A Parallel Application of the Fourier Transformation

Justin Spidell – Brett Sumser

October 2021

Abstract

A Fourier transform is a mathematical transform decomposing functions based on space and time into functions based on spatial or temporal 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:

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

Whereas the inverse Fourier transform is denoted as:

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

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," [4]. 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. When first researching the methods used to sample analog signals for conversion into digital, the Discrete Fourier Transform is impossible to avoid. 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 [3].

Unforunately, the Discrete Fourier Transform is on the slower side for algorithms, being performed in $O(n^2)$. But by using a divide and conquer strategy, an algortihm called the Fast Fourier Transform (hereby referred to as the FFT), is formulated. The FFT is quite a bit fast than the DFT, being performed in $O(n \log n)$ time. The FFT can be easily implemented using a recursive programming method, but there are benefits to using an iterative implementation. Parallel implementations can use less index computations, as well was being able to be parallized far more easily than a recursive version. This leads us nicely into the main impetus for this project, and the class as a whole.

Parallelization

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," [1]. 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

Single Instruction Multiple Data prcoessors are also known as vector processors, and allow a single instruction to process multiple pieces of data in the same clock cycle. The way that SIMD works is by packing several pieces of data into one data word. This allows the instruction loaded to act on each piece of data from a single instruction. This has applications in situations where large amounts of data are being manipulated. With respect to the Fourier Transform, this can be applied to image manipulation, audio processing, or other functions.

According to this post [2] on StackOverflow, C/C++ contains SIMD functions called vector intrinsics. These are implemented in the compiler, allowing them to load orders of magnitude faster than common library functions. The found that they could use these intrinsics to produce code up to four times faster, and even more in certain cases! For this project, getting a handle on these intrinsics would allow for an increased knowledge in C/C++, while also providing experience in possible future scenarios with performance critical code.

Multithreading

After some preliminary research into a multithreading implementation of FFT [5], it seems there are possible multithreaded implementations of the

FFT. Based on the previously cited source, Thulasiraman et al. picked two differ algorithmic approaches to investigate.

References

[1] Anthony Blake. Computing the fast fourier transform on simd microprocessors, 2012.

[2] Konstantin. Improving performance with simd intrinsics in three use cases, 2020.

[3] Bo Liu. Parallel fast fourier transform, N/A.

[4] Cory Maklin. Fast fourier transform, 2019.

[5] Parimala Thulasiraman. Multithreaded algorithms for the fast fourier transform, 2020.