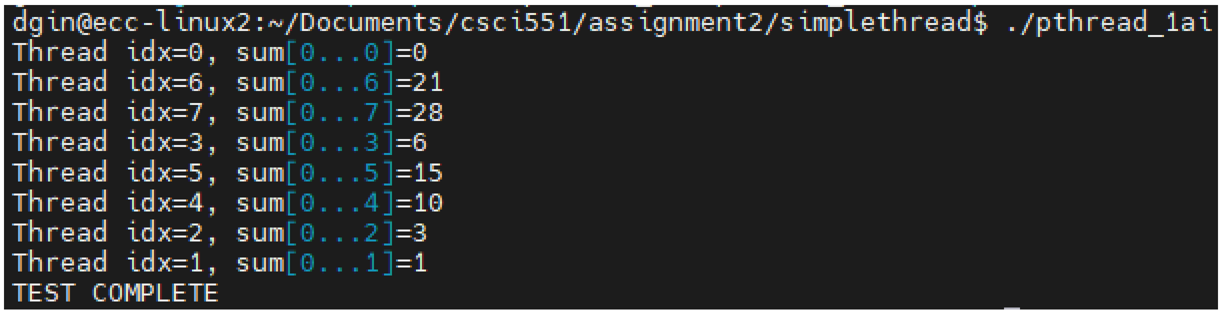
The typical naming scheme for modified code is “[original code file name]_[problem number].c”. The code will be in the specified folders. To run the code, simply run the Makefile and run the executables.

1) **Code is within the simplethread and sumdigits folders: pthread_1ai, pthread_1aii, sumdigits_1b, sumdigits_1c**

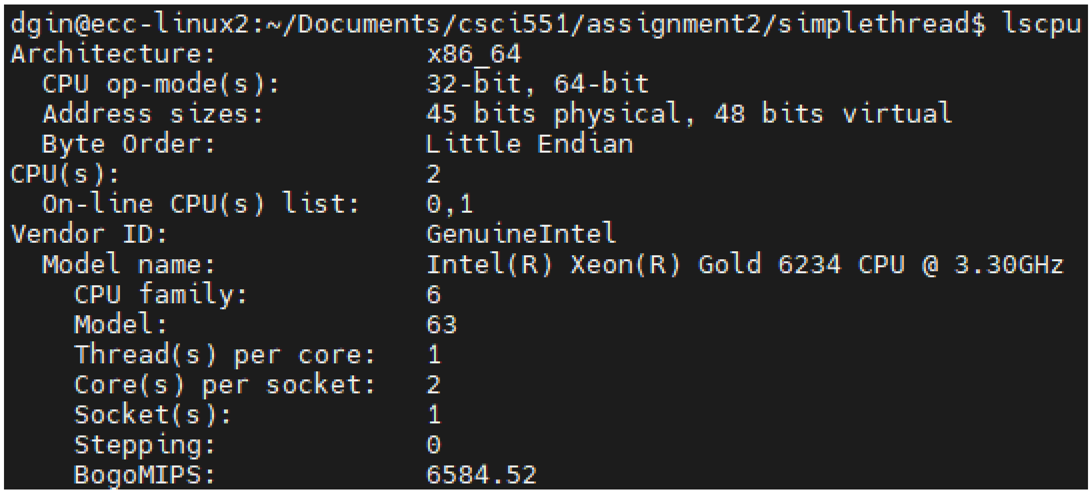
a)

i)



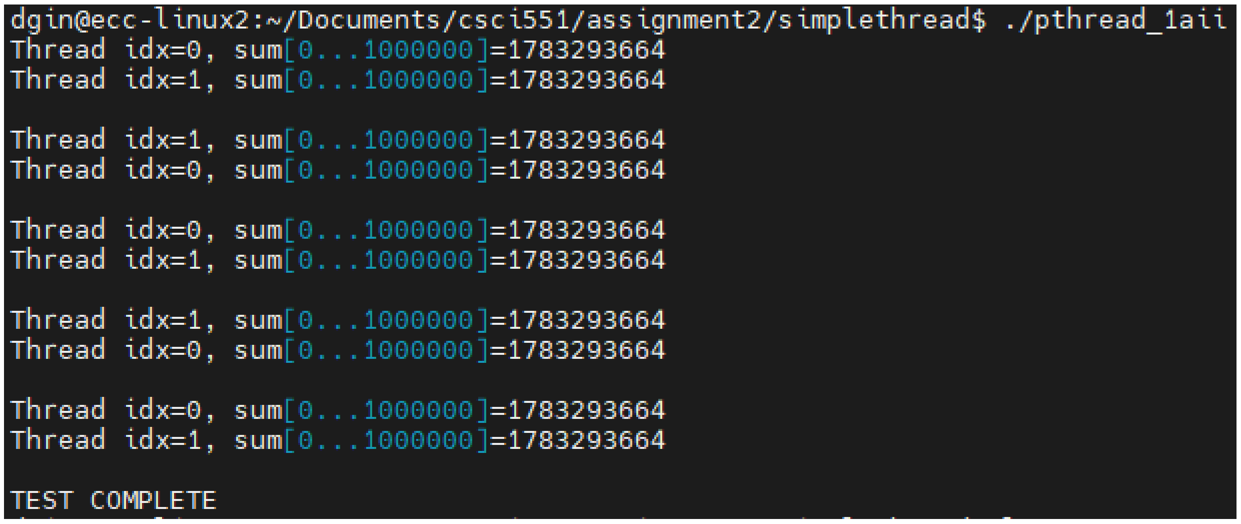
```
dgin@ecc-linux2:~/Documents/csci551/assignment2/simplethread$ ./pthread_1ai
Thread idx=0, sum[0...0]=0
Thread idx=6, sum[0...6]=21
Thread idx=7, sum[0...7]=28
Thread idx=3, sum[0...3]=6
Thread idx=5, sum[0...5]=15
Thread idx=4, sum[0...4]=10
Thread idx=2, sum[0...2]=3
Thread idx=1, sum[0...1]=1
TEST COMPLETE
```

ii)



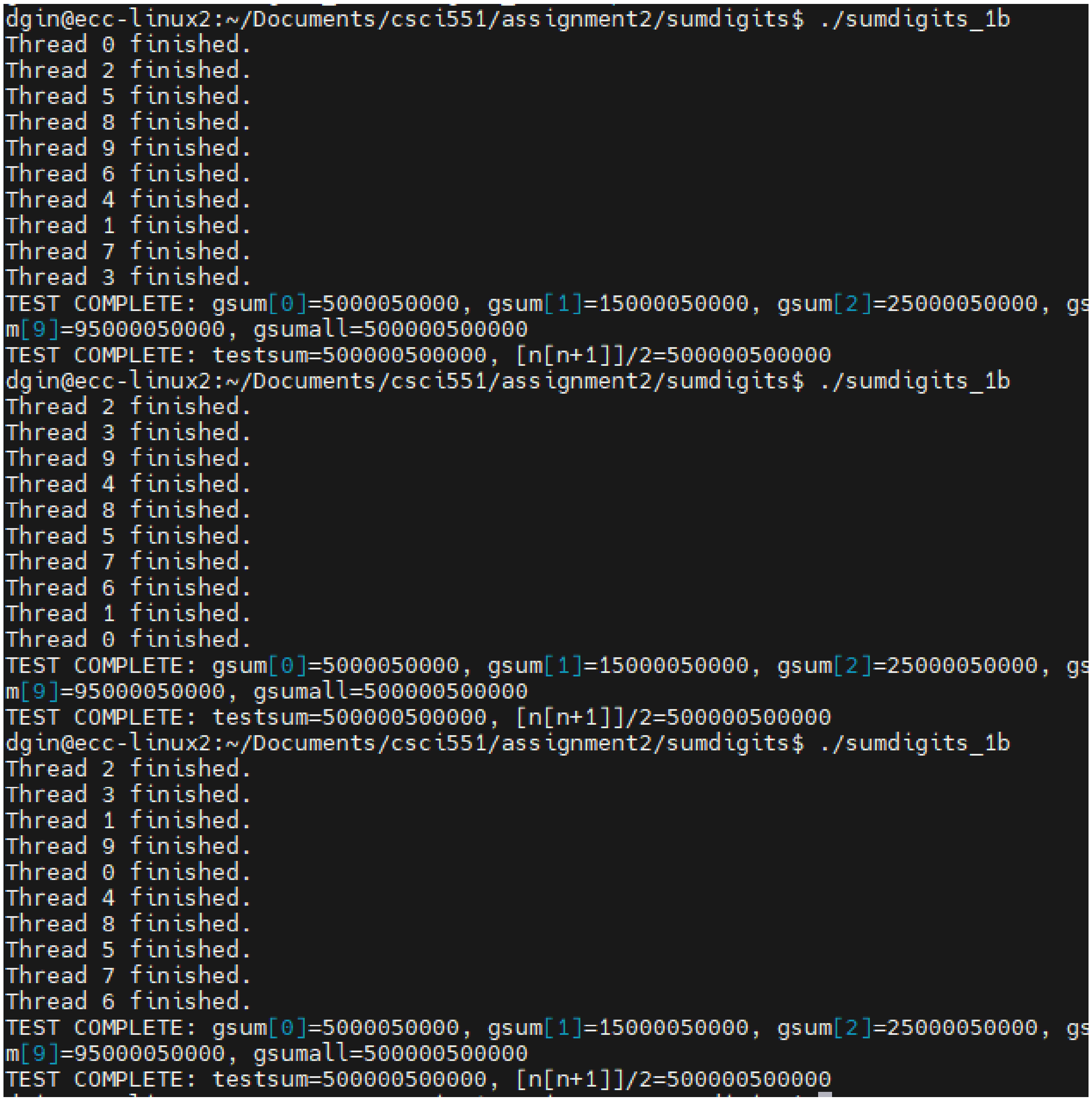
```
dgin@ecc-linux2:~/Documents/csci551/assignment2/simplethread$ lscpu
Architecture:            x86_64
  CPU op-mode(s):        32-bit, 64-bit
  Address sizes:         45 bits physical, 48 bits virtual
  Byte Order:            Little Endian
CPU(s):                  2
  On-line CPU(s) list:   0,1
Vendor ID:               GenuineIntel
  Model name:            Intel(R) Xeon(R) Gold 6234 CPU @ 3.30GHz
    CPU family:          6
    Model:               63
    Thread(s) per core:  1
    Core(s) per socket:  2
    Socket(s):           1
    Stepping:            0
    BogoMIPS:            6584.52
```

Ecc has 2 cores so m = 2



```
dgin@ecc-linux2:~/Documents/csci551/assignment2/simplethread$ ./pthread_1aii
Thread idx=0, sum[0...1000000]=1783293664
Thread idx=1, sum[0...1000000]=1783293664

Thread idx=1, sum[0...1000000]=1783293664
Thread idx=0, sum[0...1000000]=1783293664

Thread idx=0, sum[0...1000000]=1783293664
Thread idx=1, sum[0...1000000]=1783293664

Thread idx=1, sum[0...1000000]=1783293664
Thread idx=0, sum[0...1000000]=1783293664

Thread idx=0, sum[0...1000000]=1783293664
Thread idx=1, sum[0...1000000]=1783293664

TEST COMPLETE
```

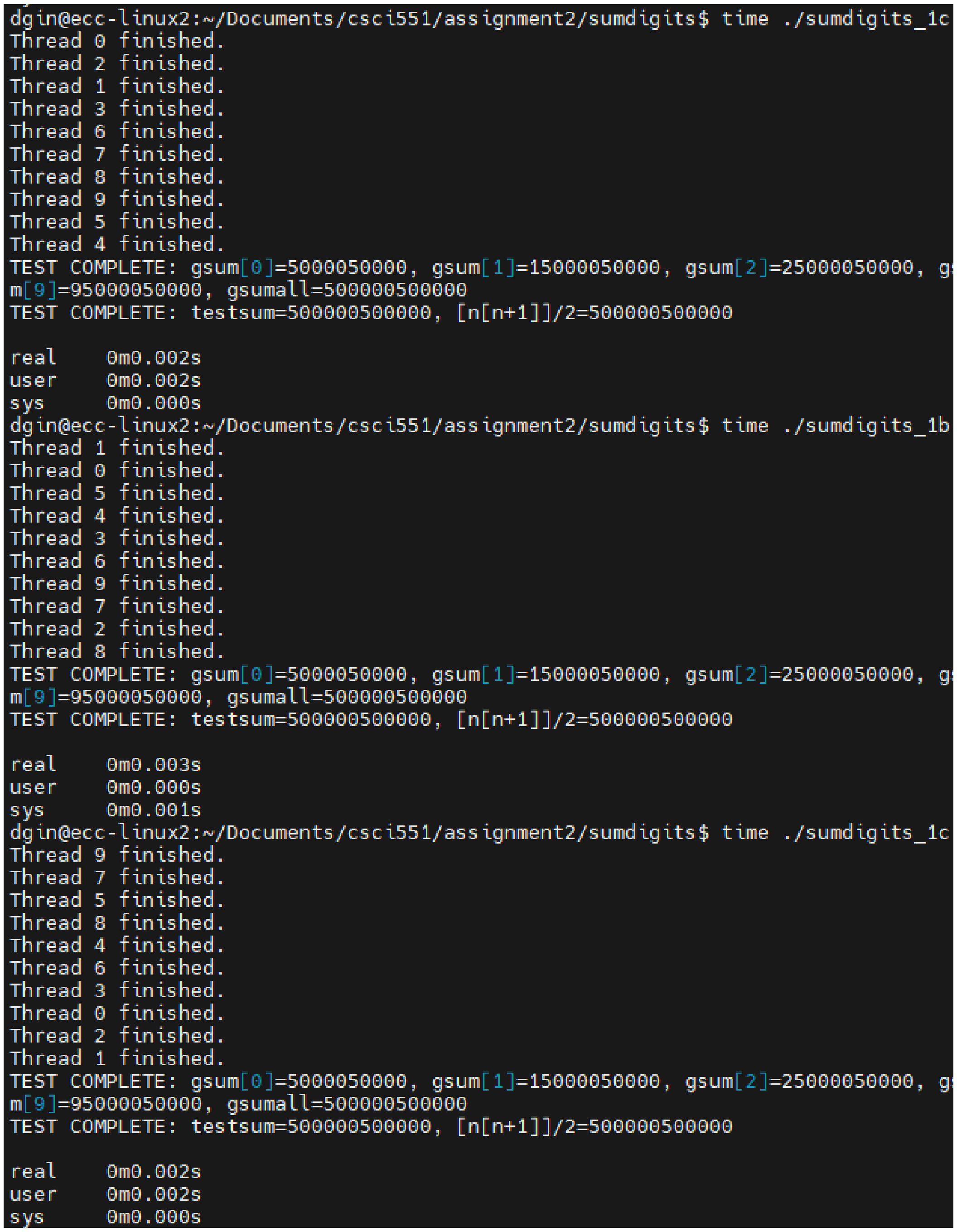
b)



```
dgin@ecc-linux2:~/Documents/csci551/assignment2/sumdigits$ ./sumdigits_1b
Thread 0 finished.
Thread 2 finished.
Thread 5 finished.
Thread 8 finished.
Thread 9 finished.
Thread 6 finished.
Thread 4 finished.
Thread 1 finished.
Thread 7 finished.
Thread 3 finished.
TEST COMPLETE: gsum[0]=5000050000, gsum[1]=15000050000, gsum[2]=25000050000, gs
m[9]=95000050000, gsumall=500000500000
TEST COMPLETE: testsum=500000500000, [n[n+1]]/2=500000500000
dgin@ecc-linux2:~/Documents/csci551/assignment2/sumdigits$ ./sumdigits_1b
Thread 2 finished.
Thread 3 finished.
Thread 9 finished.
Thread 4 finished.
Thread 8 finished.
Thread 5 finished.
Thread 7 finished.
Thread 6 finished.
Thread 1 finished.
Thread 0 finished.
TEST COMPLETE: gsum[0]=5000050000, gsum[1]=15000050000, gsum[2]=25000050000, gs
m[9]=95000050000, gsumall=500000500000
TEST COMPLETE: testsum=500000500000, [n[n+1]]/2=500000500000
dgin@ecc-linux2:~/Documents/csci551/assignment2/sumdigits$ ./sumdigits_1b
Thread 2 finished.
Thread 3 finished.
Thread 1 finished.
Thread 9 finished.
Thread 0 finished.
Thread 4 finished.
Thread 8 finished.
Thread 5 finished.
Thread 7 finished.
Thread 6 finished.
TEST COMPLETE: gsum[0]=5000050000, gsum[1]=15000050000, gsum[2]=25000050000, gs
m[9]=95000050000, gsumall=500000500000
TEST COMPLETE: testsum=500000500000, [n[n+1]]/2=500000500000
```

The threads do not finish in the same order. This is expected because the order that they are finished should be determined by the OS scheduler allocating resources as they are available, which won't be the same between runs.

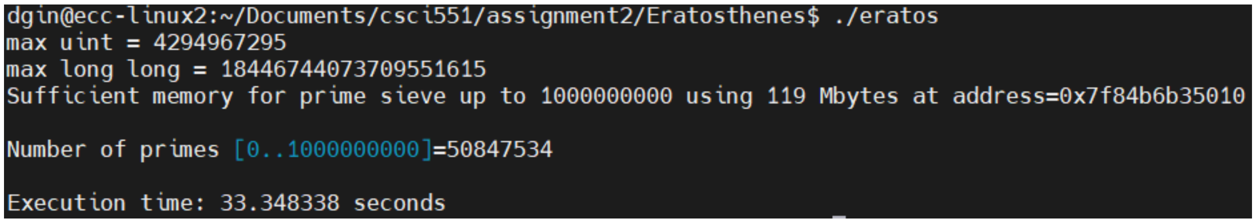
c)



```
dgin@ecc-linux2:~/Documents/csci551/assignment2/sumdigits$ time ./sumdigits_1c
Thread 0 finished.
Thread 2 finished.
Thread 1 finished.
Thread 3 finished.
Thread 6 finished.
Thread 7 finished.
Thread 8 finished.
Thread 9 finished.
Thread 5 finished.
Thread 4 finished.
TEST COMPLETE: gsum[0]=5000050000, gsum[1]=15000050000, gsum[2]=25000050000, g
m[9]=95000050000, gsumall=500000500000
TEST COMPLETE: testsum=500000500000, [n[n+1]]/2=500000500000

real    0m0.002s
user    0m0.002s
sys     0m0.000s
dgin@ecc-linux2:~/Documents/csci551/assignment2/sumdigits$ time ./sumdigits_1b
Thread 1 finished.
Thread 0 finished.
Thread 5 finished.
Thread 4 finished.
Thread 3 finished.
Thread 6 finished.
Thread 9 finished.
Thread 7 finished.
Thread 2 finished.
Thread 8 finished.
TEST COMPLETE: gsum[0]=5000050000, gsum[1]=15000050000, gsum[2]=25000050000, g
m[9]=95000050000, gsumall=500000500000
TEST COMPLETE: testsum=500000500000, [n[n+1]]/2=500000500000

real    0m0.003s
user    0m0.000s
sys     0m0.001s
dgin@ecc-linux2:~/Documents/csci551/assignment2/sumdigits$ time ./sumdigits_1c
Thread 9 finished.
Thread 7 finished.
Thread 5 finished.
Thread 8 finished.
Thread 4 finished.
Thread 6 finished.
Thread 3 finished.
Thread 0 finished.
Thread 2 finished.
Thread 1 finished.
TEST COMPLETE: gsum[0]=5000050000, gsum[1]=15000050000, gsum[2]=25000050000, g
m[9]=95000050000, gsumall=500000500000
TEST COMPLETE: testsum=500000500000, [n[n+1]]/2=500000500000

real    0m0.002s
user    0m0.002s
sys     0m0.000s
```

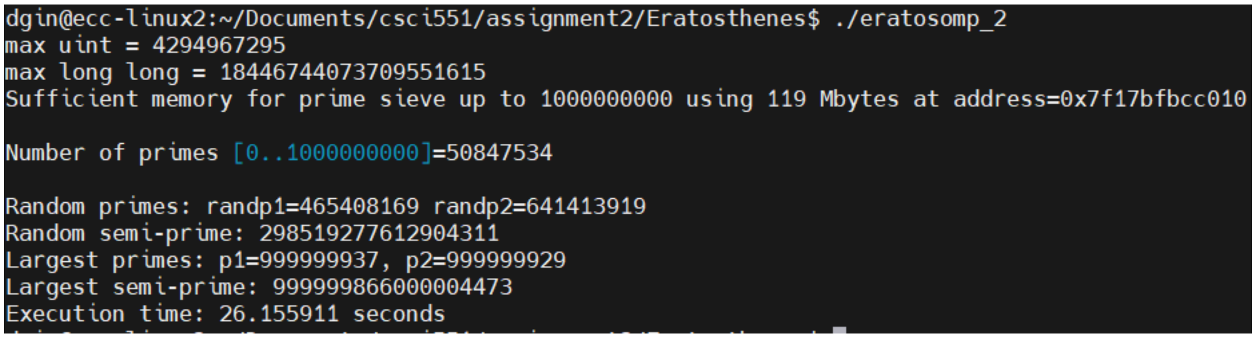
Sumdigits_1b is the pthreads implementation and sumdigits_1c is the openmp version. In comparison to the pthreads version, there wasn't much of a difference in runtime using the linux time function. Runtimes varied by 1/1000th of a second with either version being faster if not matching.

2) **Code is within the Eratosthenes folder: eratos, eratosomp_2**



```
dgin@ecc-linux2:~/Documents/csci551/assignment2/Eratosthenes$ ./eratos
max uint = 4294967295
max long long = 18446744073709551615
Sufficient memory for prime sieve up to 1000000000 using 119 Mbytes at address=0x7f84b6b35010

Number of primes [0..1000000000]=50847534

Execution time: 33.348338 seconds
```

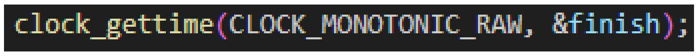
Non-parallel



```
dgin@ecc-linux2:~/Documents/csci551/assignment2/Eratosthenes$ ./eratosomp_2
max uint = 4294967295
max long long = 18446744073709551615
Sufficient memory for prime sieve up to 1000000000 using 119 Mbytes at address=0x7f17bfbcc010

Number of primes [0..1000000000]=50847534

Random primes: randp1=465408169 randp2=641413919
Random semi-prime: 298519277612904311
Largest primes: p1=999999937, p2=999999929
Largest semi-prime: 999999866000004473
Execution time: 26.155911 seconds
```

Parallel with pragma

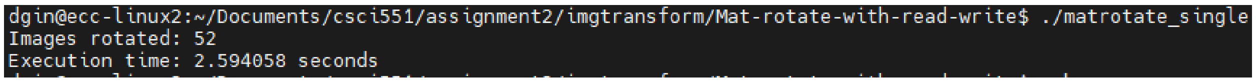
Timing was performed with posix clock_gettime(CLOCK_MONOTONIC_RAW)



```
clock_gettime(CLOCK_MONOTONIC_RAW, &finish);
```

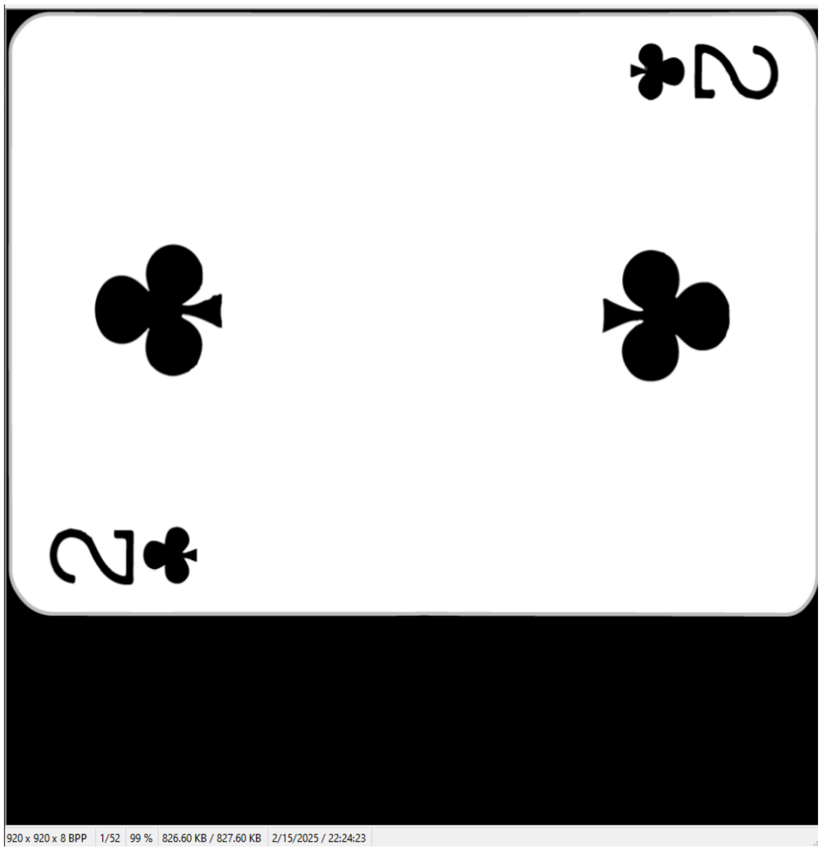
Speedup: SU = T_sequential/T_parallel = 33.348/26.156 = **1.275** times faster or **127.5%** speed up.

3) **Code is within the imgtransform folder: matrotate_single, matrotate_omp**

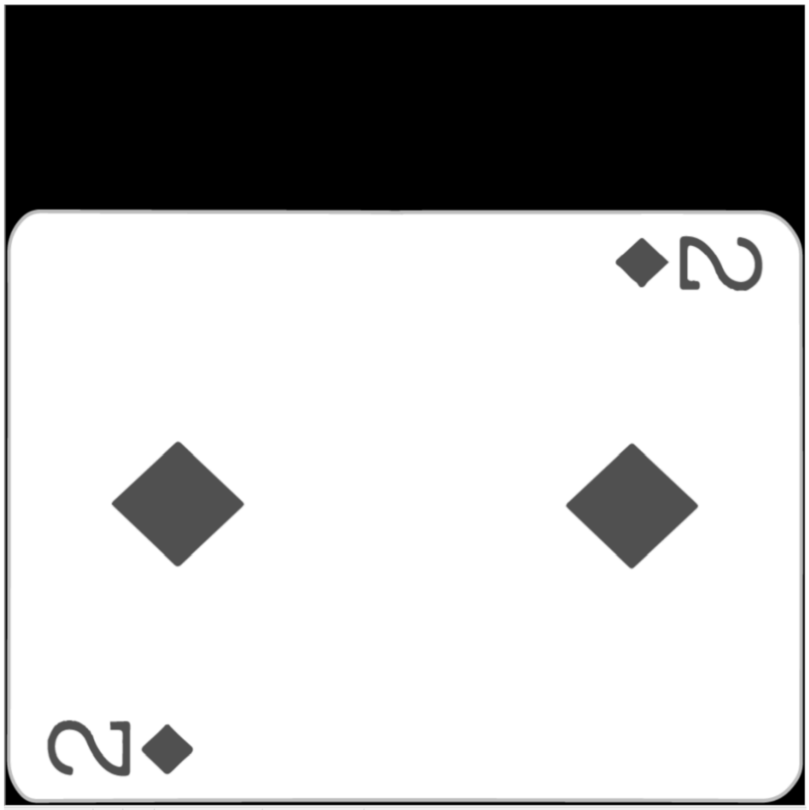


```
dgin@ecc-linux2:~/Documents/csci551/assignment2/imgtransform/Mat-rotate-with-read-write$ ./matrotate_single
Images rotated: 52
Execution time: 2.594058 seconds
```

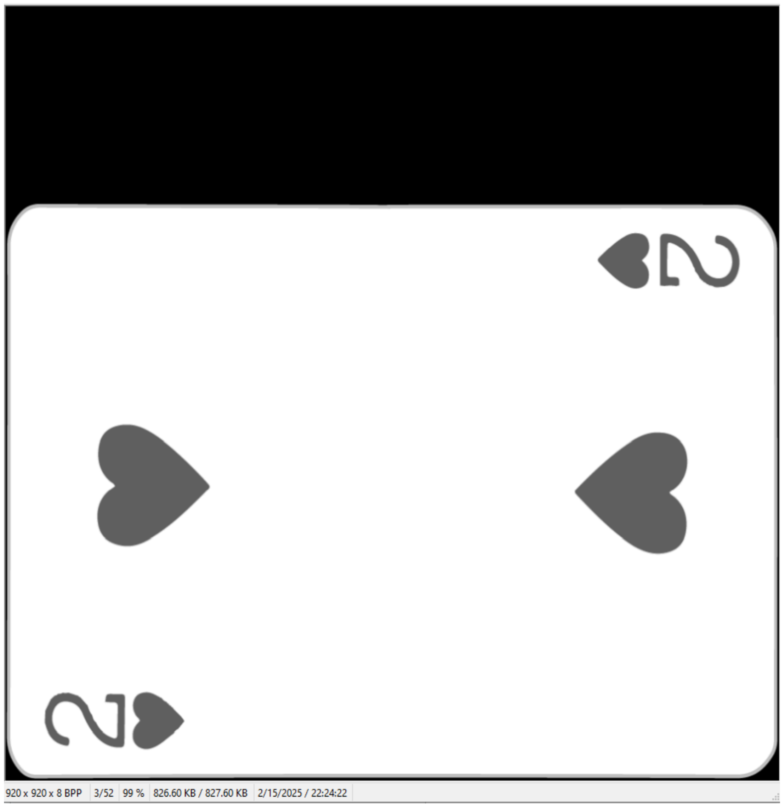
Runtimes fluctuated between ~1.5s to ~2.6s for the single-threaded version of the image rotation.



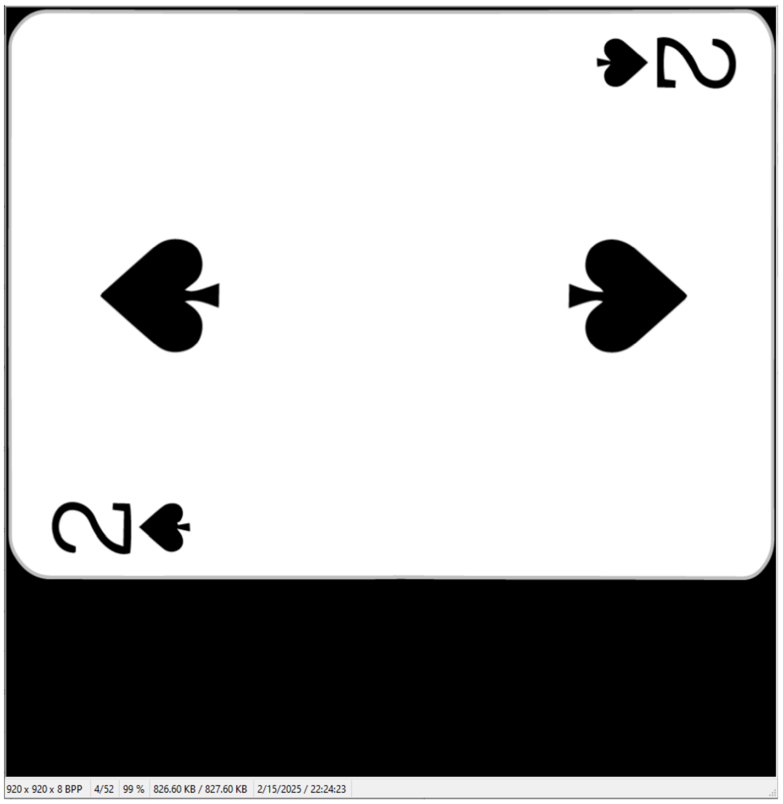
920 x 920 x 8 BPP | 1/52 | 99 % | 826.60 KB / 827.60 KB | 2/15/2025 / 22:24:23



920 x 920 x 8 BPP | 2/52 | 99 % | 826.60 KB / 827.60 KB | 2/15/2025 / 22:24:24

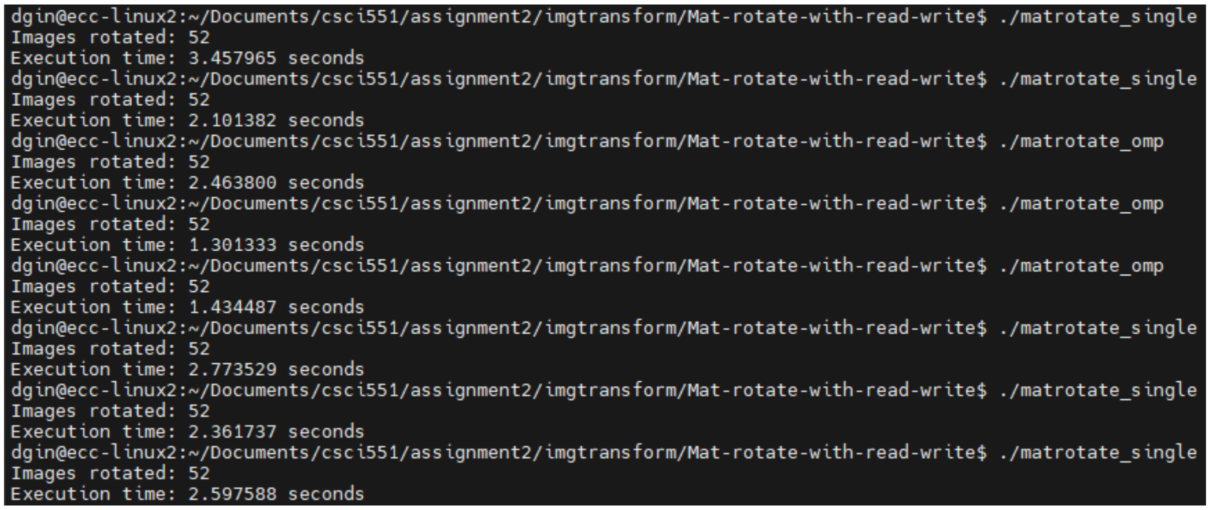


```
920 x 920 x 8 BPP | 3/52 | 99 % | 826.60 KB / 827.60 KB | 2/15/2025 / 22:24:22
```



```
920 x 920 x 8 BPP | 4/52 | 99 % | 826.60 KB / 827.60 KB | 2/15/2025 / 22:24:23
```

Spades and clubs are rotated right. Hearts and diamonds are rotated left.



```
dgin@ecc-linux2:~/Documents/csci551/assignment2/imgtransform/Mat-rotate-with-read-write$ ./matrotate_single
Images rotated: 52
Execution time: 3.457965 seconds
dgin@ecc-linux2:~/Documents/csci551/assignment2/imgtransform/Mat-rotate-with-read-write$ ./matrotate_single
Images rotated: 52
Execution time: 2.101382 seconds
dgin@ecc-linux2:~/Documents/csci551/assignment2/imgtransform/Mat-rotate-with-read-write$ ./matrotate_omp
Images rotated: 52
Execution time: 2.463800 seconds
dgin@ecc-linux2:~/Documents/csci551/assignment2/imgtransform/Mat-rotate-with-read-write$ ./matrotate_omp
Images rotated: 52
Execution time: 1.301333 seconds
dgin@ecc-linux2:~/Documents/csci551/assignment2/imgtransform/Mat-rotate-with-read-write$ ./matrotate_omp
Images rotated: 52
Execution time: 1.434487 seconds
dgin@ecc-linux2:~/Documents/csci551/assignment2/imgtransform/Mat-rotate-with-read-write$ ./matrotate_single
Images rotated: 52
Execution time: 2.773529 seconds
dgin@ecc-linux2:~/Documents/csci551/assignment2/imgtransform/Mat-rotate-with-read-write$ ./matrotate_single
Images rotated: 52
Execution time: 2.361737 seconds
dgin@ecc-linux2:~/Documents/csci551/assignment2/imgtransform/Mat-rotate-with-read-write$ ./matrotate_single
Images rotated: 52
Execution time: 2.597588 seconds
```

Gauging the runtime was very difficult for this. Runtimes had a typical range, but could fluctuate very extremely.
My single-threaded approach typically ran for about 2.5s. In very rare cases, it ran for as little as 1.23s. My openmp approach typically ran for about 1.45s. The fastest time was 1.36s
Both had very rare cases where they ran for at or over 3s. This was probably due to ECC system limitations.
In summary, there were cases that the single-threaded code could run faster, but on average, the parallel code was faster. Using the averages of 2.5s for single-threaded and 1.45s for parallel, the speedup would be 1.724 times faster.
SU = T_sequential/T_parallel = 2.5/1.45 = **<u>1.724</u>** or **<u>172.4%</u>** speed up.

Ideally, I would have implemented the parallel threads to handle a different card each of the 52, but I ran into major issues with race conditions where the threads were trying to place their information from different cards into the same image file. This created cards with corrupted appearances. I instead opted to have the threads handle the rotation functions.