Writeup

Xavier Gonzalez

xavier18

Part 1

Step 1

1. Execution time in micro seconds of `python run_benchmark.py --kernel tiled -n 25600` was 37318 µs.

```
(aws_neuronx_venv_pytorch_2_8) ubuntu@ip-172-31-33-95:~/asst4-
trainium2/part1$ python run_benchmark.py --kernel tiled -n 25600

Running vector_add_tiled with shape (25600,)

Correctness passed? True

Benchmarking performance.........

file
+---+----+---------+---------+---------+-------+--------+-------+-------+--
-----+--------+---------+---------+-------+
  B   NC   NC USED   WEIGHTS   MODE      INF/S   IRES/S   L(1)    L(50)
L(99)   NCL(1)   NCL(50)   NCL(99)   %USER
  1   1    1         dynamic   LIBMODE   26.71   26.71    37342   37388
37394   37298    37308     37318     N/A
+---+----+---------+---------+---------+-------+--------+-------+-------+--
-----+--------+---------+---------+-------+
```

2. Now with `ROW_CHUNK=128`, the execution time is 378 µs. Thus, the code is close to 100x faster.

The code is written as a loop, where `ROW_CHUNK` entries are loaded and processes in parallel. Since the NC has capacity for 128 partitions, it makes sense that if we use all 128 partitions we get around a 100x speed up.

```
(aws_neuronx_venv_pytorch_2_8) ubuntu@ip-172-31-33-95:~/asst4-
trainium2/part1$ python run_benchmark.py --kernel tiled -n 25600

Running vector_add_tiled with shape (25600,)

Correctness passed? True

Benchmarking performance.........

file
```

```
+---+----+---------+---------+---------+---------+---------+------+-------
+-------+--------+---------+---------+-------+
  B   NC   NC USED   WEIGHTS   MODE      INF/S     IRES/S    L(1)   L(50)
L(99)   NCL(1)   NCL(50)   NCL(99)   %USER
  1   1    1         dynamic   LIBMODE   1992.91   1992.91   435    439
457     378      378       378       N/A
+---+----+---------+---------+---------+---------+---------+------+-------
+-------+--------+---------+---------+-------+
```

3.

With `ROW_CHUNK=256`, there is an error because the partition dimension aka tensor shape 0 can be max 128, but in the code it's set to be `ROW_CHUNK`, leading to error if `ROW_CHUNK > 128`.

```
(aws_neuronx_venv_pytorch_2_8) ubuntu@ip-172-31-33-95:~/asst4-
trainium2/part1$ python run_benchmark.py --kernel tiled -n 25600

Running vector_add_tiled with shape (25600,)
Traceback (most recent call last):
  File "/home/ubuntu/asst4-trainium2/part1/run_benchmark.py", line 106, in
<module>
    main()
  File "/home/ubuntu/asst4-trainium2/part1/run_benchmark.py", line 102, in
main
    benchmark_kernel(kernel, *kernel_args, profile_name=args.profile_name)
  File "/home/ubuntu/asst4-trainium2/part1/run_benchmark.py", line 51, in
benchmark_kernel
    out = nki.baremetal(kernel)(*args)
  File "neuronxcc/nki/compiler/backends/neuron/TraceKernel.py", line 235,
in neuronxcc.nki.compiler.backends.neuron.TraceKernel.Kernel.__call__
  File "neuronxcc/nki/compiler/backends/neuron/TraceKernel.py", line 236,
in neuronxcc.nki.compiler.backends.neuron.TraceKernel.Kernel.__call__
  File "neuronxcc/nki/compiler/backends/neuron/TraceKernel.py", line 322,
in neuronxcc.nki.compiler.backends.neuron.TraceKernel.TraceKernel.call_impl
  File "neuronxcc/nki/compiler/backends/neuron/TraceKernel.py", line 337,
in
neuronxcc.nki.compiler.backends.neuron.TraceKernel.TraceKernel.specialize_a
nd_call
  File "neuronxcc/nki/compiler/backends/neuron/TraceKernel.py", line 339,
in
neuronxcc.nki.compiler.backends.neuron.TraceKernel.TraceKernel.specialize_a
nd_call
  File "neuronxcc/nki/compiler/backends/neuron/TraceKernel.py", line 347,
in
neuronxcc.nki.compiler.backends.neuron.TraceKernel.TraceKernel.expand_kerne
l_with_ctx
  File "neuronxcc/nki/compiler/backends/neuron/TraceKernel.py", line 367,
in
neuronxcc.nki.compiler.backends.neuron.TraceKernel.TraceKernel.expand_kerne
l_with_ctx
  File "neuronxcc/nki/compiler/backends/neuron/TraceKernel.py", line 352,
```

```
in
neuronxcc.nki.compiler.backends.neuron.TraceKernel.TraceKernel.expand_kerne
l_with_ctx
  File "/home/ubuntu/asst4-trainium2/part1/kernels.py", line 76, in
vector_add_tiled
    a_tile = nl.ndarray((ROW_CHUNK, 1), dtype=a_vec.dtype, buffer=nl.sbuf)
ValueError: number of partitions 256 exceed architecture limitation of 128.
Info on how to fix: https://awsdocs-neuron.readthedocs-
hosted.com/en/latest/general/nki/api/nki.errors.html#err-num-partition-
exceed-arch-limit
```

Step 2a

1. 186 µs. This is around 2x faster.

```
Running vector_add_stream with shape (25600,)

Correctness passed? True

Benchmarking performance.........

file
+---+----+---------+---------+---------+---------+---------+------+-------
+-------+--------+---------+---------+-------+
  B   NC   NC USED   WEIGHTS   MODE      INF/S     IRES/S    L(1)   L(50)
L(99)   NCL(1)   NCL(50)   NCL(99)   %USER
  1   1    1         dynamic   LIBMODE   2957.50   2957.50   270    272
297     186      186       186       N/A
+---+----+---------+---------+---------+---------+---------+------+-------
+-------+--------+---------+---------+-------+
```

2. I chose `FREE_DIM = 200` becasue our length is 25600 and we are using a partition size of 128.

This ran in 15 µs. This is around 10x faster than `FREE_DIM=2` and around `20x` faster than using no free dim at all.

Step 2b

2. When `FREE_DIM=2000`, the kernel execution time was 2.8e-5 seconds, and the dma transfer count was 3.

When `FREE_DIM=1000`, the kernel execution time was 2.6e-5 and the dma transfer count was 6.

4. When `FREE_DIM=2000`, first all of the loads happen, and then the addition occurs.This is because the `FREE_DIM` is large enough to load the entire length of the vectors in.

When `FREE_DIM=1000`, there are two waves of loads (as the size is half as large). A is in dark green, while B is in a blue gray. But, the result of this is that the arithmetic (in the lime green in vector E) can also happen in two waves. Importantly, the first half of the arithmetic can happen while the second vector a is loading! (this is

pipeling). thus, there is less arithmetic to do after the loads have happened, in this setting yielding wall clock speed up.

Step 3: Matrix Transpose

1. Execution Time: 86 µs
2. I think my kernel is memory-bound because all it involves is moving elements from one part of the matrix to the other (it bascially has an arithmetic intensity of zero).

Maybe technically the tranpose counts as compute, but I'm still doing so many reads and writes that I think it's memory bound.

Looking at the profiler, I believe it's still memory bound.

So far as I can tell, there are 3 important ops in the profiler:

1. copies from hbm to sbuf (in red, in DMA-E79)
2. computations (transposes) in TensorMatrixE (yellow)
3. copies from sbuf to hbm (in green, in DMA-E79)

The yellow compute is in the middle of the two memory ops, which are effectively contiguous, starting before the compute and ending after the compute. Based on this plot, it appears that if I made the compute faster, the overall time wouldn't change much, because the memory ops dominate. But, if I made bandwidth faster, the overall time would go down.

Part 2: Fused Convolution

Brief description

I tiled over both the out_channels and the in_channels. I wanted to load in large blocks, but without tiling I ran into the difficulty that the partition dimension can be max 128, so I broke up the in_channels and out_channels into tiles of size 128.

Furthermore, for max pooling, I loaded in pool_size number of rows at time.

I then computed the convolution (via a matrix multiply) in a tight inner loop, where I looped for filter_height and filter_width, accumulating the sum in PSUM. This is how I built up the convolved output, row by row. When I had finished this summation, I wrote from PSUM to SBUF.

Finally, in SBUF, I performed max pooling. Only after maxpooling did I add in the bias. Finally, I wrote to the output tensor (on HBM).

Optimization

I experimented with trying to load different numbers of rows for the maxpools, trying much larger nubmers of rows at a time. However, I found that if I tried to maxpool over a large number of rows, there was actually a period when the tensor engine was very idle, damaging performance. Thus, I settled on loading in pool_size rows at a time for the maxpooling.

The biggest optimization by far was moving ops higher in the loop structure so they weren't acted on redudnantly; and the biggest in this regard by far was moving the loading of the entire weight matrix into

HBUF, and doing the transpositions outside of the main loop. This saved many many computer, and in general allowed for me to do mat muls without any transpositions getting in the way and slowing down computation.

Achieved MFU

- (no maxpool, FP32): 41%
- (no maxpool, FP16): 90%
- (maxpool, FP32): 41%
- (maxpool, FP16): 90%

Outputs

```
(aws_neuronx_venv_pytorch_2_8) ubuntu@ip-172-31-47-171:~/asst4-
trainium2/part2$ python test_harness.py --test_maxpool --profile done

Running correctness test for conv2d kernel with smaller image... Passed 😎

Running correctness test for conv2d kernel with larger image... Passed 😎

Running correctness test for conv2d kernel with larger image + bias...
Passed 😎

Running correctness test for conv2d kernel with larger image + bias +
maxpool... Passed 😎

Comparing performance with reference kernel (no maxpool, float32)...
done_float32
+---+----+---------+---------+---------+-------+--------+------+-------+---
----+--------+---------+---------+-------+
  B   NC   NC USED   WEIGHTS   MODE      INF/S   IRES/S   L(1)   L(50)
L(99)   NCL(1)   NCL(50)   NCL(99)   %USER
  1   1    1         dynamic   LIBMODE   86.62   86.62    3891   3933
3939    3831     3832      3832      N/A
+---+----+---------+---------+---------+-------+--------+------+-------+---
----+--------+---------+---------+-------+


Execution Time for student implementation: 3832 µs
Performance test passed 😍

NEFF / NTFF files generated with names: done_float32.neff,
done_float32.ntff

Comparing performance with reference kernel (no maxpool, float16)...
done_float16
+---+----+---------+---------+---------+--------+--------+------+-------+--
-----+--------+---------+---------+-------+
  B   NC   NC USED   WEIGHTS   MODE      INF/S    IRES/S   L(1)   L(50)
```

```
L(99)   NCL(1)   NCL(50)   NCL(99)   %USER
  1   1    1         dynamic   LIBMODE   208.75   208.75   893    907
918     823      824       824       N/A
+---+----+---------+---------+---------+--------+--------+------+-------+--
-----+--------+---------+---------+-------+


Execution Time for student implementation: 824 µs
Performance test passed 😍

NEFF / NTFF files generated with names: done_float16.neff,
done_float16.ntff

Comparing performance with reference kernel (with maxpool, float32)...
done_pool_float32
+---+----+---------+---------+---------+--------+--------+------+-------+--
-----+--------+---------+---------+-------+
  B   NC   NC USED   WEIGHTS   MODE      INF/S    IRES/S   L(1)   L(50)
L(99)   NCL(1)   NCL(50)   NCL(99)   %USER
  1   1    1         dynamic   LIBMODE   113.52   113.52   3879   3886
3936    3828     3829      3830      N/A
+---+----+---------+---------+---------+--------+--------+------+-------+--
-----+--------+---------+---------+-------+


Execution Time for student implementation: 3830 µs
Performance test passed 😍

NEFF / NTFF files generated with names: done_pool_float32.neff,
done_pool_float32.ntff

Comparing performance with reference kernel (with maxpool, float16)...
done_pool_float16
+---+----+---------+---------+---------+--------+--------+------+-------+--
-----+--------+---------+---------+-------+
  B   NC   NC USED   WEIGHTS   MODE      INF/S    IRES/S   L(1)   L(50)
L(99)   NCL(1)   NCL(50)   NCL(99)   %USER
  1   1    1         dynamic   LIBMODE   290.48   290.48   892    913
924     823      824       825       N/A
+---+----+---------+---------+---------+--------+--------+------+-------+--
-----+--------+---------+---------+-------+


Execution Time for student implementation: 825 µs
Performance test passed 😍

NEFF / NTFF files generated with names: done_pool_float16.neff,
done_pool_float16.ntff
Your final score is :  60.0 / 60.0
Correctness: 10.0       Total obtainable: 10.0
Performance: 50.0       Total obtainable: 50.0
```