Parallel Fast Fourier Transform

CS597 Concurrency and Algorithms

Francis Joseph Serina

# Introduction

The Fourier Transform, or FT, is an integral transform to decompose a signal, f(t), into a sum of frequencies, F(ξ). It is an important transform used in numerous applications but commonly in signal processing. The FT can be applied on signals of higher dimensions. Due to the nature of the transform, each dimension can be computed in sequence.

However, the transform is computationally expensive. Several algorithms, Fast Fourier Transforms or FFT, have been proposed to reduce the execution time. A popular FFT implementation is the Cooley-Tukey FFT Algorithm. The Cooley-Tukey FFT requires that the length, of each dimension, of the input data should be a power of 2.

This document explores 3 ways of implementing Cooley-Tukey FFT and compares their execution times. The input data will be 2D grayscale images.

# Goals

* Implement 2D FFT in 3 different methods – Single Thread, Multi-thread, Compute Shader.
* Compare the speedup of the Multi-thread and Compute Shader implementations against the Single Thread implementation.

# Input Data

* 2D Grayscale Image
* Width and Height are powers of 2

The input will go through pre-processing as follows

1. Convert Grayscale to Complex Data
   1. The input value per pixel will be in the range of [0, 1]. This will be treated as the real part of the Complex Data.
2. Shift Frequency Domain
   1. Shift the Frequency Domain by half of the width and height by negating elements when the sum of its row and column is odd.

# Fast Fourier Transform

Cooley-Tukey FFT in 2D is as follows

1. Bit Reversal by Row
2. Apply Butterfly by Row
3. Bit Reversal by Column
4. Apply Butterfly by Column

# Output Data

The resulting data will be a 2D array of Complex Numbers. Post-processing the data will help visualize the result. The post-processing is as follows

1. Get Magnitude (or Phase) Spectrum
2. Apply logarithm and scaling
3. Normalize to be between [0, 1]
4. Convert to grayscale

# Inverse FFT

The FFT algorithm is reversible. To compute for the inverse, we perform the following

1. Take the Conjugate
2. Apply Forward FFT
3. Take the Conjugate again
4. Divide by Width \* Height

# Single Thread

# Multi-thread

# Compute Shader

# Results

# Conclusions

# References