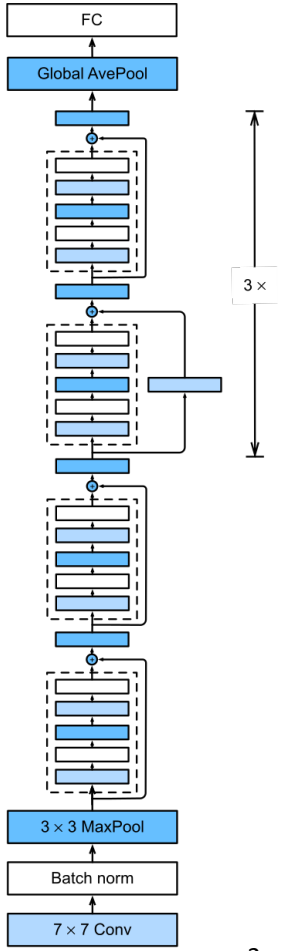
Tensor Program Optimization with Probabilistic Programs

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 slooooowww
- 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)!

```
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]
```

```
for yo in range(1024 / ty):
  for xo in range(1024 / tx):
    C[yo*ty:yo*ty+ty][xo*tx:xo*tx+tx] = 0
    for k in range(1024):
      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

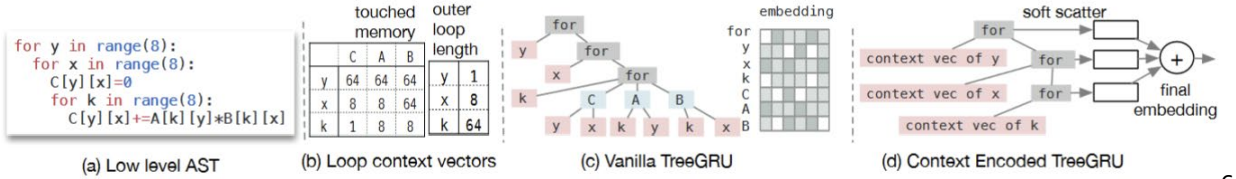
- Manual, but with syntactic sugar
- **Advantage:** large optimization space (can do anything)
- **Disadvantages:**
  - Hard to find good variant!
  - Guess/manual test and eval

```
# 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!

tion). 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*

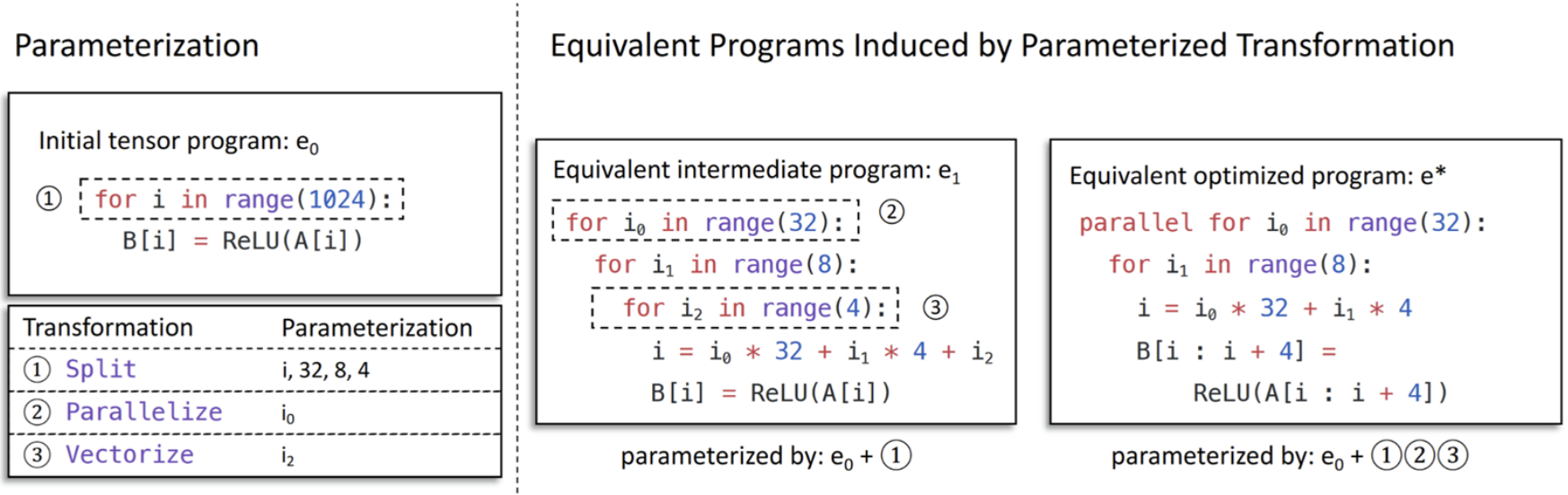
- Search space: possible optimized programs
- Search algorithm: finds good optimized program

Second idea: randomly sample parameters

- Probabilistic programming for optimizations

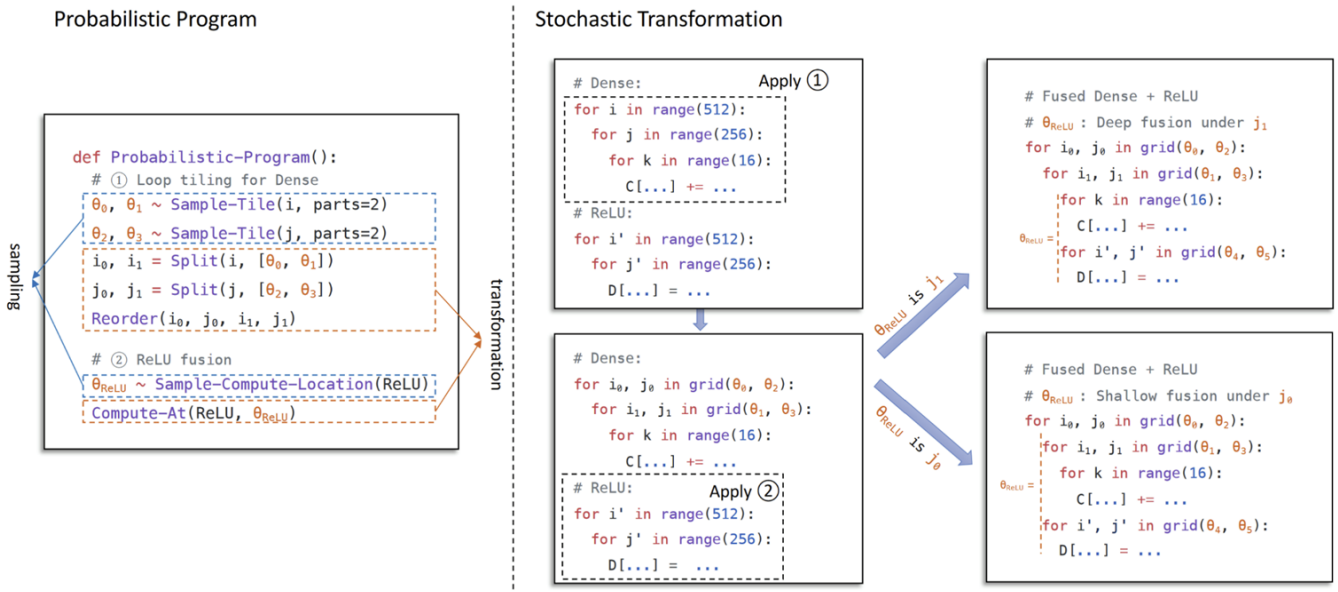
Transformations (Defining Search Space)

Search Space Definition: Program Transformations



- Define transformations (with parameters)
- Transformations composed to form optimized programs

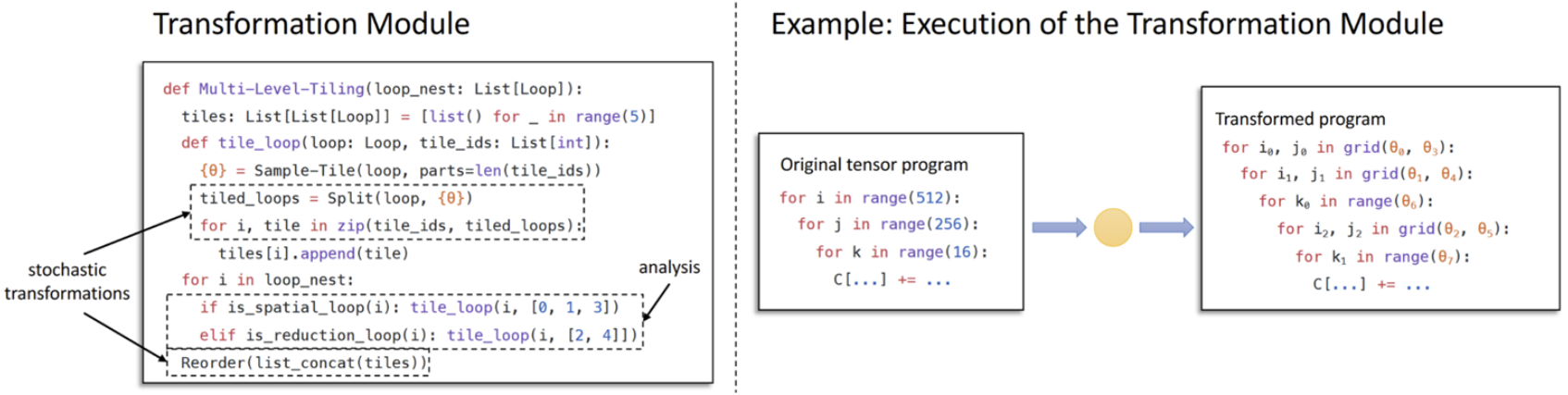
Sampling Parameters (Second Idea)



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

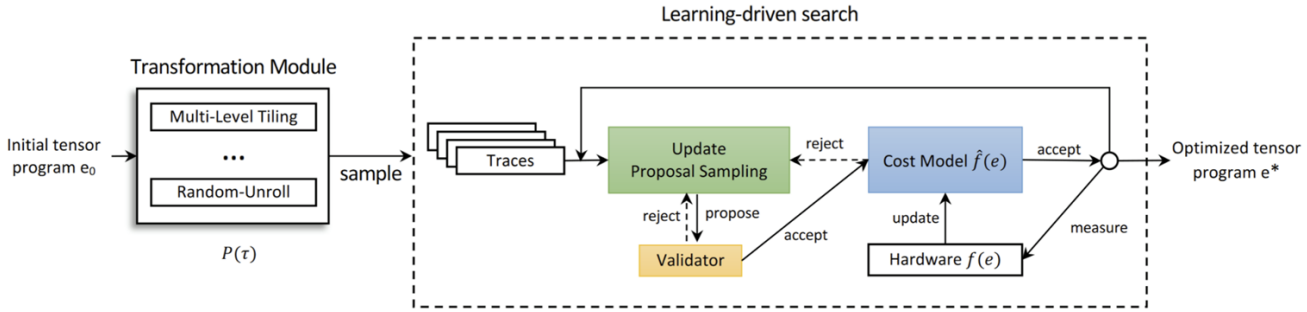
Composing Transformations

Modules: Manually Composing Transformations



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

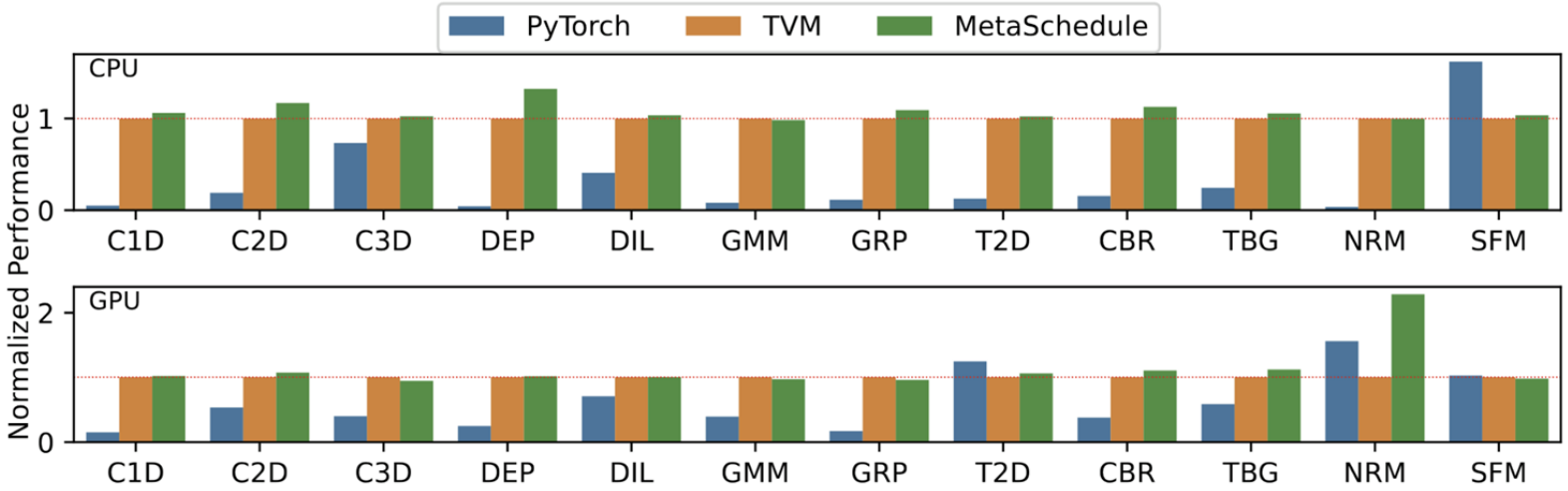
Search: Sampling Parameters, Composing Transformations



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

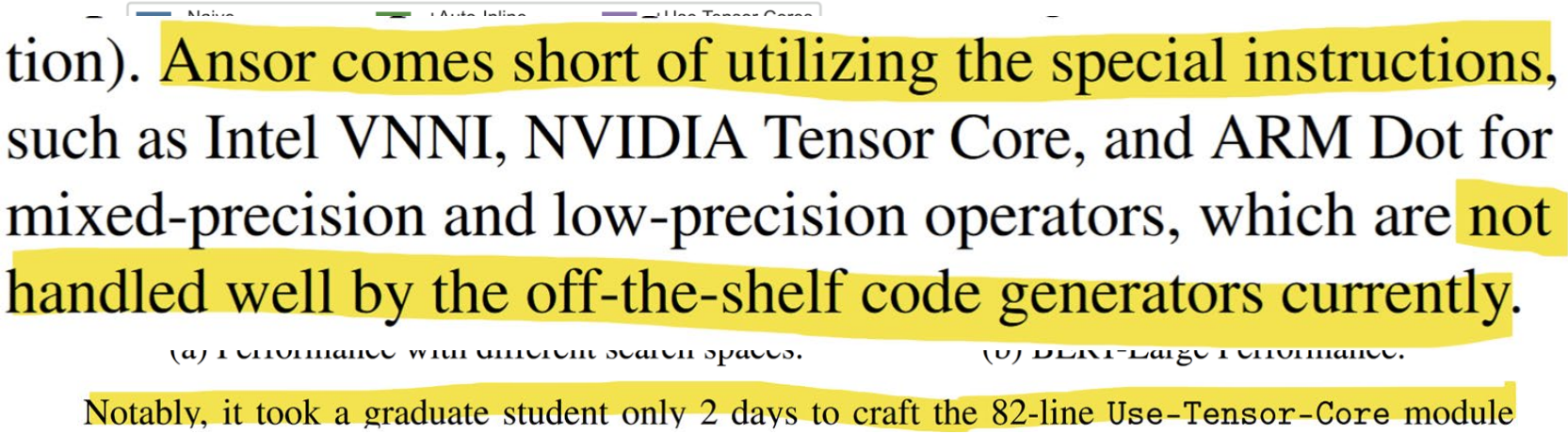
Evaluation and My Thoughts

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



tion). 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.

(a) Performance with different search spaces. (b) BERT-Large Performance.

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

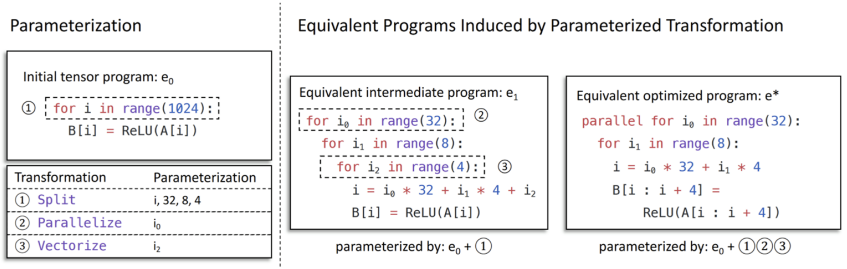
My Thoughts

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 Ansor 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 Ansor can still be reused, but we need to redesign the search space. Lastly,

How Modular?

```
sch.annotate(block_or_loop=b0, ann_key="tiling_structure", ann_val="SSSRRSRS")
```

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

A.3 Use-Tensor-Core Search Space Definition Code

```
b0 = sch.get_block(name="T_dense", func_name="main")
b1 = sch.get_block(name="T_add", func_name="main")
b2 = sch.get_block(name="T_multiply", func_name="main")
b3 = sch.get_block(name="T_cast", func_name="main")
b4 = sch.get_block(name="T_erf", func_name="main")
b5 = sch.get_block(name="T_cast_1", func_name="main")
b6 = sch.get_block(name="T_multiply_1", func_name="main")
b7 = sch.get_block(name="T_add_1", func_name="main")
b8 = sch.get_block(name="T_multiply_2", func_name="main")
b9 = sch.get_block(name="root", func_name="main")
sch.annotate(block_or_loop=b0, ann_key="tiling_structure", ann_val="SSSRRSRS")
b10 = sch.reindex(block=b0, buffer_index=0, is_write_index=True)
b11 = sch.reindex(block=b0, buffer_index=0, is_write_index=False)
b12 = sch.reindex(block=b0, buffer_index=1, is_write_index=False)
sch.transform_layout(block=b0, buffer_index=1, buffer_index_type=read, index_map=lambda j_l, k_l: (j_l, k_l, ))
sch.transform_layout(block=b0, buffer_index=0, buffer_index_type=read, index_map=lambda i_l, k_l: (i_l, k_l, ))
l13, l14, l15 = sch.get_loops(block=b0)
l16, l17 = sch.split(loop=l15, factors=[64, 16])
l18, l19 = sch.split(loop=l14, factors=[256, 16])
l20, l21 = sch.split(loop=l13, factors=[64, 16])
l22, l23, l24, l25, l26, l27 = sch.get_loops(block=b0)
sch.reorder(l24, l26, l21, l19, l17)
b28 = sch.blockize(loop=l21)
sch.annotate(block_or_loop=b0, ann_key="auto_tensorize", ann_val="wmma_sync")
sch.annotate(block_or_loop=b28, ann_key="auto_tensorize", ann_val="wmma_fill")
b29 = sch.get_block(name="root", func_name="main")
sch.annotate(block_or_loop=b29, ann_key="tensor_core_enabled", ann_val="1")
b30 = sch.get_block(name="root", func_name="main")
sch.annotate(block_or_loop=b30, ann_key="warp_execution", ann_val=1)
l31, l32, l33 = sch.get_loops(block=b28)
v34, v35, v36, v37, v38 = sch.sample_perfect_tile(loop=l31, n=5, max_innermost_factor=4, decision=[8, 1, 2, 2, 2])
l39, l40, l41, l42, l43 = sch.split(loop=l31, factors=[v34, v35, v36, v37, v38])
v44, v45, v46, v47, v48 = sch.sample_perfect_tile(loop=l32, n=5, max_innermost_factor=4, decision=[1, 32, 4, 2, 1])
l49, l50, l51, l52, l53 = sch.split(loop=l32, factors=[v44, v45, v46, v47, v48])
v54, v55, v56 = sch.sample_perfect_tile(loop=l33, n=3, max_innermost_factor=4, decision=[32, 2, 1])
l57, l58, l59 = sch.split(loop=l33, factors=[v54, v55, v56])
sch.reorder(l39, l49, l40, l50, l41, l51, l57, l58, l42, l52, l59, l43, l53)
l60 = sch.fuse(l39, l49)
sch.bind(loop=l60, thread_axis="blockIdx.x")
l61 = sch.fuse(l40, l50)
sch.bind(loop=l61, thread_axis="blockIdx.y")
l62 = sch.fuse(l41, l51)
sch.bind(loop=l62, thread_axis="threadIdx.y")
sch.annotate(block_or_loop=b28, ann_key="thread_extent_low_inclusive", ann_val=32)
sch.annotate(block_or_loop=b28, ann_key="thread_extent_high_inclusive", ann_val=1024)
b63 = sch.write_at(loop=l62, block=b28, write_buffer_index=0, storage_scope="wmma.accumulator")
sch.reverse_compute_inline(block=b10)
v64 = sch.sample_categorical(candidates=[4, 8, 16], probs=[0.0.33, 0.0.33, 0.0.33], decision=0)
sch.annotate(block_or_loop=b63, ann_key="vector_bytes", ann_val=v64)
b65 = sch.read_at(loop=l57, block=b28, read_buffer_index=0, storage_scope="shared.dyn")
v66 = sch.sample_categorical(candidates=[4, 8, 16], probs=[0.0.33, 0.0.33, 0.0.33], decision=2)
sch.annotate(block_or_loop=b65, ann_key="vector_bytes", ann_val=v66)
sch.annotate(block_or_loop=b65, ann_key="local_stage", ann_val=1)
sch.annotate(block_or_loop=b65, ann_key="double_buffer_scope", ann_val=0)
b67 = sch.read_at(loop=l57, block=b28, read_buffer_index=1, storage_scope="shared.dyn")
v68 = sch.sample_categorical(candidates=[4, 8, 16], probs=[0.0.33, 0.0.33, 0.0.33], decision=2)
sch.annotate(block_or_loop=b67, ann_key="vector_bytes", ann_val=v68)
sch.annotate(block_or_loop=b67, ann_key="local_stage", ann_val=1)
sch.annotate(block_or_loop=b67, ann_key="double_buffer_scope", ann_val=0)
b69 = sch.read_at(loop=l58, block=b28, read_buffer_index=0, storage_scope="wmma.matrix_a")
b70 = sch.read_at(loop=l58, block=b28, read_buffer_index=1, storage_scope="wmma.matrix_b")
sch.compute_inline(block=b11)
sch.compute_inline(block=b12)
sch.annotate(block_or_loop=l58, ann_key="software_pipeline_stage", ann_val=[0, 0, 1])
sch.annotate(block_or_loop=l58, ann_key="software_pipeline_order", ann_val=[0, 1, 2])
sch.annotate(block_or_loop=l57, ann_key="software_pipeline_stage", ann_val=[0, 0, 0, 0, 0, 1, 1])
sch.annotate(block_or_loop=l57, ann_key="software_pipeline_order", ann_val=[0, 3, 1, 4, 5, 2, 6])
sch.compute_inline(block=b7)
sch.compute_inline(block=b6)
sch.compute_inline(block=b5)
sch.compute_inline(block=b4)
sch.compute_inline(block=b3)
sch.compute_inline(block=b2)
sch.compute_inline(block=b1)
sch.reverse_compute_inline(block=b8)
v71 = sch.sample_categorical(candidates=[0, 16, 64, 512, 1024], probs=[0.2, 0.2, 0.2, 0.2, 0.2], decision=3)
sch.annotate(block_or_loop=b9, ann_key="unroll_explicit", ann_val=v71)
```

Questions?