TOWARDS AN OPERATIONAL MODEL OF SCHEDULES IN TENSOR EXPRESSION

Yuan Lin and Yongfeng Gu

2019-12-05

Question: is this a legal schedule?

```
A = tvm.placeholder((M, N), name='A')
B = tvm.compute((M, N), lambda bi, bj : A[bi, bj] + 1, name='B')
C = tvm.compute((M, N), lambda ci, cj : B[ci, cj] * 3, name='C')

s = tvm.create_schedule(C.op)

# step 1
s[B].compute_at(s[C], s[C].op.axis[0])

# step 2
BL = s.cache_read(B, "local", readers=[C])

print(tvm.lower(s, [A, B], simple_mode=True))
```

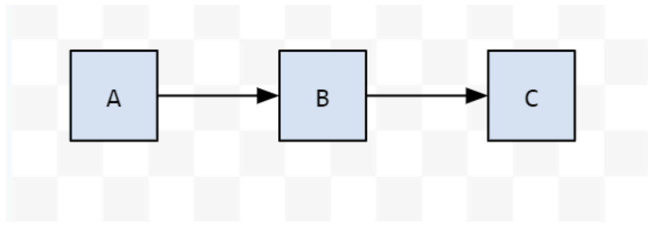
Illegal, but why?

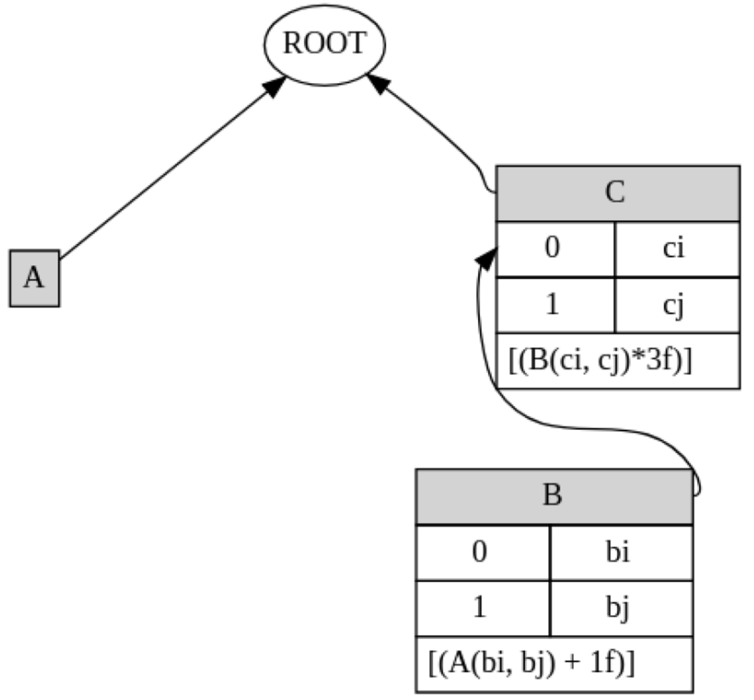
```
A = tvm.placeholder((M, N), name='A')
B = tvm.compute((M, N), lambda bi, bj : A[bi, bj] + 1, name='B')
C = tvm.compute((M, N), lambda ci, cj : B[ci, cj] * 3, name='C')

s = tvm.create_schedule(C.op)

# step 1
s[B].compute_at(s[C], s[C].op.axis[0])

# step 2
BL = s.cache_read(B, "local", readers=[C])

print(tvm.lower(s, [A, B], simple_mode=True))
```

```
TVMError: Check failed:
found_attach || stage_attach.size()
== 0: Invalid Schedule, cannot find
the producer compute(B,
0x2851e60) along the loop nest
specified by compute_at of
consumer compute(B.local,
0x2b83b00)
```

Show the Schedule Tree

```
A = tvm.placeholder((M, N), name='A')
B = tvm.compute((M, N), lambda bi, bj : A[bi, bj] + 1, name='B')
C = tvm.compute((M, N), lambda ci, cj : B[ci, cj] * 3, name='C')

s = tvm.create_schedule(C.op)

# step 1
s[B].compute_at(s[C], s[C].op.axis[0])
```

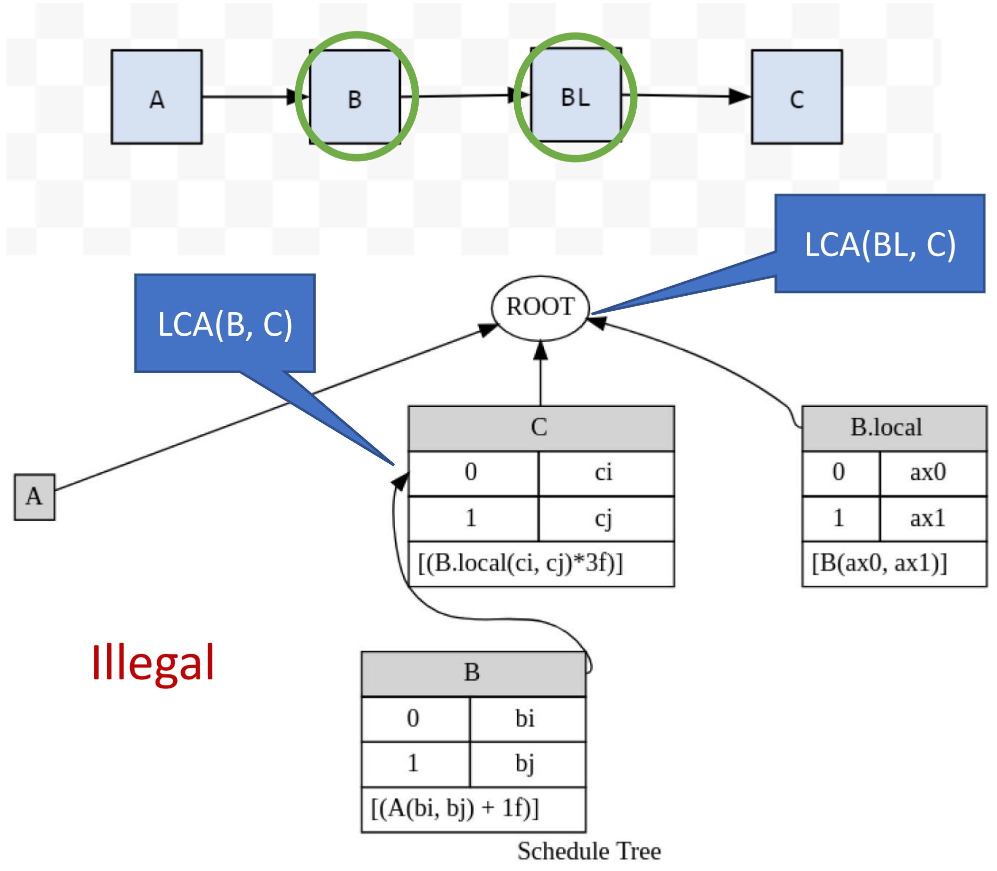




Schedule Tree

Violation of Legality

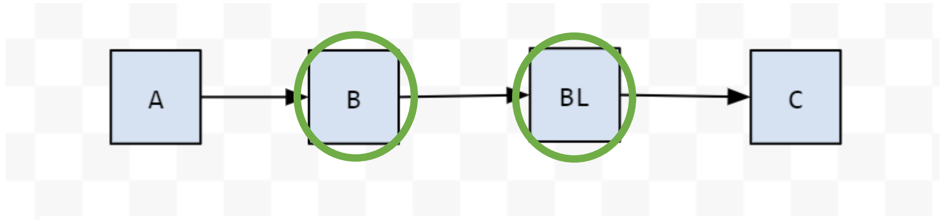
```
A = tvm.placeholder((M, N), name='A')
B = tvm.compute((M, N), lambda bi, bj : A[bi, bj] + 1, name='B')
C = tvm.compute((M, N), lambda ci, cj : B[ci, cj] * 3, name='C')

s = tvm.create_schedule(C.op)

# step 1
s[B].compute_at(s[C], s[C].op.axis[0])

# step 2
BL = s.cache_read(B, "local", readers=[C])
```

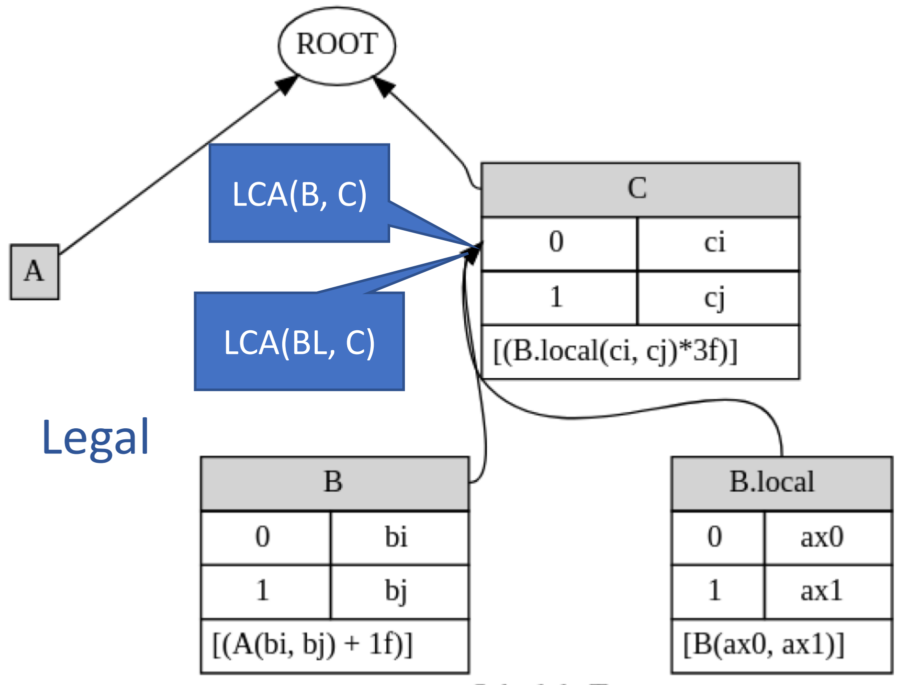


Illegal

A Possible Fix

```
A = tvm.placeholder((M, N), name='A')
B = tvm.compute((M, N), lambda bi, bj : A[bi, bj] + 1, name='B')
C = tvm.compute((M, N), lambda ci, cj : B[ci, cj] * 3, name='C')

s = tvm.create_schedule(C.op)

# step 1
s[B].compute_at(s[C], s[C].op.axis[0])

# step 2
BL = s.cache_read(B, "local", readers=[C])

# step 3 (The above schedule is illegal. This step makes it legal)
s[BL].compute_at(s[C], s[C].op.axis[0])
```





Schedule Tree

Motivations

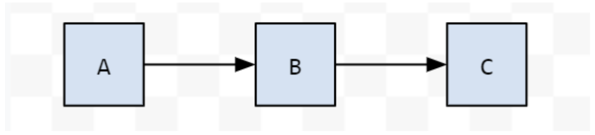
- Describe schedule primitives and transformations in a more elaborate way
- Provide guidance to new TE users
- Help debugging tricky TE programs
- Provide a framework to discuss bugs (is this a bug or feature?)
- Ensure conherent development of new features

The Operational Model

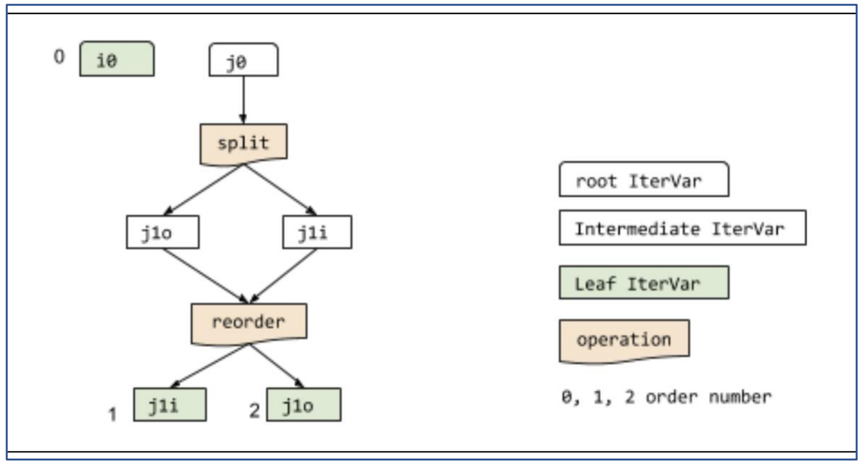
Tensor Expression

```
A = tvm.placeholder((m,n))
B = tvm.compute((m, n), lambda i, j: A[i,j] + 1)
C = tvm.compute((m, n), lambda i, j: B[i,j] * 2)
s = tvm.create_schedule(C.op)
i0, j0 = s[C].op.axis
j1o, j1i = s[C].split(j0, 2)
s[C].reorder(j1i, j1o)
```

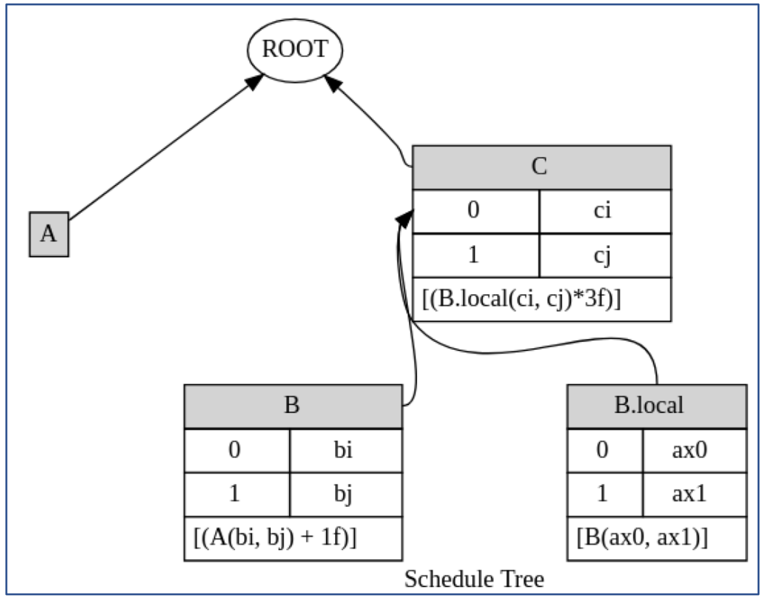
Dataflow Graph



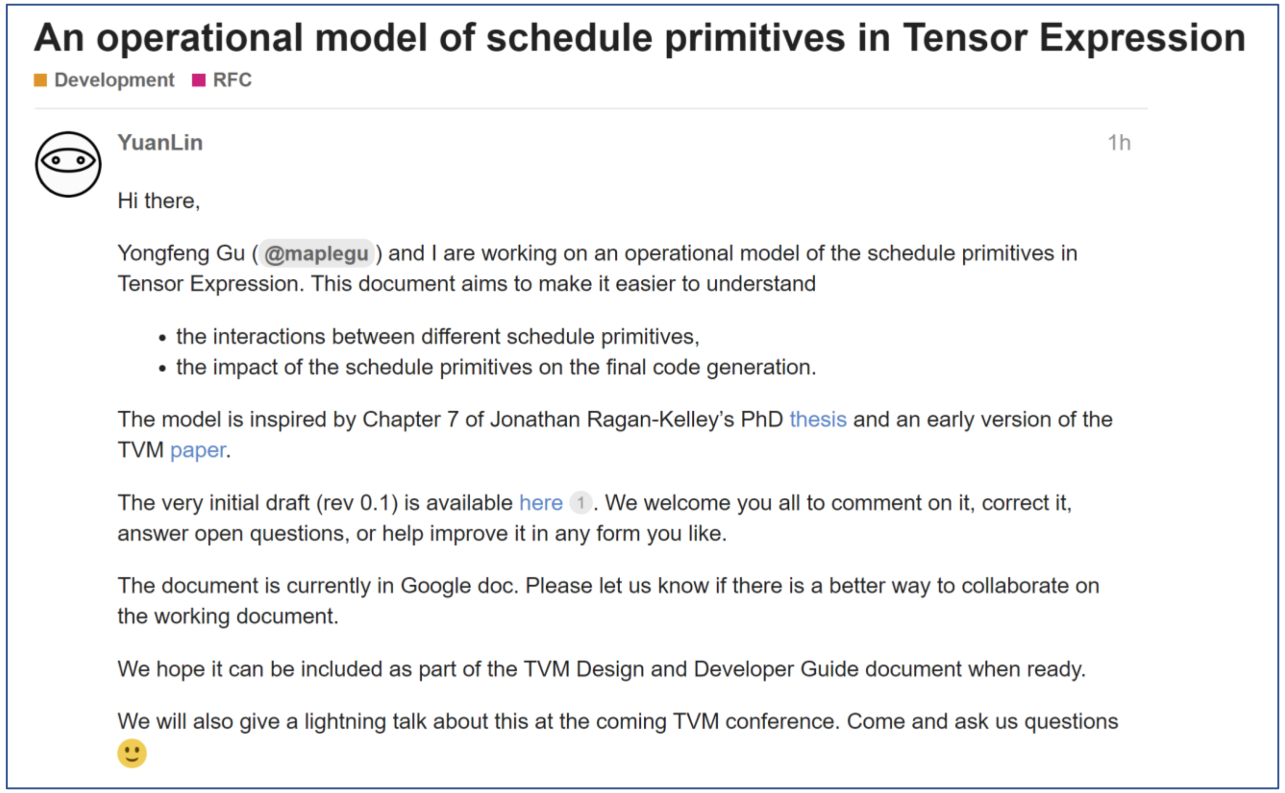
IterVar-relation DAG



Schedule Tree



The Document



An operational model of schedule primitives in Tensor Expression

Development RFC

YuanLin 1h

Hi there,

Yongfeng Gu (@maplegu) and I are working on an operational model of the schedule primitives in Tensor Expression. This document aims to make it easier to understand

- the interactions between different schedule primitives,
- the impact of the schedule primitives on the final code generation.

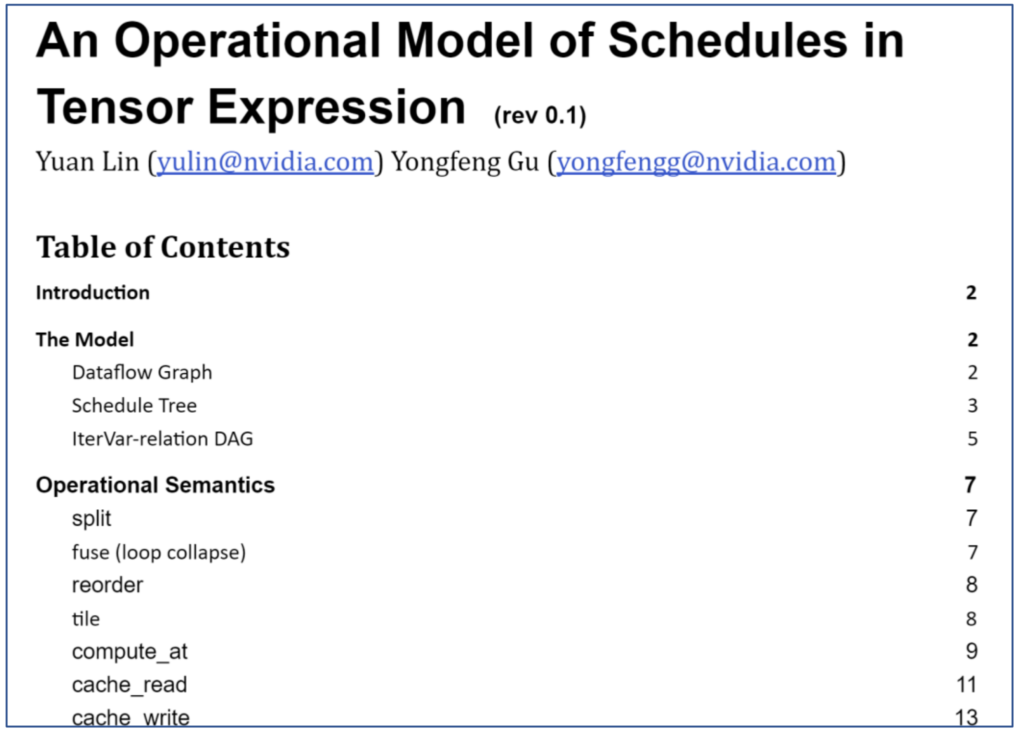
The model is inspired by Chapter 7 of Jonathan Ragan-Kelley's PhD thesis and an early version of the TVM paper.

The very initial draft (rev 0.1) is available here 1. We welcome you all to comment on it, correct it, answer open questions, or help improve it in any form you like.

The document is currently in Google doc. Please let us know if there is a better way to collaborate on the working document.

We hope it can be included as part of the TVM Design and Developer Guide document when ready.

We will also give a lightning talk about this at the coming TVM conference. Come and ask us questions 🙂



An Operational Model of Schedules in Tensor Expression (rev 0.1)

Yuan Lin (yulin@nvidia.com) Yongfeng Gu (yongfengg@nvidia.com)

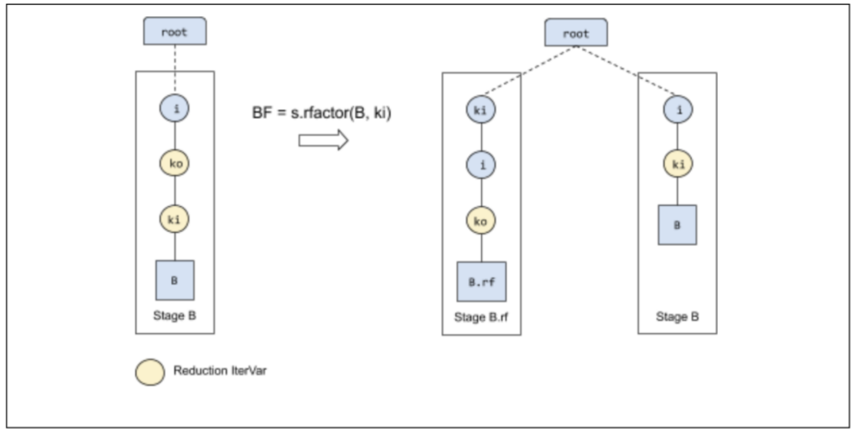
Table of Contents

Example pages

rfactor

`rfactor(tensor, axis, factor_axis=0)` inserts a refactor stage before the given (original) reduction stage for *tensor*. It copies over all the IterVars from the reduction stage, changes the reduction IterVar *axis* specified in the `rfactor` to a regular IterVar, and makes it at the position *factor_axis* (which is the outermost one by default). In the original reduction stage, all related reduction IterVars are removed except for *axis*. The refactor stage computes the partial reduction results, which are further reduced by the updated reduction stage.

```
k = tvm.reduce_axis((0, m), "k")
B = tvm.compute((n,), lambda i: tvm.sum(A[i, k], axis=k), name="B")
...
ko, ki = s[B].split(B.op.reduce_axis[0], factor=16)
BF = s.rfactor(B, ki)
```



Debug Hints	`cache_read` will create new readers. Notice the address of reader B is changed after cache_read. You need to refer to the new compute as s[B].op instead of B.op after the cache_read.

[Example 5]
The following illustrates an example where the use of `compute_at` and `cache_read` can result in an illegal schedule. Step 1 attaches B to C with compute_at. The Schedule Tree is legal as shown on the left in the figure below. In step 2, cache_read inserts a stage BL between B and C in the Dataflow Graph and adds a branch for BL under root in the Schedule Tree as shown in the middle. Although BL depends on B, LCA(BL, C) is above LCA(B, C), making the Schedule Tree illegal as explained in the compute_at section. Finally, step 3 moves the BL branch to the same point the B branch attaching to the C branch to turn the Schedule Tree legal again.

```
A = tvm.placeholder((M, N), name='A')
B = tvm.compute((M, N), lambda bi, bj : A[bi, bj] + 1, name='B')
C = tvm.compute((M, N), lambda ci, cj : B[ci, cj] * 3, name='C')

s = tvm.create_schedule(C.op)

# step 1
s[B].compute_at(s[C], s[C].op.axis[0])

# step 2
BL = s.cache_read(B, "local", readers=[C])

# step 3 (The above schedule is illegal. This step makes it legal).
s[BL].compute_at(s[C], s[C].op.axis[0])
```

Help Needed!

- We would like to contribute the document to the TVM community.
  - We are also working on a visualization tool, Tensor Expression Debug Display (TEDD).
- Questions/Survey:
  - Is this model helpful?
  - Is this or similar model well understood?
  - Will you use it for future discussion?
  - Let's work together to
    - improve the model
    - complete the document

Thank you