ECE4122/6122 Final Project Report
Three Approaches to Implementing the Two-Dimensional Fourier Transform using Thread, MPI, and Cuda

Professor: Dr. Brian Swenson

Wootae Song:	wootae@gatech.edu
Alberto Li:	ali97@gatech.edu
Edward Zhou:	edz@gatech.edu
Benjamin Thornbloom:	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

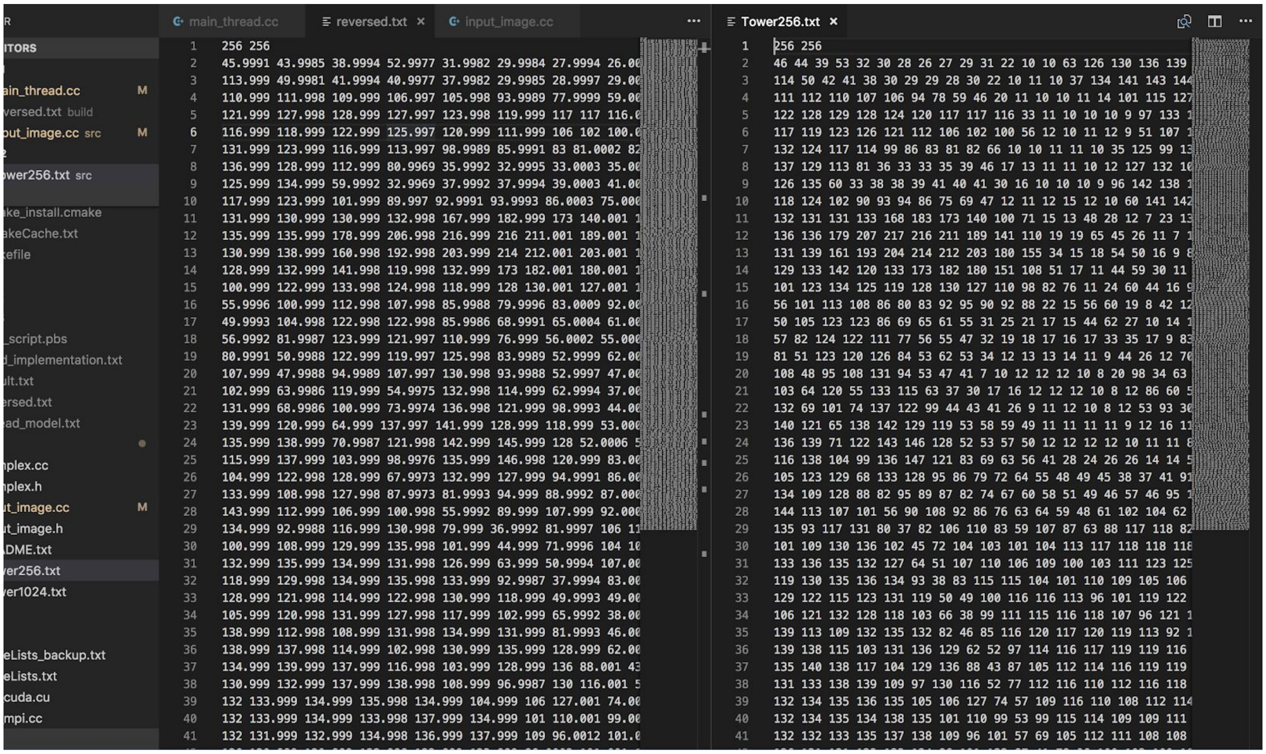
The second implementation of the two-dimensional fourier transform was an MPI approach running on a coc-ice MPI job using a pbs script with 8 MPI ranks. When the source code is compiled, it will produce an executable of the name “p32”. This method was implemented using the Danielson-Lanczos algorithm,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



```
256 256
45.9991 43.9985 38.9994 52.9977 31.9982 29.9984 27.9994 26.00
113.999 49.9981 41.9994 40.9977 37.9982 29.9985 28.9997 29.00
110.999 111.998 109.999 106.997 105.998 93.9989 77.9999 59.00
121.999 127.998 128.999 127.997 123.998 119.999 117 117 116.0
116.999 118.999 122.999 125.997 120.999 111.999 106 102 100.0
131.999 123.999 116.999 113.997 98.9989 85.9991 83 81.0002 82
136.999 128.999 112.999 80.9969 35.9992 32.9995 33.0003 35.00
125.999 134.999 59.9992 32.9969 37.9992 37.9994 39.0003 41.00
117.999 123.999 101.999 89.997 92.9991 93.9993 86.0003 75.000
131.999 130.999 130.999 132.998 167.999 182.999 173 140.001 1
135.999 135.999 178.999 206.998 216.999 216 211.001 189.001 1
130.999 138.999 160.998 192.998 203.999 214 212.001 203.001 1
128.999 132.999 141.998 119.998 132.999 173 182.001 180.001 1
100.999 122.999 133.998 124.998 118.999 128 130.001 127.001 1
55.9996 100.999 112.998 107.998 85.9988 79.9996 83.0009 92.00
49.9993 104.998 122.998 122.998 85.9986 68.9991 65.0004 61.00
56.9992 81.9987 123.999 121.997 110.999 76.999 56.0002 55.000
80.9991 50.9988 122.999 119.997 125.998 83.9989 52.9999 62.00
107.999 47.9988 94.9989 107.997 130.998 93.9988 52.9997 47.00
102.999 63.9986 119.999 54.9975 132.998 114.999 62.9994 37.00
131.999 68.9986 100.999 73.9974 136.998 121.999 98.9993 44.00
139.999 120.999 64.999 137.997 141.999 128.999 118.999 53.000
135.999 138.999 70.9987 121.998 142.999 145.999 128 52.0006 5
115.999 137.999 103.999 98.9976 135.999 146.998 120.999 83.00
104.999 122.998 128.999 67.9973 132.999 127.999 94.9991 86.00
133.999 108.998 127.998 87.9973 81.9993 94.999 88.9992 87.000
143.999 112.999 106.999 100.998 55.9992 89.999 107.999 92.000
134.999 92.9988 116.999 130.998 79.999 36.9992 81.9997 106 11
100.999 108.999 129.999 135.998 101.999 44.999 71.9996 104 10
132.999 135.999 134.999 131.998 126.999 63.999 50.9994 107.00
118.999 129.998 134.999 135.998 133.999 92.9987 37.9994 83.00
128.999 121.998 114.999 122.998 130.999 118.999 49.9993 49.00
105.999 120.998 131.999 127.998 117.999 102.999 65.9992 38.00
138.999 112.998 108.999 131.998 134.999 131.999 81.9993 46.00
138.999 137.998 114.999 102.998 130.999 135.999 128.999 62.00
134.999 139.999 137.999 116.998 103.999 128.999 136 88.001 43
130.999 132.999 137.999 138.998 108.999 96.9987 130 116.001 5
132 133.999 134.999 135.998 134.999 104.999 106 127.001 74.00
132 133.999 134.999 133.998 137.999 134.999 101 110.001 99.00
132 131.999 132.999 134.998 136.999 137.999 109 96.0012 101.0
```

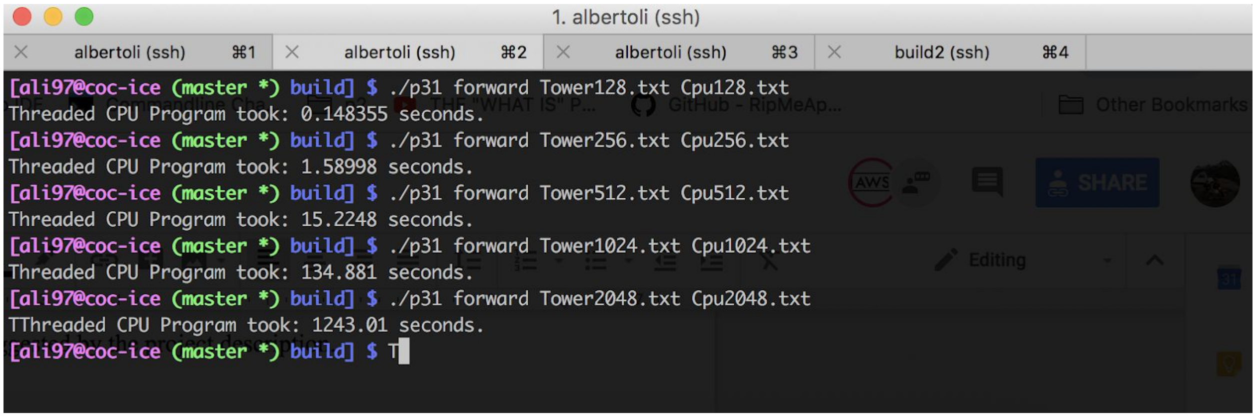
```
256 256
46 44 39 53 32 30 28 26 27 29 31 22 10 10 63 126 130 136 139
114 50 42 41 38 30 29 29 28 30 22 10 11 10 37 134 141 143 144
111 112 110 107 106 94 78 59 46 20 11 10 10 11 14 101 115 127
122 128 129 128 124 120 117 117 116 33 11 10 10 10 9 97 133 1
117 119 123 126 121 112 106 102 100 56 12 10 11 12 9 51 107 1
132 124 117 114 99 86 83 81 82 66 10 10 11 11 10 35 125 99 13
137 129 113 81 36 33 33 35 39 46 17 13 11 11 10 12 127 132 10
126 135 60 33 38 38 39 41 40 41 30 16 10 10 10 9 96 142 138 1
118 124 102 90 93 94 86 75 69 47 12 11 12 15 12 10 60 141 142
132 131 131 133 168 183 173 140 100 71 15 13 48 28 12 7 23 13
136 136 179 207 217 216 211 189 141 110 19 19 65 45 26 11 7 1
131 139 161 193 204 214 212 203 180 155 34 15 18 54 50 16 9 8
129 133 142 120 133 173 182 180 151 108 51 17 11 44 59 30 11
101 123 134 125 119 128 130 127 110 98 82 76 11 24 60 44 16 9
56 101 113 108 86 80 83 92 95 90 92 88 22 15 56 60 19 8 42 12
50 105 123 123 86 69 65 61 55 31 25 21 17 15 44 62 27 10 14 1
57 82 124 122 111 77 56 55 47 32 19 18 17 16 17 33 35 17 9 83
81 51 123 120 126 84 53 62 53 34 12 13 13 14 11 9 44 26 12 70
108 48 95 108 131 94 53 47 41 7 10 12 12 12 10 8 20 98 34 63
103 64 120 55 133 115 63 37 30 17 16 12 12 12 10 8 12 86 60 5
132 69 101 74 137 122 99 44 43 41 26 9 11 12 10 8 12 53 93 30
140 121 65 138 142 129 119 53 58 59 49 11 11 11 11 9 12 16 11
136 139 71 122 143 146 128 52 53 57 50 12 12 12 12 10 11 11 8
116 138 104 99 136 147 121 83 69 63 56 41 28 24 26 26 14 14 5
105 123 129 68 133 128 95 86 79 72 64 55 48 49 45 38 37 41 91
134 109 128 88 82 95 89 87 82 74 67 60 58 51 49 46 57 46 95 1
144 113 107 101 56 90 108 92 86 76 63 64 59 48 61 102 104 62
135 93 117 131 80 37 82 106 110 83 59 107 87 63 88 117 118 82
101 109 130 136 102 45 72 104 103 101 104 113 117 118 118 118
133 136 135 132 127 64 51 107 110 106 109 100 103 111 123 125
119 130 135 136 134 93 38 83 115 115 104 101 110 109 105 106
129 122 115 123 131 119 50 49 100 116 116 113 96 101 119 122
106 121 132 128 118 103 66 38 99 111 115 116 118 107 96 121 1
139 113 109 132 135 132 82 46 85 116 120 117 120 119 113 92 1
139 138 115 103 131 136 129 62 52 97 114 116 117 119 119 116
135 140 138 117 104 129 136 88 43 87 105 112 114 116 119 119
131 133 138 139 109 97 130 116 52 77 112 116 110 112 116 118
132 134 135 136 135 105 106 127 74 57 109 116 110 108 112 114
132 134 135 134 138 135 101 110 99 53 99 115 114 109 109 111
132 132 133 135 137 138 109 96 101 57 69 105 112 111 108 108
```

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.

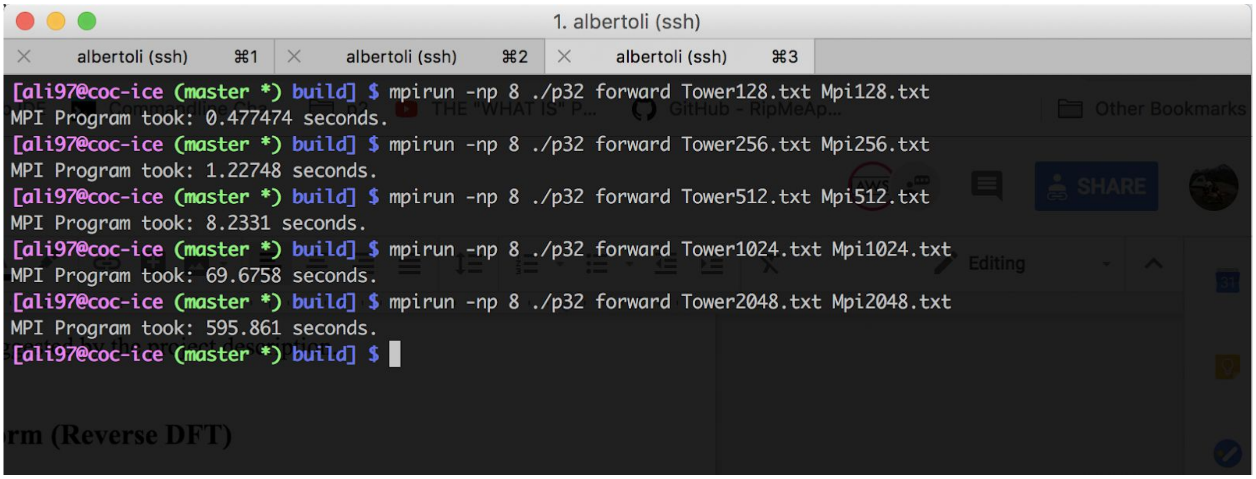
std::Thread:



```
[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
TThreaded CPU Program took: 1243.01 seconds.
[ali97@coc-ice (master *) build] $ T
```

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

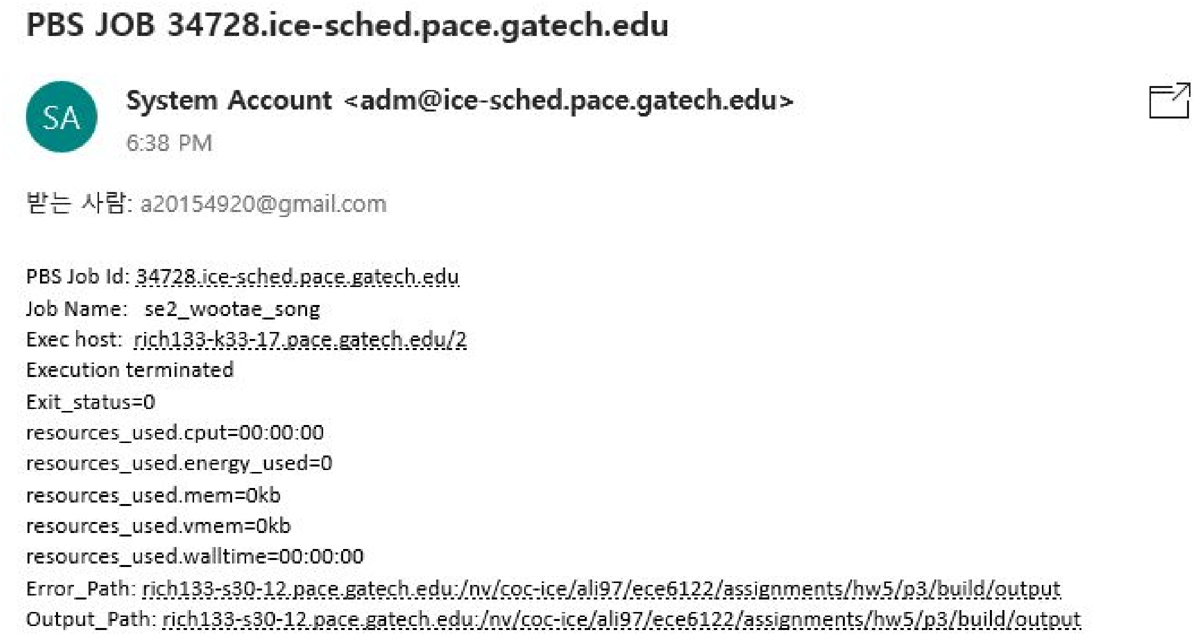
MPI:



```
[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:



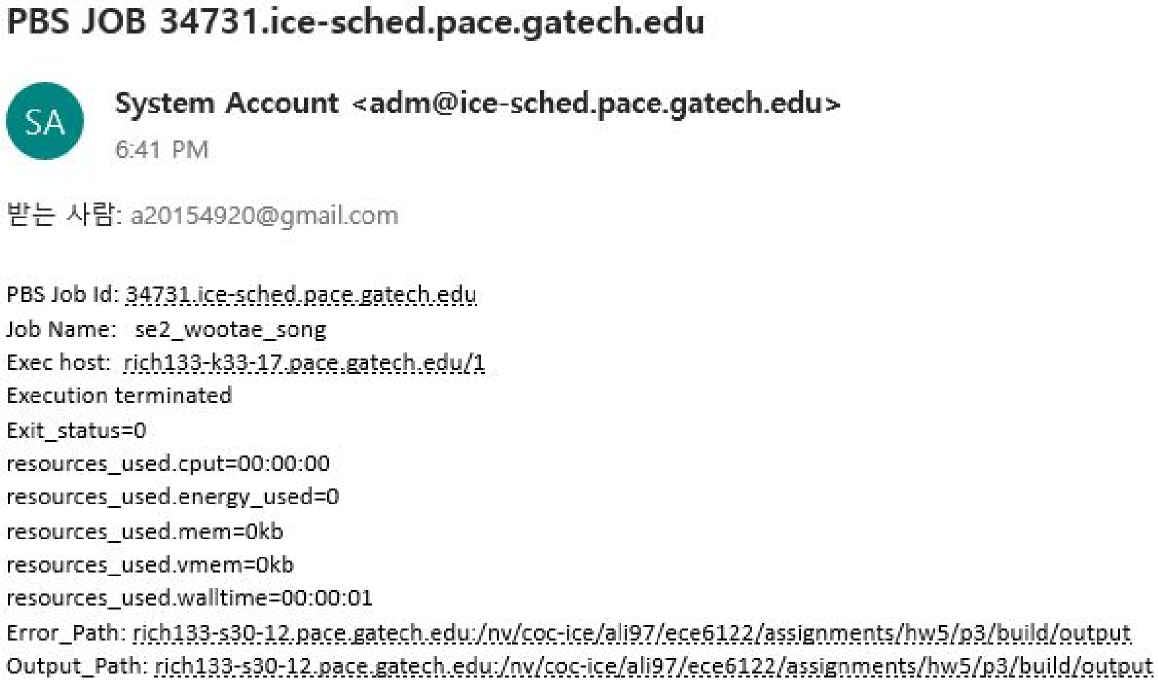
PBS JOB 34728.ice-sched.pace.gatech.edu

SA 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



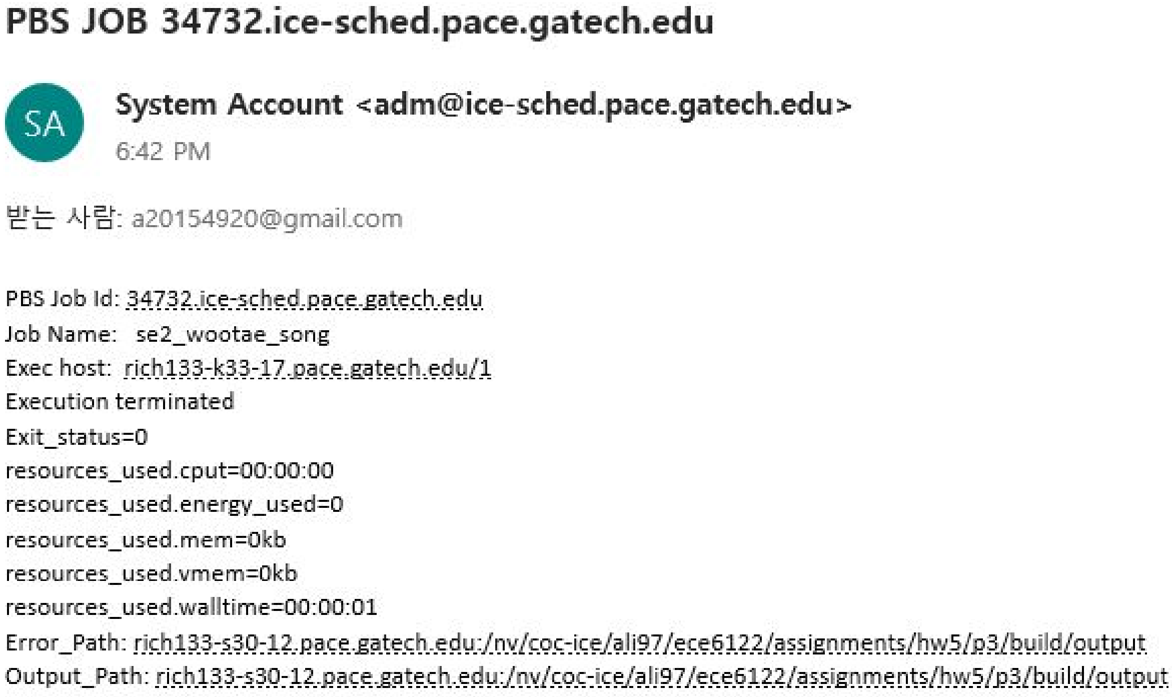
PBS JOB 34731.ice-sched.pace.gatech.edu

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



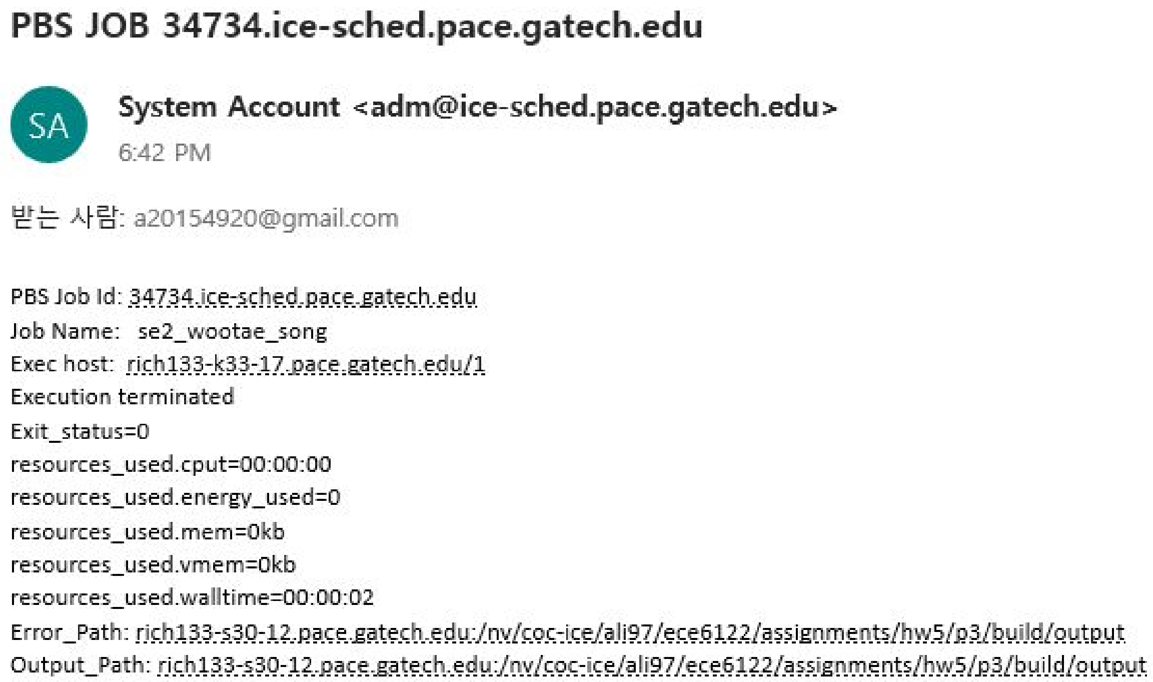
PBS JOB 34732.ice-sched.pace.gatech.edu

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



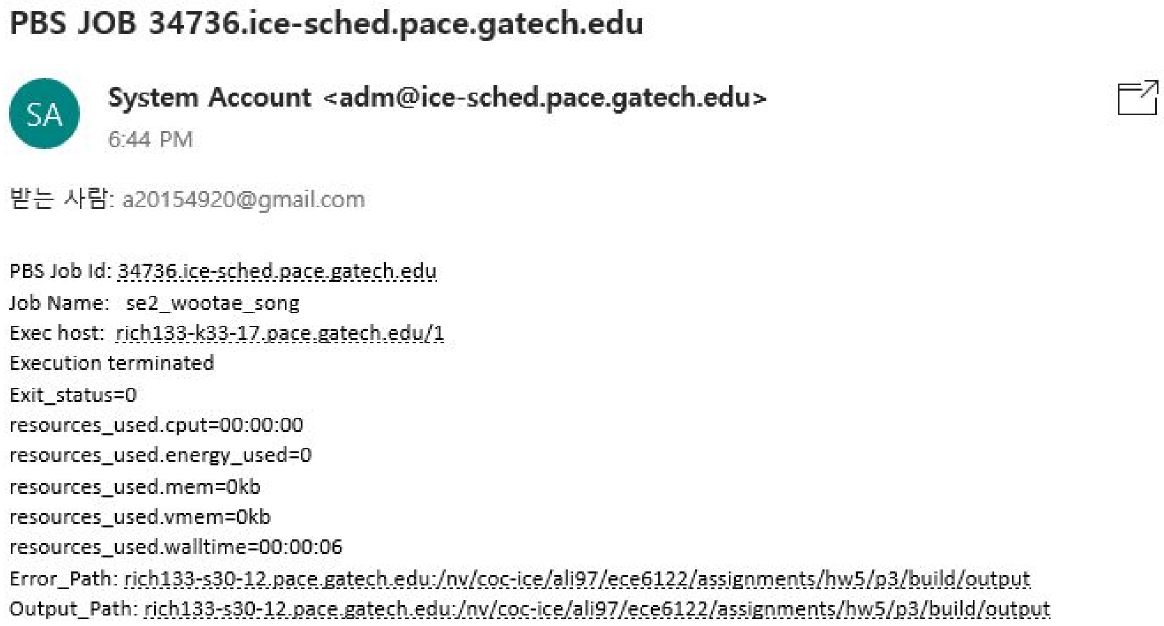
PBS JOB 34734.ice-sched.pace.gatech.edu

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

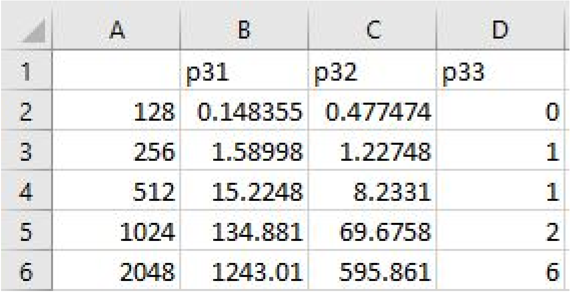
System Account <adm@ice-sched.pace.gatech.edu>
6:44 PM

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($nlog(n)$), 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.

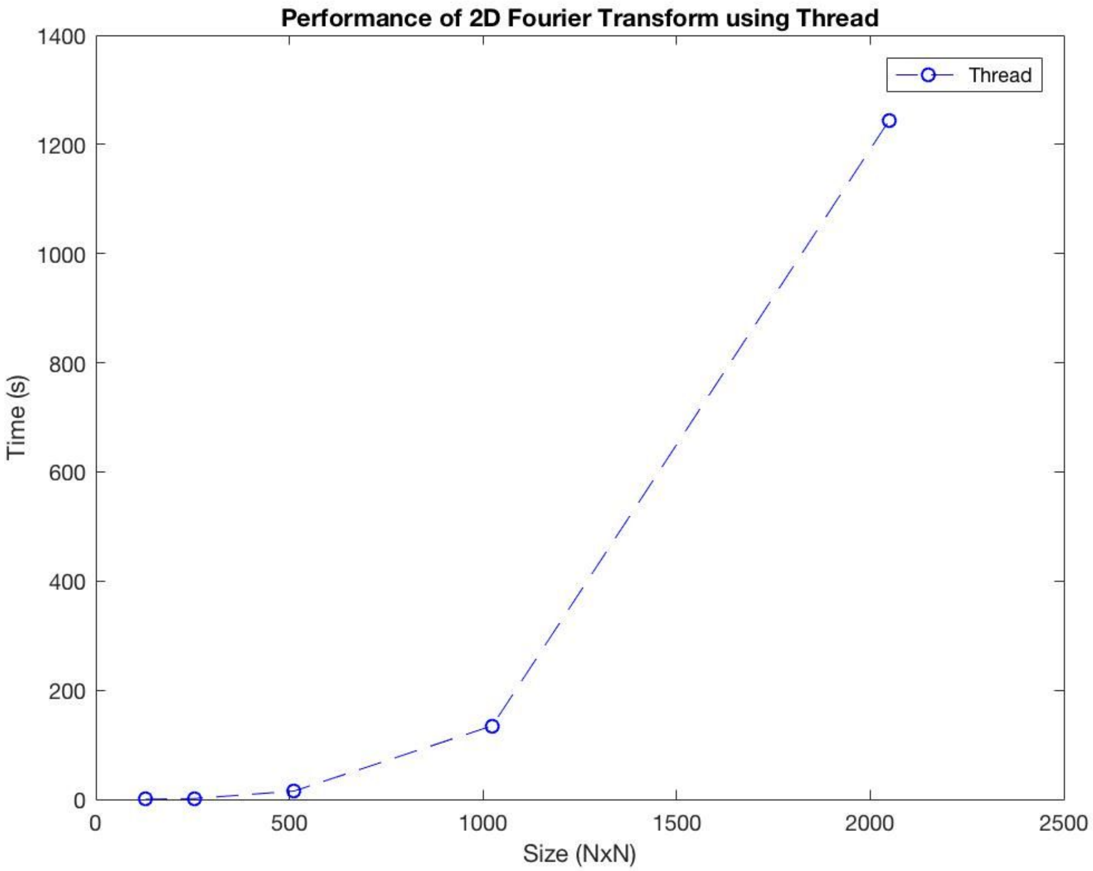


Figure 10. Time (s) vs Size (NxN) for std::Thread

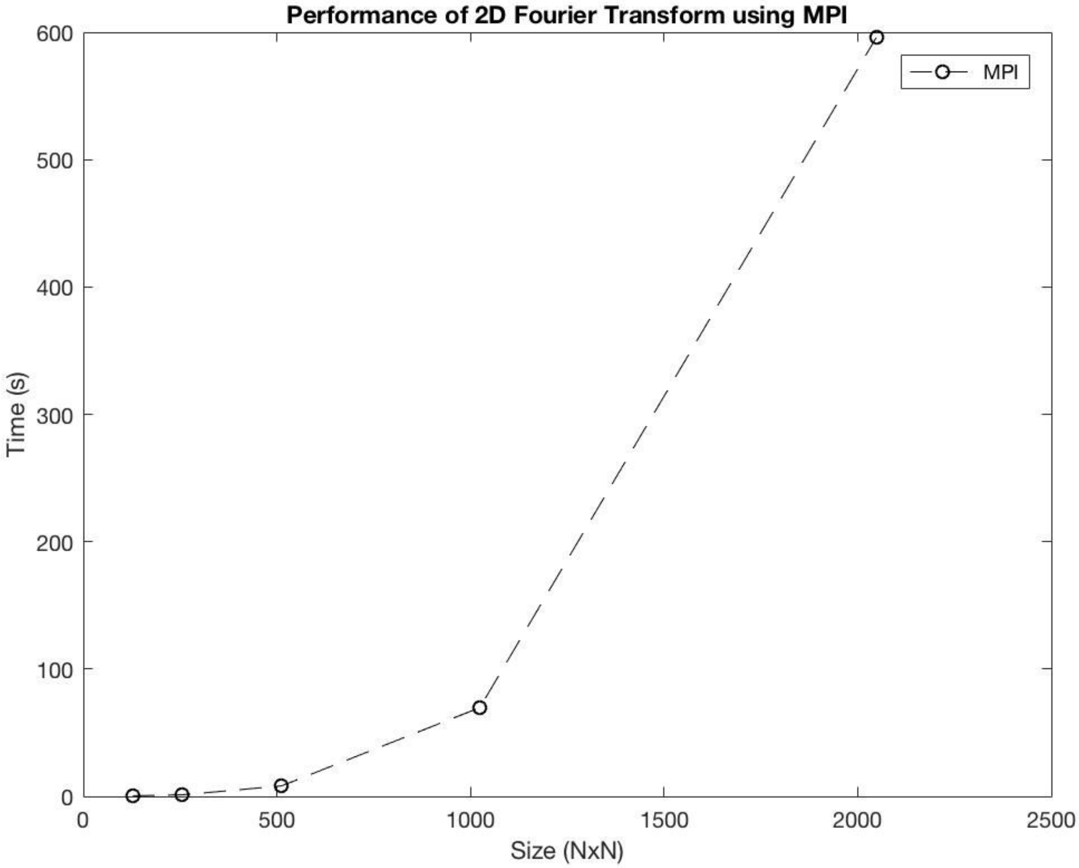


Figure 11. Time (s) vs Size (NxN) for MPI

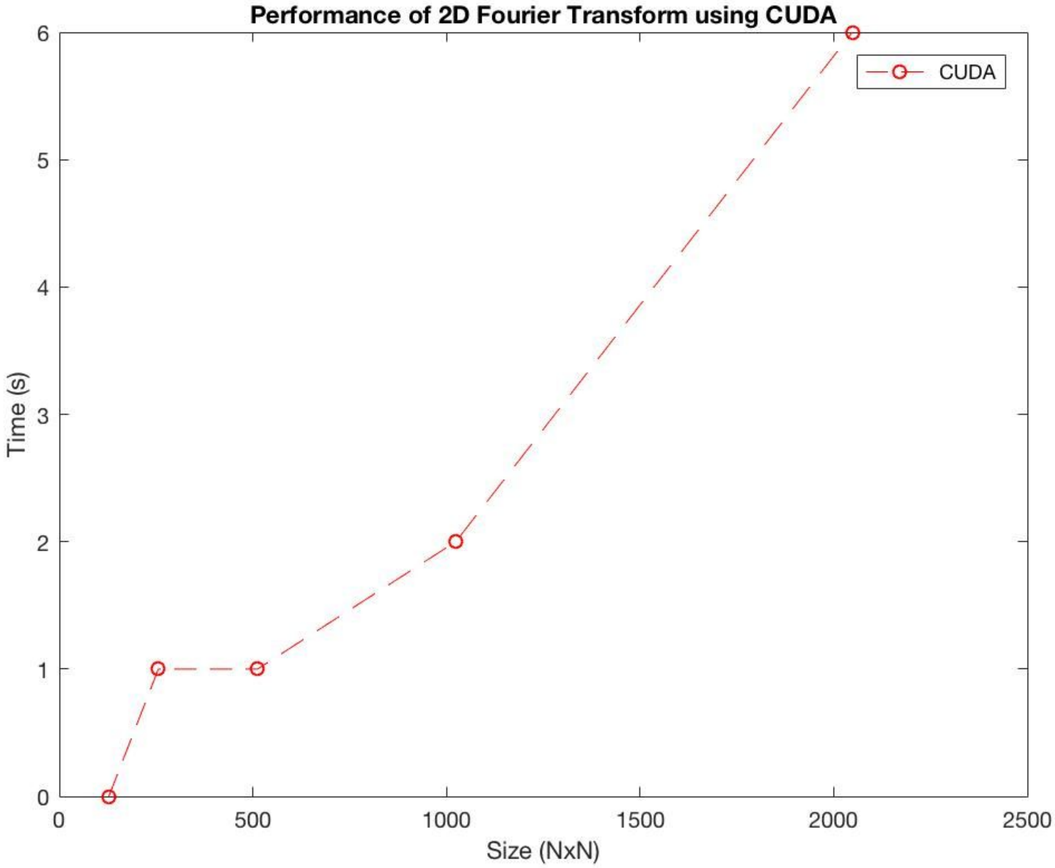


Figure 12. Time (s) vs Size (NxN) for CUDA

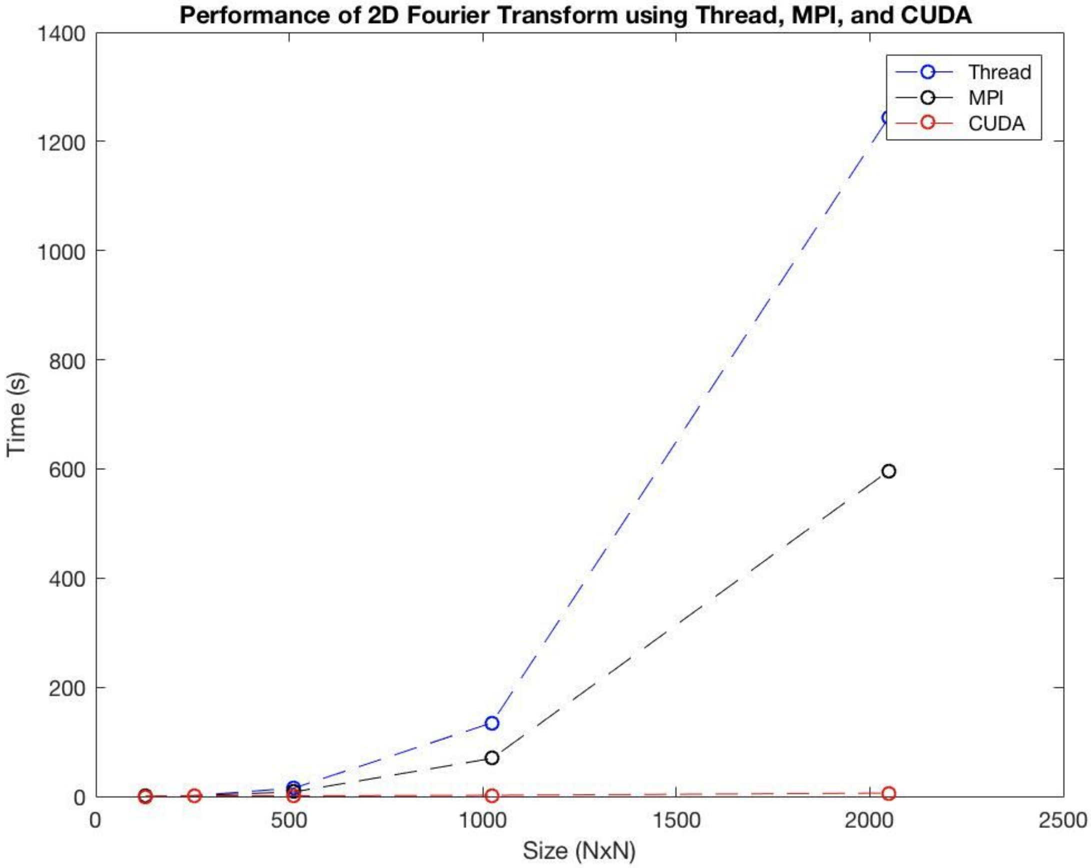


Figure 13. Time (s) vs Size (NxN) for std::Thread, MPI, and CUDA

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.