



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Jorge Francisco Teixeira Bastos da Mota

High Performance Fourier Transforms on GPUs

August 2022



Universidade do Minho

Escola de Engenharia

Departamento de Informática

Jorge Francisco Teixeira Bastos da Mota

High Performance Fourier Transforms on GPUs

Master dissertation

Integrated Master's in Informatics Engineering

Dissertation supervised by

António Ramires

August 2022

ABSTRACT

The continuous progress of the evolution of GPUs has increased the popularity of parallelizable algorithm implementations on this type of hardware. Fast Fourier Transforms is a family of algorithms which are very useful for the computation of Discrete Fourier Transforms which is applied in many practical scenarios for several areas.

Due to its usefulness and the need to optimize Fast Fourier Transforms this algorithms are effectively computed on the GPU to take advantage of its parallelizable variants

In this dissertation we provide, compare and analyze FFT implementations with popular libraries that are known to compute this efficiently, and provide specially forged GLSL implementations in the context of applications.

KEYWORDS FFT, GLSL, cuFFT, analysis, performance.

RESUMO

O progresso contínuo da evolução dos GPUs aumentou a popularidade das implementações de algoritmos paralelizáveis neste tipo de hardware. *Fast Fourier Transforms* são uma família de algoritmos úteis para o cálculo de transformadas de Fourier discretas que são aplicadas em muitos cenários práticos para diversas áreas.

Devido à sua utilidade e à necessidade de otimizar as *Fast Fourier Transforms*, esses algoritmos são efetivamente computados na GPU para aproveitar suas variantes paralelizáveis

Nesta dissertação, fornecemos, comparamos e analisamos implementações de FFT com bibliotecas populares que são conhecidas por computar isso de forma eficiente e fornecemos implementações GLSL especialmente forjadas no contexto de aplicativos.

PALAVRAS-CHAVE FFT, GLSL, cuFFT, análise, performance

CONTENTS

1	Introduction	5
1.1	Contextualization	5
1.2	Motivation	5
1.3	Objectives	5
1.4	Document Organization	6
2	State of the Art	7
2.1	Fourier Transform	7
2.1.1	What is Fourier Transform	7
2.1.2	Where it is used	8
2.2	Discrete Fourier Transform	9
2.2.1	Matrix multiplication	10
2.3	Fast Fourier Transform	12
2.3.1	Radix-2 Decimation-in-Time FFT	12
2.3.2	Radix-2 Decimation-in-Frequency FFT	14
3	Algorithms analysis	17
4	Computation of the Fourier Transform	18
4.1	Improving the Cooley-Tukey algorithm	18
4.1.1	Natural order Cooley-Tukey	18
4.1.2	Stockham algorithm	18
4.1.3	Radix-4 instead of Radix-2	18
5	Implementation on the GPU	20
5.1	Fourier Transform on the GPU	20
5.2	Implementation in GLSL	20
6	Analysis and Comparison	21
6.1	Popular implementations	21
6.1.1	cuFFT	21
6.1.2	FFTW	21
6.2	Comparison with GLSL implementation	21

6.2.1	Analysis	21
-------	----------	----

Bibliography	22
---------------------	----

LIST OF FIGURES

Figure 1	Time to frequency signal decomposition Source: NTiAudio	8
Figure 2	Radix-2 Decimation-in-Time FFT Source: Jones (2014)	13
Figure 3	Cooley-Tukey butterfly	13
Figure 4	Radix-2 Decimation-in-Frequency FFT Source: Jones (2014)	15
Figure 5	Gentleman-Sande butterfly	15
Figure 6	Natural order composition	19

LIST OF TABLES

Table 1	Dissertation schedule	6
Table 2	FFT algorithms benchmark. Results are <u>measured in milliseconds</u> for forward and inverse computation with varying input sizes	17

INTRODUCTION

1.1 CONTEXTUALIZATION

The Fast Fourier Transforms have been present in our surroundings for a long time, they're used extensively in digital signal processing including many other areas and they often need to be used in a realtime context, where the computations must be performed fast enough. Fast Fourier Transforms essentially are optimized algorithms to compute the Discrete Fourier Transform of some data, data that might be sampled from a signal, an oscilating object or even an image, which is transformed into the frequency domain allowing any kind of processing for a relatively low computational cost. Despite already existing pretty fast computations of the FFT, many applications require the processing of serveral transforms so its necessary to manage the implementations properties and achieve the best speed.

1.2 MOTIVATION

The continuous progress of the evolution of GPUs has increased the popularity of parallelizable algorithm implementations on this type of hardware. Notably the FFT algorithms family is constantly present in Computer Graphics, it's usual to find inlined implementations in shader code which offer reliable Fast Fourier Transforms [Flügge \(2017\)](#), but lack tuning of settings for a more optimized versions of these computations. On the other hand there's already out there libraries that provide efficient implementations of FFT on the GPU and CPU like cuFFT [Nvidia](#), a library provided by NVIDIA exclusively for their GPU's implemented for CUDA, and FFTW [Frigo and Johnson \(2012\)](#), a library dedicated to computations of FFT on the CPU with SIMD instructions support.

Although this libraries can provide efficient transforms with specialized cases over a proper plan, in some applications its performance might be compromised for cases where, for example, the graphics pipeline needs to be sincronized with the computation of the Fourier Transform.

1.3 OBJECTIVES

The main objective of this dissertation is to provide efficient FFT alternatives in GLSL compared with dedicated tools for high performance of FFT computations like NVIDIA cuFFT library or FFTW, while analysing the intrinsic of a good Fast Fourier Transform implementation on the GPU and even make a one to one comparison of

implementations on different frameworks. To accomplish the main objective there are two stages taken in consideration, *Analysis of CUDA and GLSL kernels* to be well settled in their differences and to have a reference for the second stage *Analysis of application specific implementations* which will cluster the study's main objective and where we'll use as case of study applications with implementation of the FFT in the field of Computer Graphics that require realtime performance.

With constant progression of the research needed for this project, some steps of the work plan were refactored to meet the needs. The two main stages of the objectives stay the same but there are some adjustments to the schedule dates and steps as shown in [Table 1](#).

- **Research Fast Fourier Transform**
- **Study cuFFT**, understand internal optimizations and prepare specialized profiles.
- **Analysis of CUDA and GLSL kernels** for FFT raw computations.
- **Research of Application driven FFT**, specialized implementations on the context of the application.
- **Writing of pre-dissertation**
- **Writing of dissertation**

	2021			2022							
	Nov	Dez	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Set
Research Fast Fourier Transform											
Study cuFFT											
Analysis of CUDA and GLSL kernels											
Research of Application driven FFT											
Writing of pre-dissertation											
Writing of dissertation											

Table 1: Dissertation schedule

1.4 DOCUMENT ORGANIZATION

This dissertation is organized in 3 chapters. Firstly, the [chapter 1](#) exposes an introduction to the subject of this dissertation with the respective background information and defines objectives including contextualization and this document organization section.

To give a state of the art overview of the theory and practice associated with Fourier Transforms, [chapter 2](#) covers most of basic understandings and algorithms needed for later chapters, this will only take simple approaches to each concept to give intuitive insight and empirical explanations without proving it formally.

STATE OF THE ART

2.1 FOURIER TRANSFORM

2.1.1 What is Fourier Transform

The **Fourier Transform** is a mathematical method to transform the domain referred to as *time* of a function, to the *frequency* domain, intuitively the Inverse Fourier Transform is the corresponding method to reverse that process and reconstruct the original function from the one in *frequency* domain representation.

Although there are many forms, the Fourier Transform key definition can be described as:

$$X(f) = \int_{-\infty}^{+\infty} x(t)e^{-ift} dt \quad (1)$$

$$x(t) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} X(f)e^{-ift} df \quad (2)$$

- $x(t), \forall t \in \mathbb{R} \rightarrow$ function in *time* domain representation with real t .
- $X(f), \forall f \in \mathbb{R} \rightarrow$ function in *frequency* domain representation with real f , also called the Fourier Transform of $x(t)$
- $i \rightarrow$ imaginary unit $i = \sqrt{-1}$

This formulation shows the usage of complex-valued domain since the imaginary unit i doesn't represent a value in the set of real numbers, making the fourier transform range from real to complex values, one complex coefficient per frequency $X : \mathbb{R} \rightarrow \mathbb{C}$

If we take into account the Euler's formula, we can replace the Fourier Transform for an equivalent, fragmenting the euler constant for a sine and cosine pair.

$$e^{ix} = \cos x + i \sin x \quad (3)$$

$$X(f) = \int_{-\infty}^{+\infty} x(t)(\cos(-ft) + i \sin(-ft))dt \quad (4)$$

Hence, we can break the Fourier Transform apart into two formulas that give each coefficient of the sine and cosine components as functions without dealing with complex numbers.

$$\begin{aligned} X_a(f) &= \int_{-\infty}^{+\infty} x(t) \cos(ft) dt \\ X_b(f) &= \int_{-\infty}^{+\infty} x(t) \sin(ft) dt \end{aligned} \quad (5)$$

The above definition of the Fourier Integral Equation 1 can only be valid if the integral exists for every value of the parameter f . This model of the fourier transform applied to infinite domain functions is called **Continuous Fourier Transform** and its targeted to the calculation of the this transform directly to functions with only finite discontinuities in $x(t)$.

2.1.2 Where it is used

It's noticable the presence of Fourier Transforms in a great variety of apparent unrelated fields of application, even the FFT is often called ubiquitous¹ due to its effective nature of solving a great hand of problems for the most intended complexity time. Some of the fields of application include Applied Mechanics, Signal Processing, Sonics and Acoustics, Biomedical Engineering, Instrumentation, Radar, Numerical Methods, Electromagnetics, Computer Graphics and more Brigham (1988).

One of the most well known cases of application is **Signal Analysis**, the Fourier Transform is probably the most important tool for analyzing signals, when representing a signal with amplitude as function of time, a signal can be translated to the frequency domain, a domain that consists of signals of sines and consines waves of varied frequencies, as demonstrated in 1, but to calculate the coefficients of those waves we need to use the Fourier Transform.

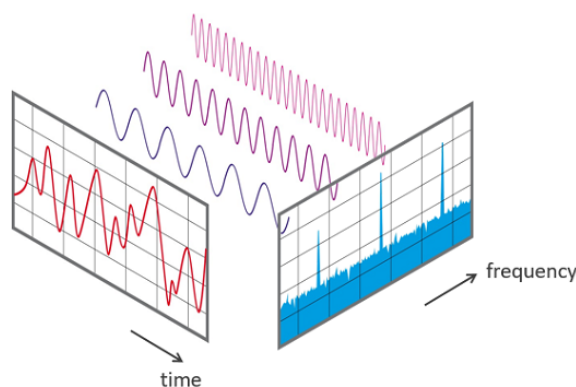


Figure 1: Time to frequency signal decomposition **Source:** NTiAudio

¹ present, appearing, or found everywhere.

Since the sine and cosine waves are in simple waveforms they can then be manipulated with relative ease. This process is constantly present in communications since the transmission of data over wires and radio circuits through signals and most devices nowadays perform it frequently

And much more applications such as polynomial multiplication [Jia \(2014\)](#), numerical integration, time-domain interpolation, x-ray diffraction ...

2.2 DISCRETE FOURIER TRANSFORM

The Fourier Transform of a finite sequence of equally-spaced samples of a function is called the **Discrete Fourier Transform** (DFT), it converts a finite set of values in *time* domain to *frequency* domain representation. It's the most important type of transform since it deals with a discrete amount of data and has the popular algorithm in which is the center of attention of fourier transforms, which can be implemented in machines and be computed by specialized hardware.

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{i2\pi}{N}kn} \quad (6)$$

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k \cdot e^{\frac{i2\pi}{N}kn} \quad (7)$$

Notably, the discrete version of the Fourier Transform has some obvious differences since it deals with a discrete time sequence, the first difference is that the sum covers all elements of the input values instead of integrating the infinite domain of the function, but we can also notice that the exponential, similar to the aforesaid, divides the values by N (N being the total number of elements in the sequence) due to the inability to look at frequency and time ft continuously we instead take the k 'th frequency over n .

We can have a more simplified expansion of this formula with:

$$X_k = x_0 + x_1 e^{\frac{i2\pi}{N}k} + \dots + x_{N-1} e^{\frac{i2\pi}{N}k(N-1)}$$

Having this sum simplified we then only need to resolve the complex exponential, and we can do that by replacing the $e^{\frac{i2\pi}{N}kn}$ by the euler formula as mentioned before to reduce the maths to a simple sum of real and imaginary numbers.

$$X_k = x_0 + x_1(\cos b_1 + i \sin b_1) + \dots + x_{N-1}(\cos b_{N-1} + i \sin b_{N-1}) \quad (8)$$

$$\text{where } b_n = \frac{2\pi}{N}kn$$

Finally we'll be left with the result as a complex number

$$X_k = A_k + iB_k$$

EXAMPLE Let us now follow an example of calculation of the DFT for a sequence x with N number of elements.

$$x = \begin{bmatrix} 1 & 0.707 & 0 & -0.707 & -1 & -0.707 & 0 & 0.707 \end{bmatrix}$$

$$N = 8$$

With this sequence we now want to transform it into the frequency domain, and for that we need to apply the Discrete Fourier Transform to each element $x_n \rightarrow X_k$, thus, for each k 'th element of X we apply the DFT for every element of x .

$$X_0 = 1 \cdot e^{-\frac{i2\pi}{8} \cdot 0 \cdot 0} + 0.707 \cdot e^{-\frac{i2\pi}{8} \cdot 0 \cdot 1} + \dots + 0.707 \cdot e^{-\frac{i2\pi}{8} \cdot 0 \cdot 7}$$

$$= (0 + 0i)$$

$$X_1 = 1 \cdot e^{-\frac{i2\pi}{8} \cdot 1 \cdot 0} + 0.707 \cdot e^{-\frac{i2\pi}{8} \cdot 1 \cdot 1} + \dots + 0.707 \cdot e^{-\frac{i2\pi}{8} \cdot 1 \cdot 7}$$

$$= (4 + 0i)$$

...

$$X_7 = 1 \cdot e^{-\frac{i2\pi}{8} \cdot 7 \cdot 0} + 0.707 \cdot e^{-\frac{i2\pi}{8} \cdot 7 \cdot 1} + \dots + 0.707 \cdot e^{-\frac{i2\pi}{8} \cdot 7 \cdot 7}$$

$$= (4 + 0i)$$

And that will produce our complex-valued output in frequency domain, as simple as that.

$$X = \begin{bmatrix} 0i & 4 + 0i & 0i & 0i & 0i & 0i & 0i & 4 + 0i \end{bmatrix}$$

2.2.1 Matrix multiplication

The example shown above is done sequentially as if each frequency bin is computed individually, but there's a way to calculate the same result by using matrix multiplication [Rao and Yip \(2018\)](#). Since the operations are done equally without any extra step we can group all analysing function sinusoids ($e^{-\frac{i2\pi}{N}kn}$), also referred to as twiddle factors.

$$W = \begin{bmatrix} \omega_N^{0 \cdot 0} & \omega_N^{1 \cdot 0} & \dots & \omega_N^{(N-1) \cdot 0} \\ \omega_N^{0 \cdot 1} & \omega_N^{1 \cdot 1} & \dots & \omega_N^{(N-1) \cdot 1} \\ \vdots & \vdots & \ddots & \vdots \\ \omega_N^{0 \cdot (N-1)} & \omega_N^{1 \cdot (N-1)} & \dots & \omega_N^{(N-1) \cdot (N-1)} \end{bmatrix} = \begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & \omega & \dots & \omega^{(N-1)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{(N-1)} & \dots & \omega^{(N-1) \cdot (N-1)} \end{bmatrix}$$

$$\text{where } \omega_N = e^{-\frac{i2\pi}{N}}$$

The substitution variable ω allows us to avoid writing extensive exponents.

The symbol W represents the transformation matrix of the Discrete Fourier Transform, also called DFT matrix, and its inverse can be defined as.

$$W^{-1} = \frac{1}{N} \cdot \begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & \omega_N & \dots & \omega_N^{(N-1)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_N^{(N-1)} & \dots & \omega_N^{(N-1) \cdot (N-1)} \end{bmatrix}$$

$$\text{where } \omega_N = e^{-\frac{i2\pi}{N}}$$

By using this matrix multiplication form we can have a more efficient way to compute the DFT in hardware.

$$X = W \cdot x$$

$$x = W^{-1} \cdot X$$

Moreover we might also want to normalize the matrix by \sqrt{N} for both Matrix DFT and IDFT instead of just normalizing the IDFT by N , that will make W a unitary matrix [Horn and Johnson \(2012\)](#). The advantage of using a unitary matrix is that we only need to reassign the constant substitution variable ω_N to be able to invert the dft, the matrix multiplication stays the same for both DFT and IDFT. Nevertheless later we will verify that the use of sqrt function isn't desirable for the implementation of any dft.

EXAMPLE Continuing the example [2.2](#), we can adapt the application of the DFT to the matrix multiplication form.

$$W = \begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & \omega_8 & \dots & \omega_8^7 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_8^7 & \dots & \omega_8^{49} \end{bmatrix}$$

$$\text{where } \omega_8 = e^{\frac{i2\pi}{8}}$$

$$X = W \cdot x = W \cdot \begin{bmatrix} 1 \\ 0.707 \\ \vdots \\ 0.707 \end{bmatrix} = \begin{bmatrix} 0 \\ 4 + 0i \\ \vdots \\ 4 + 0i \end{bmatrix}$$

It's conspicuous that the complexity time for each multiplication of every singular term of the sequence with the

complex exponential value is $O(N^2)$, hence, the computation of the Discrete Fourier Transform rises exponentially as we use longer sequences. Therefore, over time new algorithms and techniques were developed to increase the performance of this transform due to its usefulness.

2.3 FAST FOURIER TRANSFORM

The Fast Fourier Transform (FFT) is a family of algorithms that effectively compute the Discrete Fourier Transform (DFT) of a sequence and its inverse. These algorithms essentially compute the same result as the DFT but the direct usage of the DFT formulation is too slow for its applications. Thus, FFT algorithms exploit the DFT matrix structure by employing a divide-and-conquer approach [Chu and George \(1999\)](#) to segment its application.

Over time several variations of the algorithms were developed to improve the performance of the DFT and many aspects were rethought in the way we apply and produce the resulting transform, in this section we'll cover at some of those variations.

There are many algorithms and approaches on the FFT family such as the well known Cooley-Tukey, known for its simplicity and effectiveness to compute any sequence with size as a power of two, but also Rader's algorithm [Rader \(1968\)](#) and Bluestein's algorithm [Bluestein \(1970\)](#) which both deal prime sized sequences, and even the Split-radix FFT [Yavne \(1968\)](#) that recursively expresses a DFT of length N in terms of one smaller DFT of length $N/2$ and two smaller DFTs of length $N/4$.

The next two sections focus on the Cooley–Tukey algorithm, most specifically the radix-2 decimation-in-time (DIT) FFT and radix-2 decimation-in-frequency (DIF) FFT, which requires the input sequence to have a power of two size.

2.3.1 Radix-2 Decimation-in-Time FFT

The Radix-2 Decimation-in-Time FFT algorithm rearranges the original Discrete Fourier Transform (DFT) formula into two subtransforms, one as a sum over the even indexed elements and other as a sum over the odd indexed elements. The [Equation 9](#) describes this procedure with already simplified math, and hints the recursive decomposition of the DFT of size $N/2$.

$$X_k = \sum_{n=0}^{\frac{N}{2}-1} x_{2n} \cdot \omega_{N/2}^{k(2n)} + \omega_N^k \sum_{n=0}^{\frac{N}{2}-1} x_{2n+1} \cdot \omega_{N/2}^{k(2n+1)} \quad (9)$$

$$\text{where } \omega_N = e^{\frac{j2\pi}{N}}$$

This formulation successfully segments the full sized DFT into two $N/2$ sized DFT's of the even and odd indexed elements where the later is multiplied by a twiddle factor ω_N^k .

This algorithm is a Radix-2 Decimation-in-Time in the sense that the time values are regrouped in 2 subtransforms, and the decomposition reduces the time values to the frequency domain. Since the understanding of

this algorithm can be applied recursively, the [Figure 2](#) illustrates the basic behaviour and represents the $N/2$ subtransforms with boxes that can be filled by the recursive application of this algorithm to produce the frequency domain sequence.

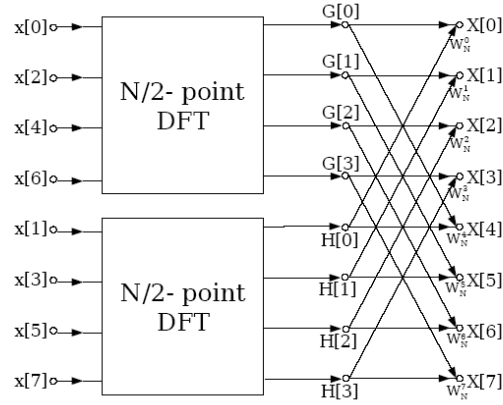


Figure 2: Radix-2 Decimation-in-Time FFT **Source:** [Jones \(2014\)](#)

Effectively, this smaller DFT's are recursively reduced by this algorithm until there's only the computation of a length-2 DFT where it's only applied the Cooley-Tukey butterfly operation [Chu and George \(1999\)](#) illustrated in [Figure 3](#).

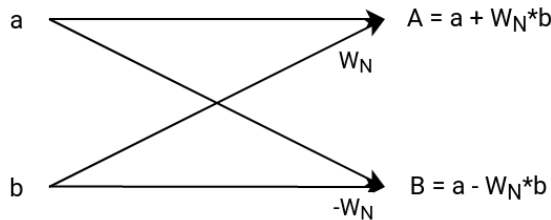


Figure 3: Cooley-Tukey butterfly

The complexity work within the algorithm is distributed with the DIT approach which decomposes each DFT by 2 having $\log N$ stages [Smith \(2007\)](#) while there are approximately N complex multiplications needed for each stage of the DIT decomposition, therefore the multiplication complexity for a N sized DFT is reduced from $O(N^2)$ to $O(N \log N)$ without any programming specific optimizations.

In practice, [algorithm 1](#) demonstrates the aforesaid with an iterative representation of a possible implementation. Although this algorithm is congruent with a code implementation, it's worth noting that the input sequence can either have real or complex numbers, since the arithmetic is the same for both domains the only thing that needs to be specialized is the operator overloading in the inner most loop.

Algorithm 1: Radix-2 Decimation-in-Time Forward FFT

Data: Sequence *in* with size *N* power of 2

Result: Sequence *out* with size *N* with the DFT of the input

```
/* Bit reversal step                                     */
foreach i = 0 to N - 1 do
    | out[bit_reverse(i)] ← in[i]
end
/* FFT                                                  */
foreach s = 1 to log N do
    | m ← 2s;
    | wm ← exp(−2πi/m);
    | foreach k = 0 to N - 1 by m do
    |     | w ← 1;
    |     | foreach j = 0 to m/2 do
    |     |     | bw ← w · out[k + j + m/2];
    |     |     | a ← out[k + j];
    |     |     | out[k + j] ← a + bw;
    |     |     | out[k + j + m/2] ← a - bw;
    |     |     | w ← w · wm;
    |     | end
    | end
end
return out;
```

2.3.2 Radix-2 Decimation-in-Frequency FFT

The Radix-2 Decimation-in-Frequency FFT algorithm is very similar to the DIT approach, its based on the same principle of divide-and-conquer but it rearranges the original Discrete Fourier Transform (DFT) into the computation of two transforms, one with the even indexed elements and other with the odd indexed elements; as in this simplified formulation [Equation 10](#).

$$\begin{aligned} X_{2k} &= \sum_{n=0}^{\frac{N}{2}-1} (x_n + x_{n+\frac{N}{2}}) \cdot \omega_{N/2}^{kn} \\ X_{2k+1} &= \sum_{n=0}^{\frac{N}{2}-1} ((x_n - x_{n+\frac{N}{2}}) \cdot \omega_{N/2}^{kn}) \cdot \omega_N^n \end{aligned} \tag{10}$$

where $\omega_N = e^{\frac{j2\pi}{N}}$

The DFT divided into these two transforms from the full sized DFT By separating these two transforms from the full sized DFT we get two distinct

Notably, this formulation distinguishes the full sized DFT into two $N/2$ sized DFT's of the even and odd indexed elements where the later is multiplied by a twiddle factor ω_N^k with both outside the same context.

This algorithm is a Radix-2 Decimation-in-Frequency since the DFT is decimated into two distinct smaller DFT's and the frequency samples will be computed separately in different groups, as if the regrouping of the DFT's would reduce directly to the frequency domain. Since the understanding of this algorithm can be applied recursively, the Figure 4 illustrates the this behaviour and represents the $N/2$ subtransforms with boxes that can be filled by the recursive application of this algorithm to produce the frequency domain sequence. Additionally this illustration can be compared to Figure 2 since both are symmetrically identical.

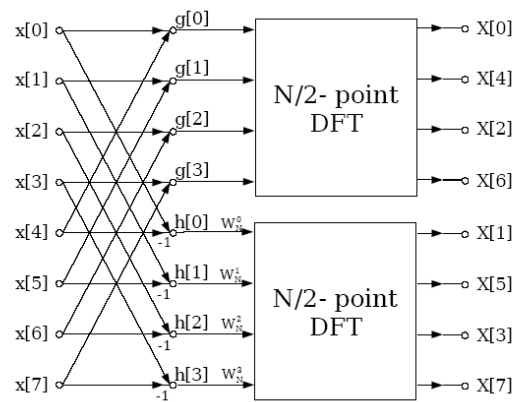


Figure 4: Radix-2 Decimation-in-Frequency FFT **Source:** Jones (2014)

Similarly to the DIT version, the DFT can be recursively reduced by the DIF algorithm until theres only the computation of a length-2 DFT where its only applied the Gentleman-Sande butterfly operation Chu and George (1999) illustrated in Figure 5.

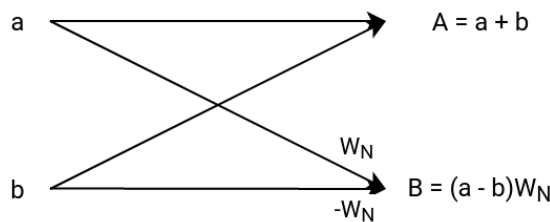


Figure 5: Gentleman-Sande butterfly

Since this algorithm has similarities with the DIT, its complexity also lives to this similarity, maintaining the same $O(N \log N)$ for number of multiplications, despite that, Figure 5 and Figure 3 might look different in number of arithmetic operations since the first has 1 addition, 1 subtraction, and 2 multiplications, and the second has 1 addition, 1 subtraction, and 1 multiplication, but effectively the $W_N \cdot b$ can be reused and only computed once as seen in algorithm 1.

In practice, [algorithm 2](#) demonstrates the aforesaid with an iterative representation of a possible implementation. Although this algorithm is congruent with a code implementation, its worth noting that the input sequence can either have real or complex numbers, since the arithmetic is the same for both domains the only thing that needs to be specialized is the operator overloading in the inner most loop.

Algorithm 2: Radix-2 Decimation-in-Frequency Forward FFT

Data: Sequence *in* with size *N* power of 2

Result: Sequence *out* with size *N* with the DFT of the input

```

/* FFT                                                    */
foreach s = 0 to log N - 1 do
    gs ← N >> s;
    wgs ← exp(2πi/gs);
    foreach k = 0 to N - 1 by gs do
        w ← 1;
        foreach j = 0 to gs/2 do
            a ← in[k + j + gs/2];
            b ← in[k + j];
            in[k + j] ← a + b;
            in[k + j + gs/2] ← (a - b) · w;
            w ← w · wgs;
        end
    end
end
/* Bit reversal step                                        */
foreach i = 0 to N - 1 do
    | out[bit_reverse(i)] ← in[i]
end
return out;

```

ALGORITHMS ANALYSIS

To flavour this pre-dissertation report some work of benchmarking and analysis were done to compete with the theoretical explanations addressed on [chapter 2](#). Hence, some implementations were tested to provide coherence to what has been studied, and algorithms such as [algorithm 1](#) Radix-2 Decimation-in-Time [algorithm 2](#) Radix-2 Decimation-in-Frequency were timed in [Table 2](#).

	Size 128	Size 256	Size 512	Size 1024	Size 2048
DFT	5.16593	17.2782	70.5689	293.104	1246.44
FFT DIT	0.169113	0.37668	0.86415	1.8793	4.47742
FFT DIF	0.159458	0.378722	0.881921	1.90661	4.13369
Recursive FFT	0.210895	0.485643	1.4421	2.32922	5.1178

Table 2: FFT algorithms benchmark. Results are measured in milliseconds for forward and inverse computation with varying input sizes

As we can see the Discrete Fourier Transform increases exponentially for higher sized sequences, as expected all FFT variants perform critically better than the original formulation.

One variant that wasn't exposed much in the above chapters is the Recursive FFT algorithm, which corresponds to a Decimation-in-Time approach with recursive reduction, therefore this algorithm aggregates the divide-and-conquer method but with the disadvantage of recursive function overhead. This recursive look at the DIT approach can be easier to implement since the bit reversal step isn't explicitly applied before starting the FFT.

Finally, as expected the DIT and DIF algorithms overrule the other alternatives for every sized input.

COMPUTATION OF THE FOURIER TRANSFORM

4.1 IMPROVING THE COOLEY-TUKEY ALGORITHM

Nowadays there is a lot more to the computation of FFT's than just the basic Cooley-Tukey algorithm described in 2.3.1 there are more algorithms, variations and improvements that enhance the computation in many aspects. Even the selected hardware can change the restrictions on the computation of this primitive.

One could optimize the fft by providing precomputed twiddle factors, or spare space for the output sequence by adopting an in-place FFT algorithm and all those factors influence the performance. Evidently a balance must be established depending on the constraints for the FFT to take advantage on the target environment.

4.1.1 *Natural order Cooley-Tukey*

The main characteristic of CooleyTukey algorithm is the presence of the bit reversal step needed to store the result in natural order. Both the DIT and DIF version apply this step, for the first it is done before the first stage, and for the later version its delayed up until the end of the last stage to be applied. This bit reversal step is essential for this algorithm to obtain the same input sequence if the inversion of the forward fft is computed, however, it is possible to change the algorithm a bit to rearrange the output in natural order.

To replace the bit reversal step we can introduce an equivalent procedure applied in each stage after the butterflies are computed, this procedure is illustrated in [Figure 6](#)

However this change gets rid of the bit reversal step, it is not optimal, for each stage there's a swap over the subtransform elements beyond the butterflies and that's what leads us to the next section, the stockham algorithm.

4.1.2 *Stockham algorithm*

4.1.3 *Radix-4 instead of Radix-2*

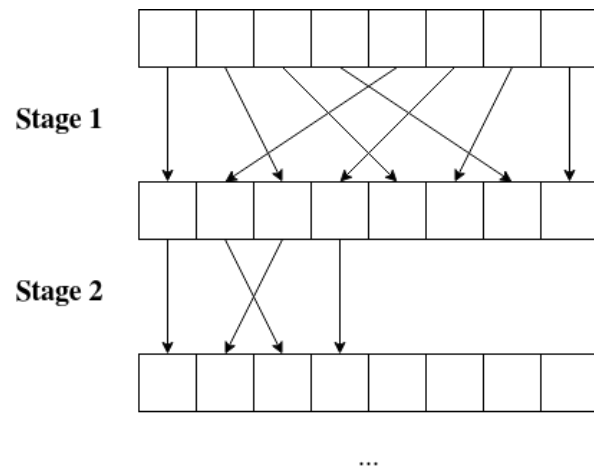


Figure 6: Natural order composition

IMPLEMENTATION ON THE GPU

5.1 FOURIER TRANSFORM ON THE GPU

5.2 IMPLEMENTATION IN GLSL

ANALYSIS AND COMPARISON

6.1 POPULAR IMPLEMENTATIONS

6.1.1 *cuFFT*

6.1.2 *FFTW*

6.2 COMPARISON WITH GLSL IMPLEMENTATION

6.2.1 *Analysis*

Popular implementations

BIBLIOGRAPHY

- Leo Bluestein. A linear filtering approach to the computation of discrete fourier transform. *IEEE Transactions on Audio and Electroacoustics*, 18(4):451–455, 1970.
- E Oran Brigham. *The fast Fourier transform and its applications*. Prentice-Hall, Inc., 1988.
- Eleanor Chu and Alan George. *Inside the FFT black box: serial and parallel fast Fourier transform algorithms*. CRC press, 1999.
- Fynn-Jorin Flügge. Realtime gpgpu fft ocean water simulation. Technical report, 2017.
- Matteo Frigo and Steven G Johnson. Fftw: Fastest fourier transform in the west. *Astrophysics Source Code Library*, pages ascl–1201, 2012.
- Roger A Horn and Charles R Johnson. *Matrix analysis*. Cambridge university press, 2012.
- Yan-Bin Jia. Polynomial multiplication and fast fourier transform. *Com S*, 477:577, 2014.
- Douglas L Jones. Digital signal processing: A user’s guide. 2014.
- NTiAudio. Fast Fourier Transformation FFT - Basics. <https://www.nti-audio.com/en/support/know-how/fast-fourier-transform-fft>. Accessed at 2022-01-28.
- CUDA Nvidia. Cufft library (2018). URL developer. [nvidia.com/cuFFT](https://developer.nvidia.com/cuFFT).
- Charles M Rader. Discrete fourier transforms when the number of data samples is prime. *Proceedings of the IEEE*, 56(6):1107–1108, 1968.
- Kamisetty Ramam Rao and Patrick C Yip. *The transform and data compression handbook*. CRC press, 2018.
- Julius Orion Smith. *Mathematics of the discrete Fourier transform (DFT): with audio applications*. Julius Smith, 2007.
- R Yavne. An economical method for calculating the discrete fourier transform. In *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*, pages 115–125, 1968.