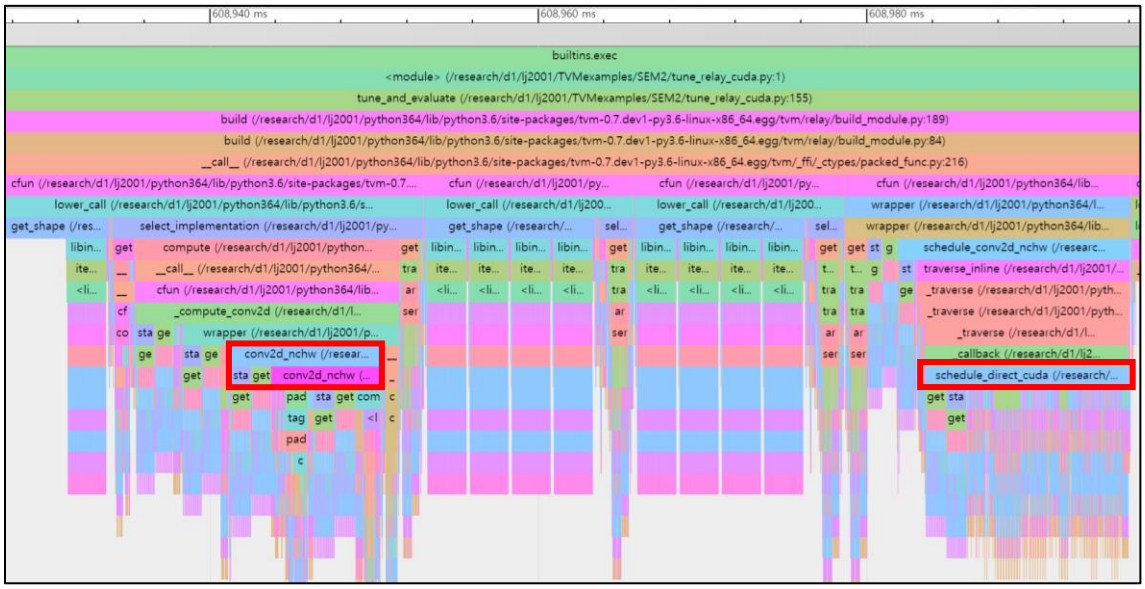
CHOI Jang Hyeon (1155077214)
Supervisors: Professors JIA Jiaya and YU Bei
Mentor: Mr. QI Sun
CSE Department
Term 2, 2020-21
April 12th, 2021

FYP Progress Report: Week 14

Using the open-source software, Viztracer, I was able to visualize the tuning process. I could easily determine which process was being run at what method or Python file by clicking on the block I am interested in.

Zooming into the “build” process, we can see how tensor values are handled in conv2d_nchw.py first and then scheduled to schedule_direct_cuda.py, which gave me a better understanding of how input data was treated.



To increase the configuration space, I took a glance at the properties for each of the 24 tasks:

```
Tuning...
config space:  ConfigSpace (len=384384, space_map=
   0 tile_f: Split(policy=factors, product=2048, num_outputs=4) len=364
   1 tile_y: Split(policy=factors, product=7, num_outputs=4) len=4
   2 tile_x: Split(policy=factors, product=7, num_outputs=4) len=4
   3 tile_rc: Split(policy=factors, product=1024, num_outputs=2) len=11
   4 tile_ry: Split(policy=factors, product=1, num_outputs=2) len=1
   5 tile_rx: Split(policy=factors, product=1, num_outputs=2) len=1
   6 auto_unroll_max_step: OtherOption([0, 512, 1500]) len=3
   7 unroll_explicit: OtherOption([0, 1]) len=2
)
[Task  1/24]  Current/Best:    0.00/   0.00 GFLOPS | Progress: (0/1500) | 0.00 s
```

There are a total of 8 parameters that were being considered for tuning under the NCHW layout. However, if we look at the “auto_unroll_max_step” parameter, it seems to be hardcoded. After some research, I found out that this parameter does not suit every hardware and needs to be modified through trial-and-error. Subsequently, I included additional values into this parameter (in “conv2d_direct.py”), which has increased the configuration space from 384384 to 512512:

```
Tuning...
config space from tune_relay_cuda:  ConfigSpace (len=512512, space_map=
   0 tile_f: Split(policy=factors, product=2048, num_outputs=4) len=364
   1 tile_y: Split(policy=factors, product=7, num_outputs=4) len=4
   2 tile_x: Split(policy=factors, product=7, num_outputs=4) len=4
   3 tile_rc: Split(policy=factors, product=1024, num_outputs=2) len=11
   4 tile_ry: Split(policy=factors, product=1, num_outputs=2) len=1
   5 tile_rx: Split(policy=factors, product=1, num_outputs=2) len=1
   6 auto_unroll_max_step: OtherOption([0, 512, 1024, 1500]) len=4
   7 unroll_explicit: OtherOption([0, 1]) len=2
)
^M[Task  1/24]  Current/Best:    0.00/   0.00 GFLOPS | Progress: (0/500) | 0.00 s
```

I could have included more values, but I was not sure how much additional training time was needed. Given the 24-hour limit on the GPUs that were compatible with my project, I first need to find out how much more time is required for one additional value and then decide whether to add in more values afterwards.

As mentioned before, the tiling outputs for NHWC only handle certain values and is difficult to change the parameters:

```
##### space definition begin #####
n, f, y, x = s[conv].op.axis
rc, ry, rx = s[conv].op.reduce_axis
cfg.define_split("tile_f", f, num_outputs=4)
cfg.define_split("tile_y", y, num_outputs=4)
cfg.define_split("tile_x", x, num_outputs=4)
cfg.define_split("tile_rc", rc, num_outputs=2)
cfg.define_split("tile_ry", ry, num_outputs=2)
cfg.define_split("tile_rx", rx, num_outputs=2)
cfg.define_knob("auto_unroll_max_step", [0, 512, 1500])
```

```
# tile and bind spatial axes
n, f, y, x = s[output].op.axis
kernel_scope, n = s[output].split(n, nparts=1)

bf, vf, tf, fi = cfg["tile_f"].apply(s, output, f)
by, vy, ty, yi = cfg["tile_y"].apply(s, output, y)
bx, vx, tx, xi = cfg["tile_x"].apply(s, output, x)
```

(NCHW scheduling parameters)

On the other hand, there is more freedom is tinkering with NHWC, as it uses the “define_knob” method instead:

```
# Schedule for autotvm
cfg.define_knob("tile_n", [2, 4, 8])
cfg.define_knob("tile_c", [2, 4, 8])
cfg.define_knob("num_thread_n", [4, 8, 16])
cfg.define_knob("num_thread_c", [4, 8, 16])
cfg.define_knob("vthread_n", [1, 2])
cfg.define_knob("vthread_c", [1, 2])
cfg.define_knob("step", [16, 3, 32, 64])
```

(NHWC schedule)

After adding a value into the “tile_n” parameter, I could verify that its configuration space increased from 1296 to 1728:

```
Tuning...
config space:  ConfigSpace (len=1296, space_map=
   0 tile_n: OtherOption([2, 4, 8]) len=3
   1 tile_c: OtherOption([2, 4, 8]) len=3
   2 num_thread_n: OtherOption([4, 8, 16]) len=3
   3 num_thread_c: OtherOption([4, 8, 16]) len=3
   4 vthread_n: OtherOption([1, 2]) len=2
   5 vthread_c: OtherOption([1, 2]) len=2
   6 step: OtherOption([16, 3, 32, 64]) len=4
)
```

```
Tuning...
config space from tune_relay_cuda:  ConfigSpace (len=1728, space_map=
   0 tile_n: OtherOption([2, 4, 8, 16]) len=4
   1 tile_c: OtherOption([2, 4, 8]) len=3
   2 num_thread_n: OtherOption([4, 8, 16]) len=3
   3 num_thread_c: OtherOption([4, 8, 16]) len=3
   4 vthread_n: OtherOption([1, 2]) len=2
   5 vthread_c: OtherOption([1, 2]) len=2
   6 step: OtherOption([16, 3, 32, 64]) len=4
```

This week, I am planning to compare the performance differences after changing such parameters. If time allows, I could experiment with the threading by changing block factors and improve the overall performance of AutoTVM.