Design Space Exploration and Analysis for AI Compilers

Dr. Size Zheng

About Me

I am now machine learning system researcher scientist at ByteDance. I am in TopSeed program. I completed my Ph.D. in the School of CS at Peking University, where I was advised by Prof. Yun Liang. I also worked with Professor Luis Ceze on LLM serving and optimization from September 2023 to January 2024 as visiting Ph.D. in SAMPL at the University of Washington. My recent publications investigate new algorithms, abstractions, and frameworks for efficient code generation on CPU and GPU. My research has been recognized with MICRO, ASPLOS, ISCA, HPCA, TPDS, DAC, and MLSys. I received my B.S. degree in the department of Computer Intelligence Science at Peking University. I am PC member of ChinaSys; reviewer of TPDS and TACO; sub-reviewer of MICRO, PPoPP, MLSys, ICS, and ICCAD.

Selected Publications

- **[MICRO 2023] Size Zheng**, Siyuan Chen, et al. TileFlow: A Framework for Modeling Fusion Dataflow via Tree-based Analysis
- **[DAC 2023] Size Zheng**, Siyuan Chen, et al. Memory and Computation Coordinated Mapping of DNNs onto Complex Heterogeneous SoC.
- **[HPCA 2023] Size Zheng**, Siyuan Chen, et al. Chimera: An Analytical Optimizing Framework for Effective Compute-intensive Operators Fusion
- **[ISCA 2022] Size Zheng**, Renze Chen, et al. AMOS: Enabling Automatic Mapping for Tensor Computations On Spatial Accelerators with Hardware Abstraction .
- **[TPDS 2021] Size Zheng**, Renze Chen, et al. NeoFlow: A Flexible Framework for Enabling Efficient Compilation for High Performance DNN Training
- **[ASPLOS 2020] Size Zheng**, Yun Liang, et al. FlexTensor: An Automatic Schedule Exploration and Optimization Framework for Tensor Computation on Heterogeneous System

Outline

1 Background

- AI Chip
- AI Algorithm
- AI Compiler

2 Techniques

- Compiler for DNN Graph
- Compiler for Operator
- Compiler for Distributed

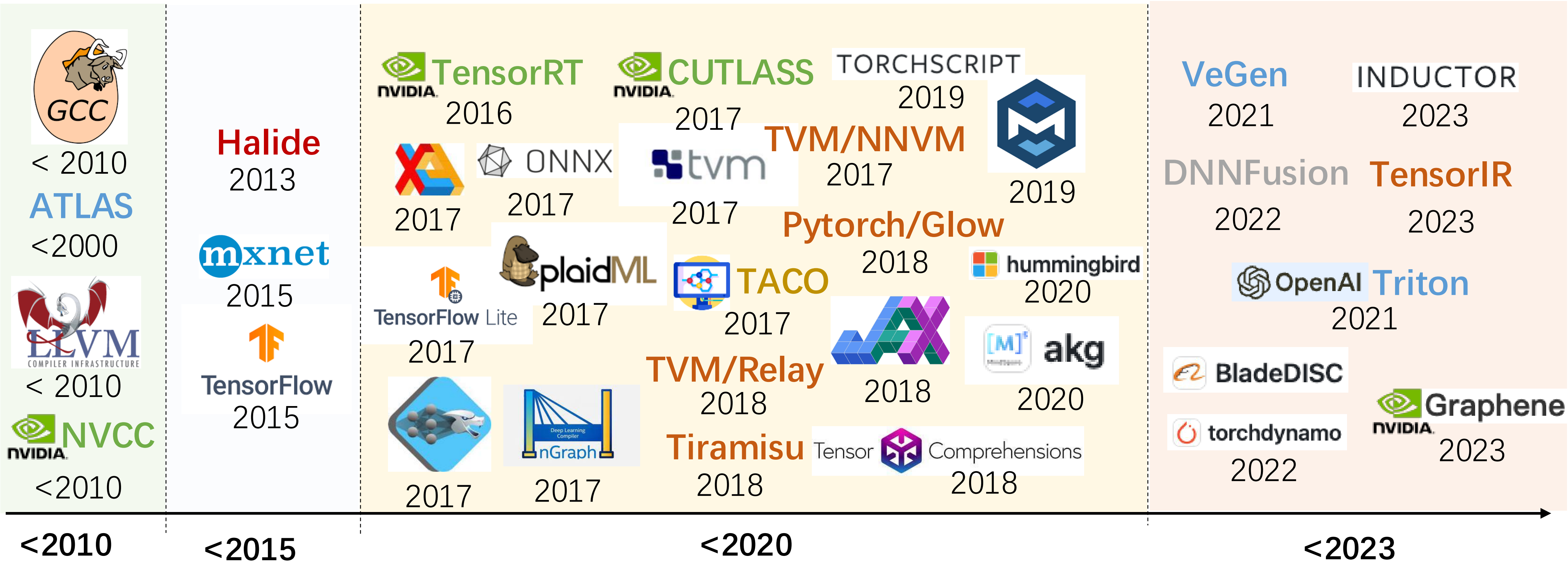
3 Future Work

- Triton-CuTe
- LLM for Compiler

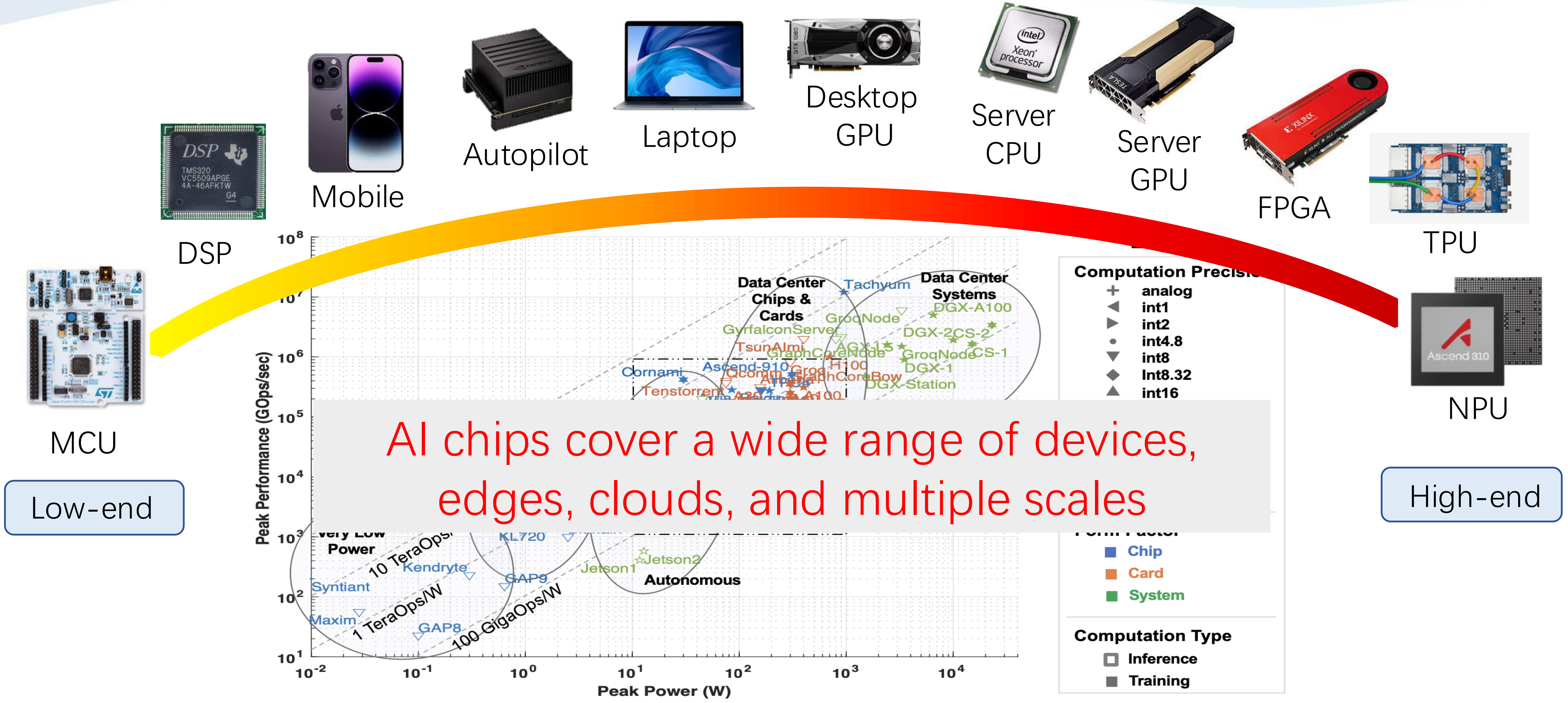
The Golden Age of Compilers

Two Streams of Techniques

- **expert-defined optimizations**
  - **pattern-matching passes**
  - **polyhedral model**
- **user-defined optimizations:** **compute-schedule decomposition**



AI Chips



AI chips cover a wide range of devices, edges, clouds, and multiple scales

Ref： MIT. Albert Reuther, et al. AI and ML Accelerator Survey and Trends.

Previous Projects

DNN Graph

- Model DSL
- Type System
- Autograd
- Convertor

- Graph IR
- Fusion
- Substitution
- Distribution

Operator

- Loop IR
- Tiling
- Polyhedral
- Transform

- Instruction
- Tensorize
- Vectorize
- Selection

NeoFlow framework (TPDS 2022 paper)

Chimera (HPCA 2023 paper)

COMB (DAC 2023 paper)

FlexTensor (ASPLOS 2020 paper)

AHS (MICRO 2021 tutorial)

SuSy (ICCAD 2020 paper)

AMOS (ISCA 2022 paper)

HASCO (ISCA 2021 paper)

ARES (ICCAD 2023 paper)

TileFlow & Domino (MICRO 2023 paper, MLSys 2024 paper)

Rubick (DAC 2023)

Outline

1 Background

- AI Chip
- AI Algorithm
- AI Compiler

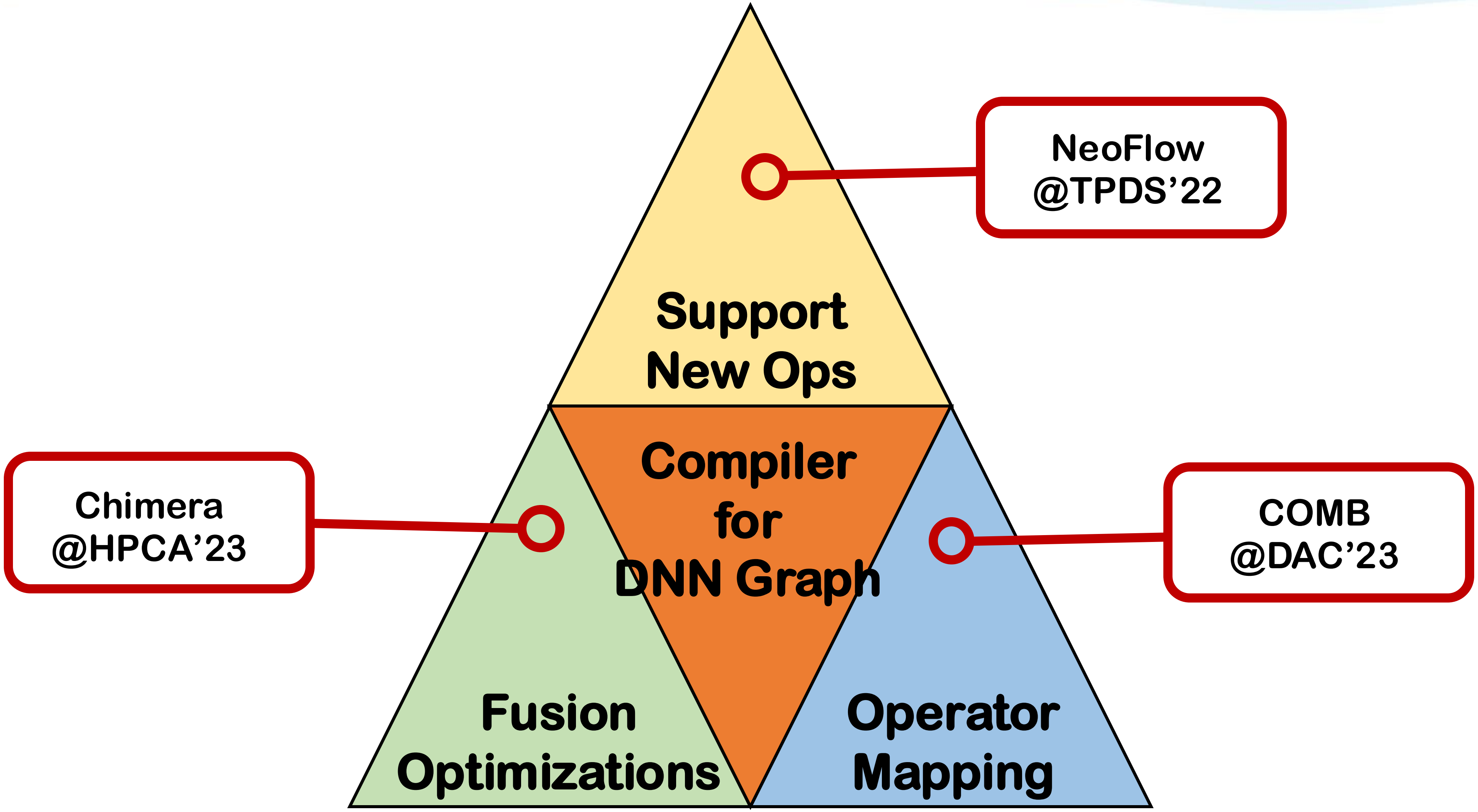
2 Techniques

- Compiler for DNN Graph
- Compiler for Operator
- Compiler for Distributed

3 Future Work

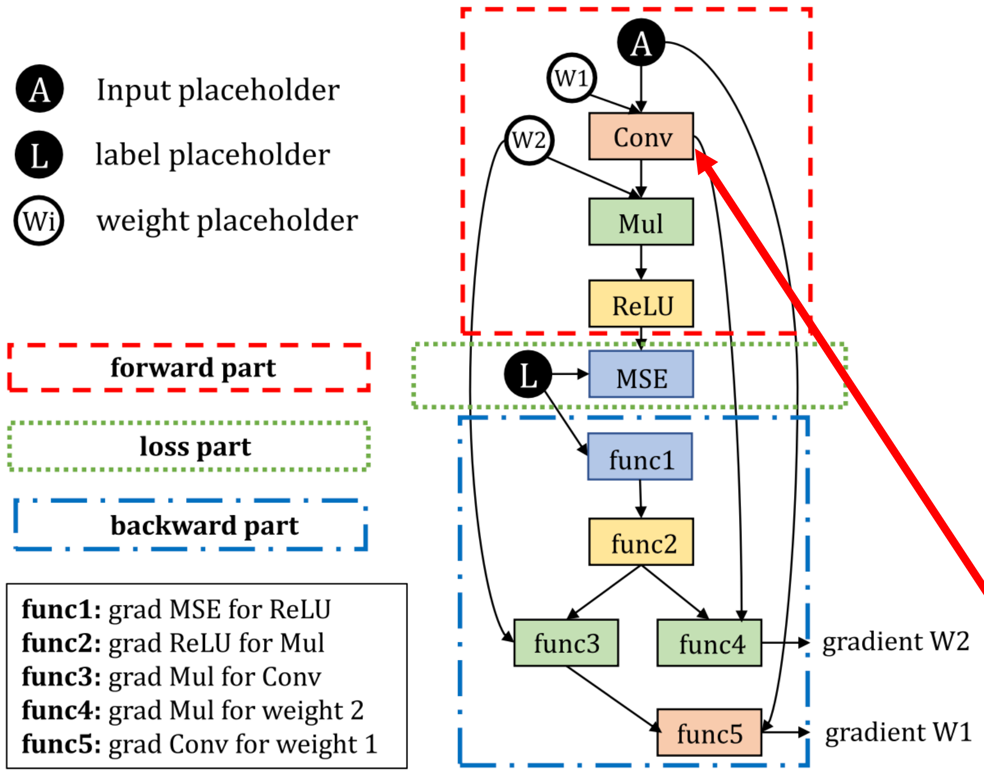
- Triton-CuTe
- LLM for Compiler

Compiler for DNN Graph



New Operator Support Challenge

1. **Kernel Implementation for both forward and backward**
2. **Generalized fusion optimization with other operators**



Shape	Batch	In_C	Out_C	Height	Width	Capsules
	1	64	256	28	28	8

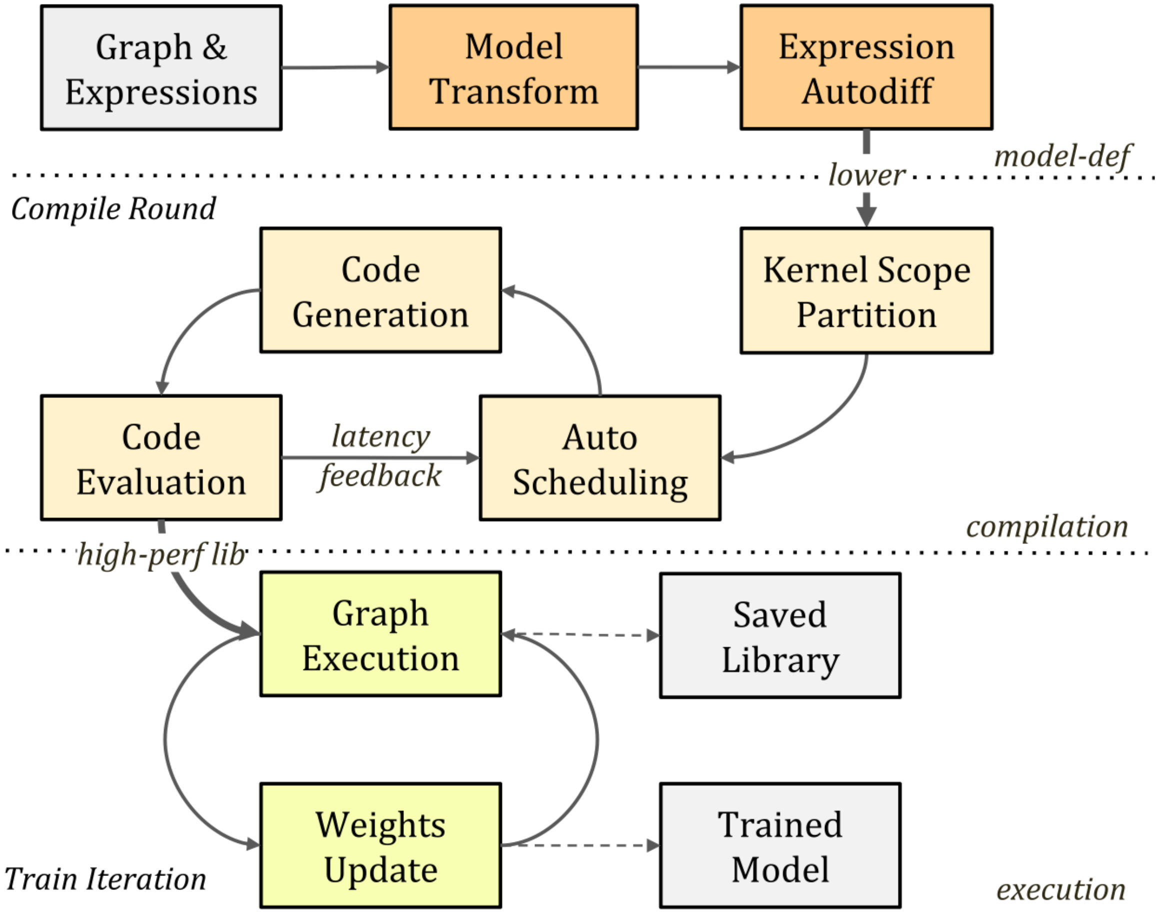
	PyTorch	TensorFlow	NeoFlow
Latency	1.529 ms	4.192 ms	0.451 ms
Launch Overhead	0.473 ms	0.717 ms	0.007 ms
Kernel Overhead	0.899 ms	2.532 ms	0.390 ms
Utilization	65.5%	77.9%	98.2%

express new operators with existing operators can be inefficient

e.g., add new op: capsule conv

$$C[b,k,p,q,i,j] = A[b,c,p*2+r,q*2+s,i,k]*B[k,c,r,s,k,j],$$

NeoFlow Framework



Overview of NeoFlow

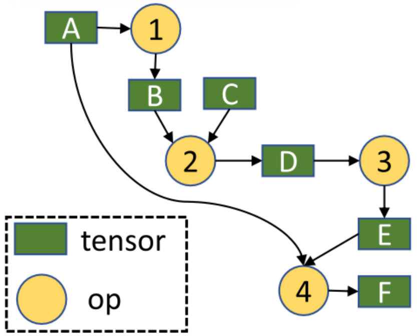
Two Techniques:

1. **Expression-based Autodiff**
2. **Generalized Fusion**

Tensor Declaration

A = tensor([1, 3, 224, 224])
B = tensor([1, 3, 228, 228])
C = tensor([64, 3, 3, 3])
D = tensor([1, 64, 112, 112])
E = tensor([1, 16, 224, 224])
F = tensor([1, 19, 224, 224])

Graph Structure



Op1: Padding

B[n, c, h, w] = Select(
h>2 && h<226 && w>2 && w<226,
A[n, c, h-2, w-2], 0)

Op2: Dilation Conv

D[n, k, p, q] = ReduceAdd({r, s},
B[n, c, p*2+r*2, q*2+s*2]
* **C**[k, c, r, s])

Op3: Depth2Space

E[n, c, h, w] = **D**[n, c*4+h%2*2+w%2, h//2, w//2]

Op4: Concatenation

F[n, c, h, w] = Select(c<3,
A[n, c, h, w], **E**[n, c-3, h, w])

Tensor Expression

Expression-based Autodiff

Insight: **Autodiff for an expression is to get the reversed mapping of index**

$$B[x_1, \ldots, x_N] = \mathbf{F}_{R=\{r_1,\ldots,r_L\}}$$
$$(A_1[f_1^1(x_1, \ldots, x_N, r_1, \ldots, r_L), \ldots, f_{M_1}^1(x_1, \ldots, x_N, r_1, \ldots, r_L)],$$
$$A_2[f_1^2(x_1, \ldots, x_N, r_1, \ldots, r_L), \ldots, f_{M_2}^2(x_1, \ldots, x_N, r_1, \ldots, r_L)],$$
$$\ldots,$$
$$A_K[f_1^K(x_1, \ldots, x_N, r_1, \ldots, r_L), \ldots, f_{M_K}^K(x_1, \ldots, x_N, r_1, \ldots, r_L)])$$

$$dA_i[z_1^i, \ldots, z_{M_i}^i] = \mathbf{H}_{R'=\{r'_1,\ldots,r'_P\}}$$
$$(dB[g_1(z_1^i, \ldots, z_{M_i}^i, r'_1, \ldots, r'_P), \ldots, g_N(z_1^i, \ldots, z_{M_i}^i, r'_1, \ldots, r'_P)],$$
$$A_1[h_1^1(z_1^i, \ldots, z_{M_i}^i, r'_1, \ldots, r'_P), \ldots, h_{M_1}^1(z_1^i, \ldots, z_{M_i}^i, r'_1, \ldots, r'_P)],$$
$$\ldots,$$
$$A_K[h_1^K(z_1^i, \ldots, z_{M_i}^i, r'_1, \ldots, r'_P), \ldots, h_{M_K}^K(z_1^i, \ldots, z_{M_i}^i, r'_1, \ldots, r'_P)]),$$

Reverse mapping of:

1. **computation operation F (easy)**
2. **index mapping f (hard)**

Solution for Affine Transformations

Insight: **For affine index transformation, the problem is reduced to solving a linear (or affine) system problem**

$$f_1^i(x_1, ..., x_N, r_1, ..., r_L) = z_1,$$

$$\dots,$$

$$f_{M_i}^i(x_1, ..., x_N, r_1, ..., r_L) = z_{M_i},$$

linear (or affine) system
x are unknowns, z are constants

for quasi-affine cases:

1. **find or create quasi-affine sub-expression pairs**
2. **substitute quasi-affine sub-expressions with new variables**

Running Example

$$Out[b,k,p,q] \mathrel{+}= In[b,c,p*2+r,q/2+s]*Weight[k,c,r,s]$$

$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} b \\ k \\ p \\ q^* \\ c \\ r \\ s \end{bmatrix}$$

solve the linear system

$$\begin{bmatrix} b \\ k \\ p^* \\ q^* \\ c \\ r \\ s \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \\ f_1 \\ f_2 \\ f_3 \end{bmatrix}$$

$q^* = q/2$

$f_4 = q \bmod 2$

$q = q^* * 2 + f_4$

$p^* = p * 2$

$p = p^*/2$

f_1, f_2, f_3 *are free variables for reduction*

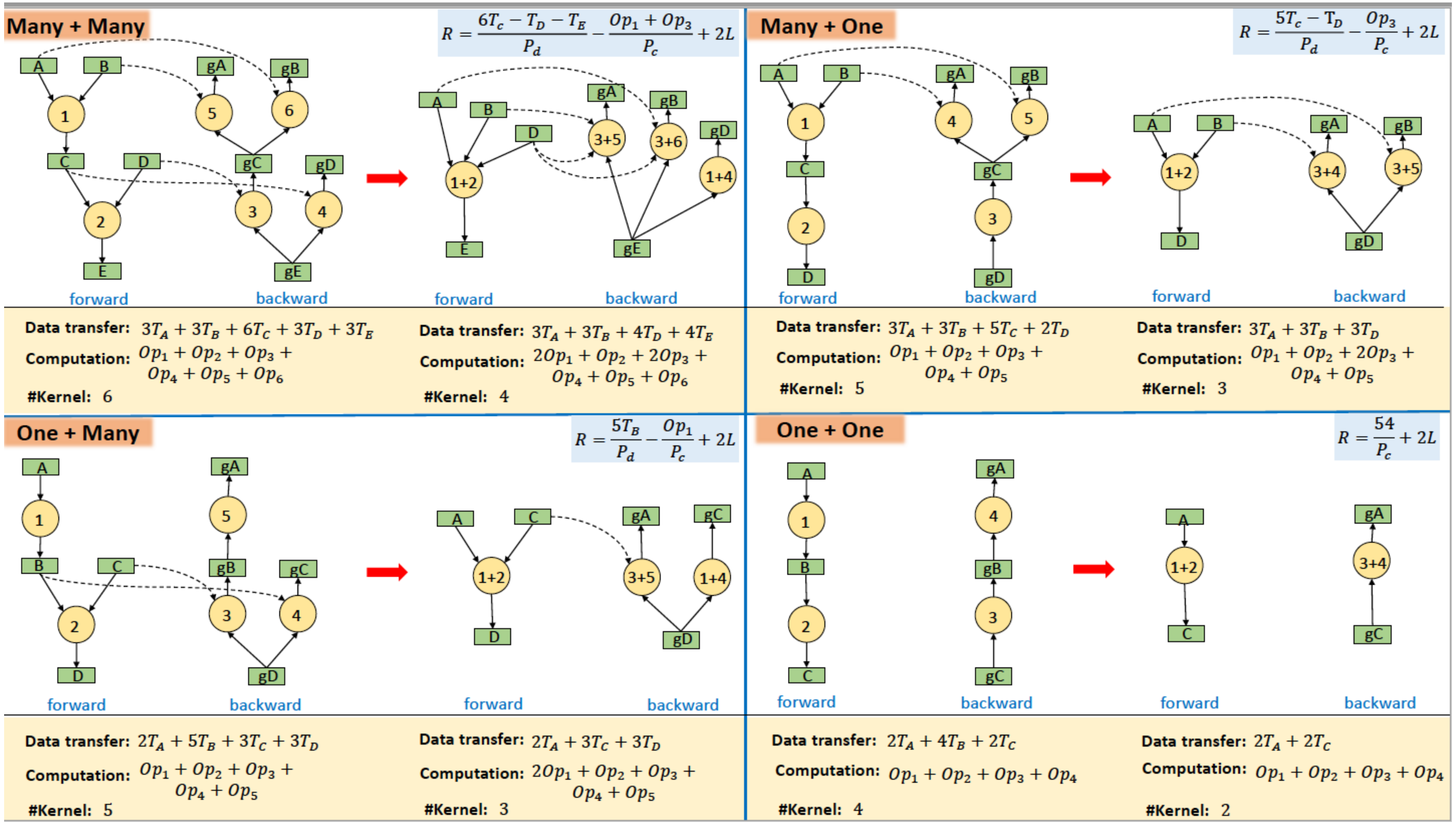
construct expressions according to the inverse

$$dIn[z_1,z_2,z_3,z_4] \mathrel{+}= dOut[z_1,f_1,(z_3-f_2)/2,(z_4-f_3)*2+f_4]*Weight[f_1,z_2,f_2,f_3]$$

Generalized Fusion

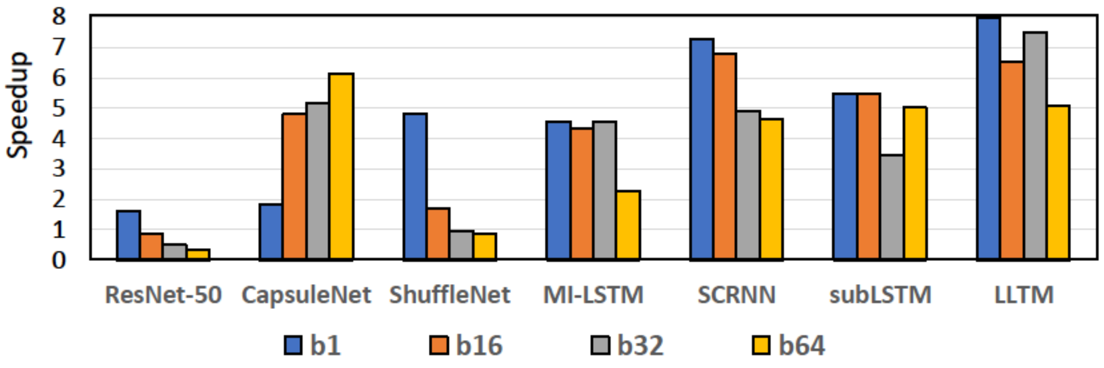
Insight: Co-optimize both forward and backward graph

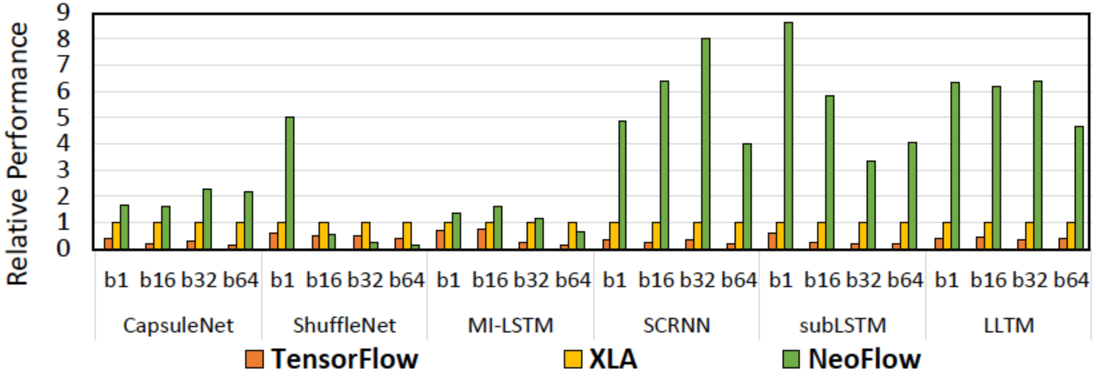
1. Four Patterns
2. Cost Model
3. Coupled effect



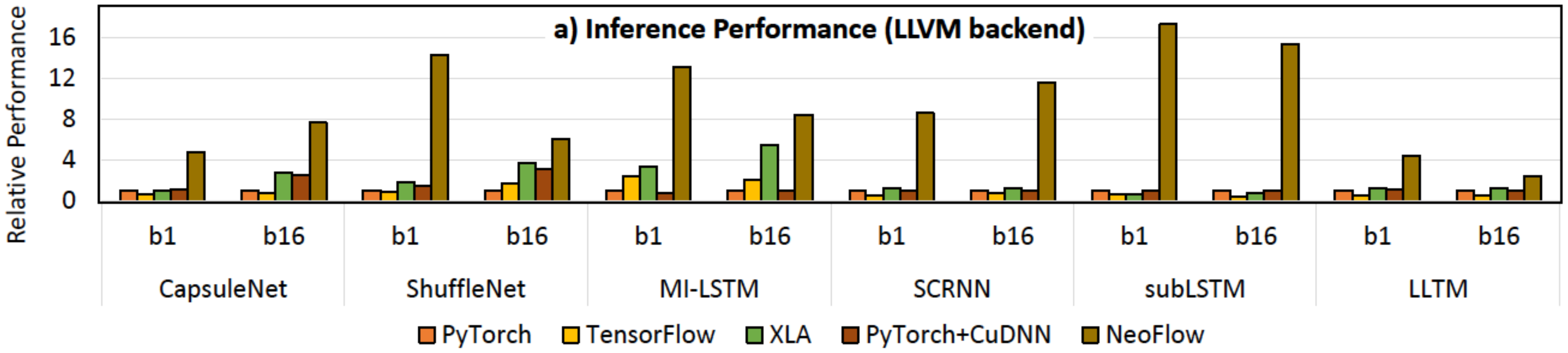
Performance

Evaluate some special networks with customized operators





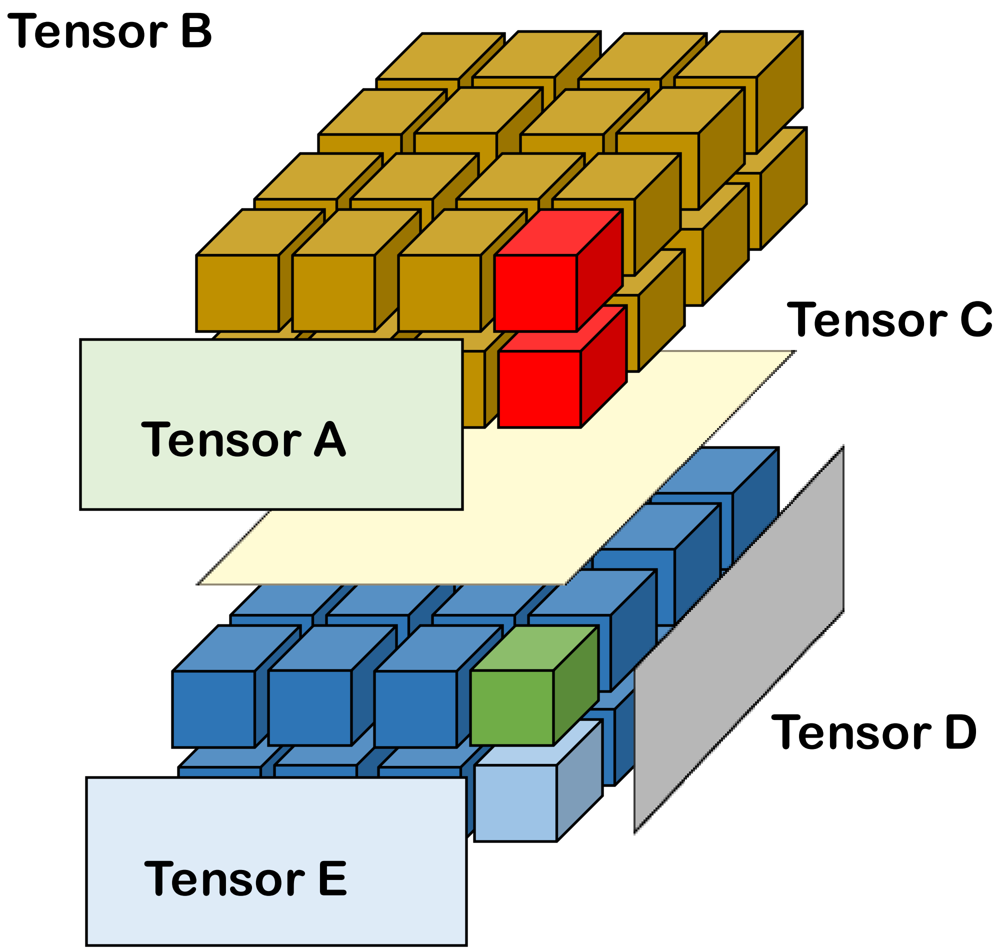
Training: 1.92x to CuDNN and 2.43x to XLA



Inference: 6.72x to CuDNN and 4.96x to XLA

Aggressive Fusion Challenges

Compute-intensive operators chains are hard to fuse



Shapes:

Tensor A: [M, K]

Tensor B: [K, L]

Tensor C: [M, L]

Tensor D: [L, N]

Tensor E: [M, N]

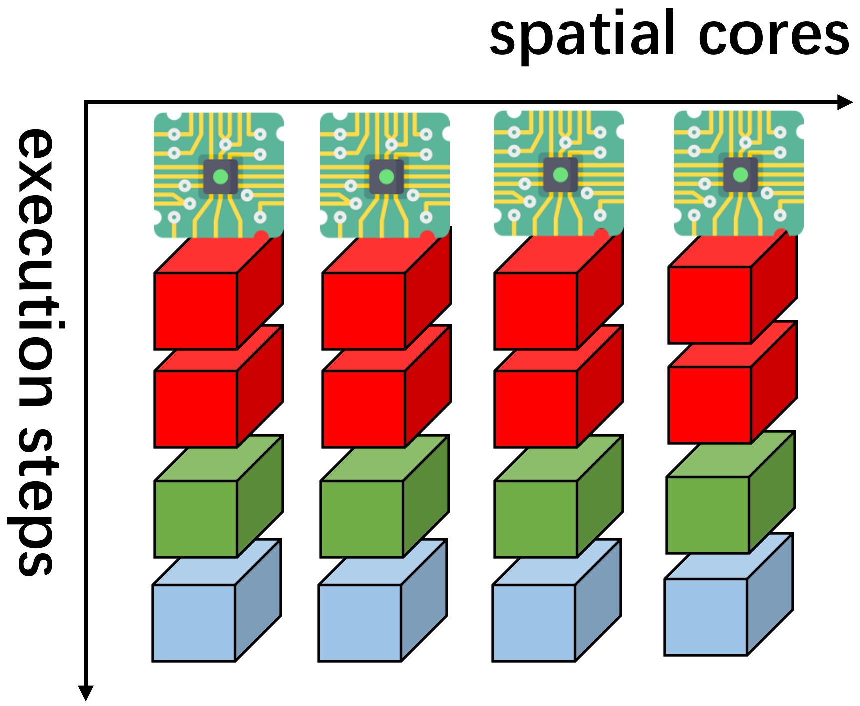
Decompose:

M→M/Tm, Tm

N→N/Tn, Tn

K→K/Tk, Tk

L→L/Tl, Tl



Example execution order: mlnk

Iterate along k-dim first, then n-dim, then l-dim, finally, m-dim

Chimera: Analysis Technique

Three insights:

1. Loop variables that are absent in tensor access won't cause data movement

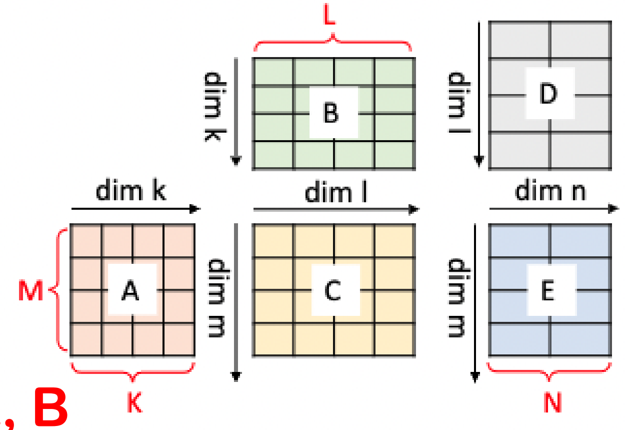
2. When inner loops cause data movement, outer loops will also cause data movement

3. Loops that are private to producer operators won't cause data movement for consumer operators

Running Examples

Insight 1: Loop variables that are absent in tensor access won't cause data movement

order: m, k, l, n

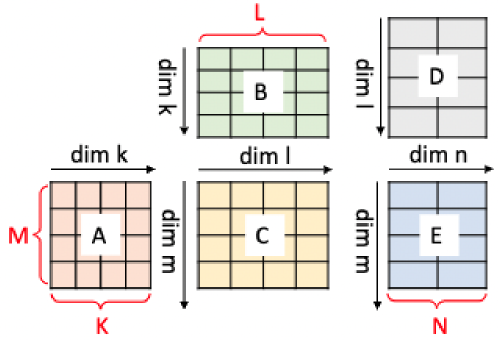



```
for m in range(0,M,Tm):        ← reuse B, D, replace A, C, E
 for k in range(0,K,Tk):       ← reuse C, D, E, replace A, B
  for l in range(0,L,Tl):      ← reuse A, D, E, replace B, C
   C[m:m+Tm,l:l+Tl]+=A[m:m+Tm,k:k+Tk]@B[k:k+Tk,l:l+Tl]
 for l in range(0,L,Tl):       ← reuse A, B, E, replace C, D
  for n in range(0,N,Tn):      ← reuse A, B, C replace D, E
   E[m:m+Tm,n:n+Tn]+=C[m:m+Tm,l:l+Tl]@D[l:l+Tl,n:n+Tn]
```

Running Examples

Insight 2: When inner loops cause data movement, outer loops will also cause data movement

order: m, k, l, n




```
for m in range(0,M,Tm):← replace A, B, C, D, E
  for k in range(0,K,Tk): ← reuse D, E, replace A, B, C
    for l in range(0,L,Tl): ← reuse A, D, E, replace B, C
      C[m:m+Tm,l:l+Tl]+=A[m:m+Tm,k:k+Tk]@B[k:k+Tk,l:l+Tl]
  for l in range(0,L,Tl):← reuse A, B, replace C, D, E
    for n in range(0,N,Tn): ← reuse A, B, C replace D, E
      E[m:m+Tm,n:n+Tn]+=C[m:m+Tm,l:l+Tl]@D[l:l+Tl,n:n+Tn]
```

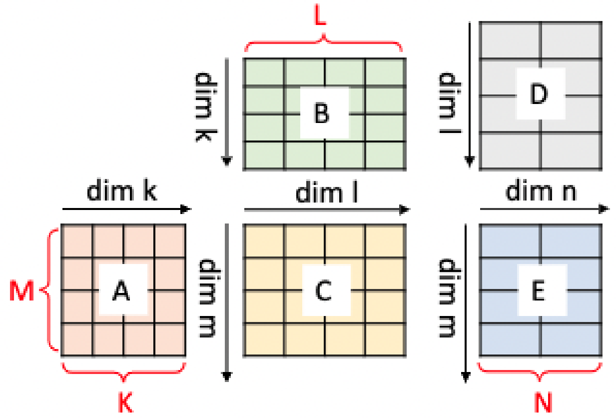
Running Examples

Insight 3: Loops that are private to producer operators won't cause data movement for consumer operators

order: m, k, l, n

```
for m in range(0,M,Tm):
 for k in range(0,K,Tk):
  for l in range(0,L,Tl):
   C[m:m+Tm,l:l+Tl]+=A[m:m+Tm,k:k+Tk]@B[k:k+Tk,l:l+Tl]
 for l in range(0,L,Tl):
  for n in range(0,N,Tn):
   E[m:m+Tm,n:n+Tn]+=C[m:m+Tm,l:l+Tl]@D[l:l+Tl,n:n+Tn]
```

Private loops, have no influence on the consumer operator



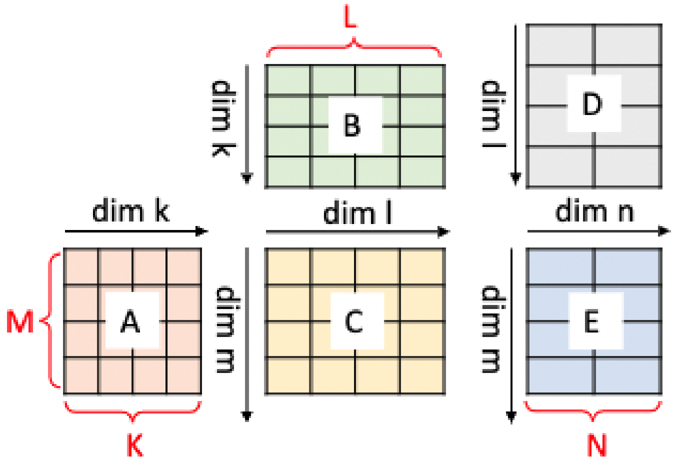
Minimize Data Movement Volume

Use Lagrange Multiplier method:

Use GEMM chain as an example (mlkn)

	A	B	C	D	E
DM	$MK\lceil \frac{L}{T_L} \rceil$	$KL\lceil \frac{M}{T_M} \rceil$	0	$NL\lceil \frac{M}{T_M} \rceil$	$MN\lceil \frac{L}{T_L} \rceil$
DF	$T_M T_K$	$T_K T_L$	$T_M T_L$	$T_L T_N$	$T_M T_N$

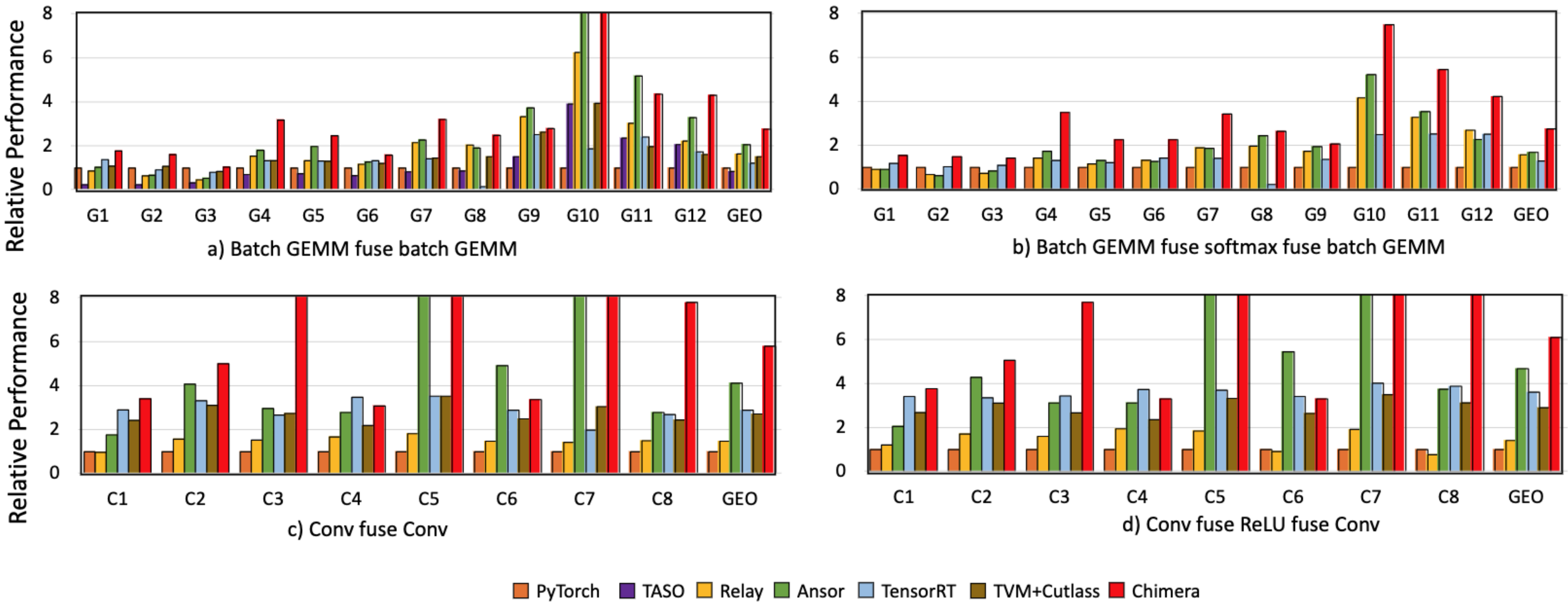
$$\mathrm{DV}_{\text{GEMM Chain}} = DM_A + DM_B + DM_C + DM_D + DM_E$$
$$= MK\lceil \frac{L}{T_L} \rceil + KL\lceil \frac{M}{T_M} \rceil + NL\lceil \frac{M}{T_M} \rceil + MN\lceil \frac{L}{T_L} \rceil$$



Constraints:

Total memory footprint should not exceed memory capacity

Performance



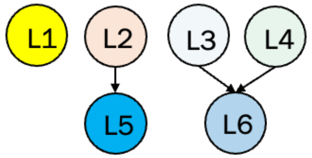
GEMM + GEMM, Conv + Conv, 2.77x to PyTorch on CPU and 5.79x to PyTorch on GPU

Operator Mapping Challenges

Design space formalization and exploration

- Hardware Resource Spatial Sharing

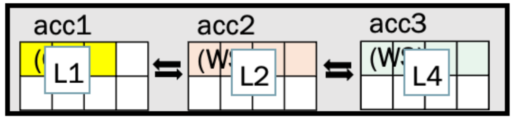
  Map more layers at the same time to hardware



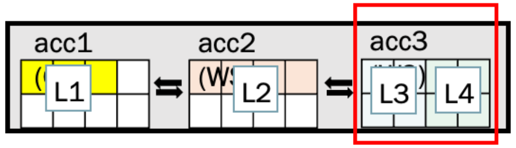
a) Sub-graph to schedule

Layer	L1	L2	L3	L4	L5	L6
Dataflow	OS	WS	OS	WS	WS	WS
Resource (unit)	3	4	3	4	4	3

b) Layer-wise optimal dataflow and resource usage



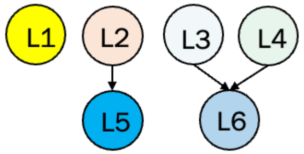
c) Dataflow-optimal mapping



d) Spatial sharing mapping (not dataflow optimal)

- Routing Distance in Mapping

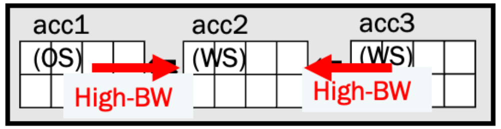
  The bandwidths between different accelerators are not the same

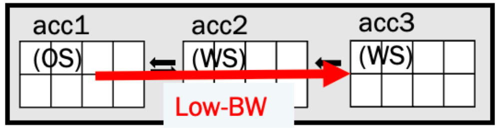


a) Sub-graph to schedule

Layer	L1	L2	L3	L4	L5	L6
Dataflow	OS	WS	OS	WS	WS	WS
Resource (unit)	3	4	3	4	4	3

b) Layer-wise optimal dataflow and resource usage



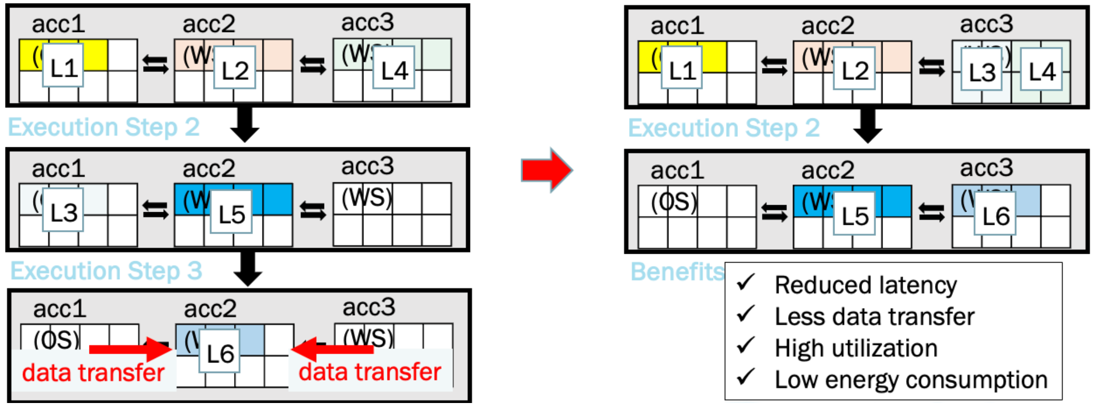


How to achieve better mapping?

1. **generate the design space**
2. **explore the design space**

- Computation and Memory Coordinated Mapping

  Consider both resource sharing and routing bandwidth



COMB: Space Design and DSE

Heterogeneous DNN and Heterogeneous SoC

DNN Graph

$$G = (V, E)$$

Multi-DNN Graph

$$\mathcal{G} = (G_1, G_2, \ldots, G_M)$$

Hetero. SoC

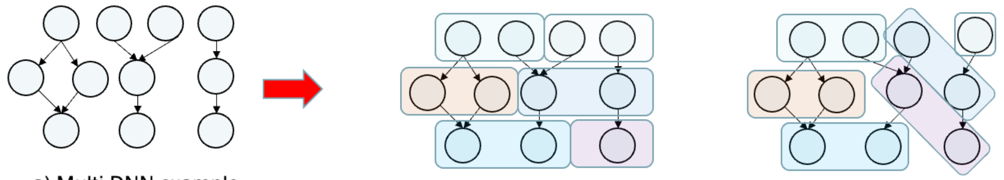
$$H = (A, Net)$$

$$A = \{\text{acc}_1, \text{acc}_2, \ldots, \text{acc}_N\}$$

$$Net = \{(\text{acc}_i, \text{acc}_j, cost) | 1 \le i, j \le N\}$$

Properties of DNN and SoC

Name	Explanation
DNN Related Methods	
Pred(L)	Get the predecessor layers of layer $L \in V$
DV(L)	Data transfer volume for outputs of layer L
GroupOf(L)	Get the dataflow group of layer L
SoC Related Methods	
NumPE(acc)	Get the number of PEs of accelerator acc
MemCap(acc)	Get the scratchpad capacity of accelerator acc
Dataflow(acc)	Get the dataflow of acceleraotr acc
Comm(acc_1, acc_2, V)	The cost of transferring data of volume V from acc_1 to acc_2 according to Net



a) Multi-DNN example (a local part)

b) Different Dataflow Grouping Choices

The layers in the same group will use the same dataflow

Mapping Multi-DNN Graph to Heterogeneous SoC

Graph Grouping

$$D_1 \preceq D_2 \preceq \cdots \preceq D_K \quad \text{where} \quad D_i = \{L_1^i, L_2^i, \ldots L_{P_i}^i\}$$

$$D_i \cap D_{i'} = \emptyset \quad \forall i \neq i', (D_1 \cup \ldots \cup D_K) = (V_1 \cup \ldots \cup V_M)$$

$$L_j^i \in (V_1 \cup \ldots \cup V_M) \quad 1 \le i \le K, \; 1 \le j \le P_i$$

Group Mapping

$$Map: \{D_1, D_2, \ldots, D_K\} \to A$$

$$Time: \{D_1, D_2, \ldots, D_K\} \to \mathcal{R}$$

The Optimization Problem

$$\min_{D_1 \preceq \ldots \preceq D_K, Map, Time} \max_i \{Time(D_i) + Cost(D_i)\}$$

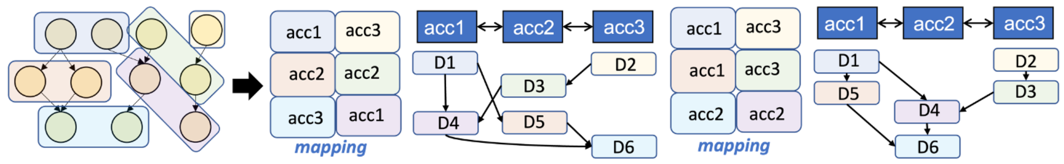
Constraints

$$\sum_j \text{PEUsage}(L_j^i, \text{Dataflow}(Map(D_i))) \le \text{NumPE}(Map(D_i))$$

$$\sum_j \text{MemUsage}(L_j^i, \text{Dataflow}(Map(D_i))) \le \text{MemCap}(Map(D_i))$$

$$Time(D_j) \ge Time(D_i) + Cost(D_i), \quad \forall D_i \preceq D_j \text{ and } D_j \npreceq D_i$$

Different Mapping Choices



Different Accelerator Mapping Choices (left: 5 hops, right: 3 hops)

The layers communicate with each other via:

1) **intra-accelerator communication (on-chip memory)**
2) **inter-accelerator communication (routing)**

DSE Algorithm

- The Steps in Algorithm

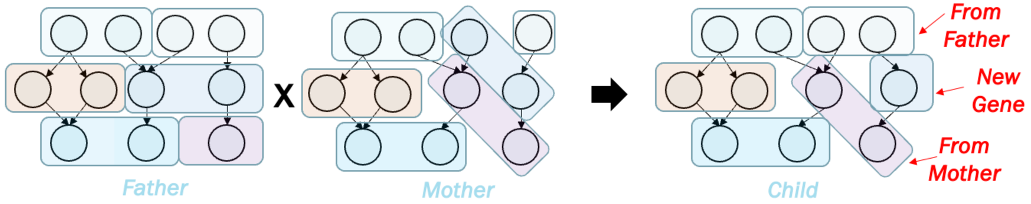
Step 1: Random Grouping — **Get the initial population**

Step 2: Greedy Mapping — **Map the groups to SoC**

Step 3: Evaluation for Fitness — **Check performance, resource, etc.**

Step 4: Crossover & Mutate — **Generate new population**

- Generate New Grouping Choices



Crossover to Generate New Grouping Choices

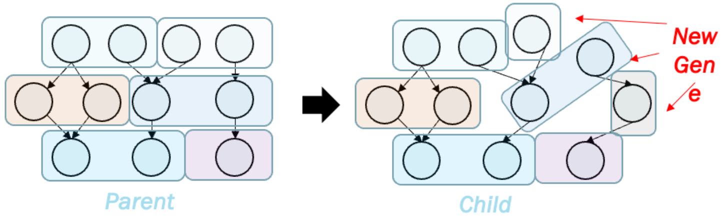
- Algorithm Skeleton: Minimize communication for each group

```
for each group D:
 Map[D] = None; Time[D] = inf;
 for acc in A:
  end_time = 0;
  for each layer L in D:
   start_time = Max(acc.cur_time, end_time_of_preds(L) + transfer_overhead)
   comp_time = ComputeLatency(L, acc)
   end_time = max(end_time, start_time + comp_time)
  if end_time < Time[D]:
   Map[D] = acc; Time[D] = end_time;
```

Get the finalize time of the group

Greedy mapping

Generate New Grouping Choices

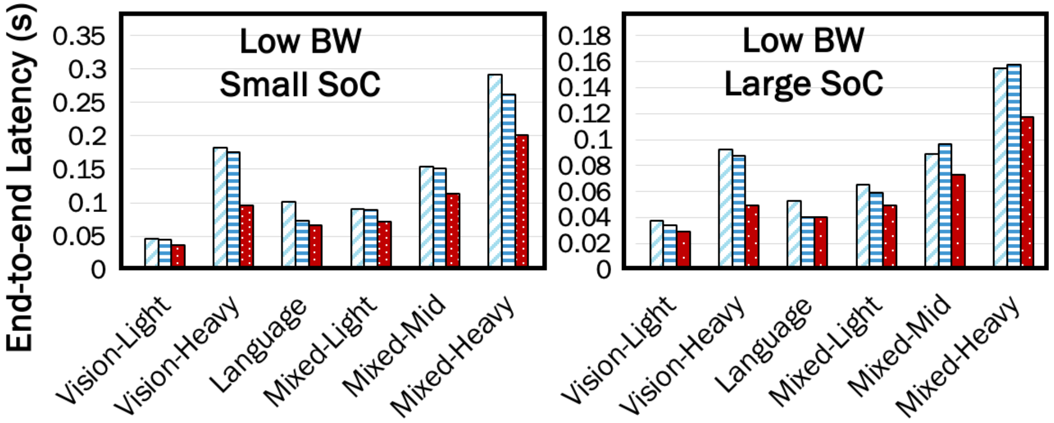


d) Mutate to Generate New Grouping Choices

Performance

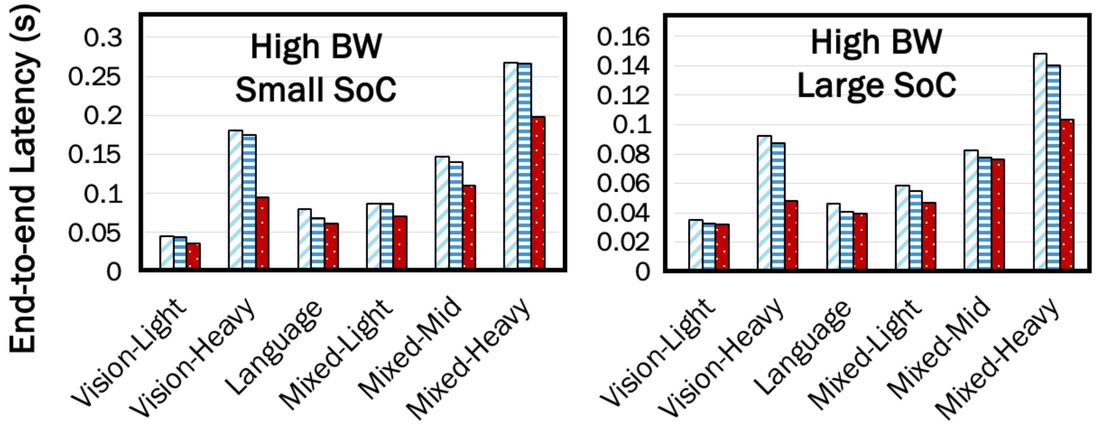
- Latency Results

Speedup to H2H: 1.23X – 1.91X
Speedup to MAGMA: 1.21X – 1.84X



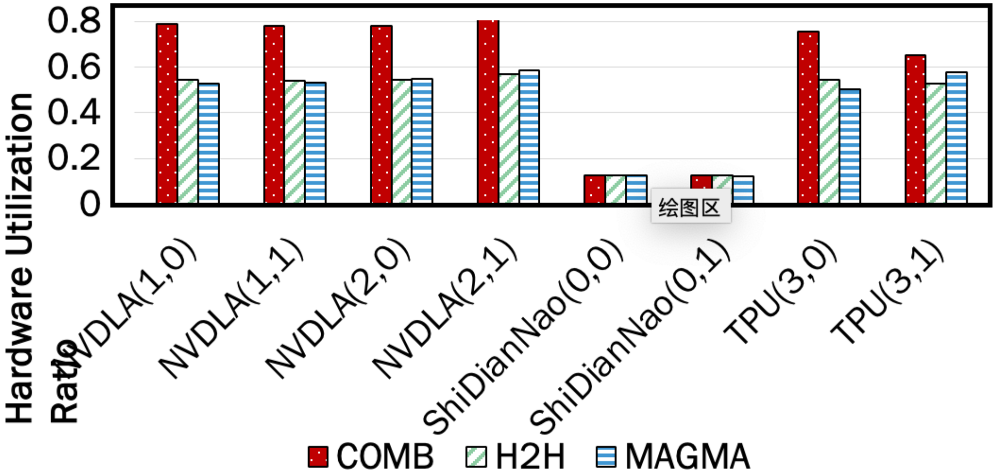
- Latency Results

Geometric Mean Speedup to H2H: 1.38X
Geometric Mean Speedup to MAGMA: 1.28X



- Hardware Utilization

1.28X to H2H and MAGMA



Outline

1 Background

- AI Chip
- AI Algorithm
- AI Compiler

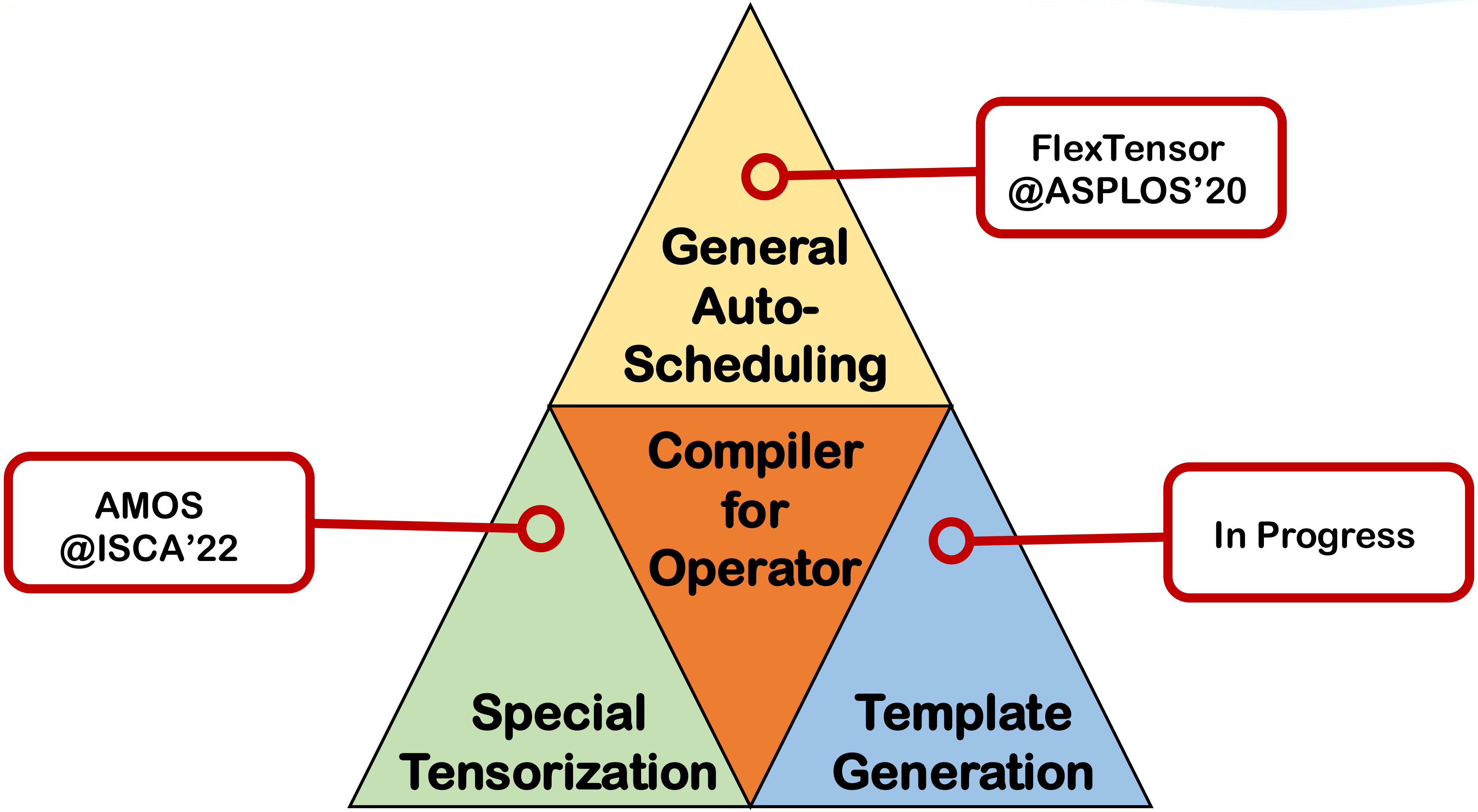
2 Techniques

- Compiler for DNN Graph
- Compiler for Operator
- Compiler for Distributed

3 Future Work

- Triton-CuTe
- LLM for Compiler

Compiler for Operator

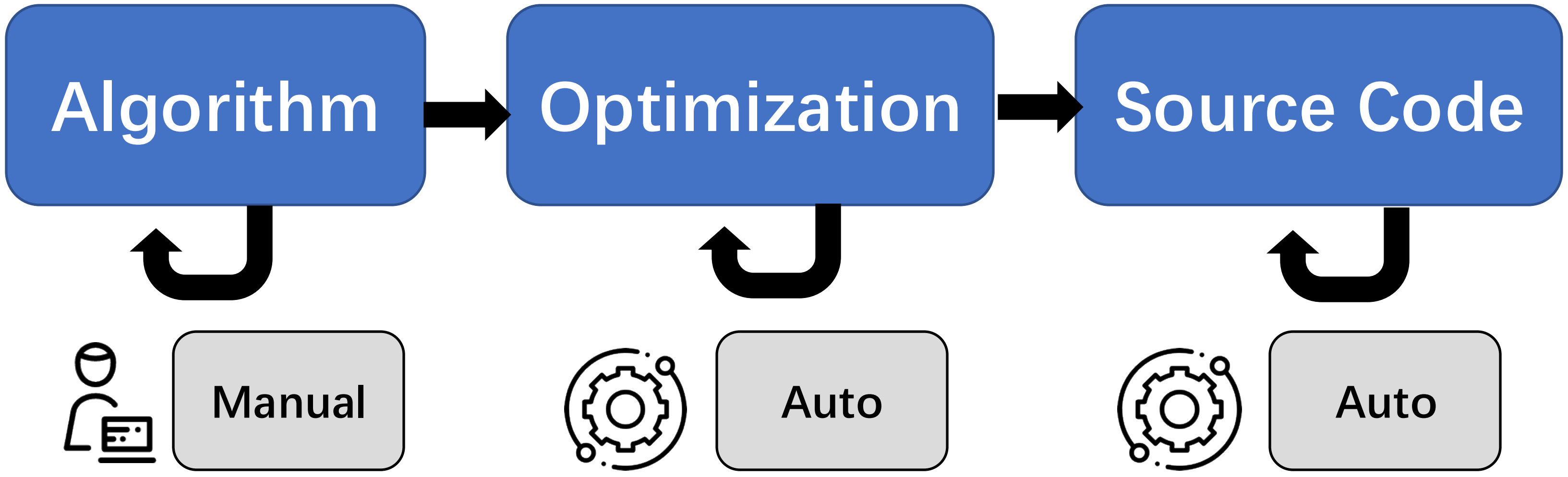


General Auto-Scheduling

Auto-Scheduling: creating passes with composable schedule primitives

Assumptions:

1. Schedule primitives are general enough for hardware
2. It is possible to produce comparable performance using schedule primitives



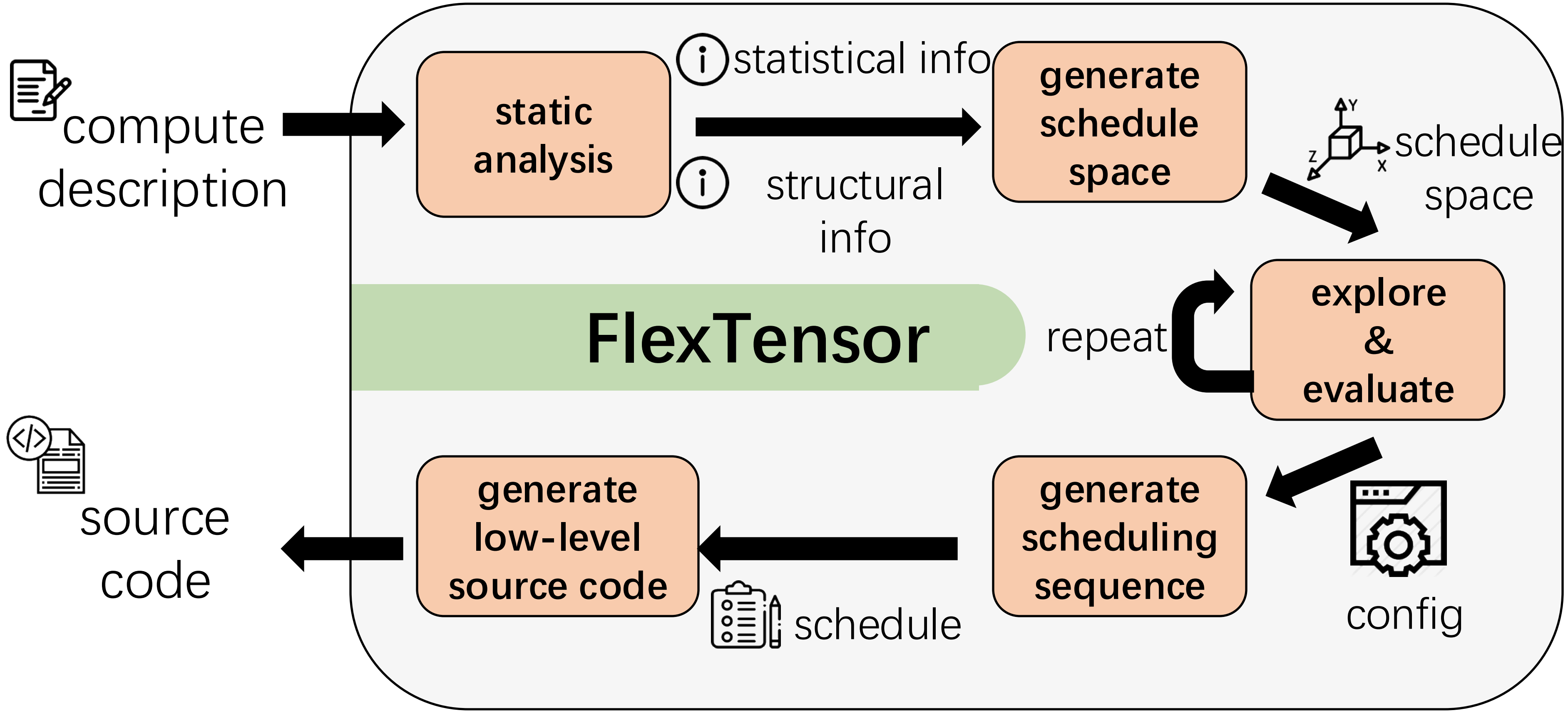
focus on algorithm

hide hardware details from users

only expertise in algorithm

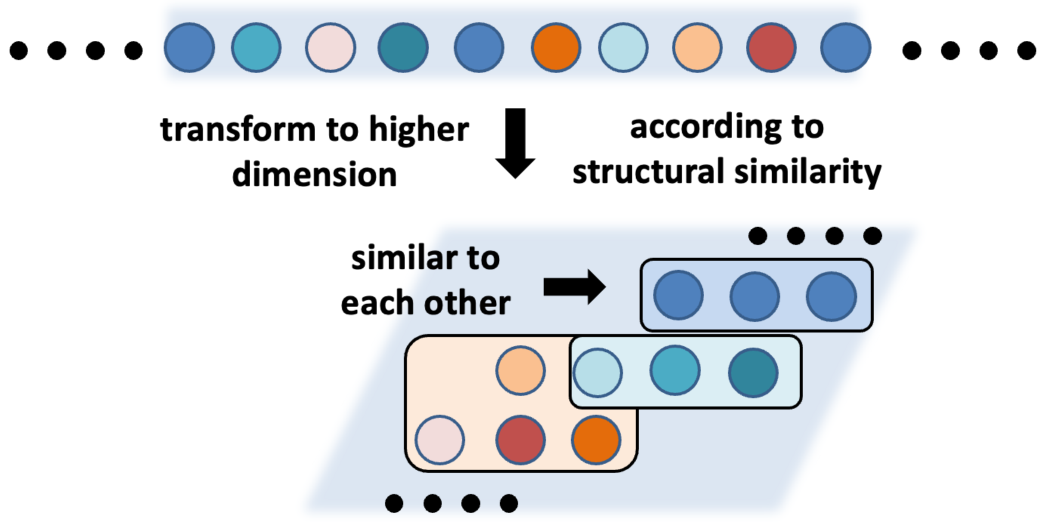
These assumptions are true for pre-Volta NV GPUs and other GPUs that are similar to NV GPUs

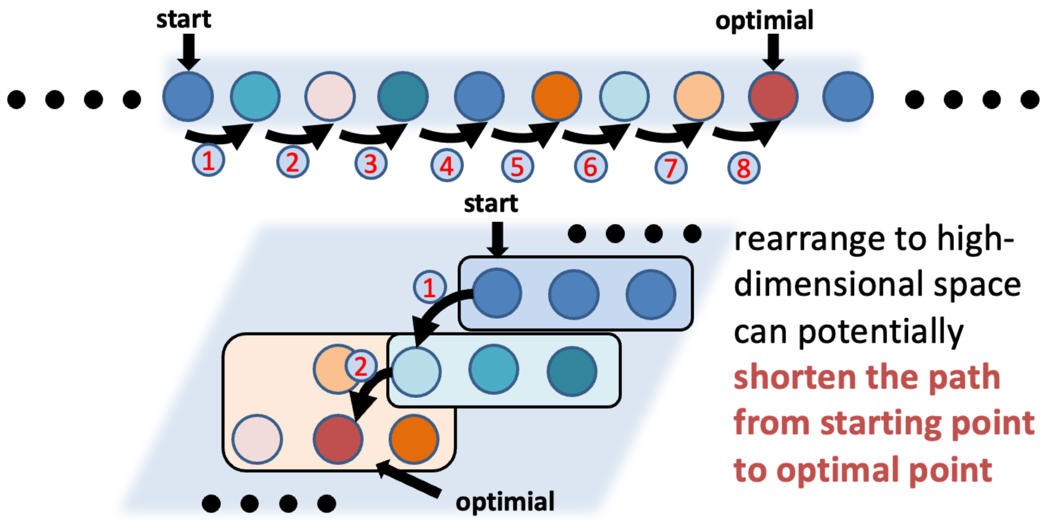
FlexTensor: Space Formalization and DSE



Space Reorganization

Insight: **Most design points are similar, the design space has locality**

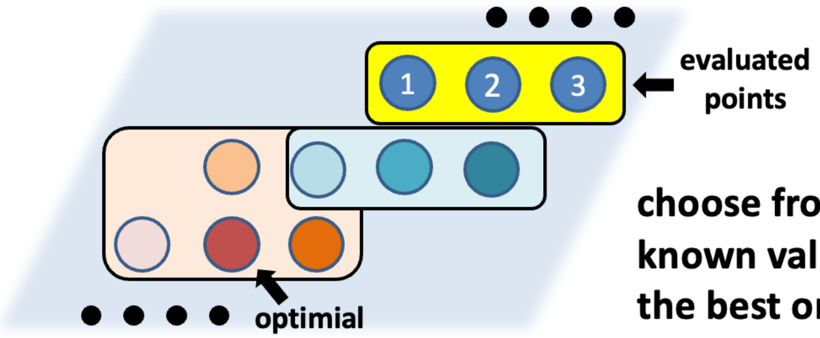




Reorganize the space into high-dimensional space enables efficient DSE

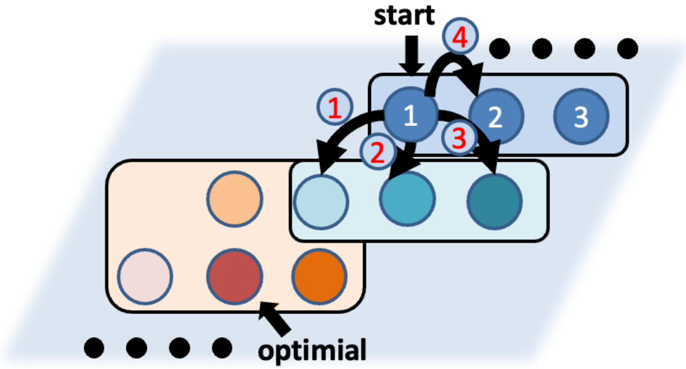
DSE with RL and Heuristic

Use Simulated Annealing to find start points



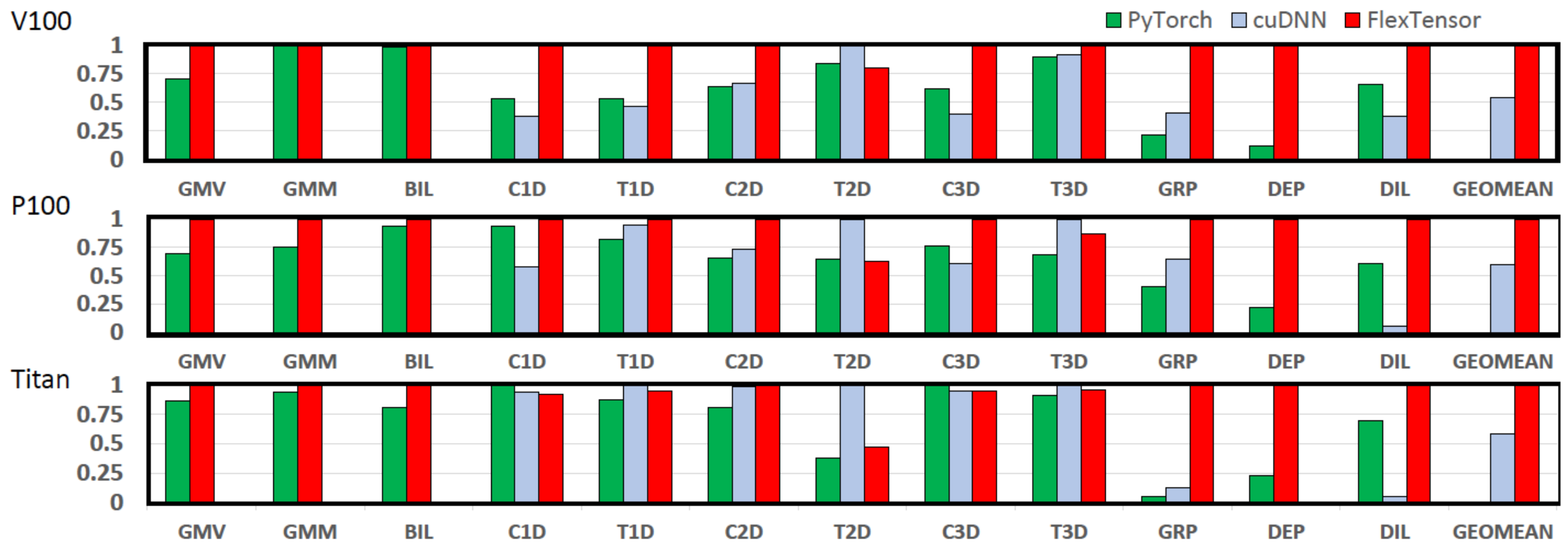
choose from 1, 2, and 3
known value: v^1, v^2, v^3
the best one known: $\boxed{v^*}$
choose according to possibility:
$\boxed{e^{-\gamma\frac{(v^*-v^i)}{v^*}}}, i = 1, 2, 3$
allow choosing multiple points

Use Q-Learning to predict modification direction of current point



1. keep record of visited points: discard 4
2. use DQN algorithm to predict Q-value of each direction: q^1, q^2, q^3
3. choose the largest one: $q^* = max(q^i), i = 1, 2, 3$

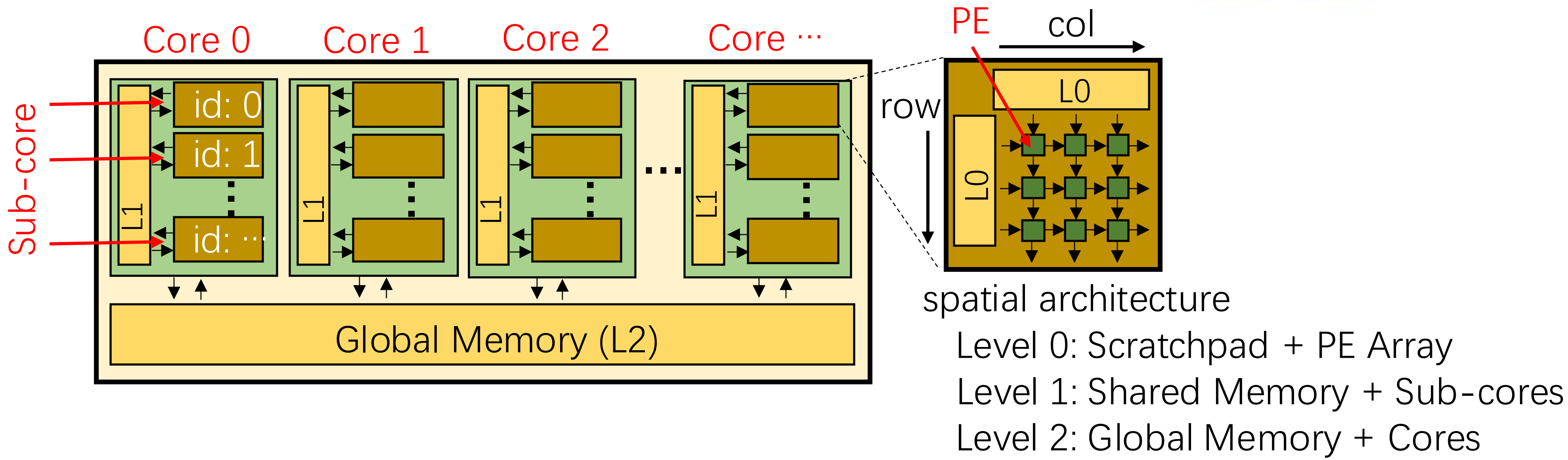
Performance



Tensor Computations	
Operator	**Abbr.**
GEMV	GMV
GEMM	GMM
Bilinear	BIL
1D convolution	C1D
Transposed 1D convolution	T1D
2D convolution	C2D
Transposed 2D convolution	T2D
3D convolution	C3D
Transposed 3D convolution	T3D
Group convolution	GRP
Depthwise convolution	DEP
Dilated convolution	DIL

only use CUDA Cores on GPUs:
P100 1.68x to CuDNN
V100 1.83x to CuDNN
Titan 1.71x to CuDNN

AI Chips are Increasingly Customized



Use dataflow architectures for higher performance and lower energy

Challenge: optimization beyond the scope of general scheduling

```
__m512d _mm512_add_pd (__m512d a, __m512d b)
```

Add two vectors — Left operand — Right operand

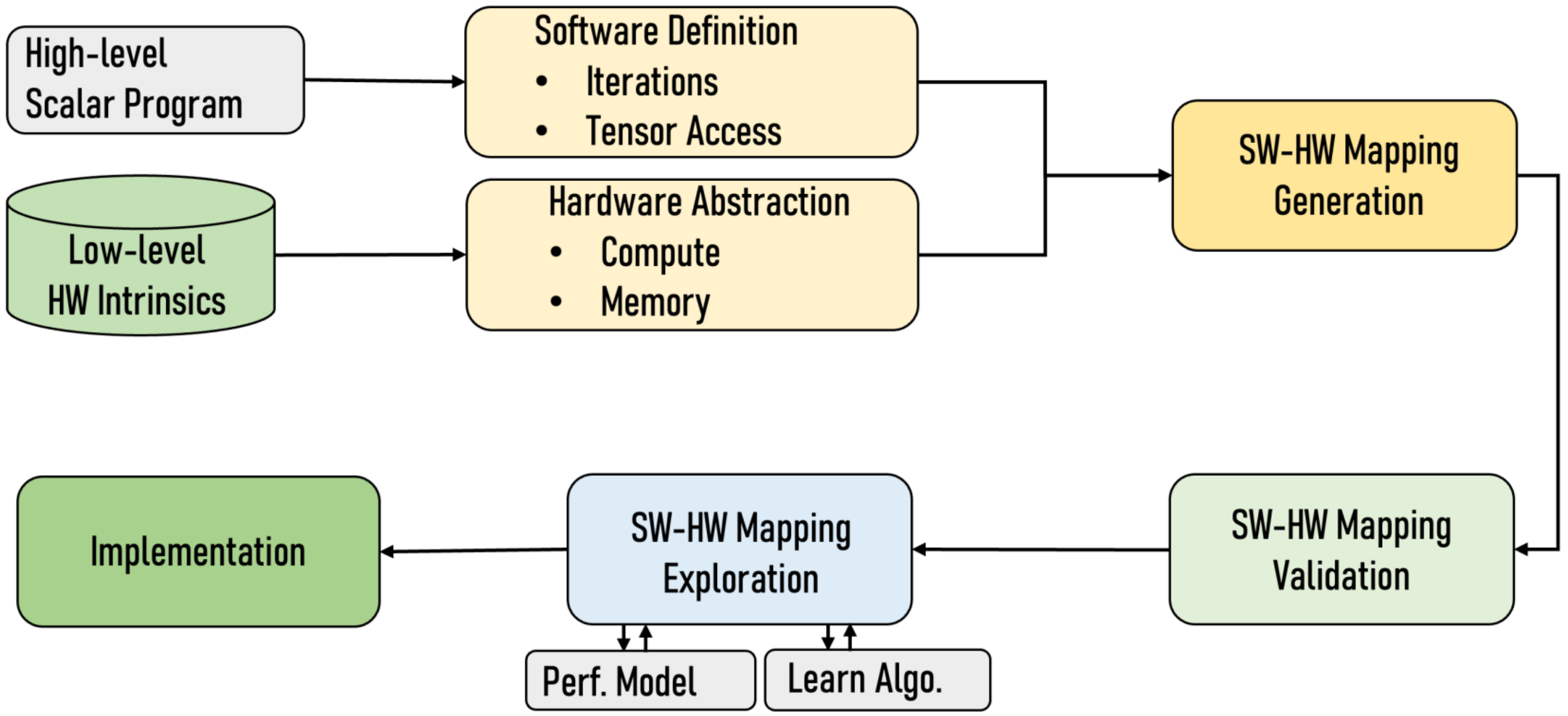
```
wmma::mma_sync(c_frag, a_frag, b_frag, d_frag)
```

Matrix multiplication — Accumulator — Operand A — Operand B — Accumulator

AMOS: Generalize Intrinsic Mapping

Insights:

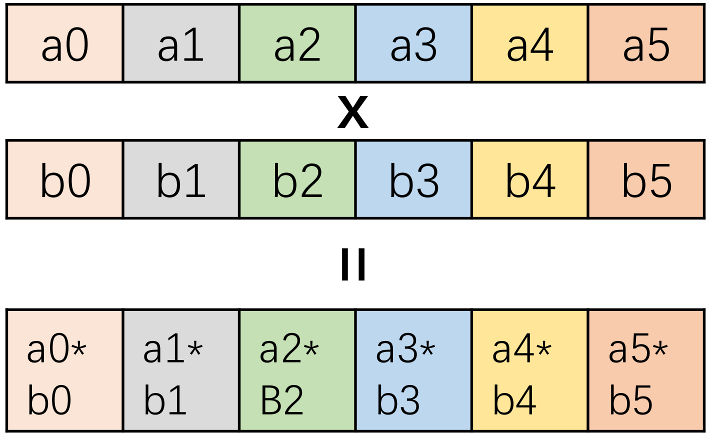
1. **Most intrinsics just represent BLAS semantics**
2. **Operator expression can be factorized into smaller BLAS operations**



Intrinsic Semantics

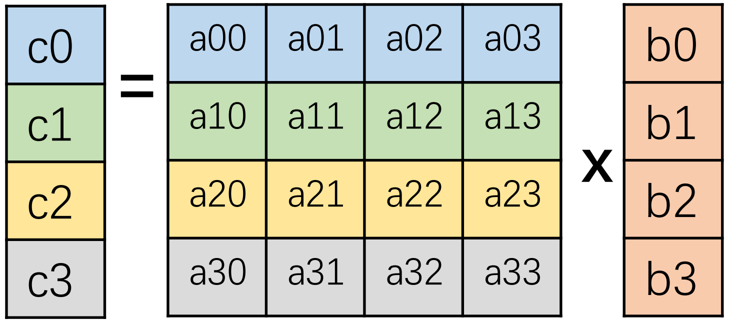
Most intrinsics just represent BLAS semantics

`c[i] = a[i]*b[i]`



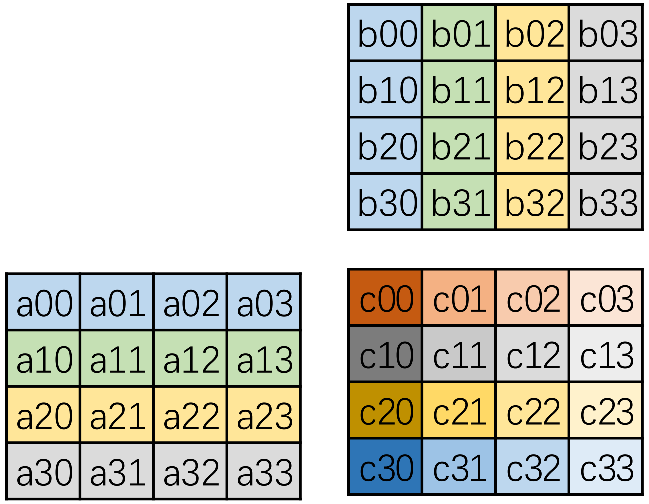
Level 1
Vector Operations

`c[i] = a[i,k]*b[k]`



Level 2
Matrix-Vector Operations

`c[i,j] = a[i,k]*b[k,j]`



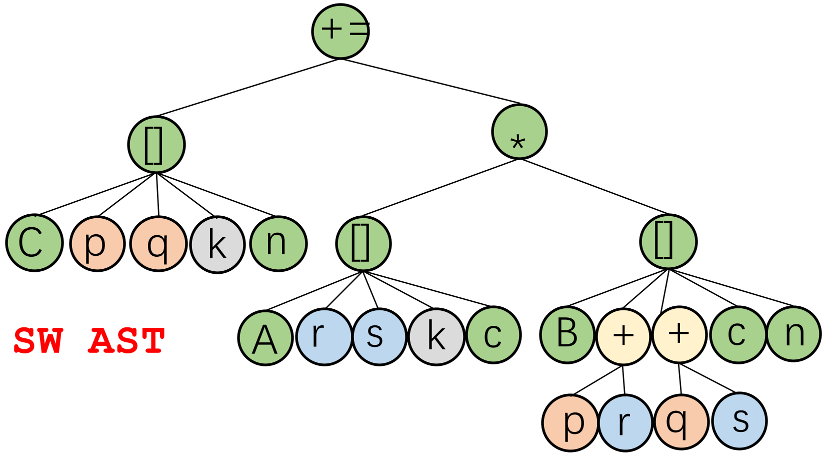
Level 3
Matrix-Matrix Operations

Matching Intrinsic and Expression

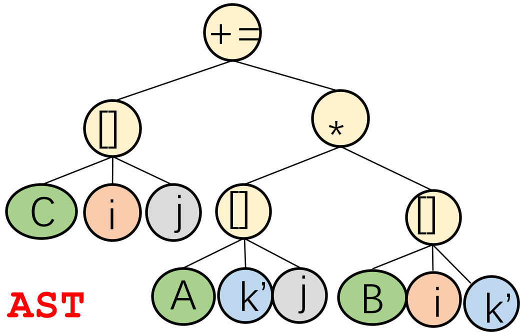
Example 1

n	-
k	j
p	i
q	i
c	-
r	k'
s	k'

Mapping



SW AST

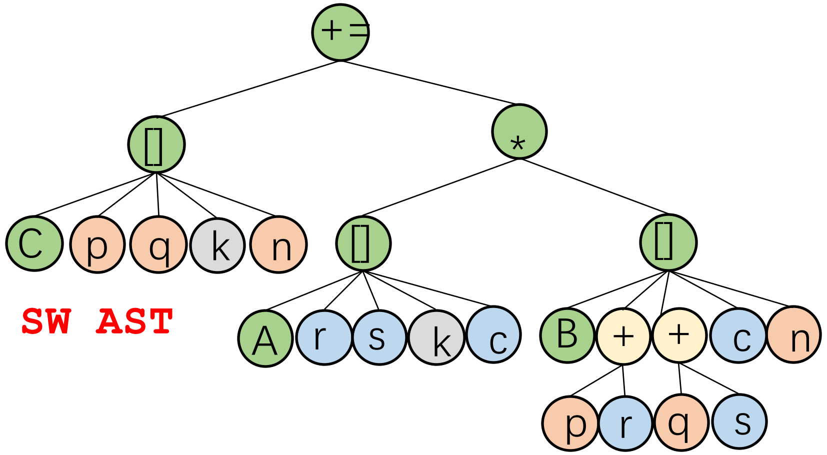


HW AST

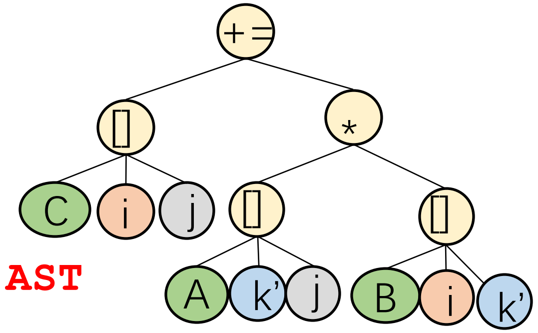
Example 2

n	i
k	j
p	i
q	i
c	k'
r	k'
s	k'

Mapping



SW AST



HW AST

Performance

Mali GPU: Bifrost architecture with dot intrinsic

```
c[0] += a[k]*b[k]
```

a0	a1	a2	a3

X

b0	b1	b2	b3

=

$\sum ai*bi$

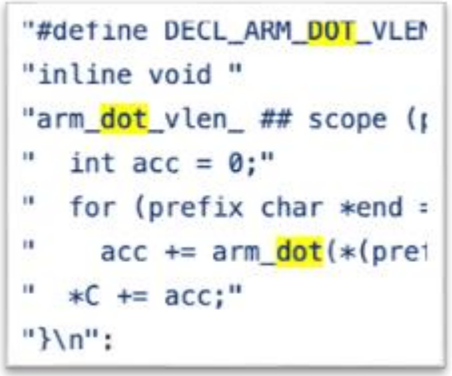
convolution data layout transformation

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

4x4 Tile

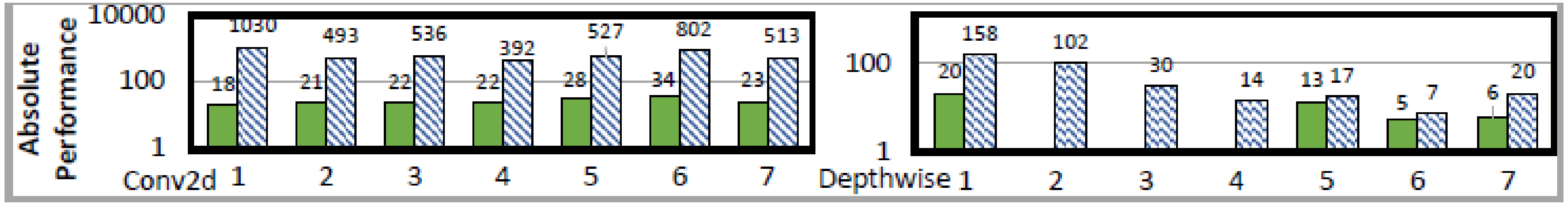
1	2	3	5	6	7	9	10	11
2	3	4	6	7	8	10	11	12
5	6	7	9	10	11	13	14	15
6	7	8	10	11	12	14	15	16

4x9 matrix



```
"#define DECL_ARM_DOT_VLEN
"inline void "
"arm_dot_vlen_ ## scope (
"  int acc = 0;"
"  for (prefix char *end =
"    acc += arm_dot(*(pre
"  *C += acc;"
"}\n";
```

code with dot intrinsic



TVM AutoTVM AMOS

Mali G76 GPU resutls

To AutoTVM, an order of magnitude speedup

Performance

AVX-512 CPU: with VNNI instructions

```
c[i] += a[i,k]*b[i,k]
```

a0	a1	a2	a3	a4	a5	…	a30	a31

X

b0	b1	b2	b3	b4	b5	…	b30	b31

||

c0	c1	c2	…	c15
a0*b0+a1*b1	a2*b2+a3*b3	a4*b4+a5*b5		a30*b30+a31*b31

Convolution data layout transformation

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

4x4 Tile

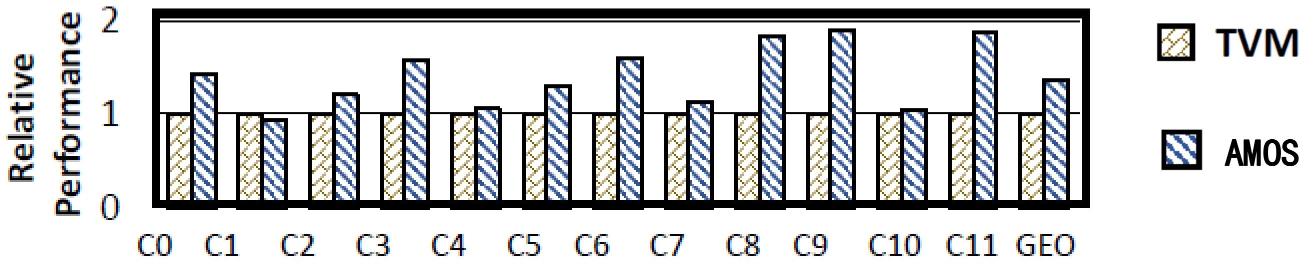
1	2	3	5	6	7	9	10	11
2	3	4	6	7	8	10	11	12
5	6	7	9	10	11	13	14	15
6	7	8	10	11	12	14	15	16

4x9 matrix

```
pair_reduction = tvm.tir.call_llvm_pure_intrin(
    "int16x32",
    "llvm.x86.avx512.pmaddubs.w.512",
    tvm.tir.const(0, "uint32"),
    vec_a,
    vec_b,
)
quad_reduction = tvm.tir.call_llvm_pure_intrin(
    "int32x16",
    "llvm.x86.avx512.pmaddw.d.512",
    tvm.tir.const(0, "uint32"),
    pair_reduction,
    vec_one,
)
```

generated code with VNNI

Xeon(R) Silver 4110 CPU results

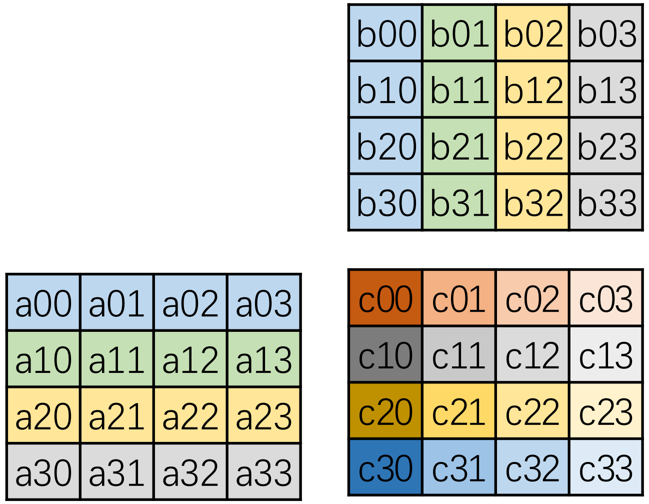


To TVM speedup: 1.37x

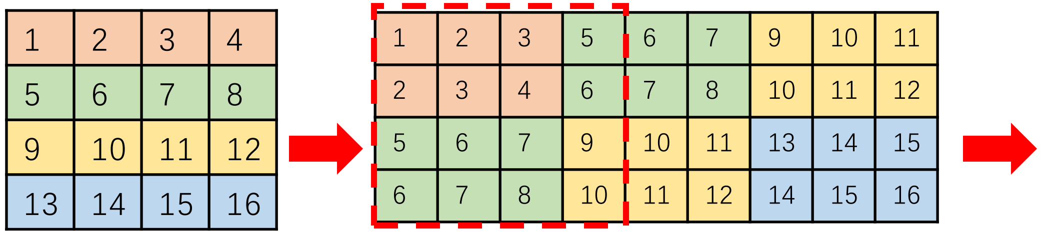
Performance

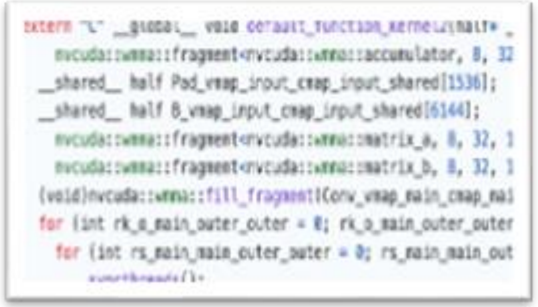
Nvidia Tensor Core GPU: with WMMA intrinsic

```
c[i,j] = a[i,k]*b[k,j]
```



Convolution data layout transformation





4x4 Tile

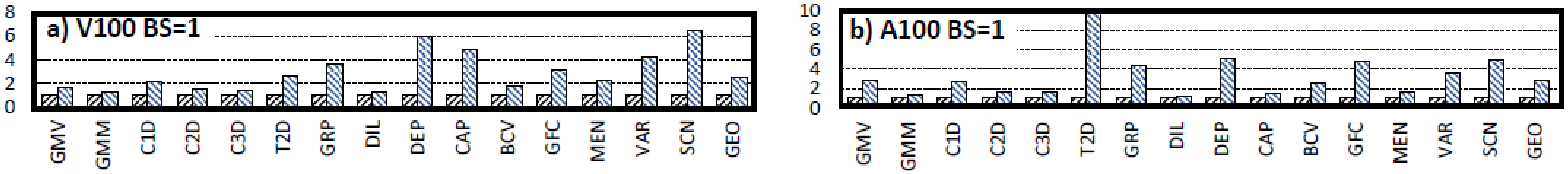
4x9 matrix

generated code with Tensor Core

Two different Tensor Core GPUs

PyTorch AMOS

To PyTorch 2x speedup



Outline

1 Background

- AI Chip
- AI Algorithm
- AI Compiler

2 Techniques

- Compiler for DNN Graph
- Compiler for Operator
- Compiler for Distributed

3 Future Work

- Triton-CuTe
- LLM for Compiler

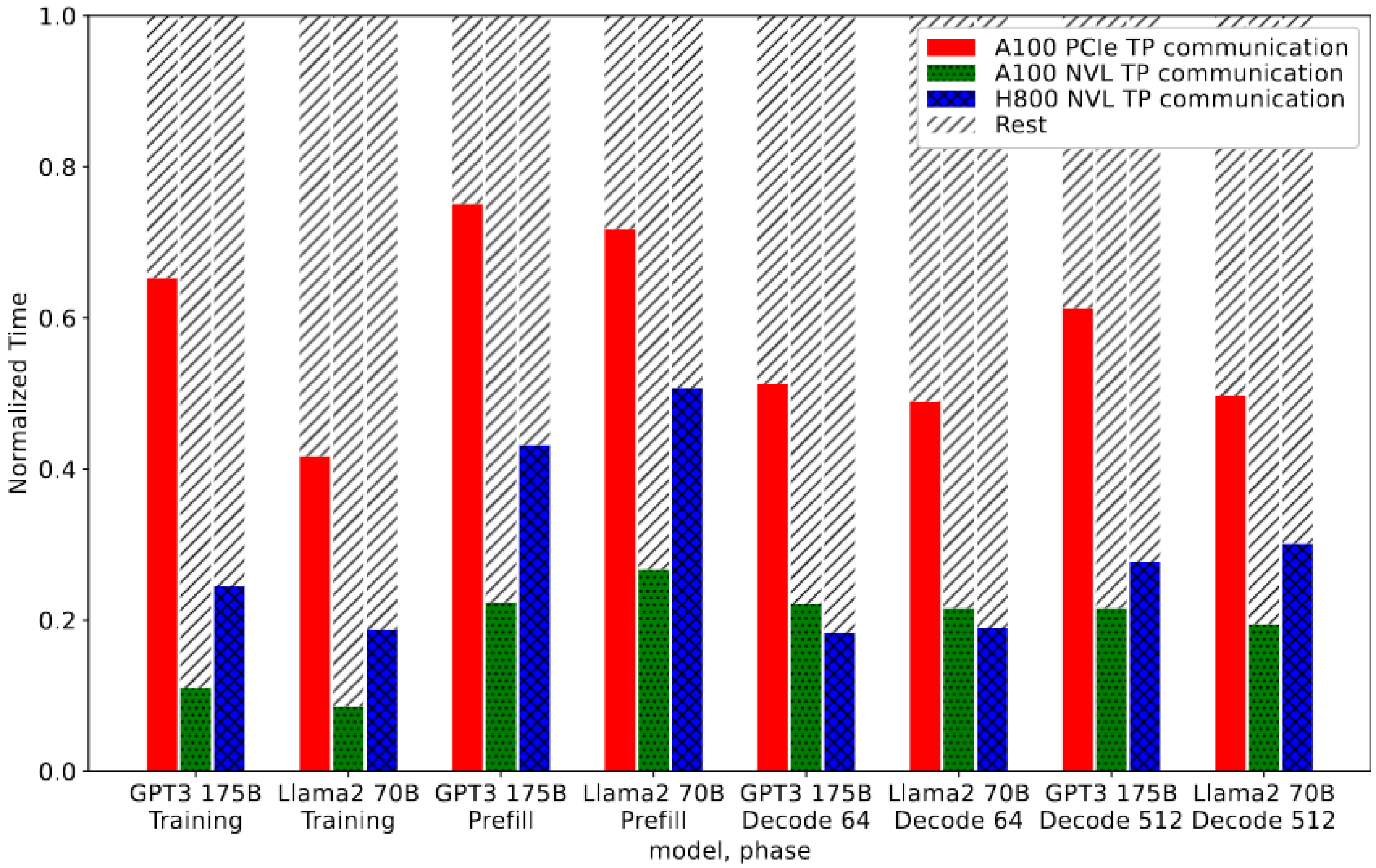
Communication Optimization Challenge

Training

	Comm. Ratio	Comp. Ratio
A100-PCIe	40%-70%	30%-60%
A100-NVLink	~10%	~90%
H800-NVLink	~20%	~80%

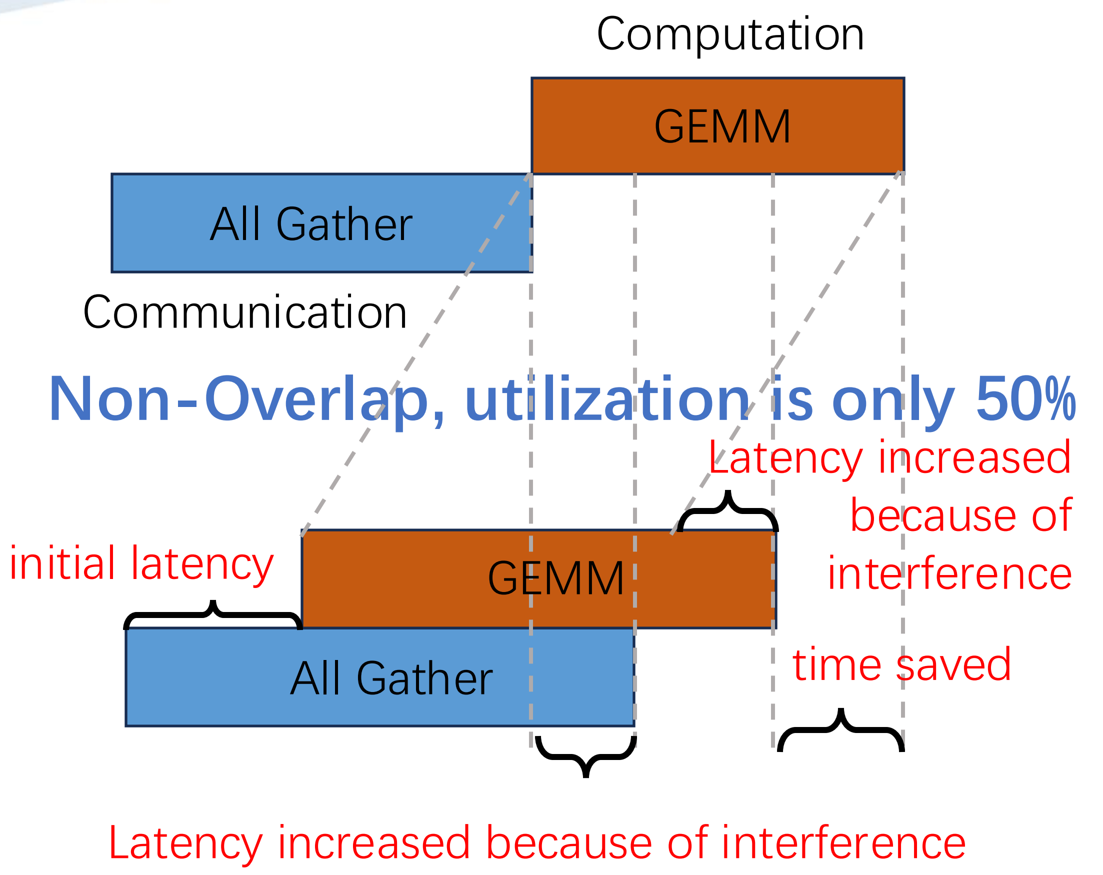
Inference

	通信占比	计算占比
A100-PCIe	50%-80%	20%-50%
A100-NVLink	>20%	<80%
H800-NVLink	20%-50%	50%-80%

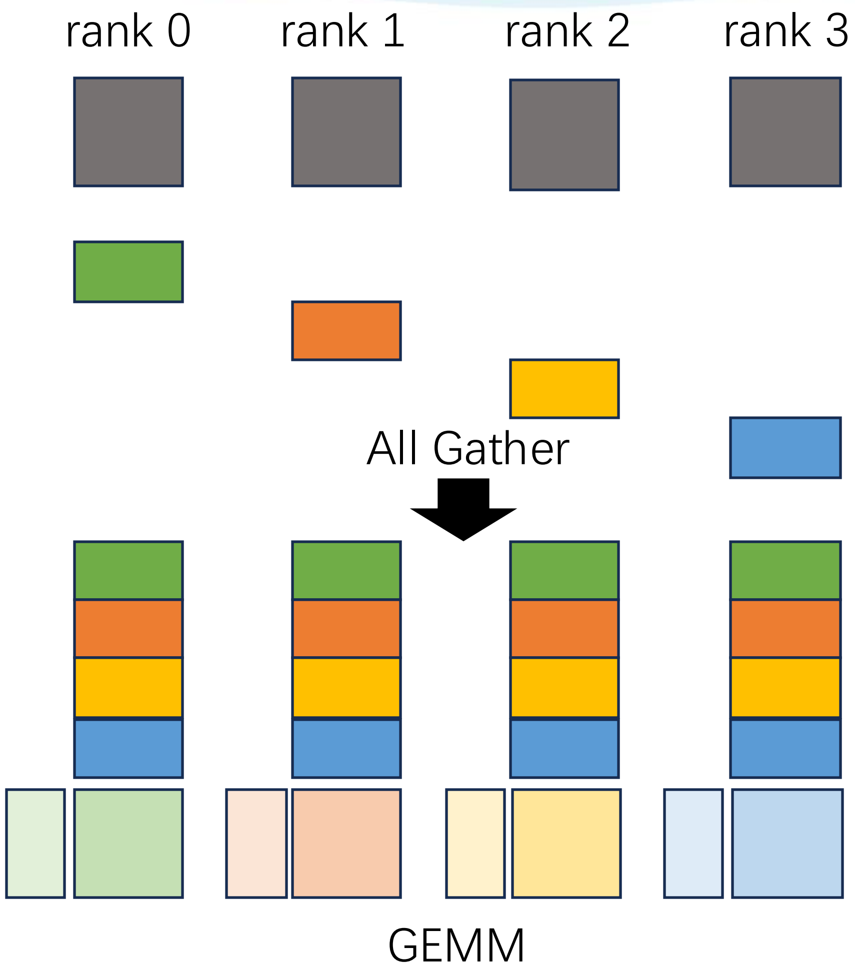


Bubble caused by communication lowers overall compute utilization

Overlapping Compute and Communication

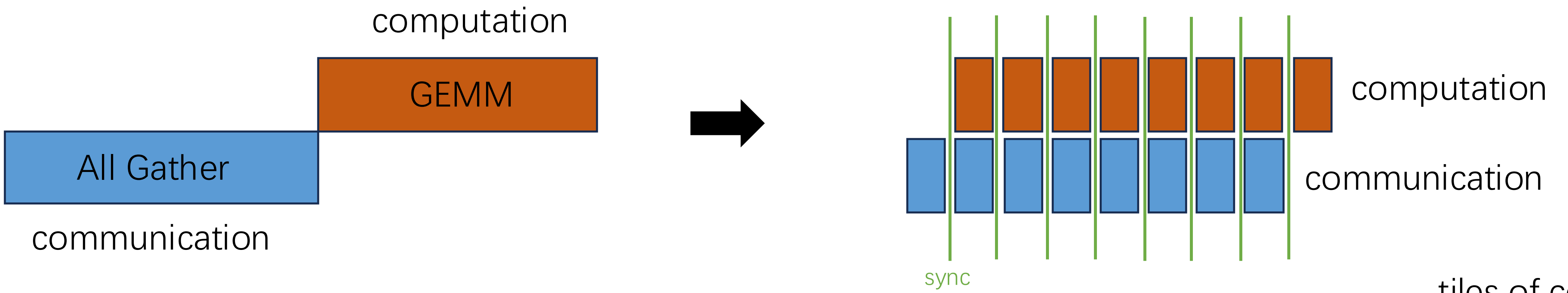


Overlapped, utilization raises to 75%

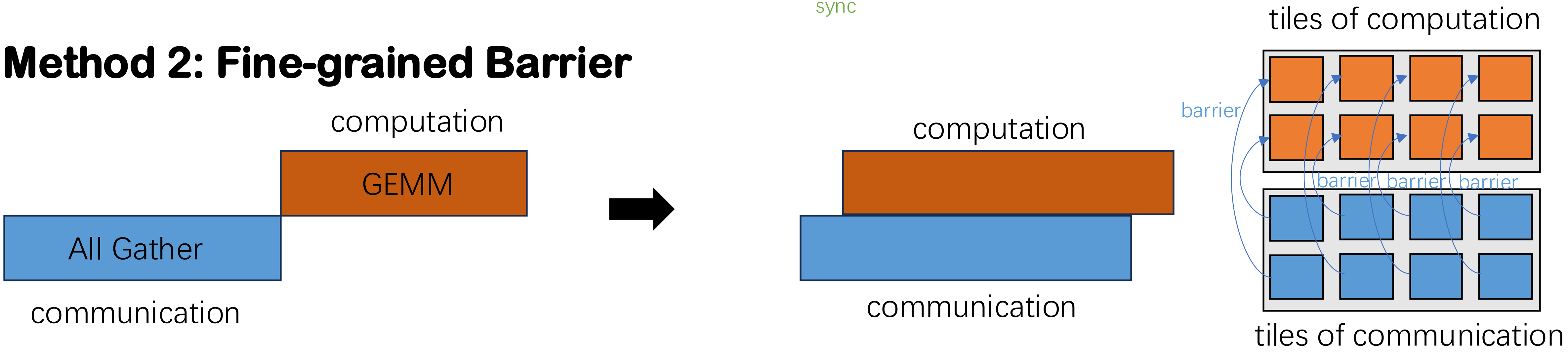


Different Methods

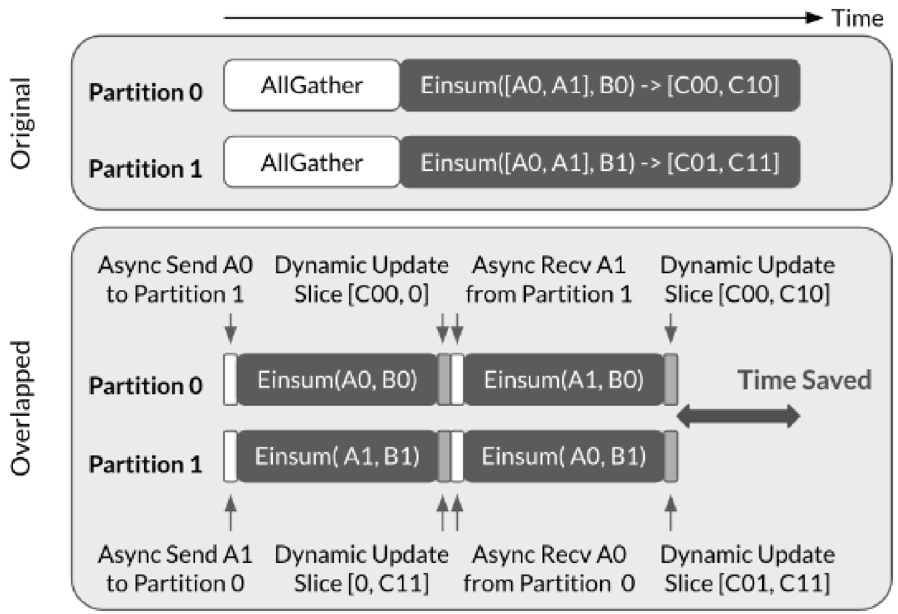
Method 1: Operator Decomposition



Method 2: Fine-grained Barrier



Operator Decomposition

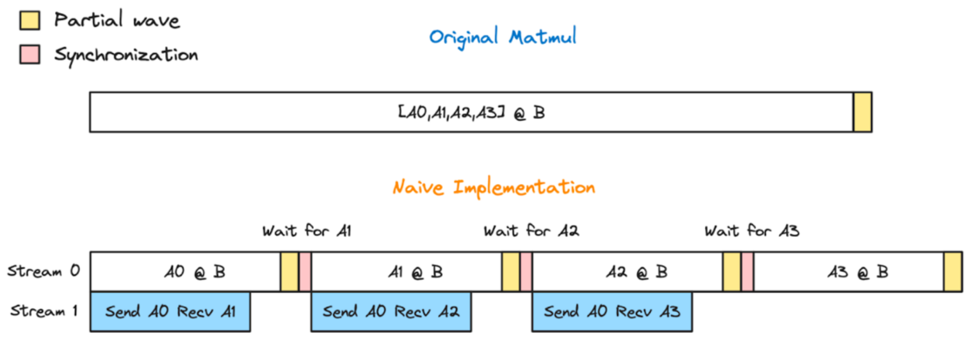


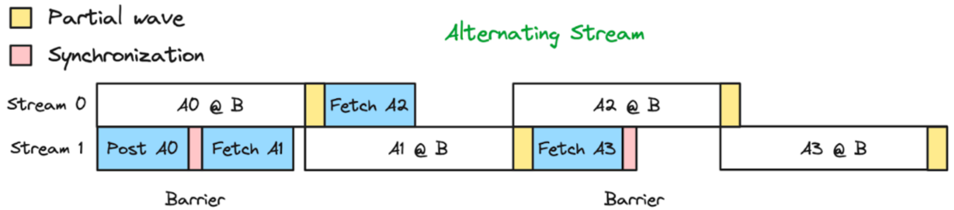
Issues：

1. **Low resource utilization**
2. **Quantization inefficiency**
3. **Stream uncertainty**

Advantages：

1. **Easy to implement**

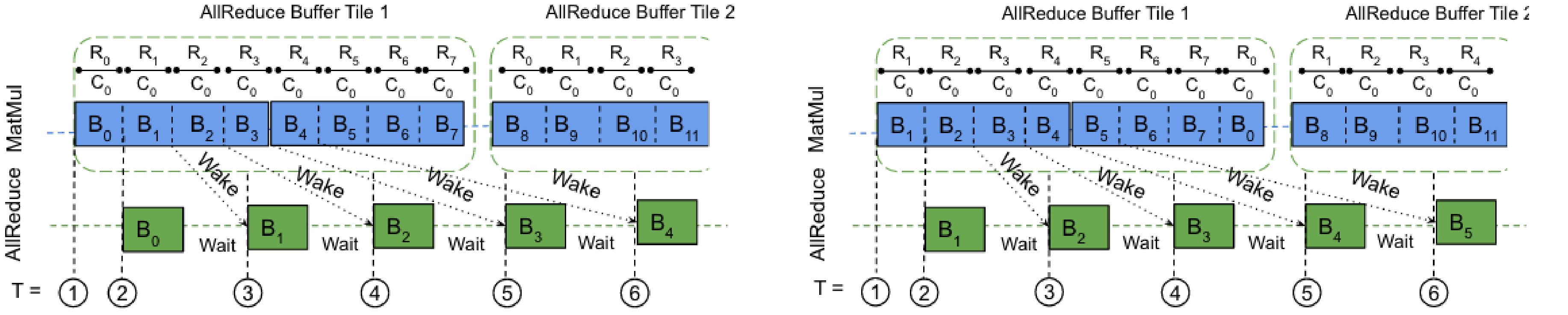




[1] Overlap Communication with Dependent Computation via Decomposition in Large Deep Learning Models
[2] PyTorch Async-TP: https://discuss.pytorch.org/t/distributed-w-torchtitan-introducing-async-tensor-parallelism-in-pytorch/209487

Fine-grained Barrier

Barrier on Device



(a) Workflow of overlap on rank 0. Rank 0 starts with chunk 0.

(b) Workflow of overlap on rank 1. Rank 1 starts with chunk 1.

[1] Breaking the Computation and Communication Abstraction Barrier in Distributed Machine Learning Workloads

Example code in CUDA

```
#if (__CUDA_ARCH__ >= 700)
    /// SM70 and newer use memory consistency qualifiers

    // Acquire pattern using acquire modifier
    asm volatile ("ld.global.acquire.gpu.b32 %0, [%1];\n" : "=r"(state) : "l"(ptr));
CUTLASS_DEVICE
static void wait_eq(void *lock_ptr, int thread_idx, int flag_idx, T val = 1)
{
  T *flag_ptr = reinterpret_cast<T*>(lock_ptr) + flag_idx;

  if (thread_idx == 0)
  {
     // Spin-loop
     #pragma unroll 1
     while(ld_acquire(flag_ptr) != val) {}
  }
  Sync::sync();
}
```

Issues:

1. hard to implement
2. resource conflict

Advantages:

1. fine-grained control
2. better performance

FLUX

Opensource: https://github.com/bytedance/flux

	M	K	N	Torch Gemm	Torch NCCL	Torch Total	Flux Gemm	Flux Comm	Flux Total
AG+Gemm (A800)	4096	12288	49152	2.438ms	0.662ms	3.099ms	2.378ms	0.091ms	2.469ms
Gemm+RS (A800)	4096	49152	12288	2.453ms	0.646ms	3.100ms	2.429ms	0.080ms	2.508ms
AG+Gemm (H800)	4096	12288	49152	0.846ms	0.583ms	1.429ms	0.814ms	0.143ms	0.957ms
Gemm+RS (H800)	4096	49152	12288	0.818ms	0.590ms	1.408ms	0.822ms	0.111ms	0.932ms

Use fine-grained barrier method.

Give the best performance on GPUs so far.

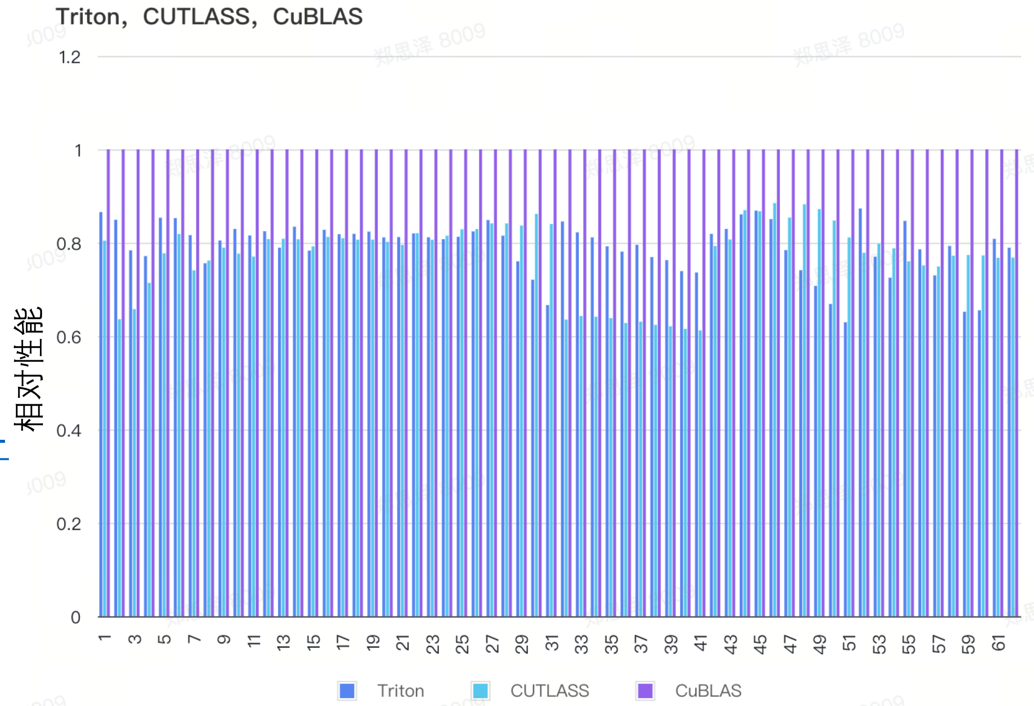
Triton-FLUX

Use Compiler for Compute-Communication Overlapping

Mostly focus on barrier-related semantics

Related Work：

[1] Breaking the Computation and Communication Abstraction Barrier in Distributed Machine Learning Workloads
[2] Overlap Communication with Dependent Computation via Decomposition in Large Deep Learning Models
[3] Triton All Gather GEMM: https://github.com/yifuwang/symm-mem-recipes/tree/main



Triton： code-gen for computation part

Triton

This is the development repository of Triton, a language and compiler for writing highly efficient custom Deep-Learning primitives. The aim of Triton is to provide an open-source environment to write fast code at higher productivity than CUDA, but also with higher flexibility than other existing DSLs.

The foundations of this project are described in the following MAPL2019 publication: Triton: An Intermediate Language and Compiler for Tiled Neural Network Computations. Please consider citing this work if you use Triton!

The official documentation contains installation instructions and tutorials. See also these third-party Triton puzzles, which can all be run using the Triton interpreter -- no GPU required.

Triton has achieved comparable performance for computation (GEMM) to hand-optimized libraries (CUTLASS)

Support Communication Instructions

Intra-GPU and Inter-GPU： synchronization and barrier

sync within threadblock

```
def __syncthreads():
  inline_asm("bar.sync 0;")
```

load barrier

```
def ld_acquire(ptr, scope):
  return inline_asm("ld.global.acquire.{scope}.b32 $0, [{ptr} ];")
```

increase barrier

```
def red_release(ptr, scope, value):
  inline_asm("red.release.{scope}.global.add.s32 [{ptr}], {value};")
```

spin lock

```
def wait_eq(ptr, value):
  while (ld_acquire(ptr, "sys") != value):
    pass
```

High-level Primitives

Block-level Communication primitives：

for peer-to-peer or producer-consumer communications

block-level producer push scatter all

```
def producer_block_push_scatter_all(block_id, data):
  for dst_rank in range(WORLD_SIZE):
    dst_ptr = retrieve_dst_ptr(block_id, dst_rank)
    store(dst_ptr, data)
```

block-level producer push signal and consumer wait signal

```
def producer_push_signal(block_id):
  __syncthreads()
  barrier_ptr = retrieve_barrier_ptr(block_id)
  if tid(axis=0)==0:
    red_release(barrier_ptr, “sys”, 1)
```

```
def consumer_block_wait(block_id, data):
  barrier_ptr = retrieve_barrier_ptr(block_id)
  if tid(axis=0) == 0:
    wait_eq(barrier_ptr, 1)
  __syncthreads()
```

Pointer-control：Get remote pointers from only rank_id and block_id

Triton Extension

Enhance Triton Compiler： Compute-Communication within one Triton kernel

All Gather Kernel Implementation

```
@triton.jit
@sc.jit(backend="triton")
def kernel_producer_all_gather_all2all_push(
  local_tensor_ptr,allgather_tensor_group,m,n,stride_m,stride_n,
  BLOCK_SIZE_M: tl.constexpr,
  BLOCK_SIZE_N: tl.constexpr,
  block_channel: scl.BlockChannel2D,
):
  pid = tl.program_id(0)
  offs_m = (pid * BLOCK_SIZE_M + tl.arange(0, BLOCK_SIZE_M)) % m
  for n_idx in range(0, tl.cdiv(n, BLOCK_SIZE_N)):
    offs_n = n_idx * BLOCK_SIZE_N + tl.arange(0, BLOCK_SIZE_N)
    mask = offs_n[None, :] < n
    local_ptrs = local_tensor_ptr + (
      offs_m[:, None] * stride_m + offs_n[None, :] * stride_n
    )
    row_data = tl.load(local_ptrs, mask=mask, other=0.0)
    scl.producer_block_push_scatter_all(
      block_channel,allgather_tensor_group,row_data,pid,n_idx,m,n,
      stride_m,stride_n,BLOCK_SIZE_M,BLOCK_SIZE_N,tl.float16,)
    scl.producer_block_push_signal(block_channel, pid, n_idx)
```

sc.jit: Python AST transformation before triton.jit

block channel is a data structure that encapsulates the mapping among block_id, rank_id, remote_pointers, and barriers

use primitives to complete communication

Triton Extension

Enhance Triton Compiler： Compute-Communication within one Triton kernel

Consumer of All Gather: Just Standard GEMM Implementation with Communication Primitives

```
offs_am = tl.max_contiguous(tl.multiple_of(offs_am, BLOCK_SIZE_M), BLOCK_SIZE_M)
offs_bn = tl.max_contiguous(tl.multiple_of(offs_bn, BLOCK_SIZE_N), BLOCK_SIZE_N)
offs_k = tl.arange(0, BLOCK_SIZE_K)
a_ptrs = a_ptr + (offs_am[:, None] * stride_am + offs_k[None, :] * stride_ak)
b_ptrs = b_ptr + (offs_k[:, None] * stride_bk + offs_bn[None, :] * stride_bn)

accumulator = tl.zeros((BLOCK_SIZE_M, BLOCK_SIZE_N), dtype=tl.float32)
scl.consumer_block_wait(block_channel, pid_m, 0)
for k in range(0, tl.cdiv(K, BLOCK_SIZE_K)):
    a = tl.load(a_ptrs, mask=offs_k[None, :] < K - k * BLOCK_SIZE_K, other=0.0)
    b = tl.load(b_ptrs, mask=offs_k[:, None] < K - k * BLOCK_SIZE_K, other=0.0)
    accumulator = tl.dot(a, b, accumulator)
    a_ptrs += BLOCK_SIZE_K * stride_ak
    b_ptrs += BLOCK_SIZE_K * stride_bk

if (c_ptr.dtype.element_ty == tl.float8e4nv):
    c = accumulator.to(tl.float8e4nv)
else:
    c = accumulator.to(tl.float16)

offs_cm = pid_m * BLOCK_SIZE_M + tl.arange(0, BLOCK_SIZE_M)
offs_cn = pid_n * BLOCK_SIZE_N + tl.arange(0, BLOCK_SIZE_N)
c_ptrs = c_ptr + stride_cm * offs_cm[:, None] + stride_cn * offs_cn[None, :]
c_mask = (offs_cm[:, None] < M) & (offs_cn[None, :] < N)
tl.store(c_ptrs, c, mask=c_mask)
```

A single line of code added to previous GEMM kernel

Performance

Support TP-MLP, TP-MoE, SP-Attention

Performance Comparable or Better than Hand-Optimized Code

Table 4. Benchmark Shapes. S is sequence length, H is hidden dimension length, I is intermediate size, E is number of experts.

Configurations of MLP

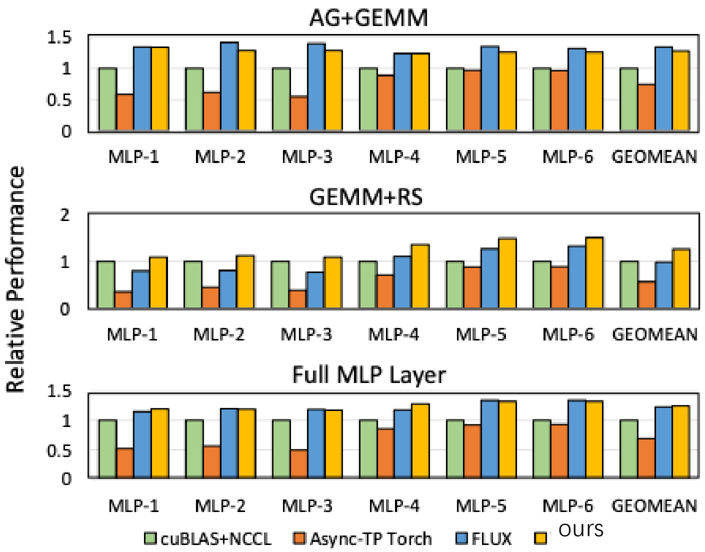
Name	S	H	I	Source Model
MLP-1	8192	4096	11008	LLaMA-7B
MLP-2	8192	4096	14336	LLaMA-3.1-8B
MLP-3	8192	3584	14336	Gemma-2-9B
MLP-4	8192	4608	36864	Gemma-2-27B
MLP-5	8192	8192	28672	LLaMA-3.1-70B
MLP-6	8192	8192	29568	Qwen-2-72B

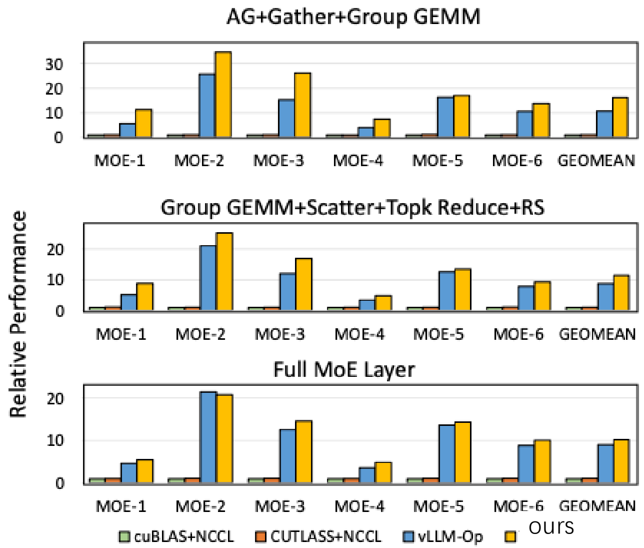
Configuration of MoE

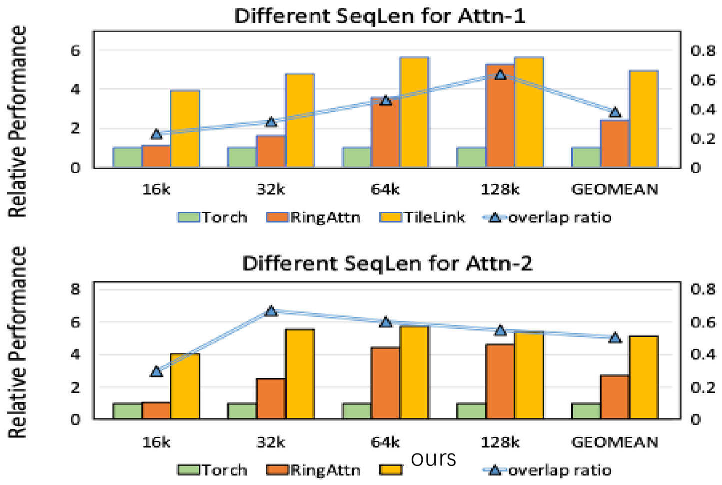
Name	S	H	I	E	topk
MoE-1	8192	2048	1536	8	2
MoE-2	8192	2048	1536	32	2
MoE-3	8192	2048	1536	32	5
MoE-4	8192	4096	2048	8	2
MoE-5	8192	4096	2048	32	2
MoE-6	8192	4096	2048	32	5

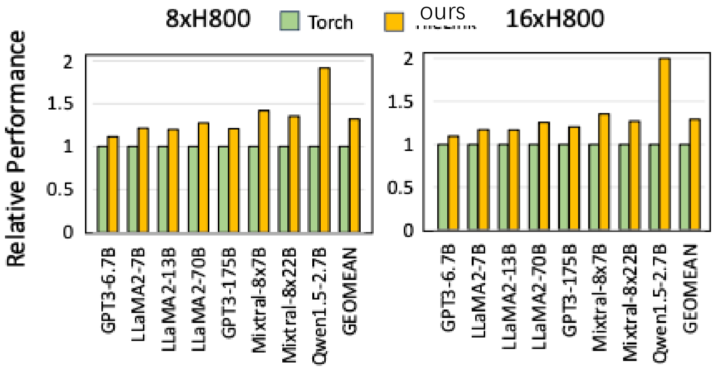
Configuration of self-attention

Name	heads	head dim	sequence length choices
Attn-1	32	128	16k, 32k, 64k, 128k
Attn-2	64	128	16k, 32k, 64k, 128k









Outline

1 Background

- AI Chip
- AI Algorithm
- AI Compiler

2 Techniques

- Compiler for DNN Graph
- Compiler for Operator
- Compiler for Distributed

3 Future Work

- Triton-CuTe
- LLM for Compiler

Future Work

Triton-CuTe From Triton Language to CUDA source code generation

Triton Performance Issue:

1. Performance is bad for some operators (e.g., GroupGemm)
2. Rigid pipeline control and resource control

Triton-CuTe Plan:

1. CUDA source code generator
2. Generate code using CuTe templates

LLM for Compiler LLM as Compiler and LLM –guided Code-gen

Manually-designed Passes are Hard to Generalize:

1. Generalize to new Ops (e.g., MMA pipeline for load with barrier)
2. Generalize to new language (e.g., pipelines for CUDA transferred to other languages)