HW5: Tiled Matrix Multiplication

The data given below can be found in “data” sub directory.

1. CUDA profile for block of 16 x 16 threads:

   ```
   # CUDA_PROFILE_LOG_VERSION 2.0
   # CUDA_DEVICE 0 GeForce GTX 480
   # CUDA_CONTEXT 1
   # TIMESTAMPFACTOR 1329e4194f13ab2c
   method,gputime,cputime,occupancy
   method=[ memcpyHtoD ] gputime=[ 4232.000 ] cputime=[ 4398.832 ]
   method=[ memcpyHtoD ] gputime=[ 2489.824 ] cputime=[ 2568.545 ]
   method=[ memcpyHtoD ] gputime=[ 3724.160 ] cputime=[ 3791.492 ]
   method=[ _Z15MatrixMulKernel6MatrixS_S_ ] gputime=[ 70208.898 ] cputime=[ 10.771 ]
   occupancy=[ 1.000 ]
   method=[ memcpyDtoH ] gputime=[ 6414.688 ] cputime=[ 77336.578 ]
   ```

2. CUDA profile for block of 32 x 32 threads:

   ```
   # CUDA_PROFILE_LOG_VERSION 2.0
   # CUDA_DEVICE 0 GeForce GTX 480
   # CUDA_CONTEXT 1
   # TIMESTAMPFACTOR 1329e4194ebfccc6
   method,gputime,cputime,occupancy
   method=[ memcpyHtoD ] gputime=[ 5857.152 ] cputime=[ 6138.350 ]
   method=[ memcpyHtoD ] gputime=[ 3548.160 ] cputime=[ 3813.570 ]
   method=[ memcpyHtoD ] gputime=[ 5291.840 ] cputime=[ 5575.102 ]
   method=[ _Z15MatrixMulKernel6MatrixS_S_ ] gputime=[ 80615.297 ] cputime=[ 11.494 ]
   occupancy=[ 0.667 ]
   method=[ memcpyDtoH ] gputime=[ 6032.064 ] cputime=[ 86973.961 ]
   ```

3. From the profile, we can see that code with block of 16 x 16 threads performs better in terms of GPU time of kernel. This is because, each SM can run 1536 thread, with 6 blocks of 256 threads each. With block of 32 x 32 threads, the occupancy is only 1024 / 1536 = 0.667.

   Another area where timings differ is the no. of global memory access each block makes. With smaller block, size of shared memory is also small. So proportionately more no. of global memory reads are done by subsequent blocks. Large block make smaller no. of global memory reads.

4. Timings for matrices of dimensions 4096 x4096 (in data/4096.log)

   ```
   GPU Time:
   Exclusive = 861.300659 ms
   Inclusive = 942.288757 ms
   CPU computation complete in 2032829.750000 ms
   Test PASSED
   Dimension M[height,width]: 4096  4096
   Dimension N[height,width]: 4096  4096
   ```

5. From trial and error, I observed that my code is failing for square matrices of dimensions 11500 elements. So I ran the program for size 11000 x 11000. Here is what I found:

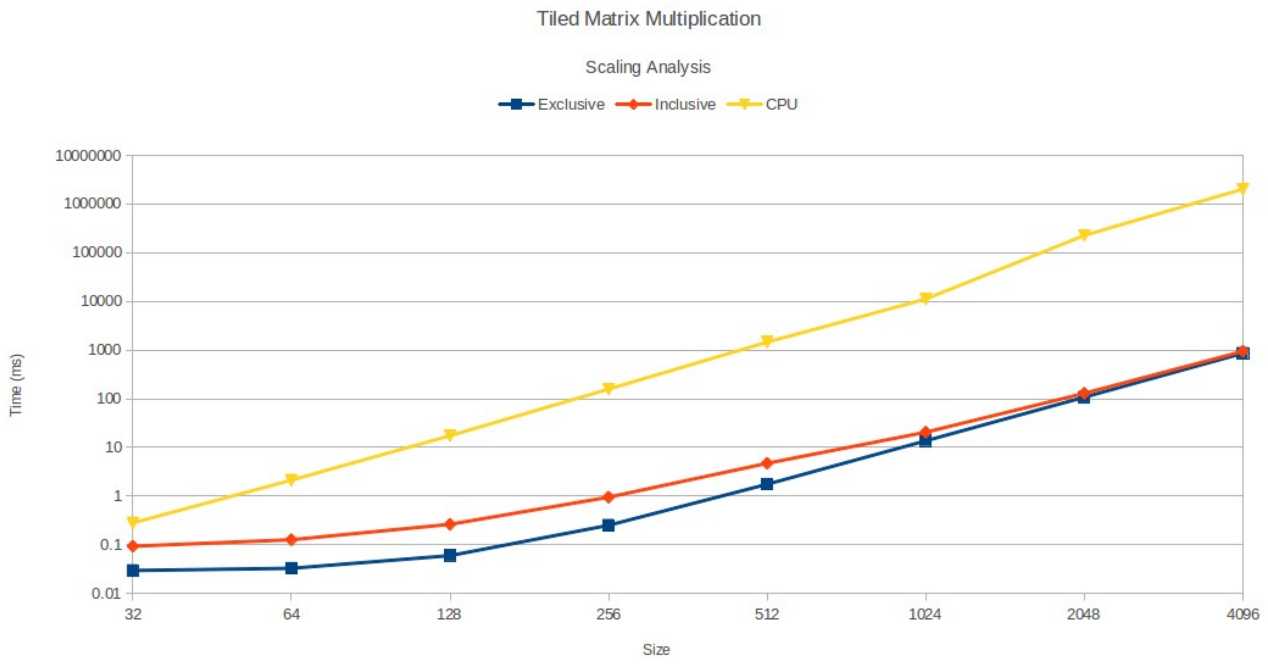
   CUDA Profile (cuda_profile_0.log in root directory):

```
# CUDA_PROFILE_LOG_VERSION 2.0
# CUDA_DEVICE 0 GeForce GTX 480
# CUDA_CONTEXT 1
# TIMESTAMPFACTOR 13272844762e2e6d
method,gputime,cputime,occupancy
method=[ memcpyHtoD ] gputime=[ 110522.531 ] cputime=[ 110694.320 ]
method=[ memcpyHtoD ] gputime=[ 118029.891 ] cputime=[ 118126.320 ]
method=[ memcpyHtoD ] gputime=[ 109445.633 ] cputime=[ 109561.898 ]
method=[ _Z15MatrixMulKernel6MatrixS_S_ ] gputime=[ 16963420.000 ] cputime=[ 7.732 ]
occupancy=[ 0.667 ]
method=[ memcpyDtoH ] gputime=[ 455258.906 ] cputime=[ 456063.781 ]
```

   Program Output:

```
GPU Time:
 Exclusive = 16963.466797 ms
 Inclusive = 17758.589844 ms
GPU computation complete
Start CPU computation
CPU computation complete in 17123908.000000 ms (Nearly 5 hours!!!)
Test PASSED
Dimension M[height,width]: 11000  11000
Dimension N[height,width]: 11000  11000
```

6. Scaling analysis:



7. I didn't quite notice this behavior. But I believe in case this happens, it could be because of floating point representations used on CPU and GPU have some differences. So these differences might accumulate as row length increases and might eventually spill over the threshold for very big matrices, making the test to fail.