Design and Analysis of Algorithms

fast fourier transform

Table of Contents

[Background: 1](#_Toc104835480)

[Introduction: 1](#_Toc104835481)

[Scope: 1](#_Toc104835482)

[Objectives: 1](#_Toc104835483)

[Timeline: 1](#_Toc104835484)

[Algorithm Explanation and Working: 2](#_Toc104835485)

[Radix-2 DIT: 2](#_Toc104835486)

[Improvements to the Algorithm: 4](#_Toc104835487)

[Bit Order Reversal – A bit Reordering Technique: 4](#_Toc104835488)

[Pseudocode: 5](#_Toc104835489)

[Complexity Analysis: 6](#_Toc104835490)

[Time Complexity: 6](#_Toc104835491)

[Space Complexity: 7](#_Toc104835492)

[For Bit Reversal: 7](#_Toc104835493)

[For FFT Algorithm: 7](#_Toc104835494)

[Project Outcomes: 7](#_Toc104835495)

[Problems Faced: 8](#_Toc104835496)

[Future Recommendations: 8](#_Toc104835497)

[Conclusion: 8](#_Toc104835498)

[References: 9](#_Toc104835499)

# **Background:**

Our initial point for this project was to resolve a complex real-world problem with the aid of algorithms. The project demanded that we find a suitable algorithm to solve the problem at hand efficiently and reliably. In today’s world signal processing is widely used. The need for mathematical computations for signal processing is indispensable. Therefore, a very suitable solution for such a problem was found to an algorithm known as Fast Fourier Transform (FFT).

# **Introduction:**

Fourier Transform is first introduced by Jean Baptiste Joseph Fourier to solve the computational complexity in wide variety of fields including earth and science, chemistry, communications, and signal processing . In signal processing, Fourier Transform has long been established as an instrumental tool applied in electrical signal spectrum and filter analysis, sampling and series, antenna, television image convolution as well as radio broadcasting . Being the limiting case of Fourier Series for non-periodic signals, FT is used to convert signal to frequency domain as the frequency domain has many superlative benefits especially for analytical purposes rather than in the classical time domain.

# **Scope:**

Fast Fourier Transform has long been established as an essential tool in signal processing. To address the computational issues while helping the analysis work for multi-dimensional signals in image processing, sparse Fast Fourier Transform model is reviewed here when applied in different applications such as lithography optimization, cancer detection, evolutionary arts and wastewater treatment. As the demand for higher-dimensional signals in various applications especially multimedia applications, the need for sparse Fast Fourier Transform grows higher.

# **Objectives:**

Our core objective for this project was to utilize our knowledge of data structures and algorithms with the further goal of expanding it. For this purpose, we investigate algorithms which are used extensively in everyday lives to get hands-off experience for algorithm analysis and their design.

# **Timeline:**

* Project Announcement: 18th March 2022
* Proposal Deadline: 2nd April 2022
* Project Initiation: 7th May 2022
* Project Submission: 30th May 2022
* Project Viva: 1st June 2022

# **Algorithm Explanation and Working:**

The most common used algorithm for calculation of DFT(Discrete Fourier Transforms) nowadays, is Cooley-Tukey Algorithm. I t is re-expression of DFT of arbitrary size N into N1 and N2; by creating N1 smaller DFT factors of size N2. Since Cooley-Tukey algorithm breaks the DFT into smaller factors it is relatively easy to combine it with any other technique for calculation of DFT where necessary. Aa recursive approach, using the principle of Divide and Conquer is applied for implementation of Cooley-Tukey algorithm.

The division of the DFT of any size N is perform most commonly using the radix-2 Decimation In Time(DIT).

## Radix-2 DIT:

The radix-2 DIT divides the DFT under consideration into two interleaved DFTs of size N/2 with each recursive stage.

The general formula for DFT is given as:

Where, k is an integer ranging from 0 to N-1.

Radix-2 DIT first computes the DFTs of the even-index inputs (x2m= x0, x2,x4,….) and then those of the odd indexed inputs (x2m+1= x1, x3,x5,….) then the results are combined to produce the DFT of the whole sequence. Using this the complexity of the algorithm can be reduced from polynomial to being logarithmic to the base 2.

After the application of the radix-2 decimation, using mathematical techniques, we get the derived result for catering the division and the general expression transforms into the expression below:

Where, for even terms,

For odd terms,

Using these we can further simplify to obtain the expressions for Xk and Xk+N/2, given as follows:

Hence, this result expresses the DFT of length and divides them into the size of N/2 recursively. It can also be noted that the result that we will obtain will be by a combination of addition(+) and subtraction (-) operations between the terms Ek and Ok . exp(-2πik/N). These +- operations when performed in combination are normally named as butterfly operations. An illustration depicting the division and recombination using radix-2 DIT is as:

|  |
| --- |
|  |

The radix-2 DIT approach to divide the DFT is also termed Danielson-Lanczos lemma.

Another technique for DIT of DFT exists in which the DFT is divided using some radix-N instead of 2. In those DIT, we also have to perform addition computations but adding a factor(called twiddle factors) to the decimated terms, to maintain the mathematical equilibrium. Those strategies are out of scope for our project.

# **Improvements to the Algorithm:**

This approach to the calculation of DFT has its advantages but a further advancement into the algorithm is offered by using an in-place approach which allows us to significantly improve the space complexity. In the in-place approach when an array of complex numbers is entered for computation of their of respective DFT, each input term is replaced by its output term in the same array. Therefore, no separate array is required for output. This is very critical since in real-life the usage of FFT algorithms is very extensive and requires maximum efficiency, maximum return.

When the in-place approach to the algorithm is used, it requires that while the outputs are updated in place of inputs, bit reordering techniques are required to correspond the input array to the output and to avoid any inconsistencies in the calculations.

## Bit Order Reversal – A bit Reordering Technique:

The most well-known reordering technique involves explicit **bit reversal** for in-place radix-2 algorithms. Bit Reversal is the permutation where the data at an index *n*, written in binary with digits *b*4*b*3*b*2*b*1*b*0 (e.g. 5 digits for *N*=32 inputs), is transferred to the index with reversed digits *b*0*b*1*b*2*b*3*b*4 . Consider the last stage of a radix-2 DIT algorithm like the one presented above, where the output is written in-place over the input: when Ek{\displaystyle E\_{k}} and Ok {\displaystyle O\_{k}} are combined with a size-2 DFT, those two values are overwritten by the outputs. However, the two output values should go in the first and second *halves* of the output array, corresponding to the *most* significant bit *b*4 (for *N*=32); whereas the two inputs {\displaystyle E\_{k}} Ek{\displaystyle E\_{k}} and Ok {\displaystyle O\_{k}} {\displaystyle O\_{k}} are interleaved in the even and odd elements, corresponding to the *least* significant bit *b*0. Thus, to get the output in the correct place, *b*0 should take the place of *b*4 and the index becomes *b*0*b*4*b*3*b*2*b*1. And for next recursive stage, those 4 least significant bits will become *b*1*b*4*b*3*b*2 and so on.

An illustration depicting this bit reversal is as:

|  |
| --- |
|  |

# **Pseudocode:**

The following is pseudocode for iterative radix-2 FFT algorithm implemented using bit-reversal permutation.

|  |
| --- |
| **Cooley-Tukey In-Place FFT:** |
| **algorithm** iterative-fft **is**  **input:** Array *a* of *n* complex values where n is a power of 2.  **output:** Array *A* the DFT of a.    bit-reverse-copy(a, A)  *n* ← *a*.length  **for** *s* = 1 **to** log(*n*) **do**  *m* ← 2*s*  *ωm* ← exp(−2π*i*/*m*)  **for** *k* = 0 **to** *n*-1 **by** *m* **do**  *ω* ← 1  **for** *j* = 0 **to** *m*/*2* – 1 **do**  *t* ← *ω* *A*[*k* + *j* + *m*/2]  *u* ← *A*[*k* + *j*]  *A*[*k* + *j*] ← *u* + *t*  *A*[*k* + *j* + *m*/2] ← *u* – *t*  *ω* ← *ω* *ωm*    **return** *A* |
| **Bit Reversal Method:** |
| **algorithm** bit-reverse-copy(*a*,*A*) **is**  **input:** Array *a* of *n* complex values where n is a power of 2.  **output:** Array *A* of size *n*.  *n* ← *a*.length  **for** *k* = 0 *to* *n* – 1 **do**  *A*[rev(k)] := *a*[k] |

|  |
| --- |
| **Explanation:** |
| **Bit Reversal Algorithm:** |
| 1. Bit reversal basically takes an array of size N, performs bit reversal, which is fundamentally array reversal. 2. Returns a bit reversed array for he input array. |
| **FFT Algorithm:** |
| 1. The implementation is based on radix-2 DIT therefore, therefore an input array of size which needs to be power of 2 is expected. 2. The input array is firstly, bit reversed. A copy ‘a’ of the original array ‘A’ is returned. 3. The first array runs to the log to the base 2(since we are using radix-2 DIT) of the size of the input array. 4. The first loop assigns the appropriate variable values based on the algorithm working above. 5. The second loop applies the summation in the formula. 6. The inner most loop performs the required mathematical operations for each of the even/odd rotation in accordance with the outer loop variables. |

# **Complexity Analysis:**

## Time Complexity:

The time complexity analysis for the algorithm are as follows:

* Firstly, we must consider the bit reversal function, it runs a loop till the length of array ‘N’, therefore it can simply be concluded to have a time complexity of O(n).
* Now for the FFT algorithm, firstly since the bit reversal function which has complexity of O(n).
* Then it consists of three further loops in which:

1. Outer Most Loop runs for log(n) iterations.
2. Second level loop runs for n iterations.
3. Inner most loop runs for m/2 iterations

* Considering the time complexity, the loops run for log(n) x n x m/2.
* Therefore, the overall complexity can be calculated as:

The logarithmic complexity overcomes the linear complexity,

The constant ½ can be removed.

Finally, the complexity can be written as:

## **Space Complexity:**

## For Bit Reversal:

In the bit reversal, the space required is just for the two arrays ‘a’ and ‘A’, both are a size of n. Therefore, we can calculate as:

Removing the constant 2, we get:

Hence, the complexity becomes linear.

## For FFT Algorithm:

In terms of space complexity, since it is in-place it only has a single array of size ‘n’. For the variables rae there, three loop variables, s, k and m. Moreover, there are also the four computation variables, t, u, ωm and ω. Hence, we can compute the complexity as:

Dropping the constant ‘7’, we get:

Dropping the constant ‘2’, we get:

Therefore, we get the linear space complexity given as:

# **Project Outcomes:**

Following are a few project outcomes, that we feel like were the most prominent.

1. **A Glance into Real World Algorithms:**

This project has as a chance to investigate real world application and appreciate the sophistication in such algorithms.

1. **Realization For Need for Efficiency:**

We got a chance to explore the scope and requirement for efficiency for algorithms that are utilized on much a larger scale, that even the most minor implementation detail has a significant effect.

1. **Sense of Teamwork:**

This project enables us a sense of teamwork and other social working skills in a professional environment. It enabled us to take a deeper glance into the realms of workaholism.

# **Problems Faced:**

The problems faced by us during the during of project were as follows.

1. **Selection of Algorithm:**

One major concern for us was the selection of appropriate algorithm, since it was our motivation to choose an algorithm that had major applicability and utility. Moreover, it should appropriately assess our knowledge of curse while ensuring appropriate challenges.

1. **Difficulty of Algorithm:**

After submitting the proposal, we realized that our choice of algorithm belonged to very specific domain and there was not as much easy-to-understand and extensive literature on it. Therefore, we had to rely on that limited content for understanding and information. Moreover, it being domain specific meant it had its certain prerequires to achieve an extensive understanding to build a project on that.

# **Future Recommendations:**

The recommendations for the project are as follows.

1. **Exploration Deeper Into the Algorithm:**

For future, we would like to explore deeper into the FFT domain of Algorithm like studying and implementing radix-N DIT and their traversals such as Row Major Order and Column Major Order. Moreover, the exploration of Multidimensional FFTs.

1. **Utilization of Practical Analysis Techniques:**

We would like to further learn and perform more algorithm analysis techniques like performance analysis, empirical analysis, graphing, and further research which, we were unable to perform currently due to time constraints.

1. **Algorithm Technique and Efficiency Improvements:**

Furthermore, we wish to perform further research to develop such an understanding for FFTs such that we are able to make and recommend improvement to the existing algorithms and introduce further improvements in the domain of signal processing.

# **Conclusion:**

Cooley-Tukey algorithm is one of the most mains stream algorithms in the domain of signal processing. It has various and widely spread applications. Although, it was difficult algorithm for us to completely grasp and perform the required analysis, it allowed us to gain experience regarding how to work out of one’s comfort zone.

# **References:**

1. <https://en.wikipedia.org/wiki/Cooley%E2%80%93Tukey_FFT_algorithm>
2. <https://eng.libretexts.org/Bookshelves/Electrical_Engineering/Signal_Processing_and_Modeling/Fast_Fourier_Transforms_(Burrus)/08%3A_The_Cooley-Tukey_Fast_Fourier_Transform_Algorithm>
3. Johnsson, S. L., & Krawitz, R. L. (1992). Cooley-tukey FFT on the connection machine. *Parallel Computing*, *18*(11), 1201-1221.
4. Bekele, A. J. A. A. (2016). Cooley-tukey fft algorithms. *Advanced algorithms*.
5. Maslen, D. K., & Rockmore, D. N. (2001). The Cooley-Tukey FFT and group theory. *Notices of the AMS*, *48*(10), 1151-1160.