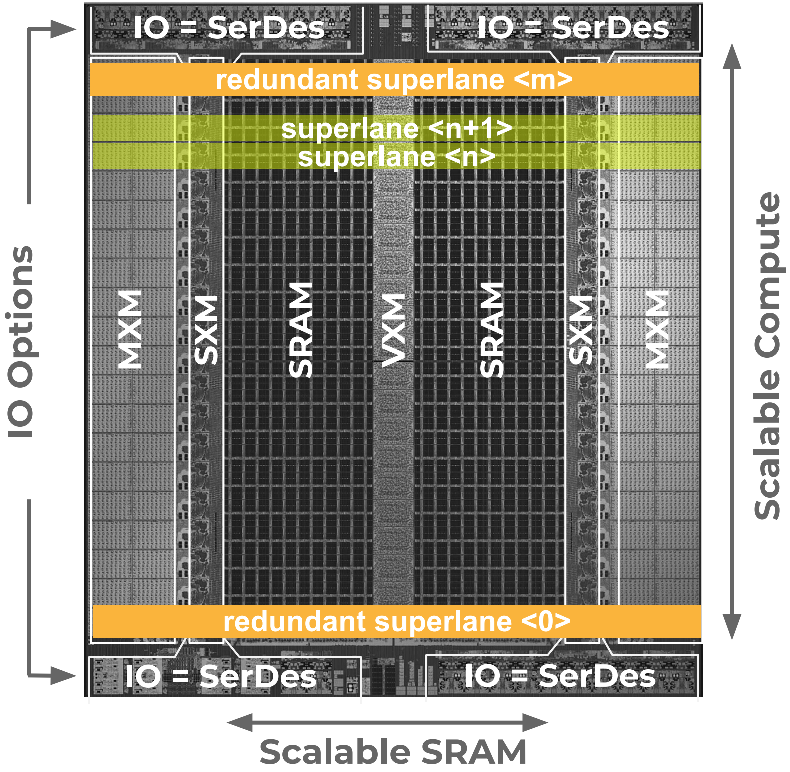
Silicon Tiler For Fast Time-to-market

Multiple Interconnect Options

- C2C for high-radix interconnect
- UCIe for MCM connected sidecar accelerator
- Scalable SXM for BW to/from IO and Compute

Scalable compute architecture

- SRAM scalable capacity
- VXM with scalable number of PEs
- MXM with scalable matrix sizes



Next-gen Silicon Compiler

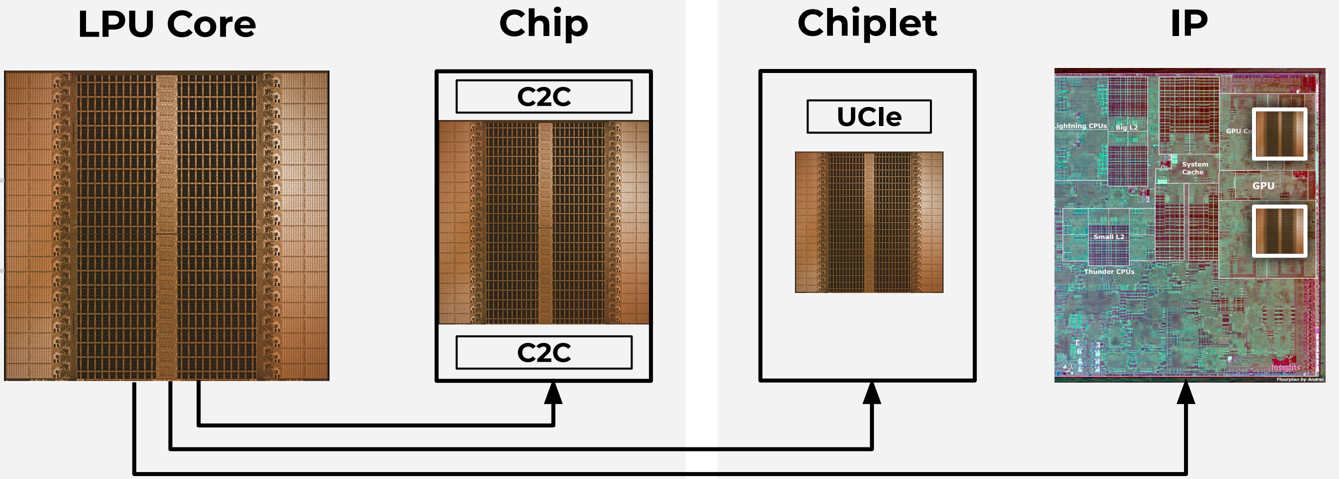
Enabling Groq Silicon Compiler & Ecosystem

Scalable SRAM

(220-440MiB)
with 3D SRAM
extension

Scalable Compute

16 SL: 256x256
20 SL:, 320x320
24 SL: 384x384

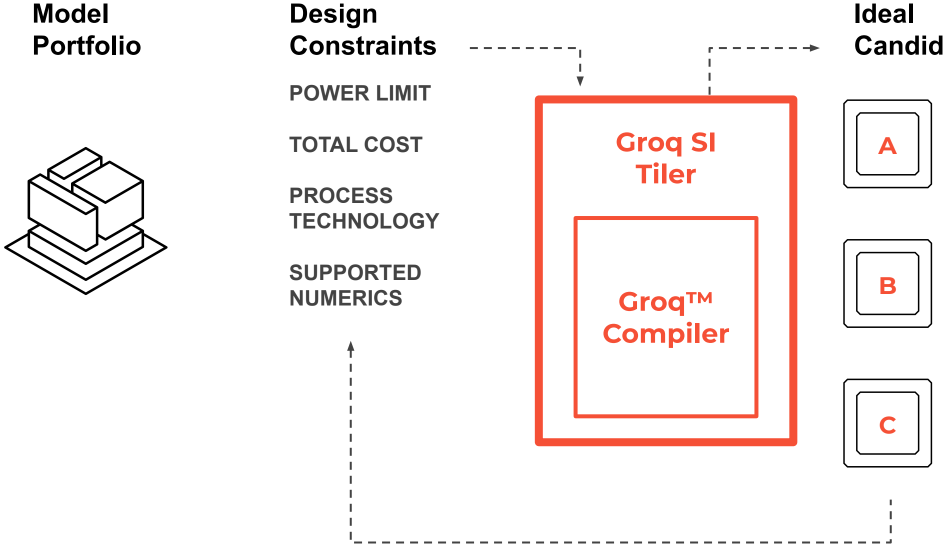


Design Space Exploration (DSE)

AI Assisted Exploration & Design

INPUTS

OUTPUTS



Model Portfolio

Design Constraints

POWER LIMIT

TOTAL COST

PROCESS TECHNOLOGY

SUPPORTED NUMERICS

Groq SI Tiler

Groq™ Compiler

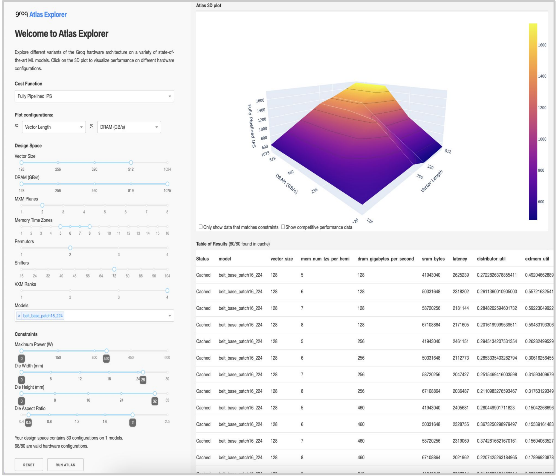
Ideal Candidates

A

B

C

Performance & SI Costs



groq Atlas Explorer

Welcome to Atlas Explorer

Explore different variants of the Groq hardware architecture on a variety of state-of-the-art ML models. Click on the 3D plot to visualize performance on different hardware configurations.

Cost Function

Fully Pipelined IPS

Plot configurations:

x: Vector Length y: DRAM (GB/s)

Design Space

Vector Size: 128 256 320 512 1024

DRAM (GB/s): 128 256 460 819 1075

MXM Planes: 1 2 3 4 5 6 7 8

Memory Time Zones: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Permutors: 1 2 3 4

Shifters: 16 24 32 40 48 56 64 72 80 88 96 104

VXM Ranks: 1 2 3 4

Models: × beit_base_patch16_224

Constraints

Maximum Power (W): 0 150 300 350 450 600

Die Width (mm): 0 6 12 18 24 25 30

Die Height (mm): 0 8 16 24 32 35

Die Aspect Ratio: 0.4 0.5 0.8 1.2 1.6 2 2.5

Your design space contains 80 configurations on 1 models.
68/80 are valid hardware configurations.

RESET RUN ATLAS

Atlas 3D plot

□ Only show data that matches constraints □ Show competitive performance data

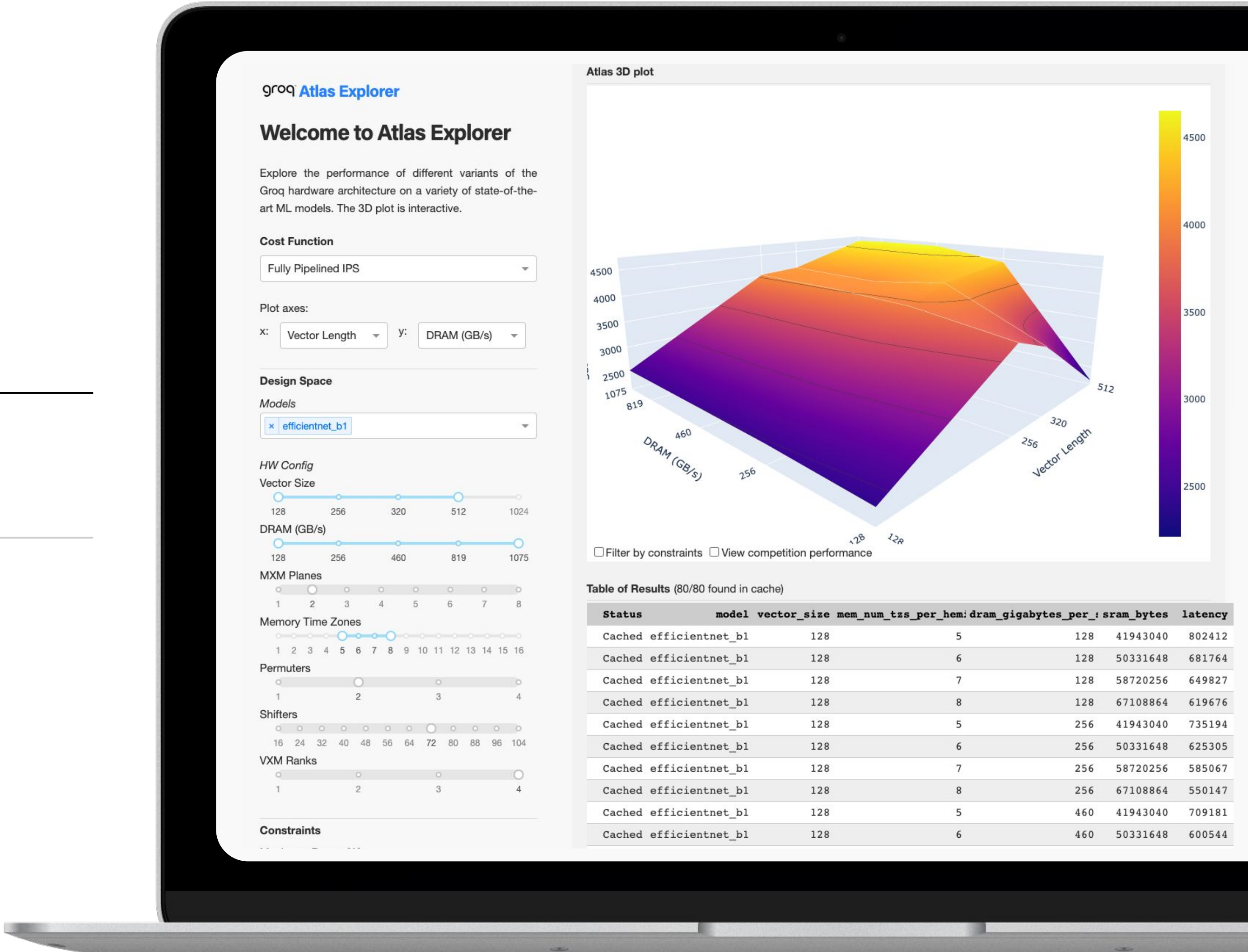
Table of Results (80/80 found in cache)

Status	model	vector_size	mem_num_tzs_per_hemi	dram_gigabytes_per_second	sram_bytes	latency	distributor_util	extmem_util
Cached	beit_base_patch16_224	128	5	128	41943040	2625239	0.2722826378855411	0.49204662889
Cached	beit_base_patch16_224	128	6	128	50331648	2318202	0.2611360010905003	0.55721632541
Cached	beit_base_patch16_224	128	7	128	58720256	2181144	0.2848202594601732	0.59223049922
Cached	beit_base_patch16_224	128	8	128	67108864	2171605	0.2016199999539511	0.59483193306
Cached	beit_base_patch16_224	128	5	256	41943040	2461151	0.2945134207531354	0.26282499529
Cached	beit_base_patch16_224	128	6	256	50331648	2112773	0.2853335403282794	0.30616256455
Cached	beit_base_patch16_224	128	7	256	58720256	2047427	0.2515469416003598	0.31593409679
Cached	beit_base_patch16_224	128	8	256	67108864	2036487	0.2110983276593467	0.31763129349
Cached	beit_base_patch16_224	128	5	460	41943040	2405681	0.280449901711823	0.15042268696
Cached	beit_base_patch16_224	128	6	460	50331648	2328755	0.3673250298979497	0.15539161483
Cached	beit_base_patch16_224	128	7	460	58720256	2319069	0.3742816621670161	0.15604063527
Cached	beit_base_patch16_224	128	8	460	67108864	2021962	0.2207425263184965	0.17896923878

Enabling highly productive and scalable discovery at **The Speed of Software**

DEMOS

DSE



groq Atlas Explorer

Welcome to Atlas Explorer

Explore the performance of different variants of the Groq hardware architecture on a variety of state-of-the-art ML models. The 3D plot is interactive.

Cost Function

Fully Pipelined IPS

Plot axes:

x: Vector Length y: DRAM (GB/s)

Design Space

Models

× efficientnet_b1

HW Config

Vector Size

DRAM (GB/s)

MXM Planes

Memory Time Zones

Permuters

Shifters

VXM Ranks

Constraints

Atlas 3D plot

☐ Filter by constraints ☐ View competition performance

Table of Results (80/80 found in cache)

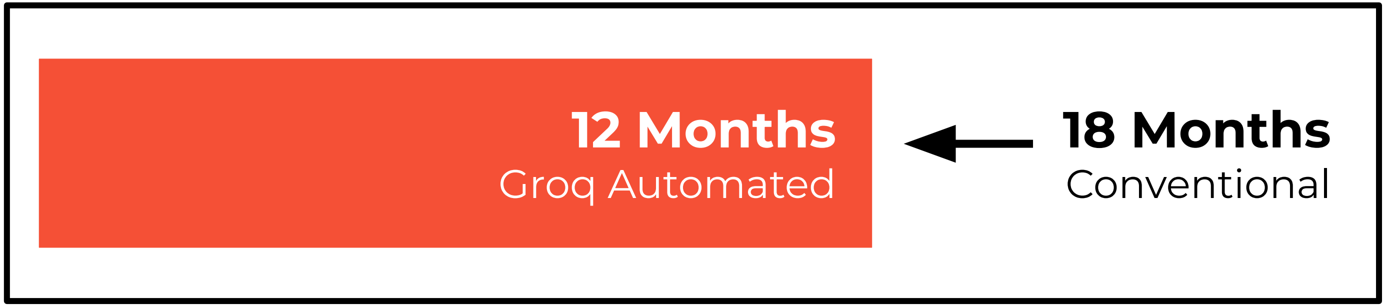
Status	model	vector_size	mem_num_tzs_per_hem:	dram_gigabytes_per_:	sram_bytes	latency
Cached	efficientnet_b1	128	5	128	41943040	802412
Cached	efficientnet_b1	128	6	128	50331648	681764
Cached	efficientnet_b1	128	7	128	58720256	649827
Cached	efficientnet_b1	128	8	128	67108864	619676
Cached	efficientnet_b1	128	5	256	41943040	735194
Cached	efficientnet_b1	128	6	256	50331648	625305
Cached	efficientnet_b1	128	7	256	58720256	585067
Cached	efficientnet_b1	128	8	256	67108864	550147
Cached	efficientnet_b1	128	5	460	41943040	709181
Cached	efficientnet_b1	128	6	460	50331648	600544

Workload to Silicon

Driving Time-to-market Improvement

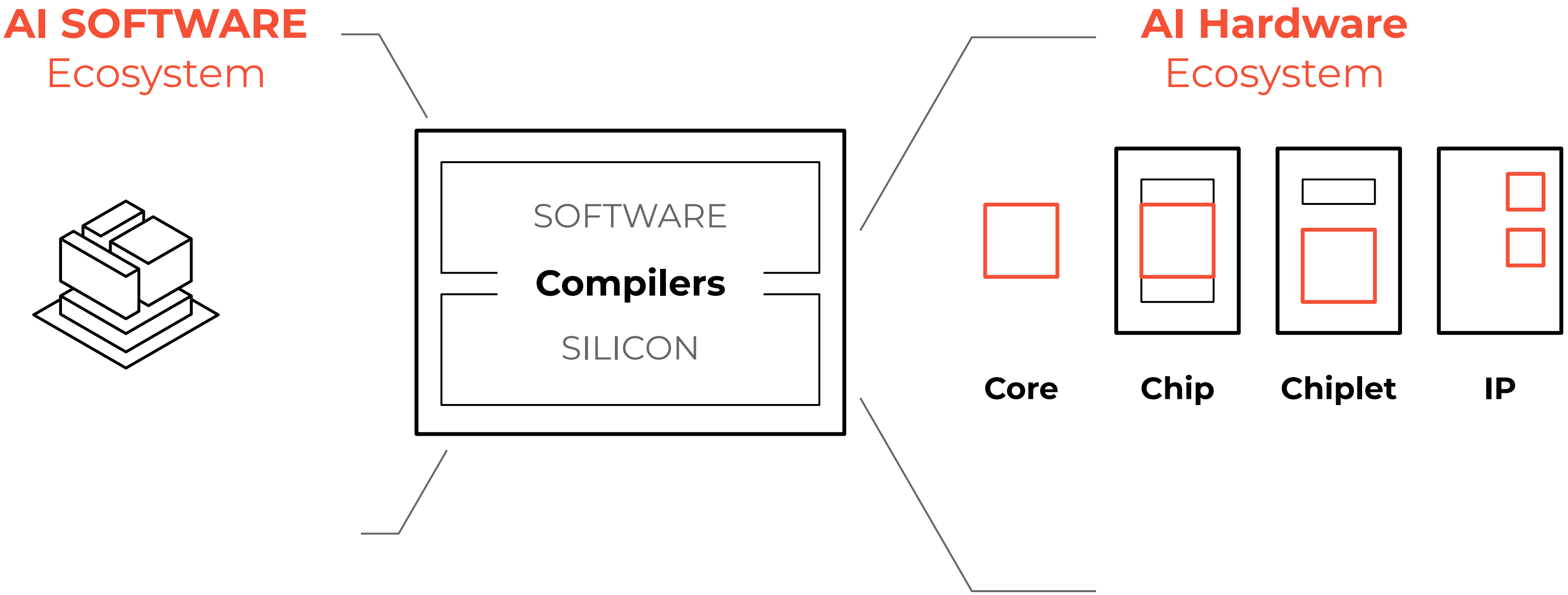
Silicon Design Cycle Improvement

Design Space
Exploration &
Silicon Tiler
TTM Improvements



12 Months
Groq Automated

18 Months
Conventional



AI SOFTWARE
Ecosystem

SOFTWARE

Compilers

SILICON

AI Hardware
Ecosystem

Core **Chip** **Chiplet** **IP**

Data Center Reliability Approaching Automotive

Large AI models
train on >100,000 AI SoCs

Silent Data
Corruption can have
>30% performance impact

Need a high reliability, testable, predictable, and reproducible hardware

Cores that don't count

Peter H. Hochschild
Paul Turner
Jeffrey C. Mogul
Google
Sunnyvale, CA, US

Rama Govindaraju
Parthasarathy
Ranganathan
Google
Sunnyvale, CA, US

David E. Culler
Amin Vahdat
Google
Sunnyvale, CA, US

Abstract

We are accustomed to thinking of computers as fail-stop, especially the cores that execute instructions, and most system software implicitly relies on that assumption. During most of the VLSI era, processors that passed manufacturing tests and were operated within specifications have insulated us from this fiction. As fabrication pushes towards smaller feature sizes and more elaborate computational structures, and as increasingly specialized instruction-silicon pairings are introduced to improve performance, we have observed ephemeral computational errors that were not detected during manufacturing tests. These defects cannot always be mitigated by techniques such as microcode updates, and may be correlated to specific components within the processor, allowing small code changes to effect large shifts in reliability. Worse, these failures are often "silent" – the only symptom is an erroneous computation.

We refer to a core that develops such behavior as "mercurial." Mercurial cores are extremely rare, but in a large fleet of servers we can observe the disruption they cause, often enough to see them as a distinct problem – one that will require collaboration between hardware designers, processor vendors, and systems software architects.

This paper is a call-to-action for a new focus in systems research; we speculate about several software-based approaches to mercurial cores, ranging from better detection and isolating mechanisms, to methods for tolerating the silent data corruption they cause.

ACM Reference Format:
Peter H. Hochschild, Paul Turner, Jeffrey C. Mogul, Rama Govindaraju, Parthasarathy Ranganathan, David E. Culler, and Amin Vahdat. 2021. Cores that don't count. In *Workshop on Hot Topics in Operating Systems (HotOS '21), May 31–June 2, 2021, Ann Arbor, MI, USA*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3458336.3465297

1 Introduction

Imagine you are running a massive-scale data-analysis pipeline in production, and one day it starts to give you wrong answers – somewhere in the pipeline, a class of computations are yielding corrupt results. Investigation fingers a surprising cause: an innocuous change to a low-level library. The change itself was correct, but it caused servers to make heavier use of otherwise rarely-used instructions. Moreover, only a small subset of the server machines are repeatedly responsible for the errors.

This happened to us at Google. Deeper investigation revealed that these instructions malfunctioned due to manufacturing defects, in a way that could only be detected by checking the results of these instructions against the expected results; these are "silent" *corrupt execution errors*, or CEEs. Wider investigation found multiple different kinds of CEEs; that the detected incidence is much higher than software engineers expect; that they are not just incremental increases in the background rate of hardware errors; that these can manifest long after initial installation; and that they typically afflict specific cores on multi-core CPUs, rather than the entire chip. We refer to these cores as "mercurial."

Because CEEs may be correlated with specific execution units within a core, they expose us to large risks appearing suddenly and unpredictably for several reasons, including seemingly-minor software changes. Hyperscalers have a responsibility to customers to protect them against such risks. For business reasons, we are unable to reveal exact CEE rates, but we observe on the order of a few mercurial cores per several thousand machines – similar to the rate reported by Facebook [8]. The problem is serious enough for us to have applied many engineer-decades to it.

While we have long known that storage devices and networks can corrupt data at rest or in transit, we are accustomed to thinking of processors as fail-stop. VLSI has always depended on sophisticated manufacturing testing to detect defective chips. When defects escaped, or manifested with aging, they were assumed to become fail-stop or at least fail-noisy: triggering machine-checks or giving wrong answers for many kinds of instructions. When truly silent failures occurred, they

HotOS '21, May 31–June 2, 2021, Ann Arbor, MI, USA
© 2021 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-8438-4/21/05.
https://doi.org/10.1145/3458336.3465297

Resilient

Language Processing Unit™ Accelerator

Interconnect resilience

Low-BER FEC enabling 99.999% uptime

- Redundant C2Cs wired at the System Level
- Bad C2C lanes bypassed in system

Compute and memory resilience

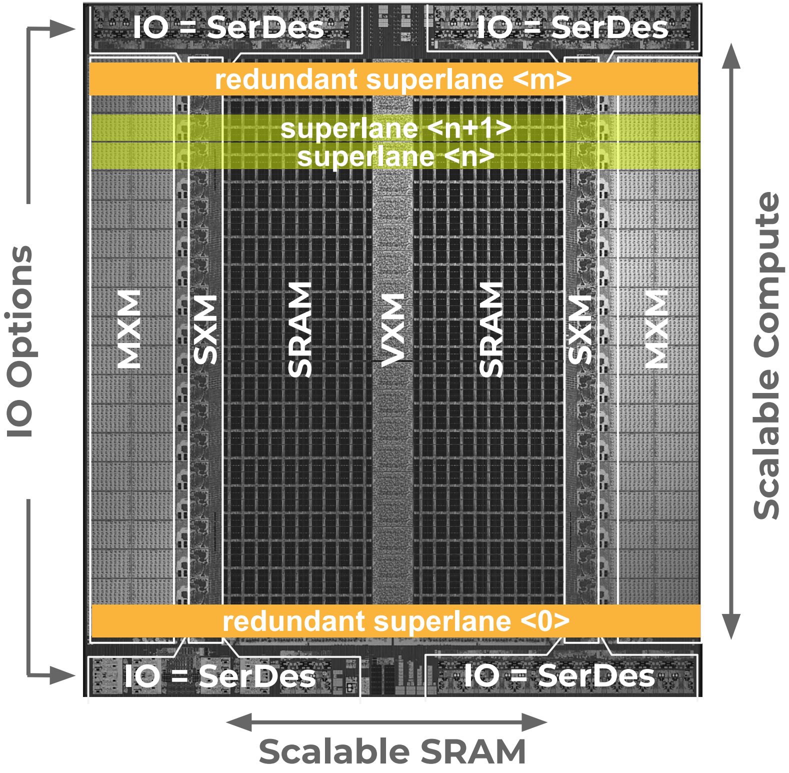
MXM checksum for SDC mitigation

- Detecting in compute errors

SRAM / Interconnect ECC protection

Repairable for yield and quality improvements

Redundant SLs for improved yield/reliability



groq™

Thank You!