**ECE 4122/6122** **Advanced Programming Techniques**

# Two-Dimensional Discrete Fourier Transform on CPU and GPU

*Assigned: Monday, November 26, 2018* *Due: Thursday, December 13, 2018*

## 1 Introduction

You may work in groups of up to five members on this assignment. It is not a requirement to work in a group. Each group member should submit their own submission.

Given a one-dimensional array of complex or real input values of Length N, the Discrete Fourier Transform consists of an array of size N computed as follows :

$$H[n] = \sum_{k=0}^{N-1} W^{nk} h[k] \tag{1.1}$$

where

$$W = e^{-j2\pi/N} = cos(2\pi/N) - jsin(2\pi/N) \tag{1.2}$$

and

$$j = \sqrt{-1} \tag{1.3}$$

For all equations in this document, we use the following notational conventions, $h$ is the discrete-time sampled signal array. $H$ is the Fourier transform of $h$. $N$ is the length of the sample array, and is always assumed to be an even power of 2. $n$ is an index into the $h$ and $H$ arrays, and is always in the range $0...(N-1)$. $k$ is also an index into $h$ and $H$, and is the summation variable needed. $j$ is the square root of negative one.

The above equation clearly requires $N^2$ computations, and as $N$ gets large the computation time can become excessive. We will be looking at an alternative approach later in this assignment.

Given a two-dimensional matrix of complex input values, the two-dimensional Fourier Transform can be computed with two simple steps. First, the one-dimensional transform is computed for each row in the matrix individually. Then a second one-dimensional transform is done on each column of the matrix individually. Note that the transformed values from the first step are the inputs to the second step.

Examining the above formula it is easy to see how some of these steps can be done simultaneously. For example if we are computing a 2D DFT of a 16 by 16 matrix, and if we had 16 threads available, we could assign each of the 16 threads to compute the DFT of a given row. In this simple example, thread 0 would compute the one-dimensional DFT for for row 0, thread 1 would compute for row 1, and so on. If we did this, the first step (computing DFT's of the rows) should run 16 times faster than when we used only one thread.

However, when computing the second step, we run into difficulties. When thread 0 computes the one-dimensional DFT for row 0, it would presumably be ready to compute the 1D DFT for column 0. Unfortunately, the compute results for all other columns may not be complete. Therefore care must be taken to ensure all threads complete their row DFTs prior to any thread beginning to work on the columns.

## 2 Danielson-Lanczos Algorithm

From equation 1.1, it appears that to compute the DFT for a sample of length $N$, it must take $N^2$ operations, since to compute each element of $H$ requires a summation overall samples in $h$. However, Danielson and Lanczos demonstrated a method that reduces the number of operations from $N^2$ to $Nlog_2(N)$. The insight of Danielson and Lanczos was that the FFT of an array of length $N$ is in fact the sum of two smaller FFT's of length $N/2$, where the first half-length FFT contains only the even numbered samples, and the second contains only the odd numbered samples. This approach is also called the Cooley-Tukey algorithm. More information can be found here :

http ://mathworld.wolfram.com/Danielson-LanczosLemma.html
https ://en.wikipedia.org/wiki/Cooley-Tukey_FFT_algorithm

# 3 What you need to do

For this assignment you will be implementing THREE different versions of the 2-dimensional DFT. One will be using CPU threads, one will be using MPI and one will be running on GPUs using CUDA. As well as being graded for producing the correct output, each submission will be timed and a portion of your grade will be dependent on how your implementation performs against other implementations in the class. Timing of your code is best done by submitting batch PBS jobs.

- The first will be a multi-threaded CPU version that utilizes the summation method shown in equation 1.1. You may use as many threads as you wish but your implementation will be run on a machine with 8 cores. 'nodes=1 ;ppn=8' You MUST use C++'s built in thread capabilities. i.e. don't use the pthreads library directly. You may synchronize your threads however you wish. Some ways may be more efficient that others. If you want to find the best approach, you'll need to research and experiment.
- The second implementation will be an MPI implementation of the Danielson-Lanczos Algorithm described previously in this document. All of your MPI runs should be done using PBS scripts. Please do NOT run MPI jobs directly on the login node. If you encounter any issues running MPI jobs, immediately submit a help request to the pace support team. I have been told all previous issues should be fixed. You should be using 8 MPI ranks when testing the performance of your code.
- For the third implementation, you must implement either of the two algorithms to run on GPUs using CUDA. You may implement either approach however recall your grade will partly be determined by the performance of the implementation you submit for grading. You may use all of the available resources on the p100 gpus available on the coc-ice job queue for this implementation.

**GRAD-STUDENTS-ONLY** : If anyone in your group is a grad student, your group must do this. Grad students must also implement reverse DFTs for all of the given implementations. It's not hard and it's an easy way to verify you have the right answer. These won't explicitly be tested for performance, only correctness.

Along with your code, you will need to submit a report, named FinalReport.pdf, for this assignment containing a graph comparing the runtimes for all three of your implementations. We will assume that all input files will be square, same width and height and will be a power of 2. Your graph should contain runtimes for the following sizes : 128x128, 256x256, 512x512, 1024x1024, 2048x2048. You will be provided with sample inputs for the 256x256 and 1024x1024 cases so you can see the required format. You should generate your own test files for the other sizes. Also if you are working in a group, you should submit the names of your group members in this report.

You MUST use CMake for configuring your project. I must be able to build your project using cmake .. from a build folder within your submission. When I type make in the build folder, it should produce THREE executables with the names p31, p32, p33. Your programs should run as follows :

./p31 forward/reverse [INPUTFILE] [OUTPUTFILE]

ex : ./p31 forward Tower256.txt Output256.txt
will perform a 2D DFT using Tower256.txt as input and producing an output file named Output256.txt.

You will need to use online resources to determine how to correctly refer to needed header files. Do NOT write your own Makefile directly. Do NOT share code with students outside of your group. I will be testing for this. For submission, compress your solution and report into p3.zip (NOT .TAR.GZ, or .RAR or anything else). zip is installed on the login node. Submit via canvas.