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

Wootae Song: Alberto Li: Edward Zhou: Benjamin Thornbloom:

wootae@gatech.edu ali97@gatech.edu edz@gatech.edu 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.

R			≣ reversed.txt ×			• = 1	Tower256.txt ×		ୟ	⊞ …
ITORS		1 256 256			10002	+ :	1 256 256		lî li	NUMBER OF
W.		2 45.9991 43.9	985 38.9994 52.9977	31.9982 29.9984 27.9	9994 26.00		2 46 44 39 53 32 30 2	28 26 27 29 31 22 10 10 63 126 3	130 136 139 🛛	
		3 113.999 49.9	981 41.9994 40.9977	37.9982 29.9985 28.9	9997 29.00		3 114 50 42 41 38 30	29 29 28 30 22 10 11 10 37 134	141 143 144	
ain_thread.cc	M	4 110.999 111.	998 109.999 106.997	105.998 93.9989 77.9	9999 59.00		4 111 112 110 107 106	5 94 78 59 46 20 11 10 10 11 14	101 115 127	
versed.txt build		5 121.999 127.	998 128.999 127.997	123.998 119.999 117	117 116.6		5 122 128 129 128 124	120 117 117 116 33 11 10 10 10	<b>9 97 133</b> 1	
out_image.cc src	м	6 116.999 118.	999 122.999 125.997	120.999 111.999 106	102 100.0		6 117 119 123 126 121	L 112 106 102 100 56 12 10 11 12	2 9 51 107 1	<i>lennim</i> -
		7 131.999 123.	999 116.999 113.997	98.9989 85.9991 83	31.0002 82		7 132 124 117 114 99	86 83 81 82 66 10 10 11 11 10 3	35 125 99 13	
wer256.txt src		8 136.999 128.	999 112.999 80.9969	35.9992 32.9995 33.0	3003 35.00		8 137 129 113 81 36 3	33 33 35 39 46 17 13 11 11 10 13	2 127 132 16	
		9 125.999 134.	999 59.9992 32.9969	37.9992 37.9994 39.0	0003 41.00		9 126 135 60 33 38 38	3 39 41 40 41 30 16 10 10 10 9 9	96 142 138 1	
ke install.cmake		10 117.999 123.	999 101.999 89.997	92.9991 93.9993 86.00	303 75.000		0 118 124 102 90 93 9	94 86 75 69 47 12 11 12 15 12 10	0 60 141 142	
-laCasha tat		11 131.999 130.	999 130.999 132.998	167.999 182.999 173	140.001 1		1 132 131 131 133 168	3 183 173 140 100 71 15 13 48 2	3 12 7 23 13	
akeCache.txt		12 135.999 135.	999 178.999 206.998	216.999 216 211.001	189.001 1		2 136 136 179 207 217	7 216 211 189 141 110 19 19 65 4	45 26 11 7 1	
tefile		13 130.999 138.	999 160.998 192.998	203.999 214 212.001	203.001		3 131 139 161 193 204	214 212 203 180 155 34 15 18 :	54 50 16 9 8	
			999 141.998 119.998	132.999 1/3 182.001	180.001	14	4 129 133 142 120 133	3 1/3 182 180 151 108 51 1/ 11 4	44 59 30 11	
8		100.999 122.	999 133.998 124.998	118,999 128 130,001	127.001	•		3 128 130 127 110 98 82 70 11 24	00 44 10 9	
di la constante de la constante		17 40 0003 104	000 122 000 122 000	05.9900 /9.9990 05.	0009 92.00	1.	7 50 105 103 100 00 0	50 65 61 55 21 25 21 17 15 44 6	2 27 10 14 1	<u>ANNER</u>
ecript phe		19 56 0002 81 0	087 173 000 121 007	110 000 76 000 56 0	004 01.00		8 57 82 124 122 111 7	7 56 55 47 32 10 18 17 16 17 3	3 35 17 0 83	
		19 80.9991 50.9	988 177 999 119 997	125 998 83 9989 52 9	999 62 80	10	9 81 51 123 120 126 8	84 53 62 53 34 12 13 13 14 11 9	44 26 12 70	
_implementation.txt		107.999 47.0	988 94.9989 107.997	130.998 93.9988 52.9	9997 47.00	21	108 48 95 108 131 9	4 53 47 41 7 10 12 12 12 10 8	20 98 34 63	
ilt.txt		102.999 63.9	986 119,999 54,9975	132,998 114,999 62.9	9994 37.00		1 103 64 120 55 133 1	115 63 37 30 17 16 12 12 12 10 1	8 12 86 60 5	
ersed.txt		131.999 68.9	986 100.999 73.9974	136,998 121,999 98.9	9993 44.00		2 132 69 101 74 137 1	122 99 44 43 41 26 9 11 12 10 8	12 53 93 30	
ad_model.txt		139.999 120.	999 64.999 137.997	141.999 128.999 118.	999 53.000		3 140 121 65 138 142	129 119 53 58 59 49 11 11 11 11	1 9 12 16 11	
		135.999 138.	999 70.9987 121.998	142.999 145.999 128	52.0006 5		4 136 139 71 122 143	146 128 52 53 57 50 12 12 12 12	2 10 11 11 8	
nlex cc		115.999 137.	999 103.999 98.9976	135.999 146.998 120	.999 83.00		5 116 138 104 99 136	147 121 83 69 63 56 41 28 24 20	6 26 14 14 5	
		26 104.999 122.	998 128.999 67.9973	132.999 127.999 94.9	9991 86.00		6 105 123 129 68 133	128 95 86 79 72 64 55 48 49 45	38 37 41 91	
ipiex.n		27 <b>133.999 108</b> .	998 127.998 87.9973	81.9993 94.999 88.99	992 87.000		7 134 109 128 88 82 9	95 89 87 82 74 67 60 58 51 49 40	5 57 46 95 1	ABOODINE.
it_image.cc	M :	28 <b>143.999 112</b> .	999 106.999 100.998	55.9992 89.999 107.9	999 92.006		8 144 113 107 101 56	90 108 92 86 76 63 64 59 48 61	102 104 62	
ıt_image.h		134.999 92.9	988 116.999 130.998	79.999 36.9992 81.99	997 106 11 <sup>11111111111111111111111111111111</sup>		9 135 93 117 131 80 3	37 82 106 110 83 59 107 87 63 8	3 117 118 82 <sup>   </sup>	111111111111111111111111111111111111111
DME.txt		100.999 108.	999 129.999 135.998	101.999 44.999 71.99	996 104 10		0 101 109 130 136 102	2 45 72 104 103 101 104 113 117	118 118 118	
ver256.txt		<b>132.999 135</b> .	999 134.999 131.998	126.999 63.999 50.99	994 107.00		1 133 136 135 132 127	7 64 51 107 110 106 109 100 103	111 123 125	
er1024 tyt		32 <b>118.999 129</b> .	998 134.999 135.998	133.999 92.9987 37.9	9994 83.00		2 119 130 135 136 134	4 93 38 83 115 115 104 101 110 :	109 105 106	
611024.1.1		<b>128.999 121</b> .	998 114.999 122.998	130.999 118.999 49.9	9993 49.00		3 129 122 115 123 131	1 119 50 49 100 116 116 113 96 3	101 119 122	
		34 <b>105.999 120</b> .	998 131.999 127.998	117.999 102.999 65.9	9992 38.00	34	4 106 121 132 128 118	3 103 66 38 99 111 115 116 118 :	107 96 121 1	
		35 138.999 112.	998 108.999 131.998	134.999 131.999 81.9	3993 45.00		5 139 113 109 132 135	5 132 82 46 85 116 120 117 120 1	119 113 92 1	
eLists_backup.txt		138.999 137.	998 114.999 102.998	130.999 135.999 128	999 62.00	30		1 130 129 62 52 97 114 116 117	119 119 116	
eLists.txt		37 134.999 139.	999 137.999 116.998	103.999 128.999 136	116 001 43	3	135 140 138 11/ 104	+ 129 130 88 43 8/ 105 112 114	110 119 119	
cuda.cu		130,999 132.	134 000 135 009 134	100.999 90.998/ 130	110.001 5	31	0 131 133 138 139 109	<u>9 9/ 130 110 32 // 112 116 110 .</u>	100 112 114	
mni co		10 122 133,999	134.999 133.996 134	000 134 000 101 110	001 74.00 001 00 00	3	a 122 134 135 130 133 a 122 124 125 124 129	0 105 101 110 00 53 00 115 114 1	100 100 111	
mpi.ce		11 132 131 000	132,999 134,998 137	.999 137.999 101 110	1001 33.00 1012 101.0	4	1 132 132 133 133 134 136	7 138 109 96 101 57 69 105 112	111 108 108	
		102 101.000	1021000 1041000 100	1999 1911993 109 3011			102 102 105 105 10	100 100 00 101 07 00 100 112	111 100 100	

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)	策1	imes al	bertoli (ssh)	₩2 ×	albertoli (ss	h) #3	×	build2 (ssh)	<b>#</b> 4		
<b>[ali9</b> Threa	7@coc-ice (mas ided CPU Progra	s <b>ter *</b> ) am tool	) <b>build]</b> k: 0.148	<pre>\$ ./p31 for 355 seconds</pre>	rward Tow	er128.txt Cpu	128.txt	Ap			ther Bot	okmarks
[ali9	7@coc-ice (mas	ster *)	) build]	\$ ./p31 for	rward Tow	er256.txt Cpu	256.txt					
Threa	ded CPU Progra	am tool	c: 1.5899	98 seconds.								1000
[ali9	7@coc-ice (mas	ster *)	) build]	\$ ./p31 for	rward Tow	er512.txt Cpu	512.txt					
Threa	ded CPU Progra	am tool	k: 15.224	48 seconds.								
[ali9	7@coc-ice (mas	ster *	) build]	\$ ./p31 for	rward Tow	er1024.txt Cp	u1024.txt	i i				
Threa	ded CPU Progra	am tool	<: 134.88	81 seconds.								101
[ali9	7@coc-ice (mas	ster *)	) build]	\$ ./p31 for	rward Tow	er2048.txt Cp	u2048.txt					
TThre	aded CPU Progr	am too	ok: 1243	.01 seconds.								
[ali9	7@coc-ice (mas	ster *)	build]	<u>\$</u> T								

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

MPI:

			1.	albertoli (ssh)				
$\times$	albertoli (ssh)	imes albertoli (ss	h) %2 ×	albertoli (ssh)	жз			
<b>[al</b> MPI	i <b>97@coc-ice (master *</b> Program took: 0.4774	<b>) build] \$</b> mpiru 74 seconds.	<b>IN -NP 8 .∕P3</b> HE WHAT IS	2 forward Tower	r <b>128.txt</b> - RipMeA	Mpi128.txt	🛅 Other I	Bookmarks
<b>[al</b> MPI	1 <b>97@coc-ice (master *</b> Program took: 1.2274	) build] \$ mpiru 8 seconds.	ın -np 8 ./p3	2 forward Towe	r256.txt	Mpi256.txt		49
[al MPI	197@coc-ice (master * Program took: 8.2331	) build] \$ mpiru seconds.	ın -np 8 ./p3	2 forward Towe	r512.txt	Mpi512.txt		
LaL MPI	Program took: 69.675	) build] \$ mpiru 8 seconds.	in -np 8 ./p3	2 forward lower	r1024.txt	t Mpi1024.txt		N 31
LaL: MPI	Program took: 595.86	) build] \$ mpiru 1 seconds.	ın -np 8 .∕p3	2 torward lowe	r2048.txt	t Mpi2048.txt		
lar	.97@coc-ice (master *	) build] \$						
rm	(Reverse DFT)							

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: <u>34728.ice-sched.pace.gatech.edu</u> Job Name: se2\_wootae\_song Exec host: <u>rich133-k33-17.pace.gatech.edu/2</u> Execution terminated Exit\_status=0 resources\_used.cput=00:00:00 resources\_used.energy\_used=0 resources\_used.mem=0kb resources\_used.walltime=00:00:00 Error\_Path: <u>rich133-s30-12.pace.gatech.edu:/nv/coc-ice/ali97/ece6122/assignments/hw5/p3/build/output</u> Output\_Path: <u>rich133-s30-12.pace.gatech.edu:/nv/coc-ice/ali97/ece6122/assignments/hw5/p3/build/output</u>

Figure 4. CUDA's timing for 128x128

67

# 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: <u>34731.ice-sched.pace.gatech.edu</u> Job Name: se2\_wootae\_song Exec host: <u>rich133-k33-17.pace.gatech.edu/1</u> Execution terminated Exit\_status=0 resources\_used.cput=00:00:00 resources\_used.energy\_used=0 resources\_used.mem=0kb resources\_used.walltime=00:00:01 Error\_Path: <u>rich133-s30-12.pace.gatech.edu:/nv/coc-ice/ali97/ece6122/assignments/hw5/p3/build/output</u> Output\_Path: <u>rich133-s30-12.pace.gatech.edu:/nv/coc-ice/ali97/ece6122/assignments/hw5/p3/build/output</u>

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: <u>34732.ice-sched.pace.gatech.edu</u> Job Name: se2\_wootae\_song Exec host: <u>rich133-k33-17.pace.gatech.edu/1</u> Execution terminated Exit\_status=0 resources\_used.cput=00:00:00 resources\_used.energy\_used=0 resources\_used.mem=0kb resources\_used.walltime=00:00:01 Error\_Path: <u>rich133-s30-12.pace.gatech.edu:/nv/coc-ice/ali97/ece6122/assignments/hw5/p3/build/output</u> Output\_Path: <u>rich133-s30-12.pace.gatech.edu:/nv/coc-ice/ali97/ece6122/assignments/hw5/p3/build/output</u>

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: <u>34734.ice-sched.pace.gatech.edu</u> Job Name: se2\_wootae\_song Exec host: <u>rich133-k33-17.pace.gatech.edu/1</u> Execution terminated Exit\_status=0 resources\_used.cput=00:00:00 resources\_used.energy\_used=0 resources\_used.mem=0kb resources\_used.mem=0kb resources\_used.walltime=00:00:02 Error\_Path: <u>rich133-s30-12.pace.gatech.edu:/nv/coc-ice/ali97/ece6122/assignments/hw5/p3/build/output</u> Output\_Path: <u>rich133-s30-12.pace.gatech.edu:/nv/coc-ice/ali97/ece6122/assignments/hw5/p3/build/output</u>

Figure 7. CUDA's timing for 1024x1024

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



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

받는 사람: a20154920@gmail.com

6:44 PM

PBS Job Id: <u>34736.ice-sched.pace.gatech.edu</u> Job Name: se2\_wootae\_song Exec host: <u>rich133-k33-17.pace.gatech.edu/1</u> Execution terminated Exit\_status=0 resources\_used.cput=00:00:00 resources\_used.energy\_used=0 resources\_used.mem=0kb resources\_used.wem=0kb resources\_used.wem=0kb resources\_used.walltime=00:00:06 Error\_Path: <u>rich133-s30-12.pace.gatech.edu:/nv/coc-ice/ali97/ece6122/assignments/hw5/p3/build/output</u> Output\_Path: <u>rich133-s30-12.pace.gatech.edu:/nv/coc-ice/ali97/ece6122/assignments/hw5/p3/build/output</u>

Figure 8. CUDA's timing for 2048x2048

67

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:

	А	В	С	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.