Tensor Program Optimization with Probabilistic Programs

Authors: J. Shao, X. Zhou, S. Feng, B. Hou, R. Lai, H. Jin, W. Lin, M. Masuda, C. H. Yu, T. Chen

Presenter: Andrew Krapivin
Problem: Optimizing Tensor Programs

- Fancy model, but sloooowww
- Not efficient on target architecture!
- **Goal:** have compiler automatically optimize!
Approaches
Solution #0

- Hand tune each model/group of models (yikes!)
- Different for each hardware (yikes)!

```python
for y in range(1024):
    for x in range(1024):
        C[y][x] = 0
    for k in range(1024):
        C[y][x] += A[k][y] * B[k][x]
```

```python
for yo in range(1024 / ty):
    for xo in range(1024 / tx):
        for yi in range(ty):
            for xi in range(tx):
                C[yo*ty+yi][xo*tx+xi] += A[k][yo*ty+yi] * B[k][xo*tx+xi]
```
Solution #1: DSL

- Manual, but with syntactic sugar
- **Advantage:** large optimization space (can do anything)
- **Disadvantages:**
  - Hard to find good variant!
  - Guess/manual test and eval

```python
# Designate a set of tile sizes
i_tiles = [16, 8, 8, 8]
j_tiles = [16, 8, 8, 8]
k_tiles = [256, 8]

# Tile the loops according to the tile sizes
i_0, i_1, i_2, i_3 = sch.split(loop=i, factors=i_tiles)
j_0, j_1, j_2, j_3 = sch.split(loop=j, factors=j_tiles)
k_0, k_1 = sch.split(loop=k, factors=k_tiles)

# Organize the loops into "SSRSRS" 6-level tiles
sch.reorder(
    i_0, j_0, # S: the 1st spatial tile
    i_1, j_1, # S: the 2nd spatial tile
    k_0,     # R: the 1st reduction tile
    i_2, j_2, # S: the 3rd spatial tile
    k_1,     # R: the 2nd reduction tile
    i_3, j_3, # S: the 4th spatial tile
)
```
Solution #2: Automation!

- **Fix** some search space $S$ of optimizations!
- Explore to find good program
- **Advantage:** no (less) manual work!
- **Disadvantage:** New hardware may have new optimizations!

*(Learning to Optimize Tensor Programs, Ansor: Generating High-Performance Tensor Programs for Deep Learning)*
Example: Special Instructions!

- Ansor comes short of utilizing the special instructions, such as Intel VNNI, NVIDIA Tensor Core, and ARM Dot for mixed-precision and low-precision operators, which are not handled well by the off-the-shelf code generators currently.

- Difficult for programmer to add new optimization!

Start over/go back to manual DSL?

Authors: Not anymore!
The New Contribution: Metaschedule

**Goal:** flexibility of DSL, but with automation
- “Expressiveness, modularity, designed for learning”

**Main idea:** decouple *search space* from *search algorithm*
- Search space: possible optimized programs
- Search algorithm: finds good optimized program

**Second idea:** randomly sample parameters
- Probabilistic programming for optimizations
Transformations (Defining Search Space)
Search Space Definition: Program Transformations

- Define transformations (with parameters)
- Transformations composed to form optimized programs
Sampling Parameters (Second Idea)

- Parameters (tiling sizes, etc.) hard to know in advance
- Sample from a set of possible parameters
Composing Transformations
Modules: Manually Composing Transformations

- Ex: multilevel tiling (ex for different parts of memory hierarchy)
- Module: creating transformation from other transformations

Transformation Module

```python
def Multi-Level-Tiling(loop_nest: List[Loop]):
tiles: List[List[Loop]] = [list() for _ in range(10)]
def tile_loop(loop: Loop, tile_ids: List[int]):
    {0} = Sample-Tile(loop, parts=len(tile_ids))
    tiled_loops = Split(loop, {0})
    for i, tile in zip(tile_ids, tiled_loops):
        tiles[i].append(tile)
    {for i in loop_nest:
        if isSpatialLoop(i): tile_loop(i, [0, 1, 3])
        elif isReductionLoop(i): tile_loop(i, [2, 4])
        Reorder(list_concat(tiles))
```

Example: Execution of the Transformation Module

Original tensor program

```python
for i in range(512):
    for j in range(256):
        for k in range(16):
            C[...] += ...
```

Transformed program

```python
for i, j in grid(0, 0, 0):
    for i, j in grid(0, 0, 0):
        for k in range(0):
            C[...] += ...
```
Search: Sampling Parameters, Composing Transformations

- Evolutionary search
- Cost model (evaluation is expensive)
- This part follows previous paper (Ansor)
- Possible to “incorporate other ways”
Evaluation and My Thoughts
Metaschedule Evaluation

- Roughly: competitive with previous best, sometimes wins
- Not about performance, necessarily
- Still, doesn’t seem too interesting...
Crux of the argument: Composability and Tensor Cores

- Ansor comes short of utilizing the special instructions, such as Intel VNNI, NVIDIA Tensor Core, and ARM Dot for mixed-precision and low-precision operators, which are not handled well by the off-the-shelf code generators currently.

- Notably, it took a graduate student only 2 days to craft the 82-line Use-Tensor-Core module.

- Limiting suffering of grad students! ❤️
  - Which (some of us) are soon to become
My Thoughts

Good:

- Competitive, in some cases much better performance
- Fast part written in C++, but programmer can write their own modules in Python
- Automated and extensible

Bad:

- No easily accessible source code
- Lack of details
- Why are previous approaches not extensive?
- Choice of possible parameters is still manual!
- How extensible is it actually?
  - Can we support sparse operators?
- How modular?

dynamic shape is an interesting future direction. Another limitation is that Anso only supports dense operators. To support sparse operators (e.g., SpMM) that are commonly used in sparse neural networks [17] and graph neural networks [25], we expect that a large portion of Anso can still be reused, but we need to redesign the search space. Lastly,
How Modular?

- To the right: use-tensor-core
- Seems to explicitly require SSSRRSRS tiling
- New GPU still has tensor cores but different arch/memory hierarchy?
Questions?