

## **Three Approaches to Implementing the Two-Dimensional Fourier Transform using Thread, MPI, and Cuda**

Wootae Song: [wootae@gatech.edu](mailto:wootae@gatech.edu)  
Alberto Li: [ali97@gatech.edu](mailto:ali97@gatech.edu)  
Edward Zhou: [edz@gatech.edu](mailto:edz@gatech.edu)  
Benjamin Thornbloom: [bthornbloom3@gatech.edu](mailto:bthornbloom3@gatech.edu)

Due: Dec 13, 2018

## Abstract

The goal of the final project was to implement the two-dimensional fourier transform using three different methods: `std::Thread` library, MPI, and Cuda. We have also implemented the inverse discrete fourier transform to validate our results. Each method of implementation was uniquely written for optimized runtime based on the computing model. We compared the performance of each method against each other. This information is shown in the results section.

### STD::Thread

The first implementation of the two-dimensional fourier transform was a multi-threaded approach using the `std::thread` library and running on a machine with eight cores. When the source code is compiled, it will produce an executable of the name “p31”. This method was implemented using an  $N^2$  approach as suggested by the project description.

### MPI

The second implementation of the two-dimensional fourier transform was an MPI approach running on a coc-ice MPI job using a pbs script with 8 MPI ranks. When the source code is compiled, it will produce an executable of the name “p32”. This method was implemented using the Danielson-Lanczos algorithm, an  $N \log(N)$  approach as suggested by the project description.

## CUDA

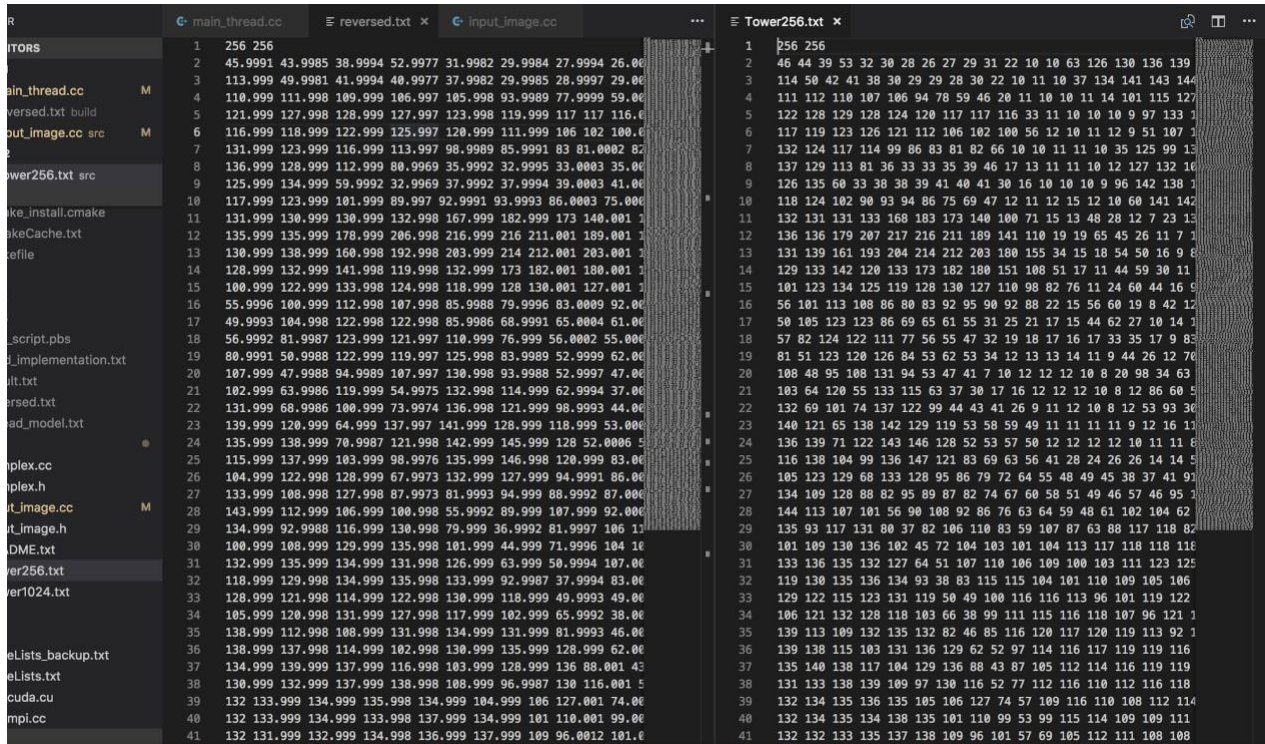
The third implementation of the two-dimensional fourier transform was a GPU approach using CUDA and running on a coc-ice job queue. When the source code is compiled, it will produce an executable of the name “p33”. This method was implemented using an  $N^2$  approach.

## Inverse Discrete Fourier Transform (Reverse DFT)

For the Thread computing model, we have implemented the inverse discrete fourier transform.

Adding this functionality has allowed us to validate the accuracy of our forward DFT since

IDFT(DFT(Image))) will reproduce the values of the original image. Below is a screenshot of the first few entries of the IDFT on Tower256.txt.



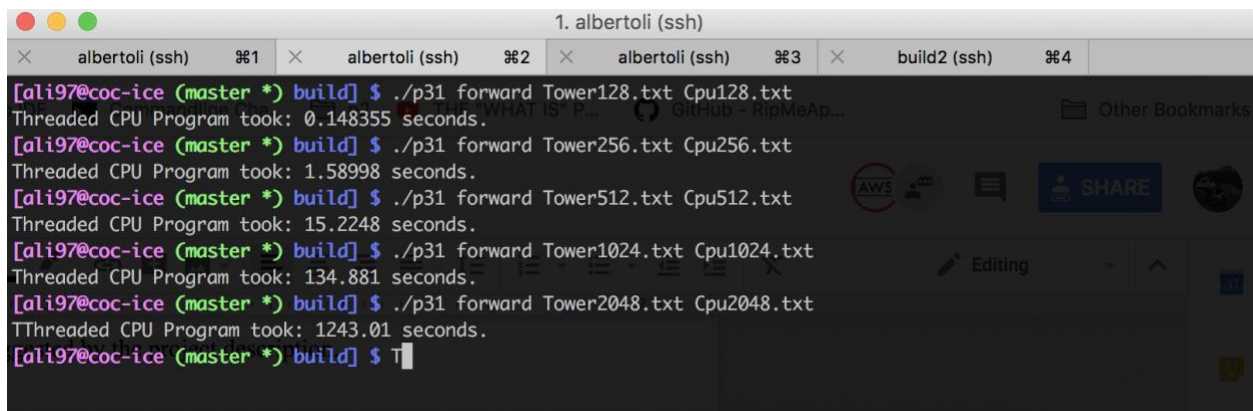
```
R
C- main_thread.cc  E reversed.txt x  C- input_image.cc  ...  E Tower256.txt x
ITORS
1 256 256
2 45.9991 43.9985 38.9994 52.9977 31.9982 29.9984 27.9994 26.0000
3 113.999 49.9981 41.9994 40.9977 37.9982 29.9985 28.9997 29.0000
main_thread.cc M
4 110.999 111.998 109.999 106.997 105.998 93.9989 77.9999 59.0000
reversed.txt build
5 121.999 127.998 128.999 127.997 123.998 119.999 117 117 116.000
6 116.999 118.999 122.999 125.997 120.999 111.999 106 102 100.000
7 131.999 123.999 116.999 113.997 98.9989 85.9991 83 81.0002 82.0000
8 136.999 128.999 112.999 80.9969 35.9992 32.9995 33.0003 35.0000
Tower256.txt src
9 125.999 134.999 59.9992 32.9969 37.9992 37.9994 39.0003 41.0000
10 117.999 123.999 101.999 89.997 92.9991 93.9993 86.0003 75.0000
11 131.999 130.999 130.999 132.998 167.999 182.999 173 140.001 140.001
12 135.999 135.999 178.999 206.998 216.999 216 211.001 189.001 177.000
13 130.999 138.999 160.998 192.998 203.999 214 212.001 203.001 177.000
14 128.999 132.999 141.998 119.998 132.999 173 182.001 180.001 177.000
15 100.999 122.999 133.998 124.998 118.999 128 130.001 127.001 140.001
16 55.9996 100.999 112.998 107.998 85.9988 79.9996 83.0009 92.0000
17 49.9993 104.998 122.998 122.998 85.9986 68.9991 65.0004 61.0000
18 56.9992 81.9987 123.999 121.997 110.999 76.9999 56.0002 55.0000
19 80.9991 50.9988 122.999 119.997 125.998 83.9989 52.9999 62.0000
20 107.999 47.9988 94.9989 107.997 130.998 93.9988 52.9997 47.0000
21 102.999 63.9986 119.999 54.9975 132.998 114.999 62.9994 37.0000
22 131.999 68.9986 100.999 73.9974 136.998 121.999 98.9993 44.0000
23 139.999 120.999 64.999 137.997 141.999 128.999 118.999 53.0000
24 135.999 138.999 70.9987 121.998 142.999 145.999 128 52.0006 52.0006
25 115.999 137.999 103.999 98.9976 135.999 146.998 120.999 83.0000
26 104.999 122.998 128.999 67.9973 132.999 127.999 94.9991 86.0000
27 133.999 108.998 127.998 87.9973 81.9993 94.999 88.9992 87.0000
28 143.999 112.999 106.999 100.998 55.9992 89.999 107.999 92.0000
29 134.999 92.9988 116.999 130.998 79.999 36.9992 81.9997 106 110.000
30 100.999 108.999 129.999 135.998 101.999 44.999 71.9996 104 116.000
31 132.999 135.999 134.999 131.998 126.999 63.999 50.9994 107.000
32 118.999 129.998 134.999 135.998 133.999 92.9987 37.9994 83.0000
33 128.999 121.998 114.999 122.998 130.999 118.999 49.9993 49.0000
34 105.999 120.998 131.999 127.998 117.999 102.999 65.9992 38.0000
35 138.999 112.998 108.999 131.998 134.999 131.999 81.9993 46.0000
36 138.999 137.998 114.999 102.998 130.999 135.999 128.999 62.0000
37 134.999 139.999 137.999 116.998 103.999 128.999 136 88.001 43.0000
38 130.999 132.999 137.999 138.998 108.999 96.9987 130 116.001 5.0000
39 132 133.999 134.999 135.998 134.999 104.999 106 127.001 74.0000
40 132 133.999 134.999 133.998 137.999 134.999 101 110.001 99.0000
41 132 131.999 132.999 134.998 136.999 137.999 109 96.0012 101.0000
```

Figure 1. First few entries of the IDFT when running ./p31 reverse Tower256.txt reversed.txt

## Results

The std::Thread, MPI, and CUDA computing methods were tested with text files of the following sizes: 128x128, 256x256, 512x512, 1024x1024, and 2048x2048. The execution times for each of the methods were timed. std::Thread and MPI were timed using std::chrono's high resolution clock at a microsecond granularity and CUDA was timed by submitting pbs scripts and looking at the walltime due to nvcc's bug with chrono.

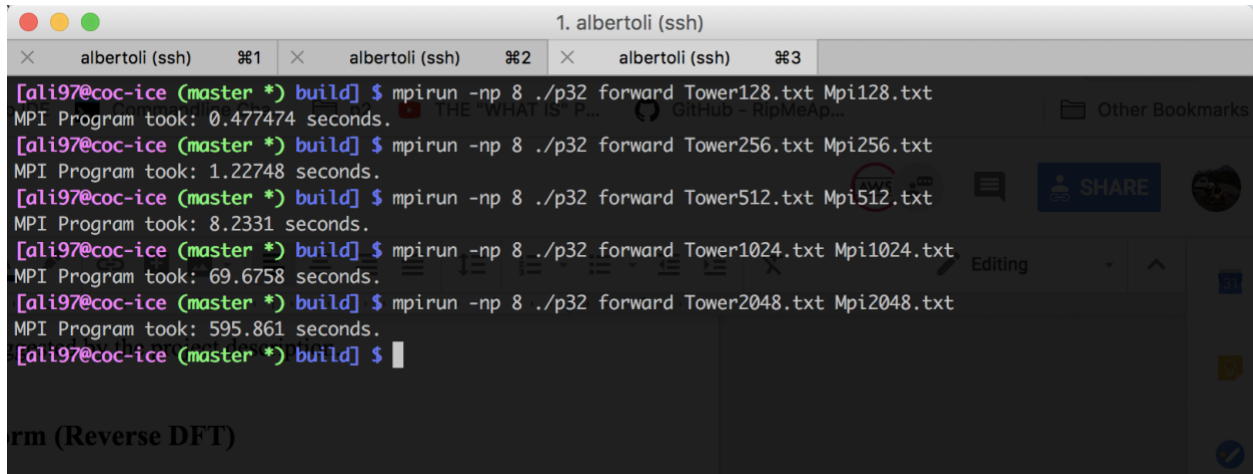
std::Thread:



```
1. albertoli (ssh)
albertoli (ssh) %1  albertoli (ssh) %2  albertoli (ssh) %3  build2 (ssh) %4
[ali97@coc-ice (master *) build] $ ./p31 forward Tower128.txt Cpu128.txt
Threaded CPU Program took: 0.148355 seconds.
[ali97@coc-ice (master *) build] $ ./p31 forward Tower256.txt Cpu256.txt
Threaded CPU Program took: 1.58998 seconds.
[ali97@coc-ice (master *) build] $ ./p31 forward Tower512.txt Cpu512.txt
Threaded CPU Program took: 15.2248 seconds.
[ali97@coc-ice (master *) build] $ ./p31 forward Tower1024.txt Cpu1024.txt
Threaded CPU Program took: 134.881 seconds.
[ali97@coc-ice (master *) build] $ ./p31 forward Tower2048.txt Cpu2048.txt
Threaded CPU Program took: 1243.01 seconds.
[ali97@coc-ice (master *) build] $ T
```

**Figure 2.** std::Thread's timing for all the sizes

MPI:



```
1. albertoli (ssh)
albertoli (ssh) %1  albertoli (ssh) %2  albertoli (ssh) %3
[ali97@coc-ice (master *) build] $ mpirun -np 8 ./p32 forward Tower128.txt Mpi128.txt
MPI Program took: 0.477474 seconds.
[ali97@coc-ice (master *) build] $ mpirun -np 8 ./p32 forward Tower256.txt Mpi256.txt
MPI Program took: 1.22748 seconds.
[ali97@coc-ice (master *) build] $ mpirun -np 8 ./p32 forward Tower512.txt Mpi512.txt
MPI Program took: 8.2331 seconds.
[ali97@coc-ice (master *) build] $ mpirun -np 8 ./p32 forward Tower1024.txt Mpi1024.txt
MPI Program took: 69.6758 seconds.
[ali97@coc-ice (master *) build] $ mpirun -np 8 ./p32 forward Tower2048.txt Mpi2048.txt
MPI Program took: 595.861 seconds.
[ali97@coc-ice (master *) build] $
```

Figure 3. MPI's timing for all the sizes

CUDA:

### PBS JOB 34728.ice-sched.pace.gatech.edu



System Account <adm@ice-sched.pace.gatech.edu>

6:38 PM



받는 사람: a20154920@gmail.com

PBS Job Id: 34728.ice-sched.pace.gatech.edu  
Job Name: se2\_wootae\_song  
Exec host: rich133-k33-17.pace.gatech.edu/2  
Execution terminated  
Exit\_status=0  
resources\_used.cput=00:00:00  
resources\_used.energy\_used=0  
resources\_used.mem=0kb  
resources\_used.vmem=0kb  
resources\_used.walltime=00:00:00

Error\_Path: rich133-s30-12.pace.gatech.edu:/nv/coc-ice/ali97/ece6122/assignments/hw5/p3/build/output  
Output\_Path: rich133-s30-12.pace.gatech.edu:/nv/coc-ice/ali97/ece6122/assignments/hw5/p3/build/output

Figure 4. CUDA's timing for 128x128

## PBS JOB 34731.ice-sched.pace.gatech.edu



System Account <adm@ice-sched.pace.gatech.edu>

6:41 PM

받는 사람: a20154920@gmail.com

PBS Job Id: 34731.ice-sched.pace.gatech.edu

Job Name: se2\_wootae\_song

Exec host: rich133-k33-17.pace.gatech.edu/1

Execution terminated

Exit\_status=0

resources\_used.cput=00:00:00

resources\_used.energy\_used=0

resources\_used.mem=0kb

resources\_used.vmem=0kb

resources\_used.walltime=00:00:01

Error\_Path: rich133-s30-12.pace.gatech.edu:/nv/coc-ice/ali97/ece6122/assignments/hw5/p3/build/output

Output\_Path: rich133-s30-12.pace.gatech.edu:/nv/coc-ice/ali97/ece6122/assignments/hw5/p3/build/output

**Figure 5.** CUDA's timing for 256x256

## PBS JOB 34732.ice-sched.pace.gatech.edu



System Account <adm@ice-sched.pace.gatech.edu>

6:42 PM

받는 사람: a20154920@gmail.com

PBS Job Id: 34732.ice-sched.pace.gatech.edu

Job Name: se2\_wootae\_song

Exec host: rich133-k33-17.pace.gatech.edu/1

Execution terminated

Exit\_status=0

resources\_used.cput=00:00:00

resources\_used.energy\_used=0

resources\_used.mem=0kb

resources\_used.vmem=0kb

resources\_used.walltime=00:00:01

Error\_Path: rich133-s30-12.pace.gatech.edu:/nv/coc-ice/ali97/ece6122/assignments/hw5/p3/build/output

Output\_Path: rich133-s30-12.pace.gatech.edu:/nv/coc-ice/ali97/ece6122/assignments/hw5/p3/build/output

**Figure 6.** CUDA's timing for 512x512

## PBS JOB 34734.ice-sched.pace.gatech.edu



System Account <adm@ice-sched.pace.gatech.edu>

6:42 PM

받는 사람: a20154920@gmail.com

PBS Job Id: 34734.ice-sched.pace.gatech.edu

Job Name: se2\_wootae\_song

Exec host: rich133-k33-17.pace.gatech.edu/1

Execution terminated

Exit\_status=0

resources\_used.cput=00:00:00

resources\_used.energy\_used=0

resources\_used.mem=0kb

resources\_used.vmem=0kb

resources\_used.walltime=00:00:02

Error\_Path: rich133-s30-12.pace.gatech.edu:/nv/coc-ice/ali97/ece6122/assignments/hw5/p3/build/output

Output\_Path: rich133-s30-12.pace.gatech.edu:/nv/coc-ice/ali97/ece6122/assignments/hw5/p3/build/output

**Figure 7.** CUDA's timing for 1024x1024

## PBS JOB 34736.ice-sched.pace.gatech.edu



System Account <adm@ice-sched.pace.gatech.edu>

6:44 PM



받는 사람: a20154920@gmail.com

PBS Job Id: 34736.ice-sched.pace.gatech.edu

Job Name: se2\_wootae\_song

Exec host: rich133-k33-17.pace.gatech.edu/1

Execution terminated

Exit\_status=0

resources\_used.cput=00:00:00

resources\_used.energy\_used=0

resources\_used.mem=0kb

resources\_used.vmem=0kb

resources\_used.walltime=00:00:06

Error\_Path: rich133-s30-12.pace.gatech.edu:/nv/coc-ice/ali97/ece6122/assignments/hw5/p3/build/output

Output\_Path: rich133-s30-12.pace.gatech.edu:/nv/coc-ice/ali97/ece6122/assignments/hw5/p3/build/output

**Figure 8.** CUDA's timing for 2048x2048

The results of each DFT computing method were compared against the results of the corresponding sizes (128x128, 256x256, ...,2048x2048) and the results of each matched. The table below summarizes the results:

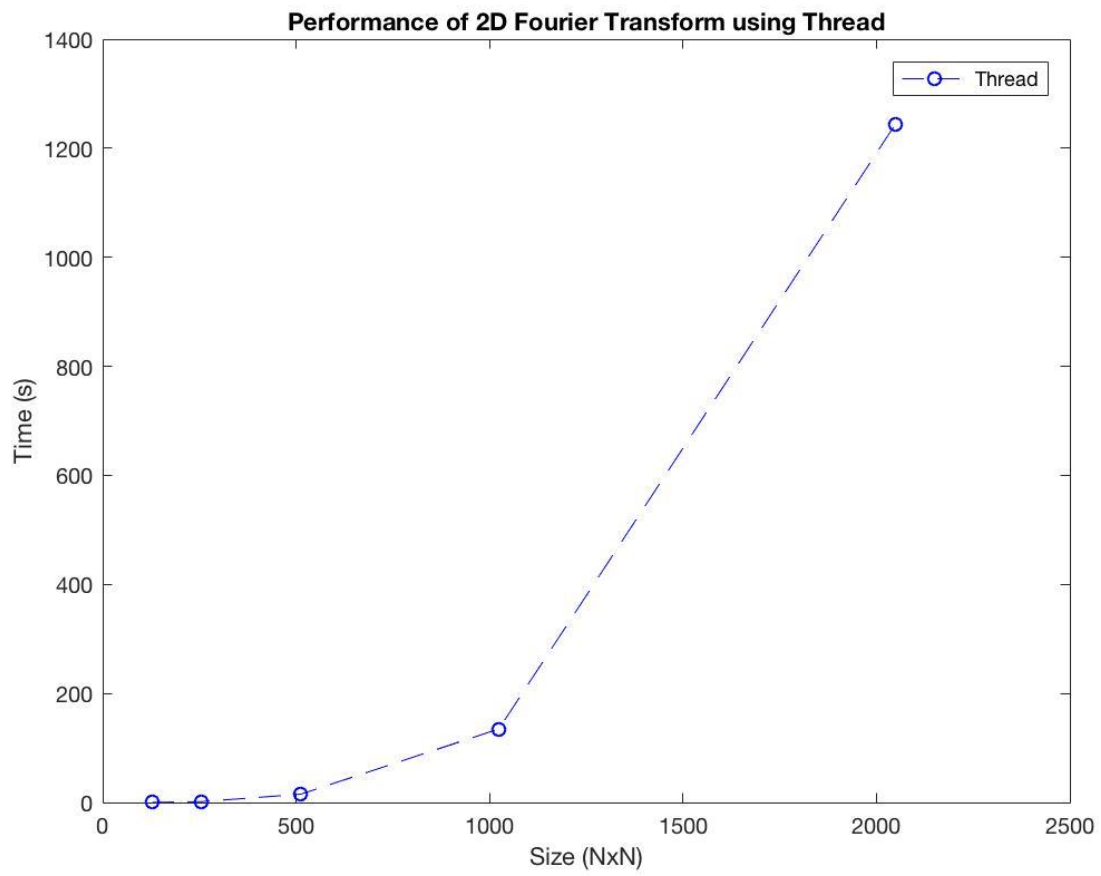
	A	B	C	D
1		p31	p32	p33
2	128	0.148355	0.477474	0
3	256	1.58998	1.22748	1
4	512	15.2248	8.2331	1
5	1024	134.881	69.6758	2
6	2048	1243.01	595.861	6

**Figure 9.** Results matrix in seconds.

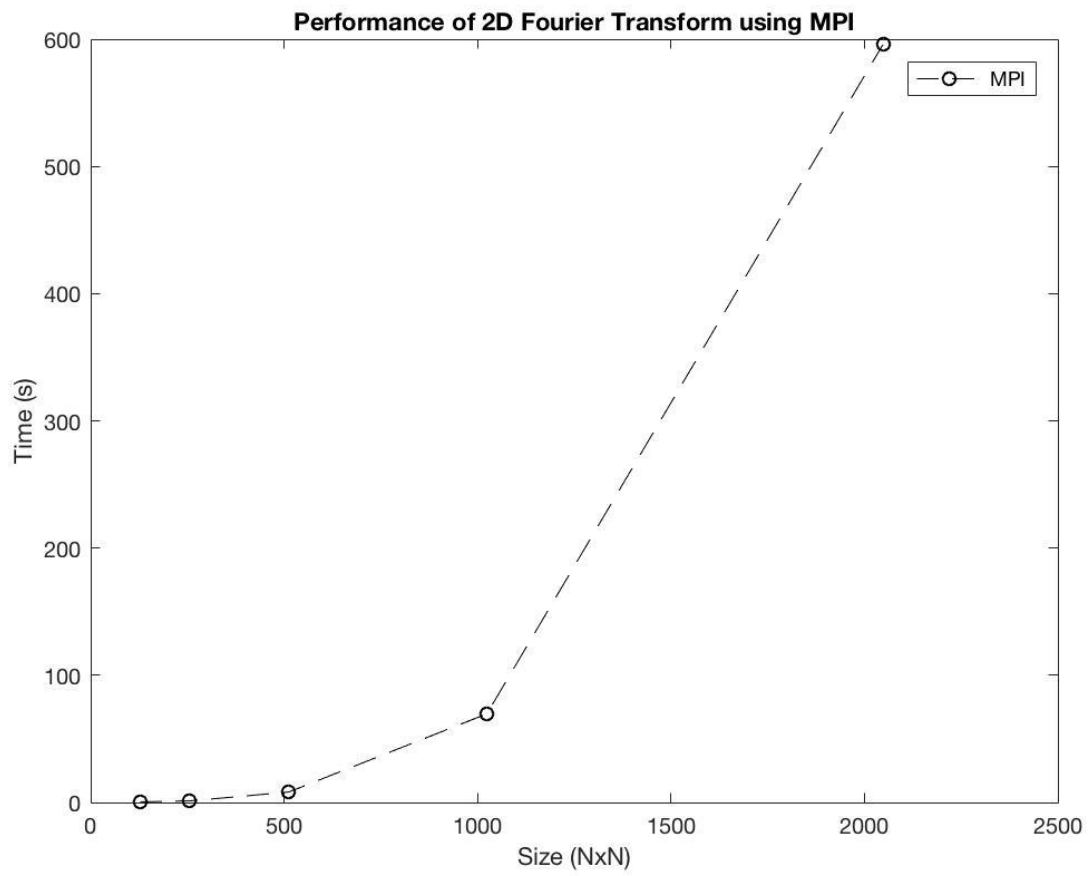
As expected, CUDA took significantly less time than the other methods and std::Thread took the most time among all the methods. For the 2048x2048 DFT, CUDA only took 6 seconds to run while std::Thread took around 21 minutes which was 207 times longer. Additionally for the 2048x2048 DFT, CUDA only took 6 seconds to run while MPI took around 10 minutes which was 99 times longer. Lastly for the 2048x2048 DFT, MPI only took 10 minutes to run while CUDA took around 21 minutes which was 2 times longer. As N increases, the difference in execution time of each of the computing methods will increase drastically as std::Thread is  $O(n^2)$ , MPI is  $O(n \log(n))$ , and CUDA is  $O(1)$ .

These performance comparisons are visualized separately (Figures. 10, 11, 12) and together (Figure. 13). By comparing the runtime results of the three different DFT implementations, the following graphs below were produced.

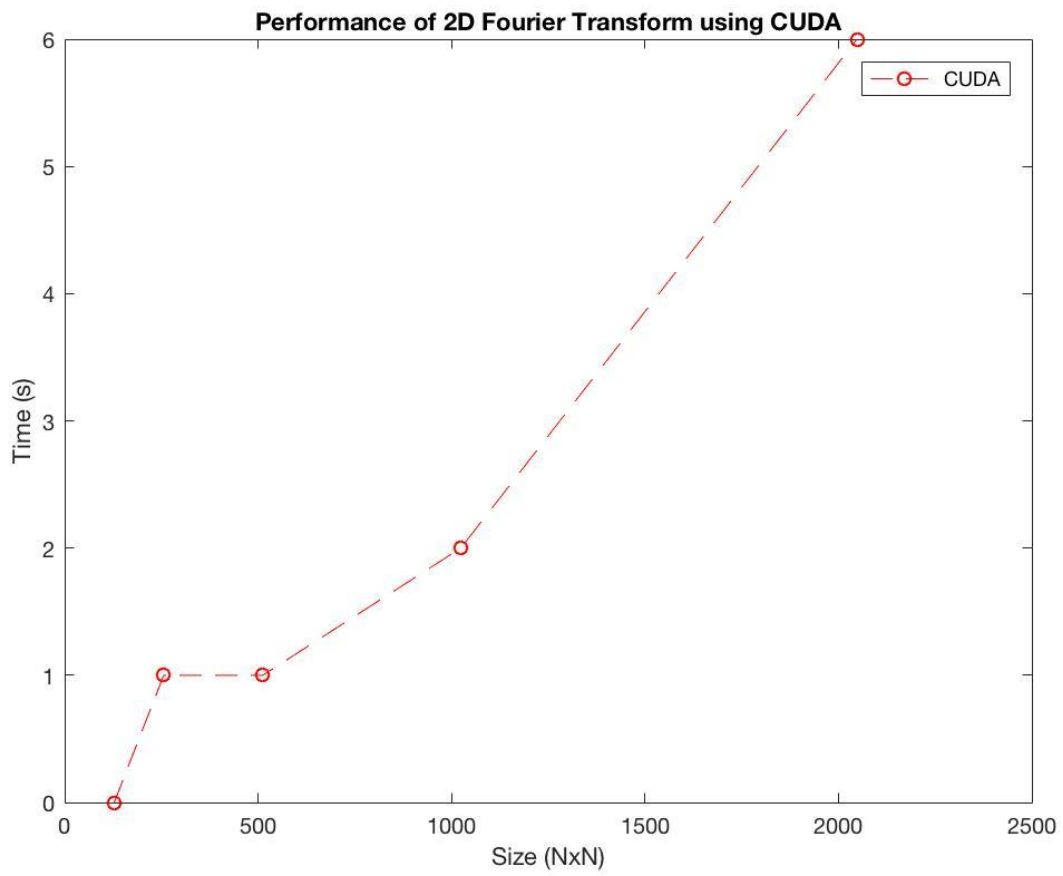




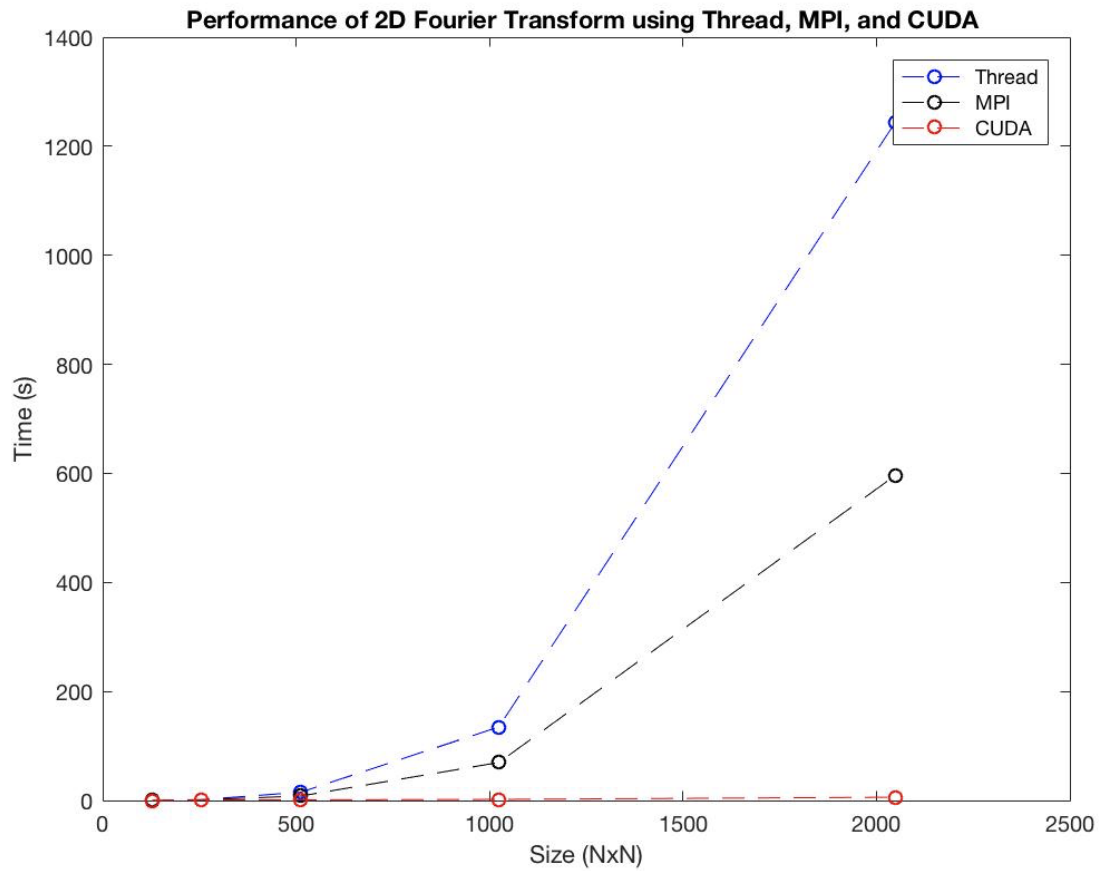
**Figure 10.** Time (s) vs Size (NxN) for std::Thread



**Figure 11.** Time (s) vs Size (NxN) for MPI



**Figure 12.** Time (s) vs Size (NxN) for CUDA



**Figure 13.** Time (s) vs Size (NxN) for std::Thread, MPI, and CUDA

In conclusion, the results of our three unique implementations based on the programming models: std::Thread, MPI, and CUDA suggest that GPU computation was the most efficient, MPI was the second most efficient, and std::Thread was the least efficient.