Anderson Nguyen
EECS 120 HW3

1. Compile and run this code, which reports the input vector size, the time to execute the kernel, and an effective bandwidth. Record these data. Explain how the effective bandwidth is being calculated.

```
=== Running 5 trials of naive ... ===
*** Trial 1 ***
N: 8388608
Timer: gettimeofday
Timer resolution: ~ 1 us (?)
Time to execute naive GPU reduction kernel: 0.003356 secs
Effective bandwidth: 10.00 GB/s
Time to execute naive CPU reduction: 0.147531 secs
SUCCESS: GPU: 41.949856         CPU: 41.949856
*** Trial 2 ***
N: 8388608
Timer: gettimeofday
Timer resolution: ~ 1 us (?)
Time to execute naive GPU reduction kernel: 0.003361 secs
Effective bandwidth: 9.98 GB/s
Time to execute naive CPU reduction: 0.147493 secs
SUCCESS: GPU: 41.947445         CPU: 41.947445
*** Trial 3 ***
N: 8388608
Timer: gettimeofday
Timer resolution: ~ 1 us (?)
Time to execute naive GPU reduction kernel: 0.003354 secs
Effective bandwidth: 10.00 GB/s
Time to execute naive CPU reduction: 0.147465 secs
SUCCESS: GPU: 41.933464         CPU: 41.933464
*** Trial 4 ***
N: 8388608
Timer: gettimeofday
Timer resolution: ~ 1 us (?)
Time to execute naive GPU reduction kernel: 0.003383 secs
Effective bandwidth: 9.92 GB/s
Time to execute naive CPU reduction: 0.147492 secs
SUCCESS: GPU: 41.939293         CPU: 41.939293
*** Trial 5 ***
N: 8388608
Timer: gettimeofday
Timer resolution: ~ 1 us (?)
Time to execute naive GPU reduction kernel: 0.003353 secs
Effective bandwidth: 10.01 GB/s
Time to execute naive CPU reduction: 0.147461 secs
SUCCESS: GPU: 41.955101         CPU: 41.955101

=== Done! ===
```

In the code, the effective bandwidth is being calculated with the formula

bw = (N * sizeof(dtype)) / (t_kernal_0 * 1e9) where

N = The size the user inputs which is 8 * 1024 * 1024 if the user does not input anything
sizeof(dtype) = the size of dtype which is a float, which is usually 4 bytes on a lot of platforms
t_kernal_0 * 1e9) = which is the timer

Effective Bandwidth Avg of 5 Trials = 9.98 GB/s (Naive)
low: 9.92 GB/s
high: 10.01 GB/s

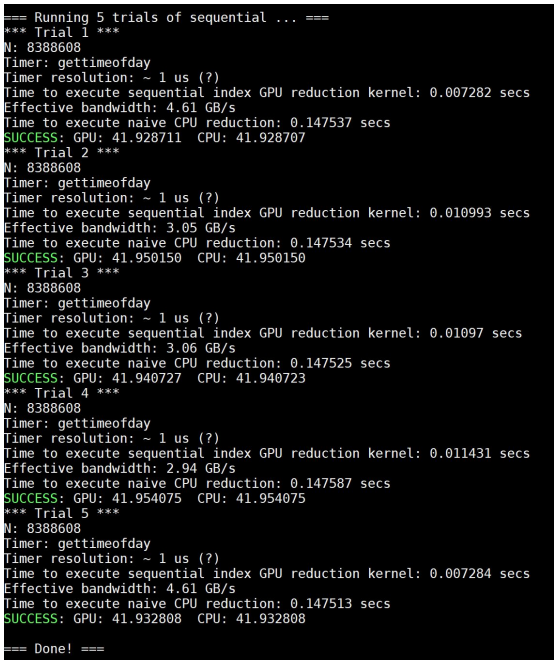
2. Implement this scheme in kernel1 of stride.cu. Measure and record the resulting performance. How much faster than the initial code is this version?

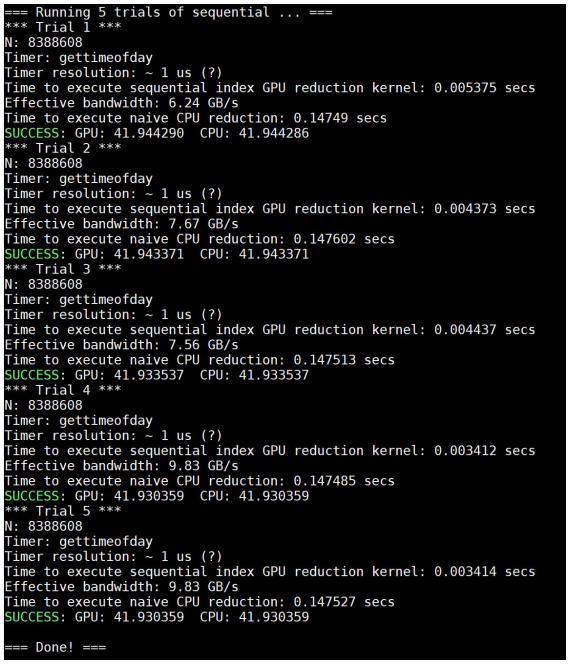
```
=== Running 5 trials of stride ... ===
*** Trial 1 ***
N: 8388608
Timer: gettimeofday
Timer resolution: ~ 1 us (?)
Time to execute strided index GPU reduction kernel: 0.003776 secs
Effective bandwidth: 8.89 GB/s
Time to execute naive CPU reduction: 0.147444 secs
SUCCESS: GPU: 41.939003  CPU: 41.939007
*** Trial 2 ***
N: 8388608
Timer: gettimeofday
Timer resolution: ~ 1 us (?)
Time to execute strided index GPU reduction kernel: 0.00376 secs
Effective bandwidth: 8.92 GB/s
Time to execute naive CPU reduction: 0.147482 secs
SUCCESS: GPU: 41.946693  CPU: 41.946693
*** Trial 3 ***
N: 8388608
Timer: gettimeofday
Timer resolution: ~ 1 us (?)
Time to execute strided index GPU reduction kernel: 0.00376 secs
Effective bandwidth: 8.92 GB/s
Time to execute naive CPU reduction: 0.147463 secs
SUCCESS: GPU: 41.936264  CPU: 41.936264
*** Trial 4 ***
N: 8388608
Timer: gettimeofday
Timer resolution: ~ 1 us (?)
Time to execute strided index GPU reduction kernel: 0.003759 secs
Effective bandwidth: 8.93 GB/s
Time to execute naive CPU reduction: 0.147466 secs
SUCCESS: GPU: 41.945580  CPU: 41.945583
*** Trial 5 ***
N: 8388608
Timer: gettimeofday
Timer resolution: ~ 1 us (?)
Time to execute strided index GPU reduction kernel: 0.003788 secs
Effective bandwidth: 8.86 GB/s
Time to execute naive CPU reduction: 0.147449 secs
SUCCESS: GPU: 41.948452  CPU: 41.948452

=== Done! ===
```

Effective Bandwidth Avg of 5 Trials = 8.90 GB/s (stride)
low: 8.86 GB/s
high: 8.93 GB/s

The stride implementation does not seem to be faster than the naive implementation since it is transferring 1GB less than the naive for my implementation of it. Even if it was supposed to be faster, the improvement is very negligible and not a big improvement.

3. Implement this scheme in kernel2 of sequential.cu. Record the new effective bandwidth.





I am not sure if there is a problem with the jobs but I get radically different effective bandwidth times and it does not seem very consistent. I have had jobs where it was around 10 GB/s and some where it was 4 GB/s. I have decided to just take the fastest one and used it as the effective bandwidth.

Effective Bandwidth Avg of 5 Trials = 8.23 GB/s (sequential)
low: 2.94 GB/s
high: 9.83 GB/s

4. Implement this scheme in kernel3 of first_add.cu and report the effective bandwidth.

```
=== Running 5 trials of first_add ... ===
*** Trial 1 ***
N: 8388608
Timer: gettimeofday
Timer resolution: ~ 1 us (?)
Time to execute first add GPU reduction kernel: 0.001755 secs
Effective bandwidth: 19.12 GB/s
Time to execute naive CPU reduction: 0.147486 secs
SUCCESS: GPU: 41.940529  CPU: 41.940533
*** Trial 2 ***
N: 8388608
Timer: gettimeofday
Timer resolution: ~ 1 us (?)
Time to execute first add GPU reduction kernel: 0.001753 secs
Effective bandwidth: 19.14 GB/s
Time to execute naive CPU reduction: 0.147569 secs
SUCCESS: GPU: 41.935719  CPU: 41.935715
*** Trial 3 ***
N: 8388608
Timer: gettimeofday
Timer resolution: ~ 1 us (?)
Time to execute first add GPU reduction kernel: 0.001795 secs
Effective bandwidth: 18.69 GB/s
Time to execute naive CPU reduction: 0.1475 secs
SUCCESS: GPU: 41.943550  CPU: 41.943550
*** Trial 4 ***
N: 8388608
Timer: gettimeofday
Timer resolution: ~ 1 us (?)
Time to execute first add GPU reduction kernel: 0.001753 secs
Effective bandwidth: 19.14 GB/s
Time to execute naive CPU reduction: 0.147511 secs
SUCCESS: GPU: 41.945744  CPU: 41.945744
*** Trial 5 ***
N: 8388608
Timer: gettimeofday
Timer resolution: ~ 1 us (?)
Time to execute first add GPU reduction kernel: 0.002932 secs
Effective bandwidth: 11.44 GB/s
Time to execute naive CPU reduction: 0.14748 secs
SUCCESS: GPU: 41.937511  CPU: 41.937508

=== Done! ===
```

Effective Bandwidth Avg of 5 Trials = 17.51 GB/s (first_add)
low: 11.44 GB/s
high: 19.14 GB/s

5. Implement this scheme in kernel4 of unroll.cu and report the effective bandwidth.

```
=== Running 5 trials of unroll ... ===
*** Trial 1 ***
N: 8388608
Timer: gettimeofday
Timer resolution: ~ 1 us (?)
Time to execute unrolled GPU reduction kernel: 0.000548 secs
Effective bandwidth: 61.23 GB/s
Time to execute naive CPU reduction: 0.163974 secs
SUCCESS: GPU: 41.952293  CPU: 41.952290
*** Trial 2 ***
N: 8388608
Timer: gettimeofday
Timer resolution: ~ 1 us (?)
Time to execute unrolled GPU reduction kernel: 0.000554 secs
Effective bandwidth: 60.57 GB/s
Time to execute naive CPU reduction: 0.183346 secs
SUCCESS: GPU: 41.952293  CPU: 41.952290
*** Trial 3 ***
N: 8388608
Timer: gettimeofday
Timer resolution: ~ 1 us (?)
Time to execute unrolled GPU reduction kernel: 0.000553 secs
Effective bandwidth: 60.68 GB/s
Time to execute naive CPU reduction: 0.160376 secs
SUCCESS: GPU: 41.958462  CPU: 41.958462
*** Trial 4 ***
N: 8388608
Timer: gettimeofday
Timer resolution: ~ 1 us (?)
Time to execute unrolled GPU reduction kernel: 0.000551 secs
Effective bandwidth: 60.90 GB/s
Time to execute naive CPU reduction: 0.176085 secs
SUCCESS: GPU: 41.958462  CPU: 41.958462
*** Trial 5 ***
N: 8388608
Timer: gettimeofday
Timer resolution: ~ 1 us (?)
Time to execute unrolled GPU reduction kernel: 0.00055 secs
Effective bandwidth: 61.01 GB/s
Time to execute naive CPU reduction: 0.160999 secs
SUCCESS: GPU: 41.943195  CPU: 41.943195
```

Effective Bandwidth Avg of 5 Trials = 60.88 GB/s (unroll)
low: 60.01 GB/s
high: 61.23 GB/s

6. Implement the algorithm cascading scheme in multiple.cu and report the effective bandwidth.

```
=== Running 5 trials of multiple ... ===
*** Trial 1 ***
N: 8388608
Timer: gettimeofday
Timer resolution: ~ 1 us (?)
Time to execute multiple add GPU reduction kernel: 0.000993 secs
Effective bandwidth: 33.79 GB/s
Time to execute naive CPU reduction: 0.178722 secs
SUCCESS: GPU: 41.946701  CPU: 41.946701
*** Trial 2 ***
N: 8388608
Timer: gettimeofday
Timer resolution: ~ 1 us (?)
Time to execute multiple add GPU reduction kernel: 0.007143 secs
Effective bandwidth: 4.70 GB/s
Time to execute naive CPU reduction: 0.169835 secs
SUCCESS: GPU: 41.954475  CPU: 41.954475
*** Trial 3 ***
N: 8388608
Timer: gettimeofday
Timer resolution: ~ 1 us (?)
Time to execute multiple add GPU reduction kernel: 0.000377 secs
Effective bandwidth: 89.00 GB/s
Time to execute naive CPU reduction: 0.159994 secs
SUCCESS: GPU: 41.940163  CPU: 41.940163
*** Trial 4 ***
N: 8388608
Timer: gettimeofday
Timer resolution: ~ 1 us (?)
Time to execute multiple add GPU reduction kernel: 0.000408 secs
Effective bandwidth: 82.24 GB/s
Time to execute naive CPU reduction: 0.152045 secs
SUCCESS: GPU: 41.937454  CPU: 41.937454
*** Trial 5 ***
N: 8388608
Timer: gettimeofday
Timer resolution: ~ 1 us (?)
Time to execute multiple add GPU reduction kernel: 0.000379 secs
Effective bandwidth: 88.53 GB/s
Time to execute naive CPU reduction: 0.185071 secs
SUCCESS: GPU: 41.929302  CPU: 41.929298

=== Done! ===
```

Effective Bandwidth Avg of 5 Trials = 59.65 GB/s (multiple)
low = 4.70 GB/s
high = 89.0 GB/s

7. Matrix Transpose Explanation. In addition to submitting your code, briefly describe how your algorithm works and optimizations attempted. Also, present performance results and a brief discussion of the results.

```
GPU transpose: 8.22889 secs ==> 1.2444e-07 billion elements/second
Time to execute CPU transpose kernel: 9e-06 secs
Transpose successful
GPU transpose: 8.21833 secs ==> 1.246e-07 billion elements/second
Time to execute CPU transpose kernel: 9e-06 secs
Transpose successful
GPU transpose: 8.21839 secs ==> 1.24599e-07 billion elements/second
Time to execute CPU transpose kernel: 9e-06 secs
Transpose successful
GPU transpose: 8.2215 secs ==> 1.24551e-07 billion elements/second
Time to execute CPU transpose kernel: 9e-06 secs
Transpose successful
GPU transpose: 8.22017 secs ==> 1.24572e-07 billion elements/second
Time to execute CPU transpose kernel: 9e-06 secs
Transpose successful
```

void gpuTranspose

For running the kernel, we need to initialize it with the gb and tb function. Those functions use the value of the size and then uses those values to then check for errors when we run the code. If it is able to pass those checks, then it will run successfully. Those errors are also there to prevent unnecessary memory leaks.

__global__ void matTrans

this allocates the memory of the matrix onto the GPU by taking the x, y and width to determine the location of the element inside of the A array and store it into the AT array.