A Parallel Application of the Fourier Transformation

Justin Spidell – Brett Sumser

October 2021

Abstract

This project intends to implement a parallized version of The Fourier Transform, called the Fast Fourier Transform (hereby referred to as the FFT) to convert analog signals into digital signals and compress images. Specifically, a Discrete Fourier Transform (hereby referred to as the DFT) is used. By taking a waveform and sampling it at a certain constant frequency, we gather discrete signals in the form of a sum of sines and cosines that can be easily stored or analyzed. This is how common storage formats like mp3 or jpg are compressed.

We intend to create an implementation of the FFT to compress images and digital audio, as well as convert analog audio to digital. Our project will be written in C++, and by taking advantage of the University of Oregon's supercomputer, we will test our compression algorithm on its speed, and loss of quality with small and large pictures and audios.

Introduction

The Fourier Transform is an important mathematical concept. It has applications in digital signal processing, convolution in neural networks, image recognition and even speech processing. The main idea behind the Fourier Transform is that it is a "mathematical operation that changes the domain (x-axis) of a signal from time to frequency," [?]. The Fourier transform is denoted by adding a circumflex to the symbol of a function:

$$f \to \hat{f}$$

The Fourier transform is defined as:

$$\hat{f}(\xi) = \int_{-\infty}^{\infty} f(x) e^{-2\pi i x \xi} dx \tag{1}$$

The particular use case for the Fourier Transform that initially started this project was that of digital signal processing, specifically the conversion of analog signals to digital for guitar effects processing. Using the Fourier Series, it is apparent that any complicated wave form can be taken, and represented as an infinite series of sine and cosine functions [?]. Unfortunately, the DFT is on the slower side for algorithms, being performed in $O(n^2)$. It is defined as:

$$Let\ x_0,\ ...,x_{N-1}\ be\ complex\ numbers,$$

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi kn/N} \quad k = 0,\ ...,\ N-1 \tag{2}$$

But by using a divide and conquer strategy over the DFT, an algorithm called the Fast Fourier Transform, is formulated. The FFT is much faster than the DFT, being performed in $O(n \log n)$ time. The FFT can be easily implemented using a recursive or iterative programming method, but there are benefits to using an iterative approach. For example, iterative implementations can use less index computations, as well was being able to be parallized far more easily than a recursive version.

Parallization

There are a few different directions to explore when developing a more parallized implementation of the Fourier Transform. According to Anthony Blake in his thesis paper titled "Computing the Fast Fourier Transform on SIMD Microprocessors", use of the FFT algorithm is extremely widespread in multiple disciplines. He goes on to state that "use of the FFT is even more pervasive, and it is counted among the 10 algorithms that have had the greatest influence on the development and practice of science and engineering in the 20th century," [?]. The widespread use of the FFT algorithm provides great evidence for the need to optimize for different applications, and to understand the methods that can be used to achieve this. Two methods of Parallelization that stand out in particular are SIMD, and multithreading.

SIMD

Multithreading