CPSC 418 - HW5

Tristan Rice, q7w9a, 25886145

2.

a)

It does one pass with the horizontal, and then one pass with the vertical.

The horizontal kernel does $11 + 9s_x$. The kernel gets called roughly $h * w$ times. Thus, total number of operations is $hw(11 + 9s_x)$.

For the vertical pass, it's similar but uses s_y instead.

Thus, the total is $hw(22 + 9s_x + 9s_y)$.

b)

There's $s_x + 1$ memory accesses per invocation of the kernel for horizontal convolution, and $s_y + 2$ accesses for the vertical.

Thus, total is $hw(s_x + s_y + 4)$ memory accesses.

c)

The compute to global memory access ratio is $\frac{22+9s_x+9s_y}{s_x+s_y+4}$. This is pretty good since there are a lot more computations than memory accesses.

d)

```
Run conv1to2_basic on a 1071x1850 image for 1000 rounds. Sigma is 3.000000.
conv1to2_basic          Total =  1.112e+01 s; Average =  1.112e-02 s.
```

$1071 * 1850/(1.112e-02s) = 1.782 * 10^8$ pixels per second.

3.

b)

Number of memory accesses for horizontal convolution is $\frac{h*w}{t} * (s_x + t)$. Vertical is $\frac{h*w}{t} * (s_y + t)$.

Total number of memory accesses is $\frac{h*w*(s_x+s_y+2t)}{t}$

Number of multiplications is roughly $h * w * (s_x + s_y)$.

Thus, computations to memory accesses is $\frac{(s_x+s_y)*t}{s_x+s_y+2t}$.

c)

```
Run conv1to2_tiled on a 1071x1850 image for 1000 rounds. Sigma is 3.000000.
conv1to2_tiled          Total =  7.436e+00 s; Average =  7.436e-03 s.
```

$1071 * 1850/(1.112e-02s) = 2.665 * 10^8$ pixels per second.

The larger the tile size is, the less overlapping data has to be loaded. However, if you make the tile size too large, you start losing benefits since each block has to do too much work.

1D tiles are also easier to implement, but you lose out on cache locality performance in the y axis. Thus, this could be more performant by making the vertical convolutions use 2D tiles.

4.

a)

1. cublasSgemv
2. cublasSnrm2
3. cublasSscal

b)

The solution is already provided.

c)

```
$ time ./hw5_blas 1024 matrix/power_A_1.500_2048.data matrix/power_b0_1.500_2048.data 500

Power iteration parameters:
  n = 1024
  filename_A = matrix/power_A_1.500_2048.data
  filename_b0 = matrix/power_b0_1.500_2048.data
  iterations = 500

After 500 iterations, magnitude of largest eigenvalue: 0.741
./hw5_blas 1024 matrix/power_A_1.500_2048.data matrix/power_b0_1.500_2048.dat  0.34s user 0.11s system 95
% cpu 0.477 total
```

It took 0.477 seconds to calculate.

d)

$$0.477s * 98.2GBps/4bytes = 11710350000$$

11 710 350 000 single precision floats could be loaded from main memory in 0.477 seconds.

e)

Matrix multiplication is O(nmp), where one matrix is $n * m$, the other $m * p$. Thus, the multiply takes n^2.

Taking the absolute value of a n vector takes n operations.

Pairwise division of two n sized vectors takes n operations.

Thus, total number of operations is $n^2 + 2n$.

f)

Number of operations $= 2048^2 + 2 * 2048 = 4198400$.

Compute per global memory access is bounded by $\frac{4198400 operations}{11710350000 accesses}$. This is a very low bound, likely caused by the `time` command including other delays such as the time it takes to load the program/matrices into main memory.