TensorIR

An Abstraction for Tensorized Program Optimization

Siyuan Feng

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Collaborators

Bohan Hou
@CMU

Junru Shao
@OctoML

Ruihang Lai
@SJTU

Hongyi Jin
@SJTU

Wuwei Lin
@OctoML

Zihao Ye
@UW

Tianqi Chen
@CMU & OctoML

TensorIR

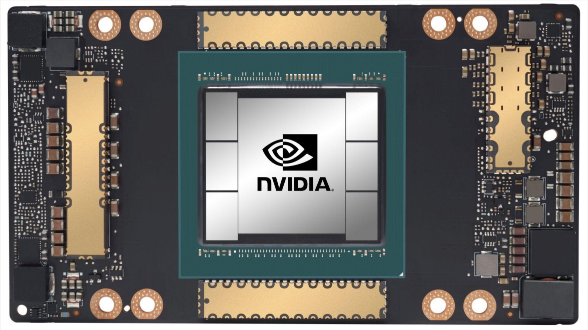
An Abstraction for Tensorized Program Optimization

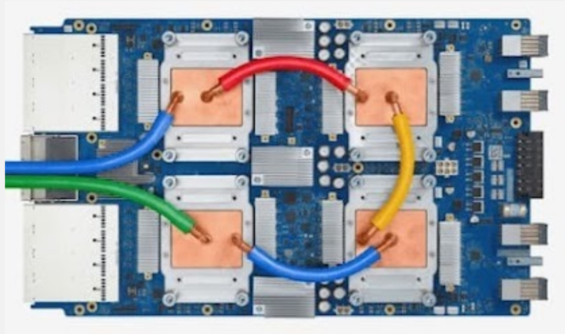
Siyuan Feng

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

tvm

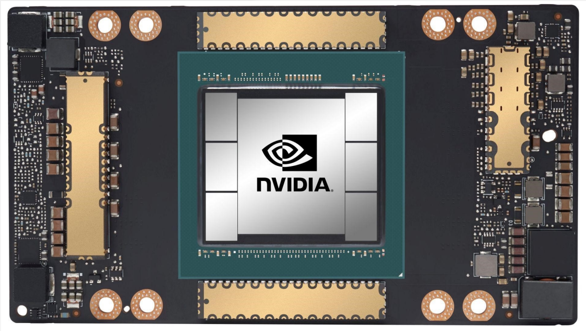
Machine Learning Hardware History

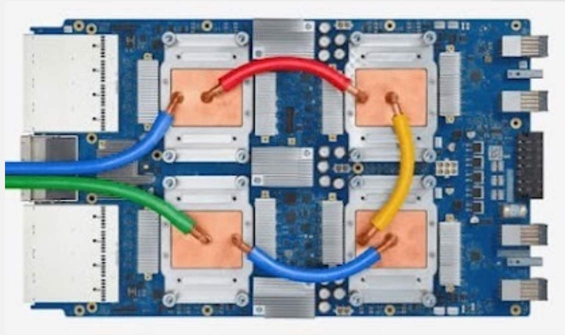


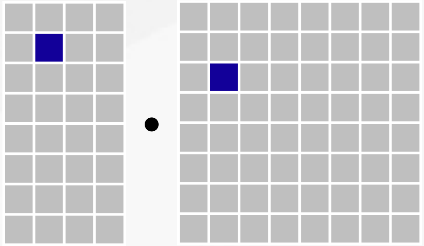


Time

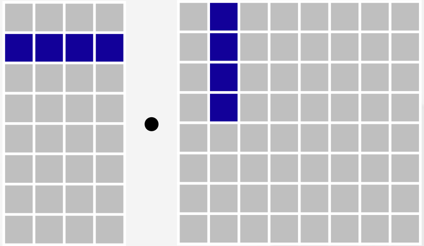
Machine Learning Hardware History



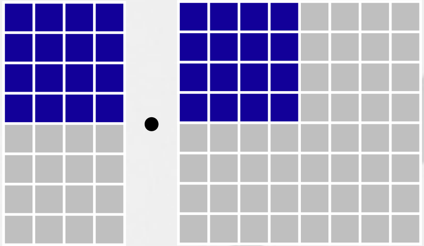




Scalar Computing

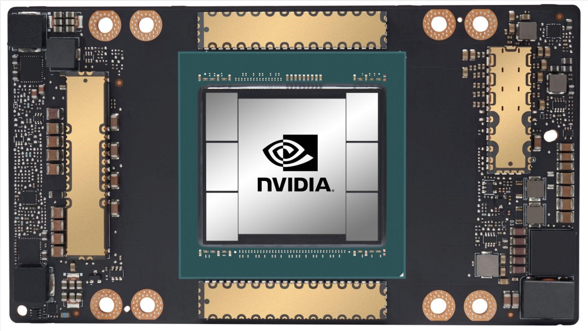


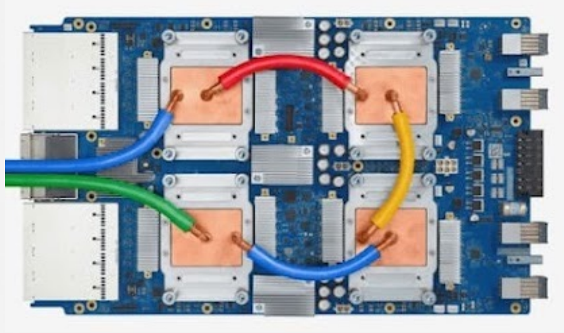
Vector Computing

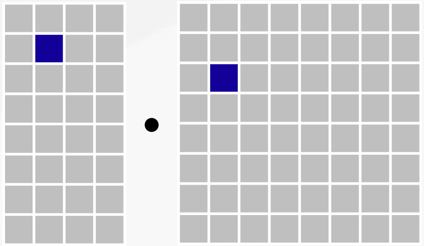


Tensor Computing

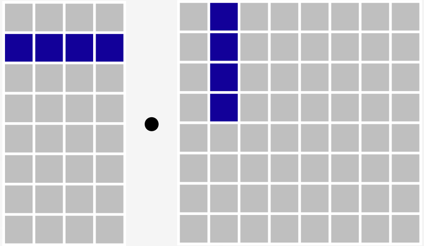
More Tensor Computing Hardware



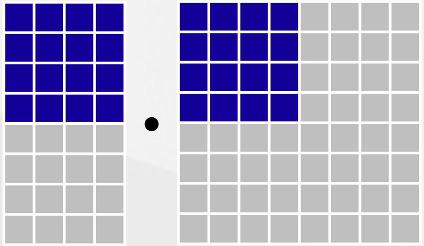




Scalar Computing



Vector Computing



Tensor Computing

- Google TPU
- Nvidia Tensor Core
- AMD Matrix Core
- Intel Matrix Engine
- Apple Neural Engine
- Arm Ethos-N
- T-Head Hanguang
- ……

Tensorized Program is the Bridge from Model to Tensor Hardware

```
for ic.outer, kh, ic.inner, kw in grid(...):

  for ax0 in range(...):
    load_matrix_sync(A.wmma.matrix_a, 16, 16, 16, ...)


  for ax0 in range(...):
    load_matrix_sync(W.wmma.matrix_b, 16, 16, 16, ...)


  for n.c, o.c in grid(...):
    wmma_sync(Conv.wmma.accumulator,
              A.wmma.matrix_a,
              W.wmma.matrix_b,
              ...)


for n.inner, o.inner in grid(...):
  store_matrix_sync(Conv.wmma.accumulator, 16, 16, 16)
```

Example Snippet: Conv2D on Tensor Core

Optimized loop nests with thread binding

Multi-dimensional data load into **specialized memory buffer**

Opaque tensorized computation body
16x16 matrix multiplication

Critical Challenges when Deploying Models to Tensor Hardware

- How to write?
- How to optimize?
- How to customize?

Popular Methods on Writing Tensorized Program

```
for i0, k0, j0, k1 in grid(...):

  for i1 in range(...):
    ...

  for j1 in range(...):
    ...

  for i1, j1 in grid(...):
    ...

for i, j in grid(...):
  ...
```

Manually Write

```
C = compute((N, M), lambda i, j:
                    sum(A[i, k]*B[k, j], reduce=k))
```

Schedule / Optimize

```
for i0 in range(...):
  for j0 in range(...):
    for k in range(...):
       for i1 in range(...):
         for j1 in range(...):
           C[...] += A[...] * B[...]
```

Auto-generating

Popular Methods on Writing Tensorized Program

```
for i0, k0, j0, k1 in grid(...):

  for i1 in range(...):
    ...

  for j1 in range(...):
    ...

  for i1, j1 in grid(...):
    ...

for i, j in grid(...):
  ...
```

Manually Write

How to write?

How to optimize?

How to customize?

Popular Methods on Writing Tensorized Program

```
C = compute((N, M), lambda i, j:
            sum(A[i, k]*B[k, j], reduce=k))
```

Schedule / Optimize

```
for i0 in range(...):
  for j0 in range(...):
    for k in range(...):
      for i1 in range(...):
        for j1 in range(...):
          C[...] += A[...] * B[...]
```

Auto-generating

How to write?

How to optimize?

How to customize?

TensorIR: Write a Program and Optimize it

```
for i in range(...):
  for j in range(...):
    for k in range(...):
      C[...] += A[...] * B[...]
```

Manually Write

Schedule / Optimize

```
for i0 in range(...):
  for j0 in range(...):
    for k in range(...):
       for i1 in range(...):
         for j1 in range(...):
           C[...] += A[...] * B[...]
```

(Automatic) Optimization

How to write?

How to optimize?

How to customize?

TensorIR: Write a Program and Optimize it

How to write?	→	TVMScript
How to optimize?	→	Interactive Schedule on Tensorized Body
How to customize?	→	Decoupled Primitives

TensorIR: Write a Program and Optimize it

		TensorIR
How to write?	→	TVMScript
How to optimize?	→	Interactive Schedule on Tensorized Body
How to customize?	→	Decoupled Primitives

Use TVMScript to Write a Program

```python
@T.prim_func
def fuse_add_exp(a: T.handle, c: T.handle):
    A = T.match_buffer(a, (64,))
    C = T.match_buffer(c, (64,))
    B = T.alloc_buffer((64,))

    for i in range(64):
        with T.block("B"):
            vi = T.axis.S(64, i)
            B[vi] = A[vi] + 1

    for j in range(64):
        with T.block("C"):
            vi = T.axis.S(64, j)
            C[vi] = exp(B[vi])
```

Multi-dimensional **buffer**

Loop nests

Computational **block**

Design Goal 0:
Write a tensor program in a python-AST based syntax.

Use TVMScript to Write a Program with Tensorized Computation

```
@T.prim_func
def fuse_add_exp(a: T.handle, c: T.handle):
    A = T.match_buffer(a, (64,))
    C = T.match_buffer(c, (64,))
    B = T.alloc_buffer((64,))

    for i in range(8):
        with tir.block("B") as [vi]:
            vi = T.axis.S(8, i)
            tir.reads(A[vi * 8: vi * 8 + 8])
            tir.writes(B[vi * 8: vi * 8 + 8])
            for k in range(8):
                B[vi * 8 + k] = A[vi * 8 + k] + 1
for j in range(64):
        with T.block("C"):
            vi = T.axis.S(64, j)
            C[vi] = exp(B[vi])
```

Multi-dimensional **buffer**

Loop nests

Block representing vectorized/
tensorized computation
Add 8 elements at a time

Design Goal 1:
Tensorized computation as the first-class citizen

Basic Unit in TensorIR: Block

```
for yo, xo, ko in grid(16, 16, 16):
    with block():
        vy = spatial_axis(length=16, value=yo)
        vx = spatial_axis(length=16, value=xo)
        vk =  reduce_axis(length=16, value=ko)

        read  A[vy*4:vy*4+4, vk*4:vk*4+4]
        read  B[vk*4:vk*4+4, vx*4:vx*4+4]
        write C[vy*4:vy*4+4, vx*4:vx*4+4]

        for yi, xi, ki in grid(4, 4, 4):
           C[vy*4 + yi, vx*4 + xi] +=
               A[vy*4 + yi, vk*4 + ki] * B[vk*4 + ki, vx*4 + xi]
```

Outside loop nesting

Block iterator domain

Producer consumer dependency relations

Only indexed by block iterator

Design Goal 2:
Isolate the internal computation tensorized computation from external loops

TensorIR: Divide and Conquer

```
for y, x, k in grid(64, 64, 64):
    C[y, x] += A[y, k] * B[k, x]
```

Introduce a key abstraction called **block** to **divide** and isolate the problem space into outer loop nests and **tensorized** body

```
for yo, xo, ko in grid(16, 16, 16):
  block (by=yo, bx=xo, bk=ko)
  for y, x, k in grid(4, 4, 4):
    C[by*16+y, bx*16+x] +=
      A[by*16+y, bk*16+k] * B[bk*16+k, bx*16+x]
```

Tensorized body (matmul4x4) **isolated** from the outer loop nests

Key Ideas

- Divide problem into sub-tensor computation blocks
- Generalize loop optimization for tensorized computation
- Combination of the above approaches in any order

Search space of loops transformations with **tensorized operations**

Tensorized Programs

```
for yo, xo, k in grid(4, 4, 16):
  for yi, xi in grid(4, 4):
    block (by, bx, bk=...)
    Tensorized body (matmul4x4)
```

Map tensorized body based on instructions provided by the backend.

Option 0: Tensorized body (matmul4x4)

```
accel.matmul_add4x4(
  C[by*16:by*16+4, bx*16:bx*16+4],
  A[by*16:by*16+4, bk*16:bk*16+4],
  B[bk*16:bk*16+4, bx*16:bx*16+4])
```

Option 1: Tensorized body (matmul4x4)

```
for y, x, k in grid(4, 4, 4):
  C[by*16+y, bx*16+x] +=
    A[by*16+y, bk*16+k] *
    B[bk*16+k, bx*16+x]
```

Transformation for Tensorized Computation

```
blockB signature
```

Iterator domain and constraints:

```
vy = spatial_axis(length=16)
vx = spatial_axis(length=16)
vk =  reduce_axis(length=16)
```

Producer consumer dependency relations:

```
read   A[vy*4:vy*4+4, vk*4:vk*4+4]
read   B[vk*4:vk*4+4, vx*4:vx*4+4]
write  C[vy*4:vy*4+4, vx*4:vx*4+4]
```

Block signature dependency information used during transformation

```
for i, j in grid(64, 64):
  ProduceA block
    A[i, j] = …
for yo, xo, k in grid(4, 4, 16):
  for yi, xi in grid(4, 4):
    blockB (ConsumeA block)
      vy = S_axis(yo*4 + yi)
      vx = S_axis(xo*4 + xi)
      vk = R_axis(ko)
      body
```

compute_at

```
for yo, xo, k in grid(4, 4, 16):
  for i, j in grid(16, 4):
    ProduceA block
      A[yo*16 + i, k*4 + j] = …
  for yi, xi in grid(4, 4):
    blockB (ConsumeA block)
      vy = S_axis(yo*4 + yi)
      vx = S_axis(xo*4 + xi)
      vk = R_axis(ko)
      body
```

Change compute location of produceA block to loop iterator k.
The system will use the dependency information to calculate the subregion of A to compute after the transformation to satisfy the read requirement of the matmul4x4 block.

Design Goal 3:
Enable loop transformations of tensorized compute body

TensorIR: Interactive Schedule in Eager Mode

Design Goal 4:
Make schedule user-friendly and show the result immediately

```python
A = te.placeholder((8,),
    dtype="float32", name="A")
B = te.compute((8,),
    lambda *i: A(*i) + 1.0, name="B")
```

```python
func = te.create_prim_func([A, B])
mod = IRModule({"main": func})
```

```python
@tvm.script.ir_module
class IRModule:
    @T.prim_func
    def main(a: T.handle, b: T.handle):
        A = T.match_buffer(a, (8,))
        B = T.match_buffer(b, (8,))
        for i in range(8):
            with T.block("B"):
                vi = T.axis.S(8, i)
                B[vi] = A[vi] + 1.0
```

```python
mod = IRModule
```

IRModule → `s.split(i,[2, 2, 2])` → IRModule → `s.reorder(i_2,i_0,i_1)` → IRModule → `tvm.build(mod)` → Runnable Module

`mod.script()`

```python
@tvm.script.ir_module
class Module:
    @T.prim_func
    def main(a: T.handle, b: T.handle):
        A = T.match_buffer(a, (8,))
        B = T.match_buffer(b, (8,))
        for i_0, i_1, i_2 in T.grid(2, 2, 2):
            with T.block("B"):
                vi = T.axis.S(8, i_0*4+i_1*2+i_2)
                B[vi] = A[vi] + 1.0
```

`mod.script()`

```python
@tvm.script.ir_module
class Module:
    @T.prim_func
    def main(a: T.handle, b: T.handle):
        A = T.match_buffer(a, (8,))
        B = T.match_buffer(b, (8,))
        for i_0, i_1, i_2 in T.grid(2, 2, 2):
            with T.block("B"):
                vi = T.axis.S(8, i_0*4+i_1*2+i_2)
                B[vi] = A[vi] + 1.0
```

TensorIR: Decoupled Schedule Primitive Design

Schedule primitives work like a **special pass**, which only based on the IRModule

```
StmtSRef ExamplePrimitive(ScheduleState self, ...) {
  // Step 1. Check correctness
  assert CheckValidation(self, old_stmt);
  // Step 2. Create wanted Stmt
  Stmt new_stmt = Mutate(old_stmt);
  // Step 3. Replace
  self->Replace(old_stmt, new_stmt);
}
```

A typical primitive skeleton

Design Goal 5:
Make it easy to use for both users and developers.

Summary

TensorIR

- TVMScript → Write a program with Tensorized body
- Block Isolation
  - Interactive Schedule on Tensorized Body → Optimize tensorized program
  - Decoupled Primitives → Easy to customize for specific hardware

Thanks

Siyuan Feng

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

tvm