Land Forces Academy Review
Vol. XXIV, No 1(93), 2019

GLANCE ON PARALLELIZATION OF FFT ALGORITHMS

Mhnd FARHAN
University of Baghdad, Iraq
mhndfarhan@yahoo.com

ABSTRACT

This paper implores the parallelization of Fast Fourier Transform (FFT) algorithms and evaluates the resultant parallelized source codes. The FFT algorithm is considered to be among the most important algorithms in the digital era. There are various FFT algorithms but just a few are considered in this paper. The Cooley-Tukey FFT is the most widely known and used. With no exception, in this paper, the radix-2 Decimation in Time (DIT) and Decimation in Frequency (DIF) are studied and implemented. Another important FFT algorithm that is the Goertzel is also considered in this paper.

KEYWORDS: parallelization, FFT, algorithms

1. Introduction

Algorithms play a very important role in life in the computations of various data. Hence it becomes crucial for the people using those algorithms to get the most efficient versions. One such algorithm is the Fast Fourier Transform (FFT) algorithm which computes the Discrete Fourier Transform (DFT).

DFT is very useful in the aid of analysis of many problems in engineering that translate to real life such as signal processing, speech and image processing (Chu & George, 2000). The number of computations done in the computation of DFT using a direct approach is given by O(n2). The FFT is basically any algorithm that is used to compute the DFT with a reduced number of computations that is O(nlogn) (Chu & George, 2000; Cuixiang, Guo-qiang & Minghe, 2005). In the ever changing world of technology, the speed of processing has always been an issue. Ranging from the internet to image and data processing to a load of other applications, the goal has always been to increase the speed (Li & Dong, 2010).

In the past, computers have had one processor with a single processing core. Such computers were only able to do one task at a time only being able to switch between tasks. With time, however, multiprocessor computers and multi-core processors were manufactured to improve on the speed of processing via the hardware concurrency which allowed for parallelization. Parallelization is the ability of a computer to do multiple tasks at the same time independently (Morris, Chowdhury & Deb, 2011; Zhang, Shen, Xu & Wang, 2013; Takahashi, 2017).

In this paper, parallelization of FFT algorithms is going to be done to compute the DFT. The C++ programming language is used to come up with the source codes.

2. Design and Methodology

A pseudo code is used to generate both the sequential and parallel source codes for the FFT algorithms. The C++ high level programming language is used in the implementation of the designs. The source codes are compiled and run using the Microsoft Visual Studio 2017

DOI: 10.2478/raft-2019-0009

software and output is displayed through the graphic user interface. Analysis of the programs is also achieved using the diagnostic tools of the software.

2.1. Design Procedure

In the actual design, the decisions on how the FFT algorithms are to be implemented are made. The sequential programs for the algorithms are first implemented. The sequential programs are then parallelized by the technique of multithreading available in the C++ programming language. The Goertzel algorithm is implemented in the signaling system dual tone multi- frequency. The Goertzel algorithm computes the k*th* DFT coefficient of the input signal unlike the other FFT algorithms which compute the DFT for all the input samples.

The required programs are represented within flow charts to give the visual impression and a road map for the actual implementations. The flow charts constructed are shown in Figure no. 1 and Figure no. 2.

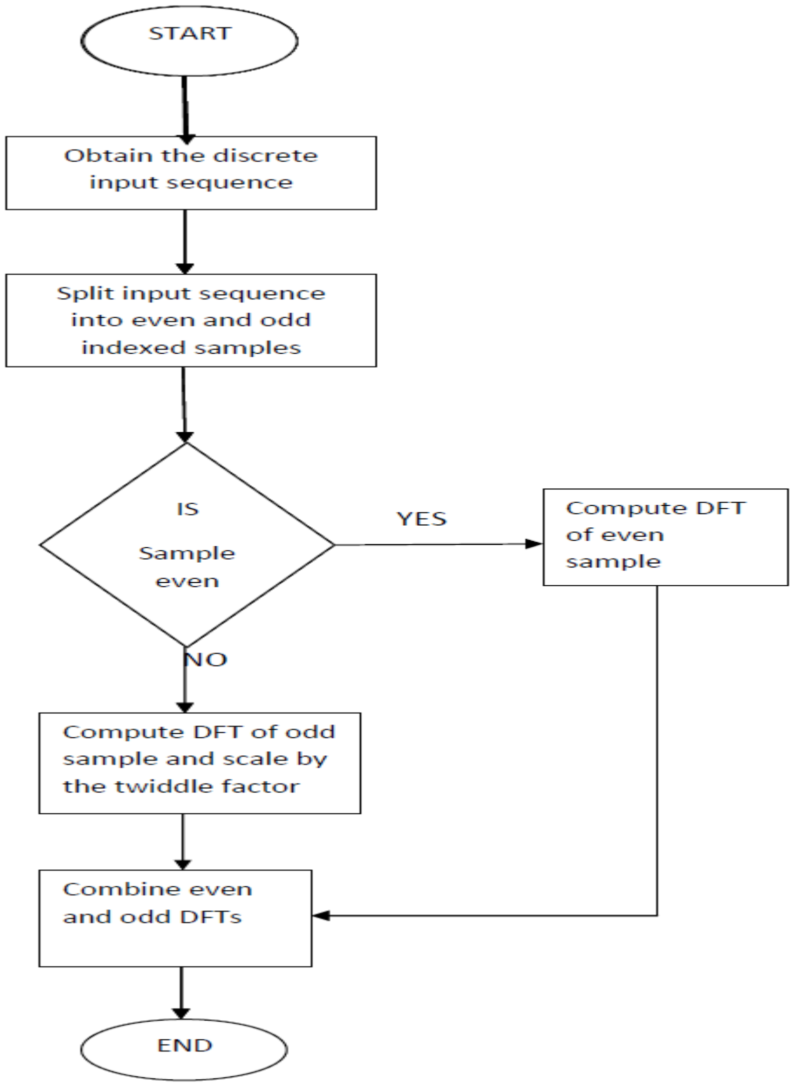
START
Obtain the discrete input sequence
Split input sequence into even and odd indexed samples
IS Sample even
YES
Compute DFT of even sample
NO
Compute DFT of odd sample and scale by the twiddle factor
Combine even and odd DFTs
END

Figure no. 1: *Flow chart for DIT FFT*

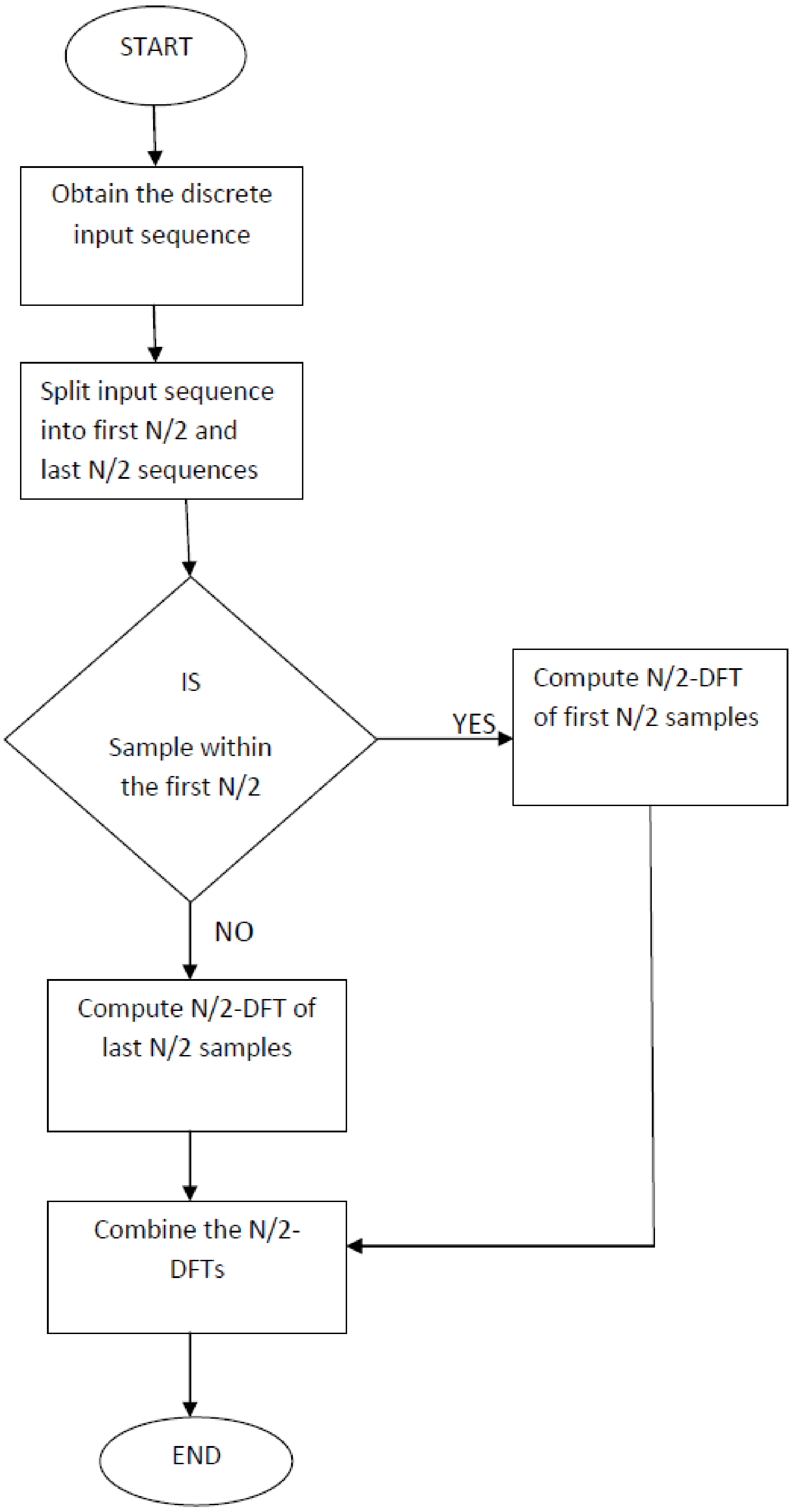
START
Obtain the discrete input sequence
Split input sequence into first N/2 and last N/2 sequences
IS Sample within the first N/2
YES
Compute N/2-DFT of first N/2 samples
NO
Compute N/2-DFT of last N/2 samples
Combine the N/2-DFTs
END

Figure no. 2: *Flow chart for DIF FFT*

2.2. Methodology

The programming language used in the implementation of the algorithms is the high level language C++. This language is chosen because of its object oriented features as well as the fact that it supports parallel programming.

Using the flow charts as reference, the source codes are implemented in the C++ programming language. The Goertzel algorithm, as well as the naïve approach for the DFT, is also implemented. Input sequences to the algorithms are taken arbitrarily whereas the focus is not entirely on the output of the programs but rather the performance.

Diagnostic tools in the Microsoft Visual Studio 2017 are used to analyze the performance of the programs. Consequently, the results from the implementations entail the time taken for the programs to accomplish the given tasks.

3. Results

After the execution of the sequential and parallel programs, the execution durations obtained are recorded in the Tables no. 1, 2, and 3.

Table no. 1
Time of execution for the sequential implementations

SEQUENTIAL IMPLEMENTATION		
Input/ Output sequence	**8**	**16**
DFT	178ms	231ms
DIT	159ms	185ms
DIF	161ms	202ms

Table no. 2
Time of execution for the parallelized implementations

PARALLEL IMPLEMENTATION		
Input/ Output sequence	**8**	**16**
DFT	136ms	164ms
DIT	155ms	170ms
DIF	150ms	166ms

Table no. 3
Time of execution for the Goertzel algorithm

GOERTZEL ALGORITHM	
Sequential Implementation	**Parallel Implementation**
238ms	202ms

The execution time of the various implementations is used to calculate the speed-up factor, that is;

Speed-up factor = (serial execution time)/(parallel execution time)

1- DFT

(i) Number of samples = 8
Speed-up factor = 178/136=1.31

(ii) Number of samples = 16
Speed-up factor = 231/164=1.41

2- DIT

(i) Number of samples = 8
Speed-up factor = 159/155=1.03

(ii) Number of samples = 16
Speed-up factor = 185/170=1.09

3- DIF

(i) Number of samples = 8
Speed-up factor = 161/150=1.07

(ii) Number of samples = 16
Speed-up factor = 202/166=1.22

4- Goertzel

Speed-up factor = 238/202=1.18

4. Discussion

In this paper, emphasis was on the computation of the DFT. Due to the shortcomings of the naïve approach in the computation, FFT algorithms are employed for to improve on the speed of computation and reduce the time required for execution. This is more evident with the increase in number of samples for which the DFT is to be computed. FFT algorithms take advantage of a number of properties of the general DFT equation in order to reduce redundancy and improve on the constraints of time and memory. Hence, for the same amount of input and output samples, FFT algorithms are faster and take less time to execute in similar conditions.

Further optimization of the computations is attained by the effective utilization of the processor unit. Computers have been sufficiently improved over time to be able to have genuine hardware concurrency made up of multi-processor computer or multi-core processors. To be able to utilize this hardware concurrency, the program must be parallelized to allow for different parts of the program being executed on different processors/ processor cores at the same time. Hence parallelization of the programs also improves the speed of execution. As evidenced by the results, it is also more economical when computing the DFT of more samples.

It is thus evident that parallel FFT algorithms are suitable for multi-processor implementation to ensure efficiency of the programs.

5. Conclusion

In this paper, computation of the DFT was attained by the implementation of the FFT algorithms in C++. Sequential programs were first implemented and then parallelized in the C++ language. The parallelized programs were compared to the corresponding sequential programs and speed factor found to be more than one. Hence the FFT algorithms were parallelized and found to have improved performance and suitable for multi-processor implementation.

REFERENCES

Chu, E., & George, A. (2000). *Inside the FFT black box: serial and parallel fast fourier transform algorithms*. Washington, D.C.: CRC Press.

Cuixiang, Z., Guo-qiang, H., & Minghe, H. (2005). Some new parallel fast fourier transform algorithms. *IEEE Sixth International Conference on Parallel and Distributed Computing Applications and Technologies*, China.

Li, P., & Dong, W. (2010). Computation oriented parallel FFT algorithms on distributed computer. *IEEE 3rd International Symposium on Parallel Architectures, Algorithms and Programming*, China.

Morris, P., Chowdhury, S., & Deb, D. (2011). An efficient methodology for realization of parallel FFT for large data set. *International Conference on Advances in Computing and Communications*, India.

Takahashi, D. (2017). An implementation of parallel 1-D real FFT on Intel Xeon Phi processors. *International Conference on Computational Science and Its Applications*, Italy.

Zhang, X., Shen, K., Xu, C., & Wang, K. (2013). Design and implementation of parallel FFT on CUDA. *IEEE 11th International Conference on Dependable, Autonomic and Secure Computing*, China.