TensorIR

An Abstraction for Tensorized Program Optimization

Siyuan Feng
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
Machine Learning Hardware History
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

Example Snippet: Conv2D on Tensor Core

```python
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)
```

- **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

Manually Write

```python
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(...):
        ...
```

Auto-generating

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

Schedule / Optimize

```python
for i0 in range(...):
    for j0 in range(...):
        for k in range(...):
            for i1 in range(...):
                for j1 in range(...):
                    C[...] += A[...] * B[...]
```
Popular Methods on Writing Tensorized Program

Manually Write

```python
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(...):
        ...
```

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

Manually Write

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

Schedule / Optimize

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

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

- 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)

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

Design Goal 0:
Write a tensor program in a python-AST based syntax.
Use TVMScript to Write a Program with Tensorized Computation

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

    for i in range(8):  # Loop nests
        with tir.block("B") as [vi]:
            vi = T.axis.spatial(8, i)
            tir.reads(A[vi * 8: vi * 8 + 8], C[vi])
            tir.writes(B[vi * 8: vi * 8 + 8], C[vi])
            for k in range(8):
                B[vi * 8 + k] = A[vi * 8 + k] + 1

    for j in range(64):  # Block representing vectorized/tensorized computation
        with T.block("C"):  # Add 8 elements at a time
            vi = T.axis.spatial(64, j)
            C[vi] = exp(B[vi])
```

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 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]

Design Goal 2:
Isolate the internal computation tensorized computation from external loops
TensorIR: Divide and Conquer

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

Search space of loops transformations with tensorized operations.

Tensorized Programs

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

```
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

Option 0: Tensorized body (matmul4x4)

```python
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)

```python
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

**Design Goal 3:**
Enable loop transformations of tensorized compute body
TensorIR: Decoupled Schedule Primitive Design

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

```cpp
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