Comparison of Sequential and Parallel Fast Fourier Transform

Rami Jurdi
Zachary Noble
Gregory Freitas
Jayden Bendezu

***Abstract*—This research is in the Digtial Signal Processing field. Where our aims are to improve the performance of realtime signal processing using parallel processing techniques coupled with 1 dimensional Fast Fourier Transforms. It has been done before by other researchers implementing multi-dimensional Fast Fourier Transforms in a multithreadeed context. The purpose of our research is to gain a better understanding of parallel processing techniques and digital signal processing. Thus the main goal is to observe the outcome of implementing the multithreaded Fast Fourier transform algorithm and learn from the state-of-the-art research.**

I. Introduction

The Fast Fourier transform (FFT) is an algorithm that uses Discrete Fourier transforms (DFT) on a time sequence to convert a signal, usually based on time, to a signal based in the frequency domain. This allows us to analyze a sequence of time-values by decomposing it into bins of different frequencies. The Fast Fourier transform is used in many applications ranging from digital signal processing, sampling, pitch correction software, and wave analysis.

The direct computation of the DFT results in n^2 multiplications and $n(n-1)$ additions, making the computation greatly expensive for sufficiently large n. Over the course of many years of research done by researchers in the field, more efficient algorithms were developed for computing DFTs, such as the Cooley-Tukey FFT algorithm. This algorithm reduces the time complexity of the computation of Fourier transforms from $O(n^2)$ to $O(nlogn)$ [**?**].

In this paper, we present our process of implementing the parallel FFT algorithm, specifically the Cooley-Tukey algorithm, and conduct performance comparisons between the sequential and parallel versions of the algorithm in C++.

II. Target Problem

The main problem at hand is to create an application that is able to process signals provided to the program as audio files such as .WAV, .MP3, and .MP4. Then displaying the audio files as a spectrograph of its signals. This problem can be broken into several steps of what we need to acheive.

- A GUI to interact with.
- Accepting audio files via local upload or recording.
- Processing the audio files.
- Processing the signals for audio files.
- Displaying the signals of the audio files on a spectrograph.

A. Approach

Tackling some of the above sub-problems. In order to create a GUI to interact with, we will write the program using the QT C++ GUI framework to simplify taking audio files from disk and decoding the raw data. Through the use of provided libraries of QT the basic functionality of loading files and playing/decoding audio is handled and we use them on a higher level. This will allow us to focus more on the problem of processing signals for the audio files. Then by handling extracting raw data from the audio files using the QAudioDecoder class provided by QT, we intend to use one-dimensional Fast Fourier Transforms to process the signals. (How do we plan to parllelize the FFT is what would be discussed here briefly since will be explained througouly within the algorithms subsection).

1) Project Architecture:

B. Plan Outline

1) Setup programming environment using Qt C++.
2) Add support using Qt multimedia API's to load and decode audio files from disk.
3) Stream audio data to an output device such as speakers or headphones.
4) Display a waveform of the audio.
5) Implement single-threaded Fast Fourier Transforms.
6) Implement multi-threaded Fast Fourier Transforms.
7) Display the frequency vs. amplitude data calculated from the Fourier transform.
8) Conducting experimental tests comparing multi-threaded implementation vs sequential implementation. Observing any performance boosts if any.

C. Algorithm

1) Discrete Fourier Transform: While we established that we will be using the Fourier Transforms. As the base implementation of fourier transforms resolves to a $O(n^2)$ runtime according to [**?**].

2) Fast Fourier Transform: It is essential that we use an implementation that yields a better runtime as in the case of our program, operating on larger audio files will begin to take much longer being inefficient. Thus we will be working with the Fast Fourier transfom algorithm to reduce our upper asymptotic bound. (Going to find a reference for this line to talk more about the $O(nlogn)$ implementation. But i need to read more).

D. Paralellized Fast Fourier Transform

E. Experimental Results

Creating testing data sets ...

Data Set 1			
Algorithm	File Type	File Size	Computation Time
Sequential FFT	foo.WAV	100 kb	N
Parallel FFT	foo.WAV	100 kb	N
Data Set 2			
Algorithm	File Type	File Size	Computation Time
Sequential FFT	foo.WAV	100 kb	N
Parallel FFT	foo.WAV	100 kb	N
Data Set 3			
Algorithm	File Type	File Size	Computation Time
Sequential FFT	foo.WAV	100 kb	N
Parallel FFT	foo.WAV	100 kb	N

III. State-of-the-Art Research

The papers that we will use to implement the parallel 1D Fast Fourier Transform algorithm are the *Proceedings of the Thirty-Seventh Southeastern Symposium on System Theory* [**?**], the .....

IV. Related Work

Here we will talk about how our work relates to currently worked on research.

V. Our Contributions