Zhuo Sheng Chin： zchin7

Xunchen Zhou: xzhou364

Xiao Yang: xyang408

Haoran Li: hli656

Yingqiao Zheng: yzheng394

ECE 6122 Final Report

1 Group Work Distribution

- C++ thread (DFT, FFT): Zhuo Sheng Chin
- MPI (FFT): Xunchen Zhou, Xiao Yang
- CUDA (DFT): Haoran Li, Yingqiao Zheng

Additionally, we implement **FFT in C++ thread** to compare the runtime of DFT and FFT.

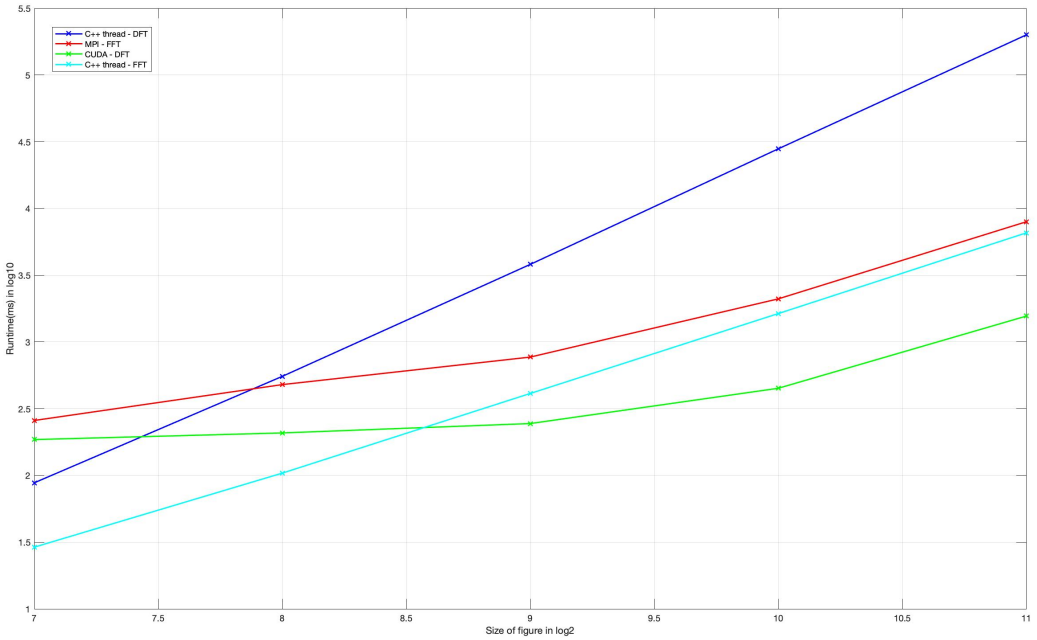
2 Runtime Analysis

Below is a table of our runtime results.

time (ms)	**128**	**256**	**512**	**1024**	**2048**
C++ thread - DFT	88	552	3810	27965	199597
C++ thread - FFT	29	104	411	1631	6561
MPI - FFT	258	479	770	2102	7946
CUDA - DFT	185.592	208.089	244.665	449.61	1561.32

Below is the graph of our runtime.

- y-coordinate: runtime(ms) in log10
- x-coordinate: size of figure in log2



Analysis

- **C++ runtime Analysis**

  Based on the graph, it is noted that the C++ thread-DFT performed the slowest among the other implementations, performing faster than the MPI-FFT and CUDA-DFT only when the input was 128x128 image size. This could be due to overheads from transferring data to the GPU, or starting up the OpenMPI interface. The DFT algorithm requires O(N^2) complexity to execute. As such, it is much slower than the MPI-FFT and C++ thread-FFT algorithm, which requires only O(NlogN) complexity to execute. While CUDA-DFT uses the same algorithm, only 8 threads are available for the C++ thread-DFT, and hence cannot take advantage of greater parallelism to execute DFT more quickly.

- **MPI runtime Analysis**

  According to the plot, MPI implementation is slower than CUDA implementation and C++ multithread-FFT but faster than C++ multithread-DFT.

  The runtime of MPI implementation should be slower than C++ multithread-FFT implementation. Multi-thread will use the same memory, so it doesn't involve send and receive from other processors which will worsen the performance, which is supported by our plot. The runtime of CUDA implementation should be always better than MPI implementation because every thread only calculate one number in the matrix, and all the number in the matrix will be calculated in the same time, which is supported by our plot.

- **CUDA runtime Analysis**

  According to the above plot, CUDA is the fastest among three methods.

In CUDA, every thread calculate only one elements in the original matrix, so for 2D DFT, ideally, we only need runtime for 2 elements in the matrix (suppose one for calculating the row, the other for calculating the column).

Based on the fact that CUDA use shared memory, if we implement FFT in CUDA, we might only be able to parallelize the calculation of a whole row instead of a single element. So it should be better to implement DFT in CUDA.