UNIVERSIDADE DO ESTADO DO AMAZONAS - UEA

ESCOLA SUPERIOR DE TECNOLOGIA

ENGENHARIA DE COMPUTAÇÃO

IASMIM SOUZA DA CUNHA

ANÁLISE COMPARATIVA DE ALGORITMOS QUE COMPUTAM TRANSFORMADA DISCRETA DE FOURIER E HARTLEY

Manaus

2013

IASMIM SOUZA DA CUNHA

ANÁLISE COMPARATIVA DE ALGORITMOS QUE COMPUTAM TRANSFORMADA DISCRETA DE FOURIER E HARTLEY

Trabalho de Conclusão de Curso apresentado à banca avaliadora do Curso de Engenharia de Computação, da Escola Superior de Tecnologia, da Universidade do Estado do Amazonas, como pré-requisito para obtenção do título de Engenheira de Computação.

Orientador: Prof. Dr. Raimundo Corrêa de Oliveira

Manaus

2013

Universidade do Estado do Amazonas - UEA
Escola Superior de Tecnologia - EST

Reitor:
Cleinaldo de Almeida Costa
Vice-Reitor:
Mario Augusto Bessa de Figueiredo
Diretor da Escola Superior de Tecnologia:
Cleto Cavalcante de Souza Leal
Coordenador do Curso de Engenharia de Computação:
Raimundo Corrêa de Oliveira
Coordenador da Disciplina TCC:
Tiago Eugênio de Melo

Banca Avaliadora composta por: *Data da Defesa: 29/11/2013.*
Dr. Raimundo Corrêa de Oliveira *(Orientador)*
Dr. Jucimar Maia da Silva Junior
Dr. Ernandes Ferreira de Melo

CIP - Catalogação na Publicação

C972a CUNHA, Iasmim

Análise comparativa de algoritmos que computam transformada discreta de Fourier e Hartley / Iasmim Cunha; [orientado por] Prof. Dr. Raimundo Corrêa de Oliveira - Manaus: UEA, 2013.

51p.: il.; 29cm

Inclui Bibliografia

Trabalho de Conclusão de Curso (Graduação em Engenharia de Computação). Universidade do Estado do Amazonas, 2013.

CDU:004.4

IASMIM SOUZA DA CUNHA

ANÁLISE COMPARATIVA DE ALGORITMOS QUE COMPUTAM TRANSFORMADA RÁPIDA DE FOURIER E HARTLEY

Trabalho de Conclusão de Curso apresentado à banca avaliadora do Curso de Engenharia de Computação, da Escola Superior de Tecnologia, da Universidade do Estado do Amazonas, como pré-requisito para obtenção do título de Engenheira de Computação.

BANCA EXAMINADORA

Prof. Raimundo Corrêa de Oliveira, Doutor
UNIVERSIDADE DO ESTADO DO AMAZONAS

Prof. Jucimar Maia da Silva Junior, Doutor
UNIVERSIDADE DO ESTADO DO AMAZONAS

Prof. Ernandes Ferreira de Melo, Doutor
UNIVERSIDADE DO ESTADO DO AMAZONAS

Agradecimentos

Primeiramente agradeço a Deus, pois sem ELE não seria nada. Aos meus pais e familiares por terem me ajudado nos momentos difíceis de minha vida, onde sempre me deram forças para continuar, e conseguir vencer mais um desafio de minha trajetória. Ao meu professor e orientador de TCC, Raimundo Corrêa de Oliveira, pela amizade, incentivo e profissionalismo. Agradeço ao meu noivo Rodrigo, por todo amor, compreensão e incentivos dados, que me ajudaram a nunca desistir.

Resumo

Este trabalho apresentará um estudo comparativo entre os algoritmos que computam as transformada rápidas de Fourier e Hartley. Tais algoritmos são essenciais em aplicações dos ramos de processamento de sinais e imagens. Conhecer o desempenho dessas transformadas rápidas é importante no desenvolvimento das aplicações que a utilizam. O ponto central deste estudo é a comparação de eficiência em termos de tempo de processamento e memória necessária para computar ambas às transformadas. Além disso, esta monografia apresenta gráficos ilustrando o desempenho de cada algoritmo implementado.

Palavras Chave: transformada, fourier , hartley ,algoritmo rápido.

Abstract

This paper presents a comparative study of algorithms that compute the fast Fourier transform and fast Hartley transform. Such algorithms are essential in applications at field of signal and image processing. Knowing the performance of these transformed fast is important in the development of applications that use it. The central point of this study is to compare efficiency in terms of processing time and memory required to compute both transformed. Additionally, this paper presents graphs showing the performance of each implemented algorithm.

Keywords: Fourier, Hartley, transform, fast algorithm.

Sumário

Lista de Tabelas **x**

Lista de Figuras **xi**

Lista de Códigos **xii**

1 Introdução **1**
1.1 Objetivo Geral 3
1.2 Objetivos Específicos 3
1.3 Metodologia 4
1.4 Organização da Monografia 4

2 Transformadas Discretas de Fourier e Hartley **6**
2.1 Transformada Discreta de Fourier 7
2.2 Discreta de Hartley 8
2.3 Algoritmos Rápidos de Fourier e Hartley 9
2.3.1 Algoritmo FFT Cooley - Tukey base 2 9
2.3.2 Algoritmo FFT Cooley - Tukey base 4 11
2.3.3 Algoritmo FFT Split Radix 12
2.3.4 Algoritmo FHT Cooley Tukey base 2 13
2.3.5 Algoritmo FHT Cooley Tukey base 4 13
2.3.6 Algoritmo FHT Split Radix 14

3 Implementações dos Algoritmos **15**
3.1 Software Eclipse 15
3.2 Monitor de Desempenho do Windows 16
3.3 System Load Indicator 17
3.4 Software MatLab 18

3.5 Software R 19
3.6 Análise de Complexidade 20
3.7 Implementações dos algoritmos 22
3.7.1 Algoritmos baseado na definição DFT 23
3.7.2 Algoritmos baseado na definição DHT 24
3.7.3 Algoritmos rápidos que DFT 25
3.7.4 Algoritmos rápidos que Computam DHT 29
3.8 Processo de extração de dados e teste 33

4 Resultados **34**
4.1 Apresentação dos resultados obtidos 34
4.1.1 Resultados de tempo de processamentos 34
4.1.2 Resultados de memória ocupada 36

5 Conclusão e Trabalhos Futuros **46**

Referências Bibliográficas **48**

Lista de Tabelas

4.1 Tabela de amostras de Tempo de processamento em segundos dos algoritmos Transformadas de Fourier em sistema operacional Windows 35
4.2 Tabela de amostras de Tempo de processamento em segundos dos algoritmos Transformadas de Fourier em sistema operacional Linux 35
4.3 Tabela de amostras de Tempo de processamento em segundos dos algoritmos Transformadas de Hartley em sistema operacional Windows 35
4.4 Tabela de amostras de Tempo de processamento em segundos dos algoritmos Transformadas de Hartley em sistema operacional Linux 36
4.5 Tabela de amostras de Memória em KB dos algoritmos Transformadas de Fourier em sistema operacional Windows 36
4.6 Tabela de amostras de Memória em KB dos algoritmos Transformadas de Fourier em sistema operacional Linux 36
4.7 Tabela de amostras de Memória em KB dos algoritmos Transformadas de Hartley em sistema operacional Windows 37
4.8 Tabela de Amostras de Memória em KB dos algoritmos Transformadas de Hartley em sistema operacional Linux 37

Lista de Figuras

2.1 Célula básica (borboleta) para a computação do Cooley - Tukey base 2 . . 10
2.2 Esquema do algoritmo Cooley - Tukey base 2 para N = 4 11
2.3 Célula básica (borboleta) para a computação do Cooley - Tukey base 4 . . 12
2.4 Célula básica (borboleta) para a computação Split Radix 13

3.1 Tela principal do eclipse 16
3.2 Tela principal do monitor de desempenho do windows. 17
3.3 Tela principal do System Load Indicator. 18
3.4 Tela principal do MATLAB 19
3.5 Tela principal do software R. 20

4.1 Amostras de Tempo de processamento, em segundos, dos algoritmos de Transformada discreta de Fourier em sistema operacional Windows 38
4.2 Amostras de Tempo de processamento, em segundos, dos algoritmos de Transformada discreta de Fourier em sistema operacional Linux 38
4.3 Amostras de memória ocupada, em KB, dos algoritmos de Transformada discreta de Fourier em sistema operacional Windows 39
4.4 Amostras de memória ocupada, em KB, dos algoritmos de Transformada discreta de Fourier em sistema operacional Linux 39
4.5 Amostras de Tempo de processamento, em segundos, dos algoritmos de Transformada discreta de Hartley em sistema operacional Windows 40
4.6 Amostras de Tempo de processamento, em segundos, dos algoritmos de Transformada discreta de Hartley em sistema operacional Linux 40
4.7 Amostras de memória ocupada, em KB, dos algoritmos de Transformada discreta de Hartley em sistema operacional Windows 41
4.8 Amostras de memória ocupada, em KB, dos algoritmos de Transformada discreta de Hartley em sistema operacional Linux 41

4.9 Amostras de Tempo de processamento em segundos dos algoritmos rápidos de transformadas de Fourier em sistema operacional Windows 42
4.10 Amostras de Tempo de processamento em segundos dos algoritmos rápidos de transformadas de Fourier em sistema operacional Linux 42
4.11 Amostras de Memória em KB dos algoritmos rápidos de transformadas de Fourier em sistema operacional Windows 43
4.12 Amostras de Memória em KB dos algoritmos rápidos de transformadas de Fourier em sistema operacional Linux 43
4.13 Amostras de Tempo de processamento em segundos dos algoritmos rápidos de transformadas de Hartley em sistema operacional Windows 44
4.14 Amostras de Tempo de processamento em segundos dos algoritmos rápidos de transformadas de Hartley em sistema operacional Linux 44
4.15 Amostras de Memória em KB dos algoritmos rápidos de transformadas de Hartley em sistema operacional Windows 45
4.16 Amostras de Memória em KB dos algoritmos rápidos de transformadas de Hartley em sistema operacional Linux 45

Lista de Códigos

3.6.1 *Pseudo código DFT* 21
3.6.2 *Pseudo código FFT Cooley Tukey base 2* 22
3.7.1 *Código principal do algoritmo DFT* 23
3.7.2 *Código principal do algoritmo DHT* 24
3.7.3 *Código principal do algoritmo FFT Cooley Tukey base 2* 25
3.7.4 *Código principal do algoritmo FFT Cooley Tukey base 4* 26
3.7.5 *Código principal do algoritmo FFT Split Radix(parte 1)* 27
3.7.6 *Código principal do algoritmo FFT Split Radix(parte 2)* 28
3.7.7 *Código principal do algoritmo FHT Cooley Tukey base 2* 29
3.7.8 *Código principal do algoritmo FHT Cooley Tukey base 4* 30
3.7.9 *Código principal do algoritmo FHT Split Radix (parte 1)* 31
3.7.10 *Código principal do algoritmo FHT Split Radix (parte 2)* 32

Capítulo 1

Introdução

A transformada Discreta de Fourier ***(DFT, do inglês Discrete Fourier Transform)*** é uma ferramenta amplamente empregada em processamento de sinais, pois, em sinais tais como áudio, fala, etc, sua avaliação no domínio da frequência carrega informações que terão aproveitamento para aplicações diversas. Este tipo de transformada consiste em decompor um sinal em suas componentes elementares seno e cosseno pois qualquer sinal pode ser representado pela soma de senos e cossenos conforme demonstrado pelo matemático e físico francês Jean Baptiste Joseph Fourier (1768-1830). [Fourier1822]

Outra transformada também utilizada para processamento de sinais é a transformada de Hartley, esta é uma transformada integral bastante relacionada com a transformada de Fourier, mas que possui as vantagens de evitar a presença de números complexos no cálculo. Ela foi proposta por R. V. L. Hartley em 1942. A versão discreta, chamada de transformada discreta de Hartley ***(DHT, do inglês Discrete Hartley Transform)***, foi introduzida por R. N. Bracewell em 1983. [Bracewell1983]

A Transformada Discreta de Fourier, assim como a Transformada Discreta de Hartley, são ferramentas que computam o espectro de freqüência de uma seqüência finita de comprimento N, com uma complexidade computacional de N^2 multiplicações e $N(N-1)$ adições. O sucesso das aplicações de técnicas de transformação é devido, principalmente, à existência dos chamados *algoritmos rápidos*. Com isso, técnicas para computação de

transformadas discretas com uma baixa complexidade multiplicativa vêm sendo objeto de interesse há um longo tempo. [Cooley1967] [Oppnheim2013]

Todavia, as transformadas de Fourier e Hartley possuem uma grande complexidade computacional o que as tornam inúteis em aplicações reais. Com a idéia de tornar a utilização de transformação nessas aplicações o conceito de transformadas rápidas foi criado, conceito esse que utiliza a ideia de dividir para conquistar [Cooley and Tukey1965].

Em 1965, J.W. Cooley e J.W. Tukey introduziram uma idéia revolucionária que posteriormente tornou-se conhecida como a Transformada Rápida de Fourier ***(FFT, do inglês Fast Fourier Transform)***. Entretanto, pode-se atribuir a Gauss algumas das idéias propostas nesse trabalho. Os algoritmos de J.W. Cooley e J.W. Tukey são chamados de Cooley-Tukey base 2 e Cooley-Tukey base 4, que utilizam da estratégia de divisão e conquista. A FFT é um marco na teoria de algoritmos e, mais especificamente, no campo de Processamento de Sinais. [Cooley and Tukey1965]

Outro algoritmo rápido bastante pesquisado no meio acadêmico é o Split Radix ***(SRFFT, do inglês, Split Radix Fast Fourier Transform)***, proposto por Duhamel e Holmanm em 1986 [Duhamel1986]

Um algoritmo rápido foi desenvolvido para o cálculo do DHT e foi apresentado no trabalho de R. N. Bracewell Fellow, em 1984. [Bracewell1984]. Por analogia com o algoritmo de transformada rápida de Fourier (FFT), este algoritmo foi nomeado como transformada rápida de Hartley ***(FHT, do inglês, Fast Hartley Transform)***. Em 1985 no trabalho de H. V. Sorrensen, C. L. Jonick, C. S. Burrus e M. T. Headman afirmou - se as filosofias de todos os algoritmos FFT são podem ser igualmente aplicáveis ao cálculo da DHT, no trabalho de Sorrensen foi desenvolvido um conjunto de algoritmos rápidos para o cálculo da DHT, incluindo Cooley - Tukey base 2, Cooley Tukey base 4 e split radix.

Em muitos casos, a informação contida em um sinal está no domínio da frequência. Nestes casos, a forma de onda no domínio do tempo não é importante, mas sim a ampli-

tude, frequência e fase das componentes senoidais. Uma função pode ser convertida do domínio do tempo para o da frequência através da transformada de Fourier e transformada de Hartley.

Este estudo, preocupou-se em implementar os algoritmos rápidos que computam a transformada rápida de Fourier e Hartley que são: Cooley Tukey base 2, Cooley Tukey base 4 e Split Radix, utilizando a linguagem de programação C++ e a IDE eclipse,o utilizar o software MATLAB para assegurar que os resultados dos algoritmos estejam corretos. Os algoritmos foram codificados para receber um arquivo de entrada com a extensão .txt e computar a sua transformada. E assim ,obter os valores de tempo processamento e memória de cada algoritmo, para uma análise comparativa de desempenho destes, que pretende ser mostrada de forma gráfica com o auxilio do software R e utilizando tabelas.

1.1 Objetivo Geral

O principal objetivo deste projeto é analisar os desempenhos de algoritmos, que calculam transformada discreta de Fourier e Hartley.

1.2 Objetivos Específicos

- Implementar um algoritmo de Transformada discreta de Fourier pela definição;
- Implementar um algoritmo de Transformada discreta de Hartley pela definição;
- Implementar um algoritmo FFT de Cooley - Tukey base-2;
- Implementar um algoritmo FFT de Cooley - Tukey base-4;
- Implementar um algoritmo FHT de Cooley - Tukey base-2;
- Implementar um algoritmo FHT de Cooley - Tukey base-4;
- Implementar um algoritmo FFT Split Radix;

- Implementar um algoritmo FHT Split Radix;
- Analisar os dados de tempo de processamento e custo de memória para os algoritmos implementados;

1.3 Metodologia

A execução do trabalho desta pesquisa envolverá a implementação de algoritmos computacionais, que calculam a transformada de Fourier e Hartley pela definição e seus algoritmos rápidos, Cooley - Tukey base 2 e Cooley - Tukey base 4 e Split Radix, totalizando assim 8 algoritmos.

As análises serão feitas em sistema operacional Windows e Linux. As ferramentas definidas para extrair e monitorar os dados no Windows é o Monitor de Desempenho, que pertencem ao Windows ADK (Kit de Avaliação e Implantação do Windows). Para S.O Linux foi definida a ferramenta System Load Indicator para o monitoramento de custo de memória e tempo.

Para a comparação entre as transformadas de Hartley e de Fourier a implementação e teste foi feita nas condições mais semelhantes possíveis. Para a implementação foi utilizada a linguagem C++, com a Eclipse IDE for C/C++ Developers, versão Juno Service Release 2. Para verificar se os algoritmos estavam certos foi usado o software MATLAB 2012, versão 7.14.0.736 para 64 bits. Para os testes foi utilizando um computador com processador de 2,20 GHz e 4 GB de RAM rodando Windows 8 de 64 bits.

1.4 Organização da Monografia

Este trabalho está estruturado nos seguintes capítulos:

- **Capítulo 2:**

  Neste capítulo será exposto os conceitos das transformadas discretas de Fourier e das transformadas de discretas de Hartley. O capítulo também explanará a respeito das

transformadas rápidas de Fourier e das transformadas rápidas de Hartley chamadas Cooley - Tukey base 2, Cooley Tukey base 4 e Split Radix.

- **Capítulo 3:**

  Este capítulo trata do desenvolvimento do projeto, ferramentas utilizadas bem como apresenta-se o pseudocódigo para detalhar partes importantes do trabalho. O capítulo demonstra também, alguns cálculos de complexidades dos algoritmos, para que possamos provar a afirmação de que o algoritmo da transformada de Fourier e Hartley por definicão possuem complexidade de $O(N^2)$ enquanto seus algoritmos rápidos evoluíram sua complexidade para $O(NlogN)$.

- **Capítulo 4:**

  Neste capítulo, será mostrado o resultado da análise de desempenho dos algoritmos em forma de tabelas e gráficos.

- **Capítulo 5:**

  Este capítulo apresenta a conclusão, a respeito dos resultados dos teste em sistemas operacionais diferentes e aponta qual o algoritmo que, a partir das análises gráficas possuiu melhor desempenho em relação aos demais, com base nos resultados apresentados no capítulo anteriores a este e sugestões de trabalhos futuros.

Capítulo 2

Transformadas Discretas de Fourier e Hartley

A análise espectral de sinais em tempo contínuo tem como uma de suas principais ferramentas a transformada de Fourier. No entanto, muitas aplicações envolvendo esta transformada dependem de um computador digital para efetuar o processamento dos dados, sendo que seu cálculo fica inviável para sinais em tempo contínuo. Da mesma forma que a transformada discreta de Fourier ***DFT, do inglês, Discrete Fourier Transfom***. A transformada discreta de Hartley ***(DHT, do inglês Transform Discrete Hartley)*** é, uma transformada utilizada no estudo de sinais. A principal vantagem da DHT sobre a DFT é que ela transforma uma sequência de valores reais em outra sequência de valores reais, evitando a necessidade de se trabalhar com números complexos.

Existem algoritmos bastante eficientes para cálculo da DHT, a transformada rápida de Hartley ***(FHT, do inglês, Fast Hartley Transform)*** a FHT foi proposta originalmente por Bracewell em 1984 [Bracewell1984]. Assim como existem algoritmos rápidos para computar a transformada discreta de Fourier ***(FFT, do inglês, Fast Fourier Transform)***. Assim, nossa abordagem consiste em definir a DFT, sua definição e suas principais propriedades, a **DHT, do inglês, Discrete Fourier Transfom** e seus algoritmos rápidos Cooley Tukey base 2, Cooley Tukey base 4 e Split Radix.

2.1 Transformada Discreta de Fourier

A DFT é uma sequência, em vez de uma função contínua, e corresponde a amostras em frequência, igualmente espaçadas. Além de sua importância teoria como uma representação de Fourier de sequências. A DFT desempenha um importante papel na implementação de uma variedade de algoritmos de processamento digital de sinais. [Oppnheim2013] A transformada discreta de Fourier é definida matematicamente pela expressão:

$$X_k := \sum_{n=0}^{N-1} X_n e^{\frac{-i2\pi nk}{N}} \tag{2.1}$$

Considerando que :

$$e^{\frac{-2\pi nk}{N}} = cos(\frac{2\pi nk}{N}) - isen(\frac{2\pi nk}{N})$$

A equação 2.1 pode ser melhor expressada como:

$$X_k := \sum_{k=0}^{n=1} x_n[cos(\frac{2\pi nk}{N}) - isen(\frac{2\pi nk}{N})] \tag{2.2}$$

A expressão obtida da aplicação foi, mostrada em Oppnheim2013

Algumas áreas do conhecimento que tem se beneficiado da análise de Fourier são:

- Astronomia [Leonidas2000] [Ables1968];
- Imagens Médicas [Birkfellner2011];
- Áudio digital [Kondoz2004];
- Codificação de Canais [Campello2009];
- Marcas d'água [Kitamura2001];
- Comunicações [Wysocki2005];

2.2 Discreta de Hartley

Em 1983 R. N Bracewell define em [Bracewell1983] a transformada discreta de Harthey de maneira muito similar a transformada de Fourier. O par transformado é relacionado pela seguinte expressão:

$$H_k := \sum_{n=0}^{N-1} h_n cas\left(\frac{2\pi nk}{N}\right) \tag{2.3}$$

em que : $cas(x) = cos(x) + sen(x)$
a fórmula pode ser melhorada para:

$$H_k := \sum_{n=0}^{N-1} h_n \left[cos\left(\frac{2\pi nk}{N}\right) + sen\left(\frac{2\pi nk}{N}\right)\right] \tag{2.4}$$

As expressões mensionadas nesta seção foram retiradas de [Sorrensen1985]

Algumas áreas do conhecimento que tem se beneficiado da análise de Hartley são:

- Imagens Médicas [Furlani2005];
- Processamento Digital de Imagens [Massey1998];
- Criptografia [Toivonen1979];
- Codificação de Canais [Blahut1979];
- Eletrogastrografia [Cintra2005];

2.3 Algoritmos Rápidos de Fourier e Hartley

Os algoritmos de FFT e FHT são baseados, respectivamente, no principio fundamental de decomposição do cálculo da transformada de Fourier discreta e da transformada de Hartley discreta de uma sequência de comprimento N em transformadas discretas de comprimento menor, que são combinadas para formar a transformada de N potos. [Oppnheim2013]. Essas transformadas de comprimento menor podem ser decompostas novamente em transformadas ainda menores. A forma como esse princípio é implementado leva a uma variedade de algoritmos diferentes.

2.3.1 Algoritmo FFT Cooley - Tukey base 2

Existem várias formas de estruturar o algoritmo da FFT, sendo uma variante conhecida como Cooley - Tukey base 2, também chamado **Radix 2**, que opera sobre um vetor de N elementos, onde N é potência de 2. Devido a sua representação gráfica, a operação básica do algoritmo Cooley - Tukey base 2 é conhecida como "borboleta"e consiste de duas somas e uma multiplicação complexa. O algoritmo de Cooley - Tukey base 2 realiza cada operação sobre dois pontos, fornecendo a menor unidade computacional possível para a FFT . Esta configuração permite maior flexibilidade para a avaliação deste algoritmo. O algoritmo é definido matematicamente pela expressão:

$$X_k = \sum_{n=0}^{\frac{N}{2}-1} x_n \omega_n^{-2\pi(2nk)} + \sum_{n=0}^{\frac{N}{2}-1} x_n \omega_n^{-2\pi(2n+1)k} \tag{2.5}$$

em que ω_n é chamado de fator de giro e equivale a $e^{\frac{-i2\pi nk}{N}}$

Expressão mostrada foi obtida de [Cooley and Tukey1965].

O cálculo mais básico ou borboleta da FFT pode ser ilustrada na figura 2.1.

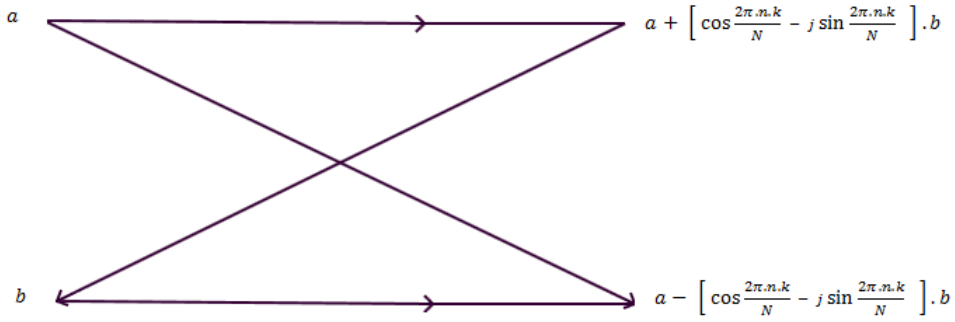

Figura 2.1: Célula básica (borboleta) para a computação do Cooley - Tukey base 2

Assim, pode-se obter a transformada de um conjunto de pontos através da transformada de seus subconjuntos par e ímpar, e a aplicação sucessiva deste processo caracteriza a FFT. Normalmente as combinações são feitas duas a duas através da borboleta, que consiste no mais básico cálculo de uma FFT. A figura 2.2 ilustra o cálculo de uma DFT de N = 4, com as entradas $x[1, 1, 1, -1]$, exemplificando a sucessão de borboletas no esquema do algoritmo Cooley Tukey base 2.

Sabendo que :

$$\omega_N^{kn} = e^{\frac{-j2\pi kn}{N}} = cos\left(\frac{2\pi kn}{N}\right) - jsen\left(\frac{2\pi kn}{N}\right)$$

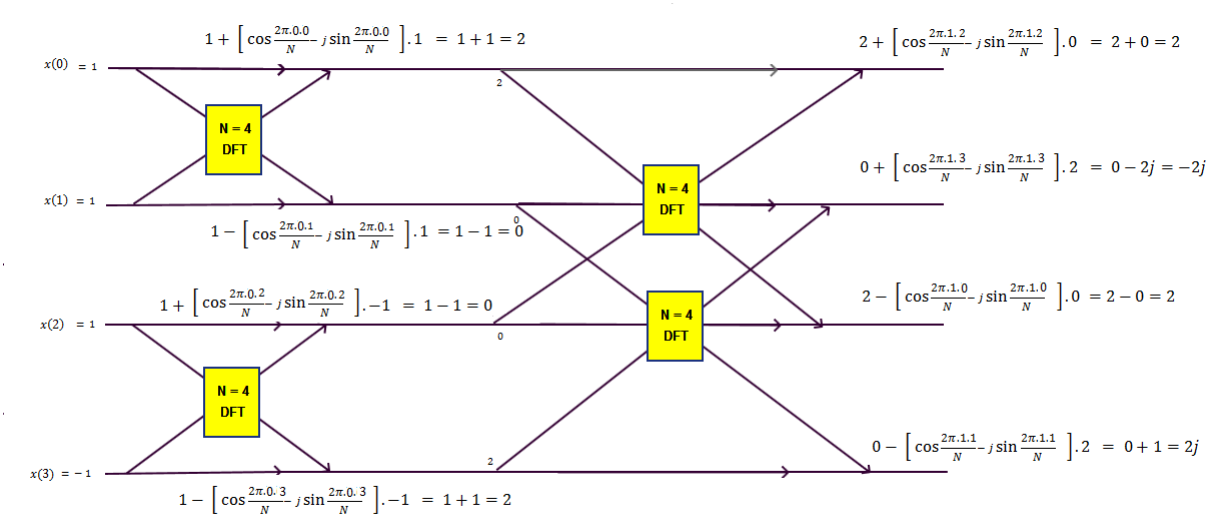

Figura 2.2: Esquema do algoritmo Cooley - Tukey base 2 para N = 4

2.3.2 Algoritmo FFT Cooley - Tukey base 4

O algoritmo Cooley Tukey base 4, também chamado **Radix 4**, consiste em dividir a seqüência de entrada em 4 subseqüência , conforme as expressões:

$$X_{4k} = \sum_{n=0}^{(N/4)-1} [(x(n) + x(n+\frac{N}{4}) + x(n+\frac{N}{2}) + x(n+\frac{3N}{4})]W_N^0 W_{\frac{N}{4}}^{kn} \tag{2.6}$$

$$X_{4k+1} = \sum_{n=0}^{(N/4)-1} [(x(n) - jx(n+\frac{N}{4}) - x(n+\frac{N}{2}) + jx(n+\frac{3N}{4})]W_N^n W_{\frac{N}{4}}^{kn} \tag{2.7}$$

$$X_{4k+2} = \sum_{n=0}^{(N/4)-1} [(x(n) - x(n+\frac{N}{4}) + x(n+\frac{N}{2}) - x(n+\frac{3N}{4})]W_N^{2n} W_{\frac{N}{4}}^{kn} \tag{2.8}$$

$$X_{4k+3} = \sum_{n=0}^{(N/4)-1} [(x(n) + jx(n+\frac{N}{4}) - x(n+\frac{N}{2}) - jx(n+\frac{3N}{4})]W_N^{3n} W_{\frac{N}{4}}^{kn} \tag{2.9}$$

Assim como Cooley Tukey base 2. O algoritmo FFT Cooley Tukey base 4 obtem a transformada de um conjunto de pontos de cálculos básicos denominados de ”borboleta”, porém com um diferencial de utilizar combinações 4 em 4 pontos. O esquema do algoritmo

é composto por sequencias de ”borboletas”. A figura 2.3 ilustra o cálculo de uma DFT de N = 4, com as entradas $x[1, 1, 1, -1]$, exemplificando a borboleta Cooley Tukey base 4.

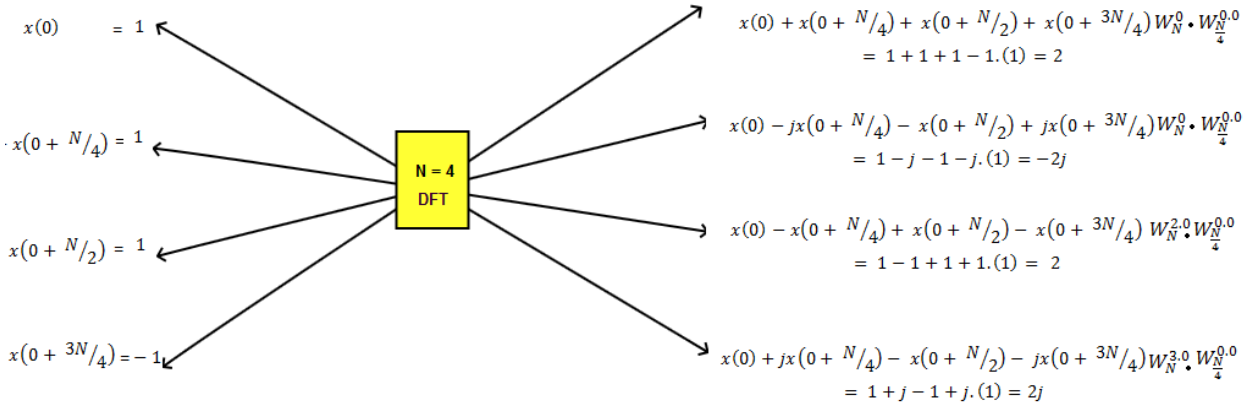

Figura 2.3: Célula básica (borboleta) para a computação do Cooley - Tukey base 4

2.3.3 Algoritmo FFT Split Radix

A SRFFT ***(do inglês, Split Radix Fast Fourier Transform)*** aplica o Cooley-Tukey base 2 nas amostras de índices de números pares de uma DFT de N entradas:

$$X_{2k} = \sum_{n=0}^{(N/2)-1} [x(n) + x(n + \frac{N}{2})] W_{N/2}^{n} \tag{2.10}$$

Com o algoritmo de Cooley-Tukey base 4 calcula - se as amostras de índices de números impares. Conforme a expressão:

$$X_{4k+1} = \sum_{n=0}^{(N/4)-1} [(x(n) - x(n + \frac{N}{2})] - j[x(n + \frac{N}{4}) - x(n + \frac{3N}{4})] W_N^n W_{\frac{N}{4}}^{kn} \tag{2.11}$$

$$X_{4k+3} = \sum_{n=0}^{(N/4)-1} [(x(n) - x(n + \frac{N}{2})] + j[x(n + \frac{N}{4}) - x(n + \frac{3N}{4})] W_N^{3n} W_{\frac{N}{4}}^{kn} \tag{2.12}$$

As expressões mencionadas nesta seção foram retiradas de [Sorrensen1985].

A figura 2.4, é ilustra o cálculo mais básico da SRFFT em que ocorre a aplicação dos Cooley Tukey base 2 e 4, para os índices pares e ímpares, respectivamente.

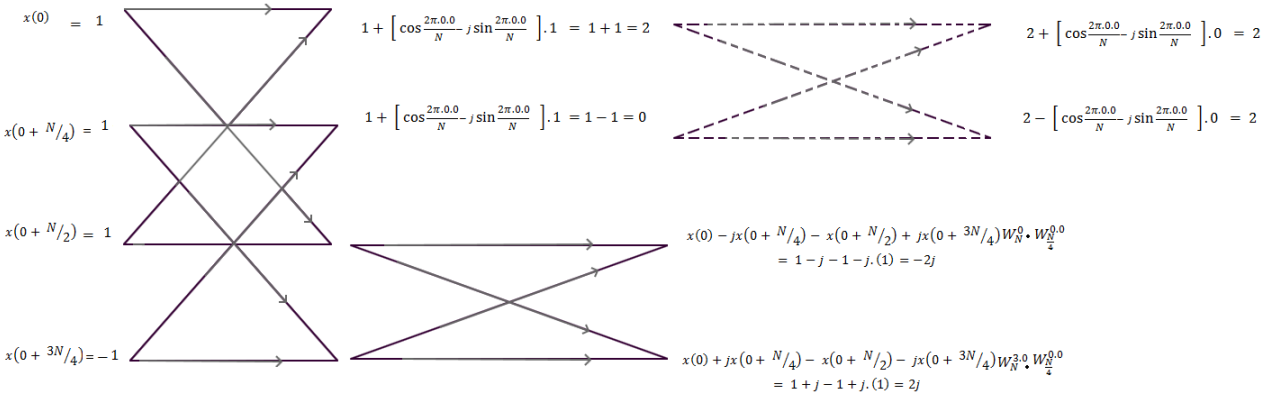

Figura 2.4: Célula básica (borboleta) para a computação Split Radix

2.3.4 Algoritmo FHT Cooley Tukey base 2

Similar ao algoritmo FFT Cooley Tukey base 2 o algoritmo FHT Cooley Tukey 2 decompõe a DHT em DHTs sucessivamente menores, e segunda [Sorrensen1985] o algoritmo é definido matematicamente pela expressão:

$$H_k = H_2n(k) + H_{2n+1}(k)cos(\frac{2\pi k}{N}) + H_{2n+1}(N-k)sen(\frac{2\pi k}{N}) \tag{2.13}$$

2.3.5 Algoritmo FHT Cooley Tukey base 4

O algoritmo FHT Cooley Tukey base 4, respeita a expressão provada no artigo [Sorrensen1985]

$$\begin{aligned} H_k = H_4n(k) &+ H_{4n+1}cos(\frac{2\pi k}{N}) + H_{4n+1}(N-k)sen(\frac{2\pi k}{N}) \\ &+ H_{4n+2}cos(\frac{2\pi 2k}{N}) + H_{4n+2}(N-k)sen(\frac{2\pi 2k}{N}) \\ &+ H_{4n+3}cos(\frac{2\pi 3k}{N}) + H_{4n+3}(N-k)sen(\frac{2\pi 3k}{N}) \end{aligned} \tag{2.14}$$

2.3.6 Algoritmo FHT Split Radix

O algoritmo FHT split radix, respeita a expressão provada no artigo [Sorrensen1985]:

$$H_{2k} = \sum_{n=0}^{\frac{N}{2}-1} [x_n + x_{n+\frac{N}{2}}] cas(\frac{2\pi}{N} 2kn) \tag{2.15}$$

$$H_{4k+1} = \sum_{n=0}^{\frac{N}{4}-1} \left[\left\{ x_n - x_{n+\frac{N}{2}} + x_{\frac{N}{4}-n} - x_{\frac{3N}{4}-n} \right\} cos\left(\frac{2\pi n}{N}\right) + \left\{ x_{n+\frac{3N}{4}} - x_{n+\frac{N}{4}} + x_{\frac{N}{2}-n} - x_{N-n} \right\} sen\left(\frac{2\pi n}{N}\right)) \right] cas\left(\frac{2\pi}{N} 4kn\right) \tag{2.16}$$

$$H_{4k+3} = \sum_{n=0}^{\frac{N}{4}-1} \left[\left\{ x_n - x_{n+\frac{N}{2}} + x_{\frac{3N}{4}-n} - x_{\frac{N}{4}-n} \right\} cos\left(\frac{2\pi 3n}{N}\right) + \left\{ x_{n+\frac{3N}{4}} - x_{n+\frac{N}{4}} - x_{\frac{N}{2}-n} + x_{N-n} \right\} sen\left(\frac{2\pi 3n}{N}\right)) \right] cas\left(\frac{2\pi}{N} 4kn\right) \tag{2.17}$$

Capítulo 3

Implementações dos Algoritmos

Neste capítulo, serão introduzidos os softwares utilizados no trabalho e serão mostradas as implementações os algoritmos rápidos que computam a transformada rápida de Fourier e Hartley, Cooley Tukey base 2, Cooley Tukey base 4 e Split Radix.

3.1 Software Eclipse

Eclipse é um IDE para desenvolvimento Java, que várias linguagens a partir de plugins como C/C++, PHP, ColdFusion, Python, Scala e plataforma Android. Ele foi feito em Java e segue o modelo open source de desenvolvimento de software. O projeto Eclipse foi iniciado na IBM que desenvolveu a primeira versão do produto e doou-o como software livre para a comunidade. O gasto inicial da IBM no produto foi de mais de 40 milhões de dólares. Possui como características marcantes o uso da SWT e não do Swing como biblioteca gráfica, utiliza orientação ao desenvolvimento baseado em plug-ins e o amplo suporte ao desenvolvedor com centenas de plug-ins que procuram atender as diferentes necessidades de diferentes programadores [EclipseJUNO].

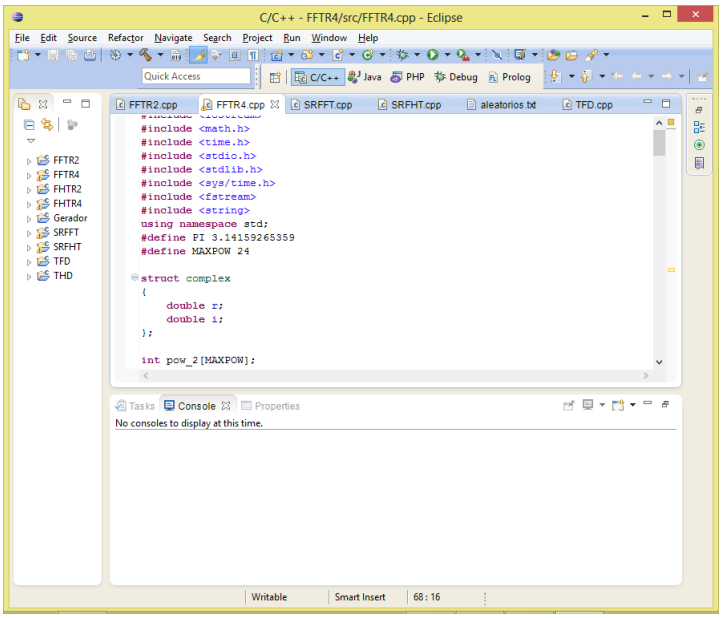

Figura 3.1: Tela principal do eclipse

3.2 Monitor de Desempenho do Windows

O Monitor de Desempenho do Windows é uma ferramenta do Console de Gerenciamento Microsoft (MMC) que fornece ferramentas para analisar o desempenho do sistema. A partir de um console único, pode-se monitorar o desempenho de aplicativo e hardware em tempo real, personalizar que dados deseja -se coletar nos logs, definir limites para alertas e ações automáticas, pode -se gerar relatórios e exibir os dados do desempenho passado de várias maneiras. O Monitor de Desempenho do Windows executa coleta e registro de dados usando Conjuntos de Coletores de Dados, que podem conter contadores desempenho, dados de rastreamento de eventos e informações de configuração do sistema (chaves de registro). Pode-se usar o Monitor de Desempenho para exibir dados de desempenho em tempo real ou de um arquivo de log. As opções de visualização incluem gráficos, histogramas e relatórios. Pode -se também pode usar o Monitor de Desempenho para criar conjuntos de coletores de dados e sessões de Rastreamento de Eventos. [MonitorWindows2013].

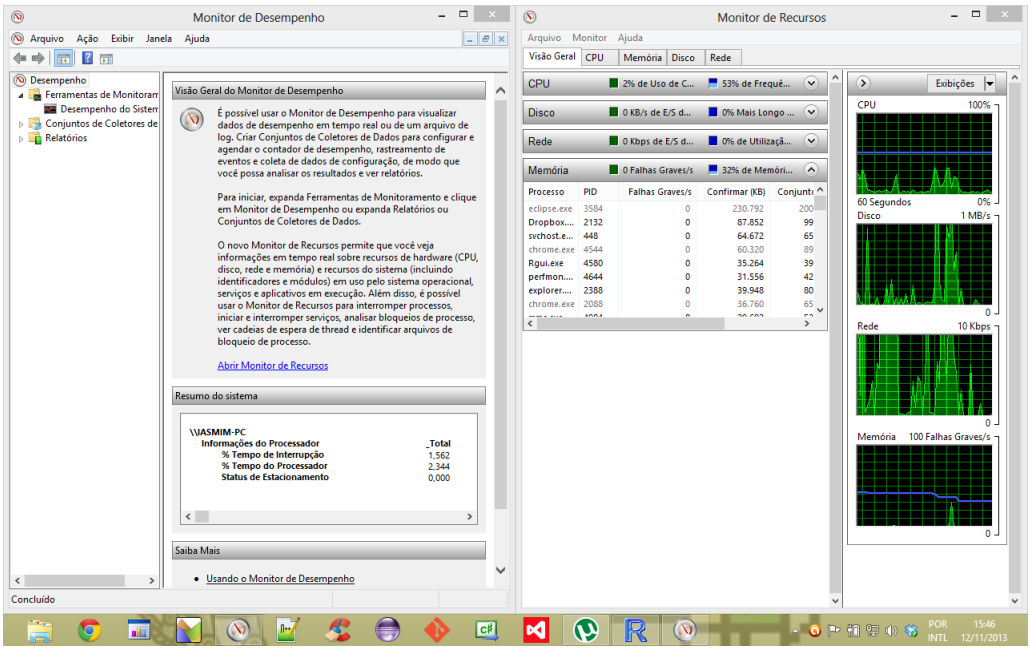

Figura 3.2: Tela principal do monitor de desempenho do windows.

3.3 System Load Indicator

Existem vários indicadores de sistema para o Ubuntu, alguns talvez não sejam muito úteis, mas outros, como o System Load Indicator, pode ser considerado verdadeiramente essencial. A ferramenta é indicada para monitorar a velocidade da rede e uso da CPU, uso de memória, o espaço restante no disco rígido e muito mais. O System Load Indicator é bem semelhante as ferramentas das versões antigas do Ubuntu (até a 10.10), devem se lembrar de uma ferramente que o Gnome 2 possuía para monitorar o sistema, porém engloba mais funcionalidades. Ele também tem algumas opções de configuração e preferências, como a escolha das cores e quais recursos para mostrar ao usuário controlar, diretamente no painel superior do Ubuntu.

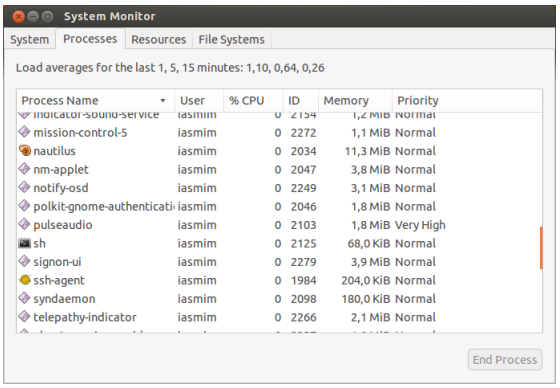

Figura 3.3: Tela principal do System Load Indicator.

3.4 Software MatLab

O Matlab é um software destinado a fazer cálculos com matrizes (Matlab = MATrix LABoratory). MATLAB foi criada no fim dos anos 1970 por Cleve Moler, então presidente do departamento de ciências da computação da Universidade do Novo México. Ela logo se espalhou para outras universidades e encontrou um forte uso no âmbito da comunidade matemática aplicada. Jack Little, um engenheiro, conheceu a linguagem MATLAB, durante uma visita feita por Moler a Universidade de Stanford em 1983. Reconhecendo o seu potencial comercial, ele juntou-se a Moler e Steve Bangert. Eles reescreveram MATLAB em C, em 1984 fundaram a MathWorks e prosseguiram no seu desenvolvimento. As bibliotecas reescritas ficaram conhecidas como LAPACK.

MATLAB foi adotado pela primeira vez por engenheiros de projeto de controle, a especialidade de Little, e rapidamente se espalhou para outros campos de aplicação. Agora, é também utilizado nas áreas da educação, em especial o ensino da álgebra linear e análise numérica, e é muito popular entre os cientistas envolvidos com o processamento de imagem.

MATLAB é um software interativo de alta performance voltado para o cálculo numérico. O MATLAB integra análise numérica, cálculo com matrizes, processamento de sinais e construção de gráficos em ambiente fácil de usar onde problemas e soluções são expressos somente como eles são escritos matematicamente, ao contrário da programação tradicional.

O MATLAB é um sistema interativo cujo elemento básico de informação é uma matriz que não requer dimensionamento. Esse sistema permite a resolução de muitos problemas numéricos em apenas uma fração do tempo que se gastaria para escrever um programa semelhante em linguagem Fortran, Basic ou C. Além disso, as soluções dos problemas são expressas quase exatamente como elas são escritas matematicamente. [Matlab2012]

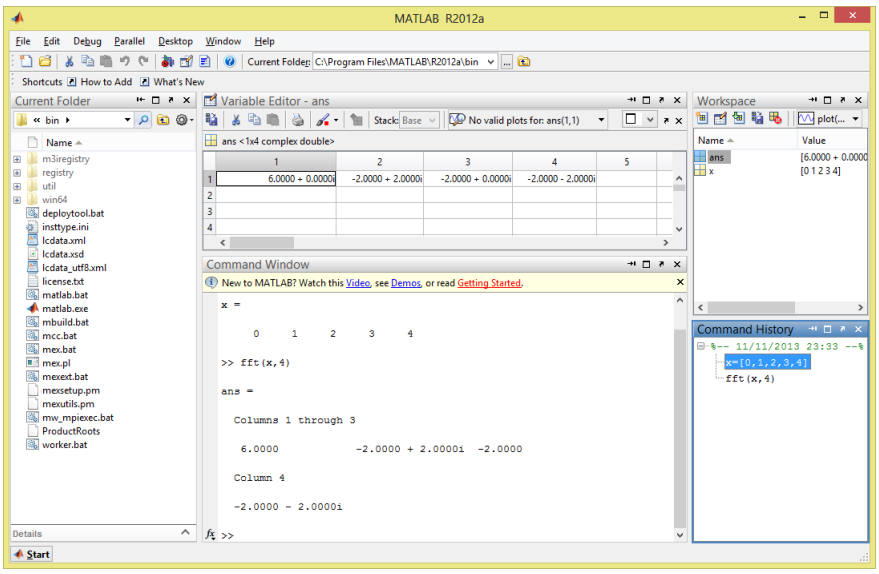

Figura 3.4: Tela principal do MATLAB

3.5 Software R

O R é um software livre para computação estatística e construção de gráficos que pode ser baixado e distribuído gratuitamente de acordo com a licença GNU. O R está disponível para as plataformas UNIX, Windows e MacOS. [rproject2013]

O R tem muitas funções para análises estatísticas e gráficos, este último são visualizados imediatamente em sua própria janela e podem ser salvos em diversos formatos (jpg, png, bmp, ps, pdf, emf, PiCTeX, xfig). Os formatos disponíveis podem depender do sistema operacional). Os resultados de uma análise estatística são mostrados na tela, alguns resultados intermediários (valores de P, os coeficientes de regres-

são, resíduos, e etc) podem ser guardados, escritas num ficheiro, ou utilizados em análises subsequentes facilitando muito os trabalhos dos pesquisadores. [Paradis2005]

O R permite ao usuário, por exemplo, analisar sucessivamente vários conjuntos de dados. Também é possível combinar um único programa diferente de funções estatísticas para realizar análises mais complexas. Uma característica proeminente do R é a sua flexibilidade. Considerando um software clássico que exibe diretamente os resultados de uma análise, R armazena estes resultados em um "objeto", de modo que a análise pode ser feita com qualquer resultado exibido. O usuário pode se surpreender com isso, pois esse recurso é muito útil, cujo, o usuário pode extrair apenas a parte dos resultados que é de seu interesse.

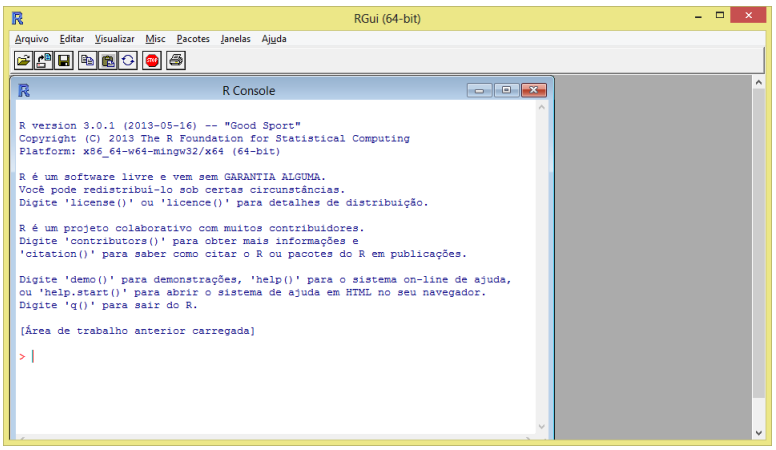

Figura 3.5: Tela principal do software R.

3.6 Análise de Complexidade

Na versão discreta da transformada, tanto para a direta quanto para a inversa, para cada posição do vetor é necessário realizar N multiplicações complexas e $N-1$ somas complexas. Para a computação total da série a ser transformada é realizado um esforço computacional na ordem de $O(N^2)$.

Os algoritmos das Transformadas Rápidas de Fourier e Hartley foram propostos para otimizar a computação da Transformada de Fourier e Hartley, reduzindo suas complexi-

dades de $O(N^2)$ para $O(NlogN)$. Estas reduções são possíveis através da computação eficiente de operações com a eliminação de multiplicações por 1 que são encontradas nesta transformada.A estratégia utilizado na computação da Transformada Rápida de Fourier e Hartley é dividir a transformada em transformadas menores recursivamente para reduzir o esforço computacional.

Neste capítulo será calculado a complexidade dos algoritmos aqui estudados.Como a análise de complexidade não é o foco deste trabalho apenas seram feitas as provas para o algoritmo por definicão de Fourier e o algoritmo de FFT cooley tukey base 2. Uma vez que para os demais algoritmos as códificações são semelhantes tendo assim a mesma complexidade.

Primeiramente será provada a complexidade do algoritmo por definição de Fourier que possui a mesma lógica de *loops* que o algoritmo por definição de Hartley, o que faz com que ambos tenham a mesma complexidade.O pseudo código 3.6.1 mostra apenas o bloco que computa a transformada de fourier que de forma geral é uma multiplicação de uma matriz de N-1 termos com um vetor de N posições.

```
1  para i de 0 ate N-1
2    Xr[i] <- 0;
3    Xi[i] <- 0;
4  para j de 0 ate N-1
5    Xr[i] recebe borboleta real
6    Xi[i] recebe borboleta complexa
```

Código 3.6.1: *Pseudo código DFT*

$$C(n) = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} 1 = \sum_{i=1}^{N}\sum_{j=1}^{N} 1$$
$$= \sum_{i=0}^{N-1} n = n\sum_{i=0}^{N-1} 1 = n.n = n^2$$

Assim conclui-se que a complexidade equivale a $O(N^2)$.

Por conseguinte será provada a complexidade do algoritmo de Cooley Tukey base 2,

que possui a mesma lógica de *loops* que os outros algoritmos rápidos, o que faz com que ambos tenham a mesma complexidade.
Sabe-se que os algorimos rápidos utilizam a estratégia de divisão e conquista. Podemos notar no código 3.6.2 que com a utilização dessa estratégia, são necessários apenas $log_2 n$ passos para se bisseccionar toda a base de dados.

```
1   para m de 0 ate N
2       para j de 0 ate passo
3         faca operas de borboleta
4        passo/2
```

Código 3.6.2: *Pseudo código FFT Cooley Tukey base 2*

Observando o **loop** mais interno segundo [Graham2008], o logaritmo de um número n na base b indica quantas vezes n deve ser dividido por b até alcançar o valor 1. A expressão exata para o número de vezes que $n > 1$ é dada por: $C(n) = log_2 n + 1$

Dessa forma a análise do ANR (algoritmo não recursivo) equivale a:

$$C(n) = \sum_{m=0}^{N} log_2 n = N log_2 N \tag{3.1}$$

Portanto o número de operações necessárias para se calcular cada uma das transformadas não é mais da ordem de $O(N^2)$ e sim da ordem de $O(N(log_2 N))$.

3.7 Implementações dos algoritmos

Os algoritmos foram implementados na linguagem de programação C++ no IDE eclipse. A escolha do IDE eclipse foi baseada no fato deste ser multiplataforma, o que possibilita a análise em sistemas windows e linux. Os algoritmos foram codificados com base nas fórmulas de cada transformadas já apresentadas. As saídas dos algoritmos foram comparadas com transformadas criadas no software MATLAB com os mesmos valores, para confirmar a veracidade dos algoritmos implementados.

3.7.1 Algoritmos baseado na definição DFT

Bloco principal do código baseado na expressão 2.2 da definição da transformada de Fourier que retorna valores reais e imaginários. Tal algoritmo foi implementado para nível de conhecimento pois não será considerado para os testes de análises de desempenho.

```
1       double *Xreal = new double [ N ];
2           double *Ximag = new double [ N ];
3           double Fpi = ( M_PI + M_PI );
4           double pi_N;
5           for ( int n = 0; n < N; n++ )
6           {
7                   Xreal[n] = 0;
8                   Ximag[n] = 0;
9                   pi_N = n * Fpi;
10                  for ( int k = 0; k < N; k++ )
11                  {
12                          Xreal[ n ]  += x[ k ] * cos( (pi_N * k)/N );
13                          Ximag[ n ]  -= x[ k ] * sin( (pi_N * k)/N );
14                  }
15          }
```

Código 3.7.1: *Código principal do algoritmo DFT*

3.7.2 Algoritmos baseado na definição DHT

Bloco principal do código implementado com na base na expressão 2.4. O algoritmo possui uma lógica muito semelhante com o algoritmo do código 3.7.2, porém como estudado no capítulo 2 a transformada de hartley utiliza a função cas(x) = cos(x) + sen(x) em seus cálculos, retornado apenas valores reais. Tal algoritmo foi implementado para nível de conhecimento pois não será considerado para os testes de análises de desempenho.

```
1
2           double *Xreal = new double [ N ];
3           double Fpi = ( M_PI + M_PI );
4           double pi_N;
5
6           for ( int n = 0; n < N; n++ )
7           {
8                   Xreal[n] = 0;
9                   pi_N = n * Fpi;
10                  for ( int k = 0; k < N; k++ )
11                  {
12                          Xreal[ n ]  += x[ k ]
13                                          *( cos( (pi_N * k)/N )
14                                                          + sin((pi_N * k)/N ));
15                  }
16          }
```

Código 3.7.2: *Código principal do algoritmo DHT*

3.7.3 Algoritmos rápidos que DFT

Bloco principal do algoritmo baseado na expressão 2.5 do conceito de divisão e conquista de Cooley Tukey base 2, onde ocorre a multiplicação com o núcleo de transformação. A variável passo é utilizado para o controle dos índices que devem ser multiplicados.

```
1  for (int m = 0; m < k; m++)
2                  {
3                          for (int j = 0; j < passo; j++)
4                          {
5                                  indice = indicebase + j;
6
7                                  a = tmpreal[ indice + passo ];
8                                  b = tmpimg[ indice + passo ];
9                                  c = cos(pi2_N * pot[ indice ]);
10                                 d = -sin(pi2_N * pot[ indice ]);
11                                 real[ indice ] = tmpreal[ indice ] + (a*c - b*d);
12                                 imag[ indice ] = tmpimg[ indice ] + (b*c + a*d);
13
14                                 c = cos(pi2_N * pot[ indice + passo ]);
15                                 d = -sin(pi2_N * pot[ indice + passo ]);
16
17                                 real[ indice + passo ] = tmpreal[ indice ] + (a*c - b*d);
18                                 imag[ indice + passo ] = tmpimg[ indice ] + (b*c + a*d);
19                         }
20                         indicebase += passo + passo;
21                 }
22                 passo = passo * 0.5;
```

Código 3.7.3: *Código principal do algoritmo FFT Cooley Tukey base 2*

Bloco principal que computa a transformada rápida de Cooley Tukey base 4 de acordo com as expressões apresentadas em 2.6, 2.7, 2.8 e 2.9.

```
1           N1=4;
2                   N2=N/4;
3                   for (n2=0; n2<N2; n2++)
4                   {
5
6                           bfly[0].r = (x[n2].r + x[N2 + n2].r + x[2*N2+n2].r + x[3*N2+n2].r);
7                           bfly[0].i = (x[n2].i + x[N2 + n2].i + x[2*N2+n2].i + x[3*N2+n2].i);
8
9                           bfly[1].r = (x[n2].r + x[N2 + n2].i - x[2*N2+n2].r - x[3*N2+n2].i);
10                          bfly[1].i = (x[n2].i - x[N2 + n2].r - x[2*N2+n2].i + x[3*N2+n2].r);
11
12                          bfly[2].r = (x[n2].r - x[N2 + n2].r + x[2*N2+n2].r - x[3*N2+n2].r);
13                          bfly[2].i = (x[n2].i - x[N2 + n2].i + x[2*N2+n2].i - x[3*N2+n2].i);
14
15                          bfly[3].r = (x[n2].r - x[N2 + n2].i - x[2*N2+n2].r + x[3*N2+n2].i);
16                          bfly[3].i = (x[n2].i + x[N2 + n2].r - x[2*N2+n2].i - x[3*N2+n2].r);
17
18
19                          for (k1=0; k1<N1; k1++)
20                          {
21                                  twiddle(&W, N, (double)k1*(double)n2);
22                                  x[n2 + N2*k1].r = bfly[k1].r*W.r - bfly[k1].i*W.i;
23                                  x[n2 + N2*k1].i = bfly[k1].i*W.r + bfly[k1].r*W.i;
24                          }
25                  }
```

Código 3.7.4: *Código principal do algoritmo FFT Cooley Tukey base 4*

Bloco principal que computa a transformada rápida Split Radix de acordo com a expressão apresentada em 2.10.

```
1            do
2             {
3               for(I=IS;I<N;I+=ID)
4                         {
5                                 I1 = I + 1;
6                                 I2 = I1 + N4;
7                                 I3 = I2 + N4;
8                                 I4 = I3 + N4;
9                                 T1 = X[I4] +X[I3];
10                                X[I4] = X[I4] - X[I3];
11                                X[I3] = X[I1] - T1;
12                                X[I1] = X[I1] + T1;
13                                if(N4!=1)
14                                {
15                                        I1 += N8;
16                                        I2 += N8;
17                                        I3 += N8;
18                                        I4 += N8;
19                                        T1 = (X[I3] + X[I4])*.7071067811865475244f;
20                                        T2 = (X[I3] - X[I4])*.7071067811865475244f;
21                                        X[I4] = X[I2] - T1;
22                                        X[I3] = -X[I2] - T1;
23                                        X[I2] = X[I1] - T2;
24                                        X[I1] = X[I1] + T2;
25                                }
26                        }
27                        IS = 2 * ID - N2;
28                        ID = 4 * ID;
29                }while(IS<N);
30
```

Código 3.7.5: *Código principal do algoritmo FFT Split Radix(parte 1)*

Bloco principal que computa a transformada rápida Split Radix de acordo com as fórmulas apresentadas em 2.11 e 2.12.

```
1                   for(J= 2;J<=N8;J++)
2                   {
3                           A3 = 3.0 * A;
4                           CC1   = cos(A);
5                           SS1   = sin(A);
6                           CC3   = cos(A3);
7                           SS3   = sin(A3);
8                           A = (float)J * E;
9                           IS = 0;
10                          ID = 2 * N2;
11                          do
12                          {
13                                  for(I=IS;I<N;I+=ID)
14                                  {
15                                          I1 = I + J;
16                                          I2 = I1 + N4;
17                                          I3 = I2 + N4;
18                                          I4 = I3 + N4;
19                                          I5 = I + N4 - J + 2;
20                                          I6 = I5 + N4;
21                                          I7 = I6 + N4;
22                                          I8 = I7 + N4;
23                                          T1 = X[I3] * CC1 + X[I7] * SS1;
24                                          T2 = X[I7] * CC1 - X[I3] * SS1;
25                                          T3 = X[I4] * CC3 + X[I8] * SS3;
26                                          T4 = X[I8] * CC3 - X[I4] * SS3;
27                                          T5 = T1 + T3;
28                                          T6 = T2 + T4;
29                                          T3 = T1 - T3;
30                                          T4 = T2 - T4;
31                                          T2 = X[I6] + T6;
32                                          X[I3] = T6 - X[I6];
33                                          X[I8] = T2;
34                                          T2    = X[I2] - T3;
35                                          X[I7] = -X[I2] - T3;
36                                          X[I4] = T2;
37                                          T1    = X[I1] + T5;
38                                          X[I6] = X[I1] - T5;
39                                          X[I1] = T1;
40                                          T1    = X[I5] + T4;
41                                          X[I5] = X[I5] - T4;
42                                          X[I2] = T1;
43                                  }
44                                  IS = 2 * ID - N2;
45                                  ID = 4 * ID;
46                          }while(IS<N);
47                  }
```

Código 3.7.6: *Código principal do algoritmo FFT Split Radix(parte 2)*

3.7.4 Algoritmos rápidos que Computam DHT

Bloco principal do código baseado da expressão 2.13 de Cooley Tukey base 2.

```
1  for (int m = 0; m < passo; m++)
2          {
3                  result[ indicebase ] = tmpresult[ indicebase ]+ tmpresult[ indicebase + k ];
4                  result[ indicebase + k ] = tmpresult[ indicebase ]- tmpresult[ indicebase + k ];
5                  for (int j = 1; j < k; j++)
6                  {
7
8                          indice = indicebase + j;
9                          result[ indice ] = tmpresult[ indice ]
10                                          + ( tmpresult[ indice + k ] * cos( pi2_N * j ) )
11                                          + ( tmpresult[ indicebase + k + k - j ] *sin( pi2_N * j ) );
12
13                         result[ indice + k ] = tmpresult[ indice ]
14                                         - ( tmpresult[ indice + k ] *cos( pi2_N * j ) )
15                                         - ( tmpresult[ indicebase + k + k - j ] * sin( pi2_N * j ) );
16                 }
17                 indicebase += k + k;
18         }
19    passo = passo * 0.5;
```

Código 3.7.7: *Código principal do algoritmo FHT Cooley Tukey base 2*

Bloco principal do código baseado da expressão 2.14 de Cooley Tukey base 4 ,utilizando o conceito de hartley.

```
1     for (n2=0; n2<N2; n2++)
2           {
3
4                   bfly[0] =  x[n2]
5                                + x[N2 + n2]
6                                    + x[(2*N2) + n2]
7                                        + x[(3*N2) + n2 ];
8
9
10                  bfly[1] =  x[n2]
11                               + (x[N2 + n2])*cos((2*n2* PI)/N)
12                               + (x[N2 - 1 + n2])*sin(2*n2  * PI/N)
13                               + x[(2*N2) + n2]*cos((4*n2) * PI/N)
14                               + x[(2*N2) - 1 + n2]*sin((4*n2) *PI/N)
15                               + x[(3*N2) + n2 ]*cos((6*n2) * PI / N)
16                               + x[(3*N2) - 1 + n2]*sin((6*n2) * PI/N);
17
18                  bfly[2] =  x[n2]
19                               + (x[N2 + n2])*cos(4*n2 * PI/N)
20                               + (x[N2 - 2 + n2])*sin(4*n2 * PI/N)
21                               + x[(2*N2) + n2]*cos(8*n2  * PI/N)
22                               + x[(2*N2) - 2 + n2]*sin(8*n2 *PI/N)
23                               + x[(3*N2) + n2 ]*cos(12*n2 * PI / N)
24                               + x[(3*N2) - 2 + n2]*sin(12*n2 * PI/N);
25
26                  bfly[3] =  x[n2]
27                               + (x[N2 + n2])*cos(6*n2 * PI/N)
28                               + (x[N2 - 3 + n2])*sin(6*n2 * PI/N)
29                               + x[(2*N2) + n2]*cos(12*n2 * PI/N)
30                               + x[(2*N2) - 3 + n2]*sin(12*n2 *PI/N)
31                               + x[(3*N2) + n2 ]*cos(18*n2 * PI / N)
32                               + x[(3*N2) - 3 + n2]*sin(18*n2 * PI/N);
33
34                  for (k1=0; k1<N1; k1++)
35                  {
36                          x[n2 + N2*k1].r = bfly[k1];
37                  }
38          }
```

Código 3.7.8: *Código principal do algoritmo FHT Cooley Tukey base 4*

Bloco principal baseado na expressão 2.15 que utiliza dos conceitos de Cooley tukey base 2 do algoritmo de Split Radix para transformada de rápida de Hartley.

```
1                   do
2                   {
3                           for(I=IS;I<N;I+=ID)
4                           {
5                                   I1 = I + 1;
6                                   I2 = I1 + N4;
7                                   I3 = I2 + N4;
8                                   I4 = I3 + N4;
9                                   if(I != 1)
10                                  {
11                                          T1 = X[I1] + X[I2];
12                                          T2 = X[I1] - X[I2];
13                                          X[I3] = X[I1] - T1;
14                                          X[I1] = X[I1] + T1;
15                                          X[I4] = X[I2] - T2;
16                                          X[I2] = X[I2] + T2;
17                                  }
18
19                                  T1 = X[I4] +X[I3];
20                                  X[I4] = X[I4] - X[I3];
21                                  X[I3] = X[I1] - T1;
22                                  X[I1] = X[I1] + T1;
23                                  if(N4!=1)
24                                  {
25                                          I1 += N8;
26                                          I2 += N8;
27                                          I3 += N8;
28                                          I4 += N8;
29                                          if(I != (N8 + 1))
30                                          {
31                                                  T1 = X[I3]*.7071067811865475244f;
32                                                  T3 = X[I4]*.7071067811865475244f;
33                                                  X[I4] = X[I2] - T3;
34                                                  X[I2] = X[I2] + T3;
35                                                  X[I3] = X[I1] - T1;
36                                                  X[I1] = X[I1] + T1;
37                                          }
38                                          T1 = (X[I3] - X[I4])*.7071067811865475244f;
39                                          T2 = (X[I3] + X[I4])*.7071067811865475244f;
40                                          X[I4] = X[I2] - T1;
41                                          X[I3] = -X[I2] - T1;
42                                          X[I2] = X[I1] - T2;
43                                          X[I1] = X[I1] + T2;
44                                  }
45                          }
46                          IS = 2 * ID - N2;
47                          ID = 4 * ID;
48                  }while(IS<N);
```

Código 3.7.9: *Código principal do algoritmo FHT Split Radix (parte 1)*

Bloco principal baseado na fórmula 2.16 e 2.17 que utiliza dos conceitos de Cooley Tukey base 4 do algoritmo de Split Radix para transformada de rápida de Hartley.

```
A = E;
                for(J= 2;J<=N8;J++)
                {
                        A3 = 3.0 * A;
                        CC1   = cos(A);
                        SS1   = sin(A);  /*typo A3--really A?*/
                        CC3   = cos(A3); /*typo 3--really A3?*/
                        SS3   = sin(A3);
                        CPS1 = CC1 + SS1;
                        CPS3 = CC3 + SS3;
                        CMS1 = CC1 - SS1;
                        CMS3 = CC3 - SS3;
                        A = (float)J * E;
                        IS = 0;
                        ID = 2 * N2;
                        do
                        {
                                for(I=IS;I<N;I+=ID)
                                {
                                        I1 = I + J;
                                        I2 = I1 + N4;
                                        I3 = I2 + N4;
                                        I4 = I3 + N4;
                                        I5 = I + N4 - J + 2;
                                        I6 = I5 + N4;
                                        I7 = I6 + N4;
                                        I8 = I7 + N4;
                                        if(I != J)
                                        {
                                                T1 = X[I3] * CPS1 + X[I7] * CMS1;
                                                T2 = X[I7] * CMS1 - X[I3] * CPS1;
                                                T3 = X[I4] * CPS3 + X[I8] * CMS3;
                                                T4 = X[I8] * CMS3 - X[I4] * CPS3;
                                                T5 = T1 + T3;
                                                T6 = T2 + T4;
                                                T3 = T1 - T3;
                                                T4 = T2 - T4;
                                                T2 = X[I6] + T6;
                                                X[I3] = T6 - X[I6];
                                                X[I8] = T2;
                                                T2    = X[I2] - T3;
                                                X[I7] = -X[I2] - T3;
                                                X[I4] = T2;
                                                T1    = X[I1] + T5;
                                                X[I6] = X[I1] - T5;
                                                X[I1] = T1;
                                                T1    = X[I5] + T4;
                                                X[I5] = X[I5] - T4;
                                                X[I2] = T1;
                                        }
                                        T1 = X[I3] * CC1 + X[I7] * SS1;
                                        T2 = X[I7] * CC1 - X[I3] * SS1;
                                        T3 = X[I4] * CC3 + X[I8] * SS3;
                                        T4 = X[I8] * CC3 - X[I4] * SS3;
                                        T5 = T1 + T3;
                                        T6 = T2 + T4;
                                        T3 = T1 - T3;
                                        T4 = T2 - T4;
                                        T2 = X[I6] + T6;
                                        X[I3] = T6 - X[I6];
                                        X[I8] = T2;
                                        T2    = X[I2] - T3;
                                        X[I7] = -X[I2] - T3;
                                        X[I4] = T2;
                                        T1    = X[I1] + T5;
                                        X[I6] = X[I1] - T5;
                                        X[I1] = T1;
                                        T1    = X[I5] + T4;
                                        X[I5] = X[I5] - T4;
                                        X[I2] = T1;
                                }
                                IS = 2 * ID - N2;
                                ID = 4 * ID;
                        }while(IS<N);
                }
```

Código 3.7.10: *Código principal do algoritmo FHT Split Radix (parte 2)*

3.8 Processo de extração de dados e teste

Os dados extraídos para o trabalho foram tempo de processamento e memória ocupada dos algoritmos. Os testes foram feitos em sistema operacional windows e linux. O comprimento das amostras de entradas estavam na faixa de 512 a 131072. O tempo de processamento foi extraído em tempo de execução da seguinte forma:

No Windows

- **Tempo:** Incluiu -se a biblioteca <*windows.h*> e usou -se a função *GetTickCount()* que retorna o número de milisegundos desde uma determinada data (não importa qual é essa data). [libraryGetTickCount]

- **Memória:** Para extração dos dados de memória no sistema operacional windows utilizou-se, em tempo de processamento dos algoritmos, o software monitor de desempenho do windows.

No Linux

- **Tempo:**Incluir a biblioteca <*sys/time.h*> e usar a função gettimeofday() que recebe um ponteiro para uma estrutura struct timeval e preenche seus campos com os dados do tempo decorrido desde o último boot. A estrutura struct timeval é uma estrutura que contem dois campos:*tv_sec* e *tv_usec*. O primeiro indica o número de segundos decorridos desde o último boot, e o segundo o número de microssegundos decorridos desde o último segundo. Ao chamar a função *gettimeofday()* você deve passar como primeiro parâmetro um ponteiro para uma struct timeval e como segundo parâmetro o valor NULL. A função então popula os campos *tv_sec* e *tv_usec*. [librarygettimeofday]

- **Memória:** Para extração dos dados de memória no sistema operacional linux utilizou-se, em tempo de processamento dos algoritmos, o software System Load Indicator.

Capítulo 4

Resultados

Este capítulo apresenta os resultados obtidos para os algoritmos apresentados tendo como parâmetros tempo de processamento e memória.

Implementou - se 4 algoritmos utilizando conceitos de transformada de Fourier e 4 algoritmos utlizando conceitos de trasnsformada de Hartley, tendo como dados de entradas números aleatórios reais de tamanho $512 \leq N \geq 131072$.

4.1 Apresentação dos resultados obtidos

Ao se fazer os teste não foi possível extrair os dados de amostras de tamanho menores que $2^9 = 512$, pois os algoritmos eram rápidos de forma que a função em C++ (gettimeofday()) utilizada nos códigos, não conseguiu fornecer um valor diferente de zero . Assim para valores maiores que 512 conseguimos extrair valores. As seguir veremos os dados extraídos dos algoritmos das transformadas de Fourier utilizando a definição, assim como os dados extraídos dos algoritmos rápidos estudados, nos sistemas operacionais Windows e Linux.

4.1.1 Resultados de tempo de processamentos

Os dados das tabelas abaixo foram extraídos com as ferramentas de desempenho mencionadas anteriormente . Cada algoritmo foi testado individualmente nos dois sistemas operacionais:Windows e Linux.

Tabela 4.1: Tabela de amostras de Tempo de processamento em segundos dos algoritmos Transformadas de Fourier em sistema operacional Windows

Amostras	**TFD (s)**	**FFTR2 (s)**	**FFTR4 (s)**	**SRFFT (s)**
512	0,109	0		0
1024	0,484	0,005	0,003	0
2048	1,014	0,01		0,001
4096	7,020	0,015603	0,018002	0,001001
8192	26,926	0,06		0,002
16384	101,183	0,1092	0,059007	0,007001
32768	400,954	0,244802		0,01200
65536	1210,084	0,510008	0,273047	0,028003
131072	2268,908	1,04521		0,07201

Tabela 4.2: Tabela de amostras de Tempo de processamento em segundos dos algoritmos Transformadas de Fourier em sistema operacional Linux

Amostras	**TFD (s)**	**FFTR2 (s)**	**FFTR4 (s)**	**SRFFT (s)**
512	0,018607	0,000752		0,0000064
1024	0,074052	0,00163	0,000432	0,000129
2048	0,300218	0,003345		0,000261
4096	1,17396	0,00742	0,001468	0,000533
8192	4,69973	0,015933		0,001114
16384	18,5571	0,035434	0,007494	0,002428
32768	73,9402	0,075476		0,009957
65536	300,614	0,16179	0,032148	0,010933
131072	1202,05	0,350101		0,025124

Tabela 4.3: Tabela de amostras de Tempo de processamento em segundos dos algoritmos Transformadas de Hartley em sistema operacional Windows

Amostras	**THD (s)**	**FHTR2 (s)**	**FHTR4 (s)**	**SRFHT (s)**
512	0,105	0		0,000005
1024	0,409	0,002	0,002	0,00112
2048	1,002	0,005001		0,001123
4096	6,989	0,015602	0,011001	0,001456
8192	25,987	0,048007		0,00209
16384	101,098	0,10703	0,053008	0,007234
32768	399,007	0,205035		0,013
65536	1209,094	0,431065	0,270034	0,029708
131072	2259,888	0,909114		0,073234

Tabela 4.4: Tabela de amostras de Tempo de processamento em segundos dos algoritmos Transformadas de Hartley em sistema operacional Linux

Amostras	**THD (s)**	**FHTR2 (s)**	**FHTR4 (s)**	**SRFHT (s)**
512	0,018271	0,000483		0,000045
1024	0,073748	0,001138	0,000284	0,000321
2048	0,295751	0,002461		0,0004
4096	1,17157	0,005479	0,001374	0,00098
8192	4,64196	0,012189		0,00136
16384	18,3919	0,025515	0,00651	0,00298
32768	72,9208	0,064102		0,010002
65536	294,532	0,119375	0,031032	0,01169
131072	1179,64	0,258543		0,026267

4.1.2 Resultados de memória ocupada

As tabelas seguintes possuem os valores de memória ocupada no tempo de processamento dos algoritmos estudados que computam a transformada de Fourier utilizando a definição, e os algoritmos rápidos mencionados nos tópicos anteriores.

Tabela 4.5: Tabela de amostras de Memória em KB dos algoritmos Transformadas de Fourier em sistema operacional Windows

Amostras	**TFD (KB)**	**FFTR2 (KB)**	**FFTR4 (KB)**	**SRFFT (KB)**
512	2932	2944		2944
1024	2948	2984	2044	2948
2048	3036	3036		2956
4096	3076	3064	2872	2960
8192	3144	3200		2980
16384	3388	3392	2972	3024
32768	3728	3796		3072
65536	4528	4800	3364	3148
131072	6032	6076		3416

Tabela 4.6: Tabela de amostras de Memória em KB dos algoritmos Transformadas de Fourier em sistema operacional Linux

Amostras	**TFD (KB)**	**FFTR2 (KB)**	**FFTR4 (KB)**	**SRFFT (KB)**
512	136	140		132
1024	148	168	148	132
2048	180	208		140
4096	228	304	196	144
8192	324	452		160
16384	528	588	388	196
32768	912	912		256
65536	1600	1600	1100	388
131072	3100	3100		640

Tabela 4.7: Tabela de amostras de Memória em KB dos algoritmos Transformadas de Hartley em sistema operacional Windows

Amostras	**THD (KB)**	**FHTR2 (KB)**	**FHTR4 (KB)**	**SRFHT (KB)**
512	3184	2044		2950
1024	3084	2952	2892	2955
2048	3232	3020		2957
4096	3088	3072	2956	2963
8192	3244	3180		2986
16384	3340	3380	3156	3030
32768	3716	3832		3080
65536	4236	4736	3912	3150
131072	6112	6584		3430

Tabela 4.8: Tabela de Amostras de Memória em KB dos algoritmos Transformadas de Hartley em sistema operacional Linux

Amostras	**THD (KB)**	**FHTR2 (KB)**	**FHTR4 (KB)**	**SRFHT (KB)**
512	128	136		140
1024	136	152	140	146
2048	160	176		160
4096	192	240	164	220
8192	240	356		260
16384	396	460	260	280
32768	652	652		310
65536	1100	1100	644	390
131072	2100	2100		680

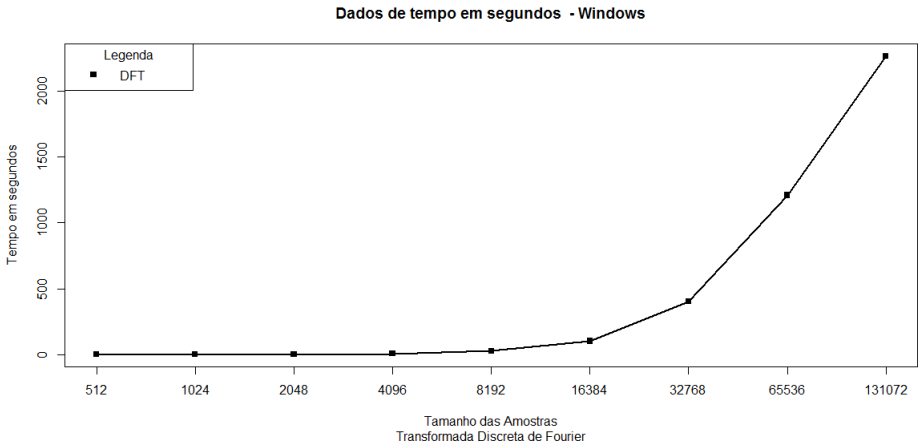

Figura 4.1: Amostras de Tempo de processamento, em segundos, dos algoritmos de Transformada discreta de Fourier em sistema operacional Windows

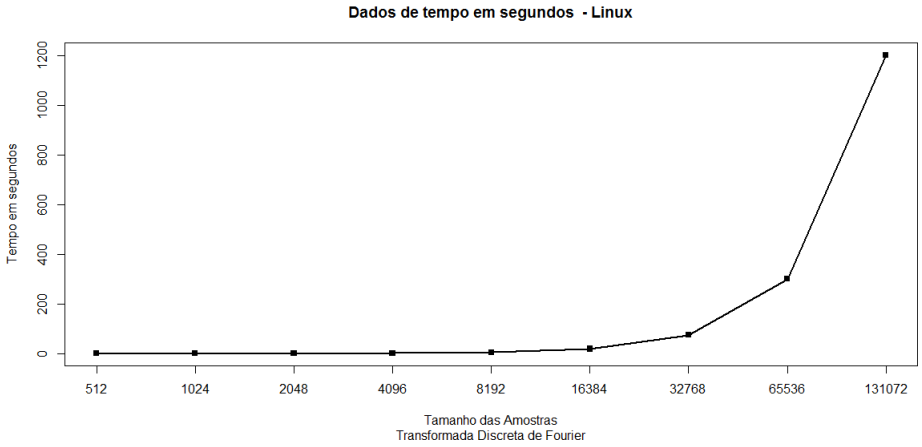

Figura 4.2: Amostras de Tempo de processamento, em segundos, dos algoritmos de Transformada discreta de Fourier em sistema operacional Linux

Nos gráficos das figuras 4.1 e 4.2, pode - se ver o diferencial de desempenho entre os sistemas operacionais estudados, onde os tempos dos algoritmos em sistema linux diminuiram pela para a amostra $N = 131072$ em 800 segundos .

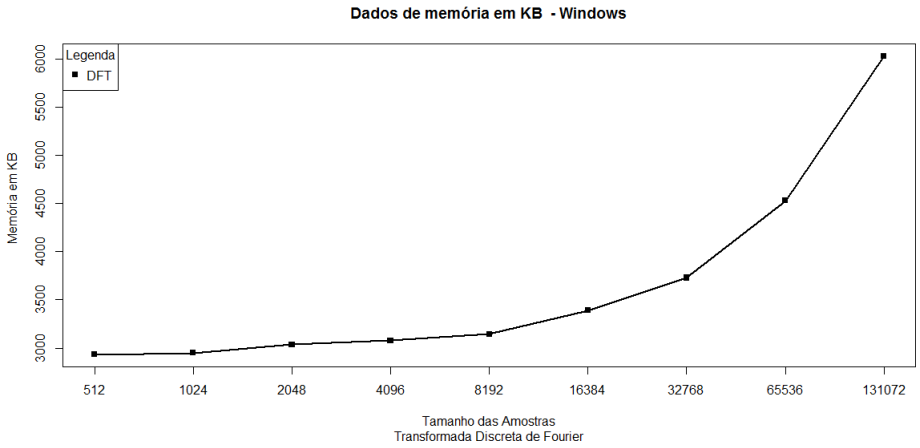

Figura 4.3: Amostras de memória ocupada, em KB, dos algoritmos de Transformada discreta de Fourier em sistema operacional Windows

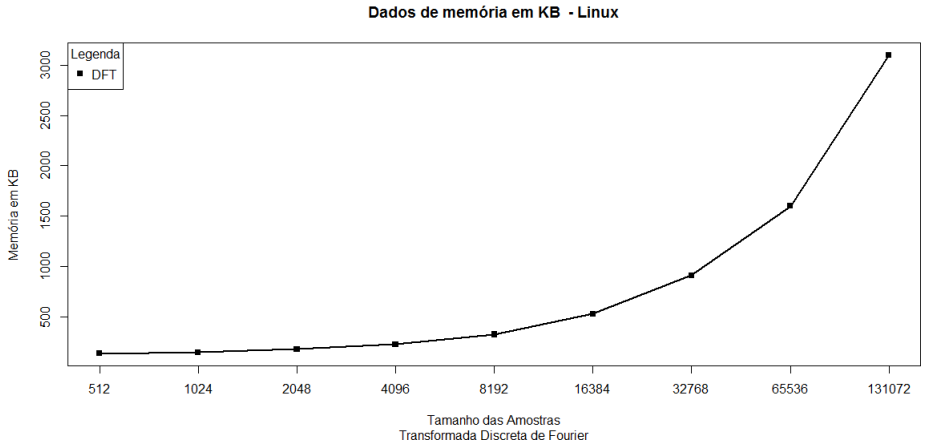

Figura 4.4: Amostras de memória ocupada, em KB, dos algoritmos de Transformada discreta de Fourier em sistema operacional Linux

Nos gráficos das figuras 4.3 e 4.4, pode - se ver o diferencial de desempenho entre os sistemas operacionais estudados, onde os tempos dos algoritmos em sistema linux diminuiram pela metade em relação aos valores dos tempos em sistema windows .

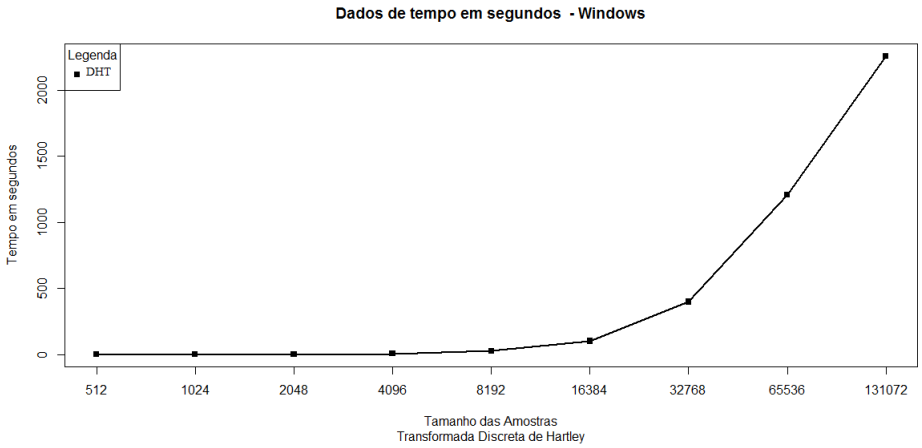

Figura 4.5: Amostras de Tempo de processamento, em segundos, dos algoritmos de Transformada discreta de Hartley em sistema operacional Windows

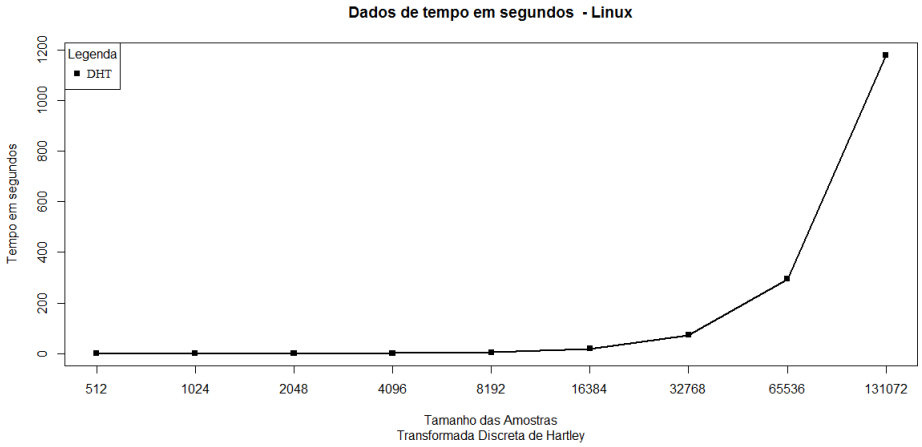

Figura 4.6: Amostras de Tempo de processamento, em segundos, dos algoritmos de Transformada discreta de Hartley em sistema operacional Linux

Nos gráficos dos resultados dos algoritmos que computam transformada de Hartley não ocorreram diferença significativa dos resultados de algoritmos de transformada discreta de Fourier.

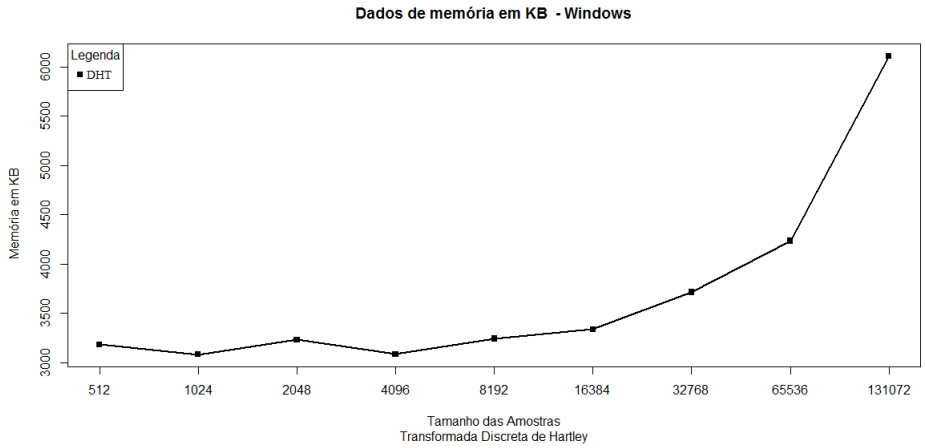

Figura 4.7: Amostras de memória ocupada, em KB, dos algoritmos de Transformada discreta de Hartley em sistema operacional Windows

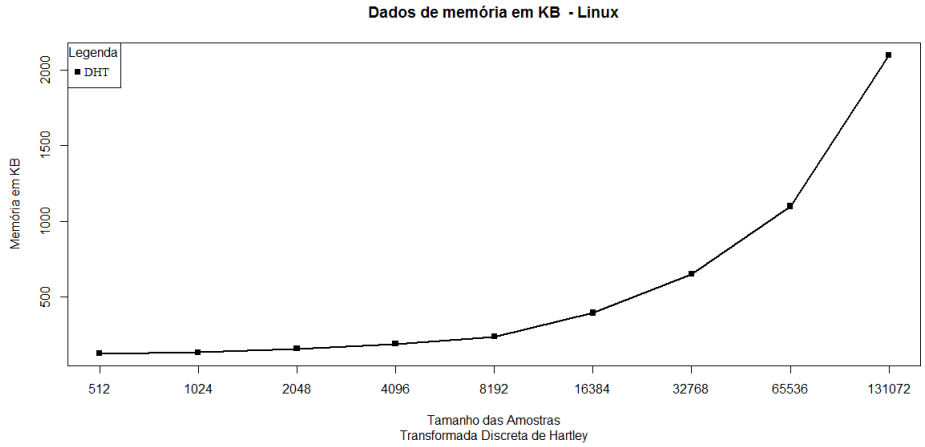

Figura 4.8: Amostras de memória ocupada, em KB, dos algoritmos de Transformada discreta de Hartley em sistema operacional Linux

Visto os gráficos dos valores por definição pode se ter uma noção melhor de quanto os algoritmos rápidos possuem um melhor desempenho em relação aos algoritmos segundo a definição.

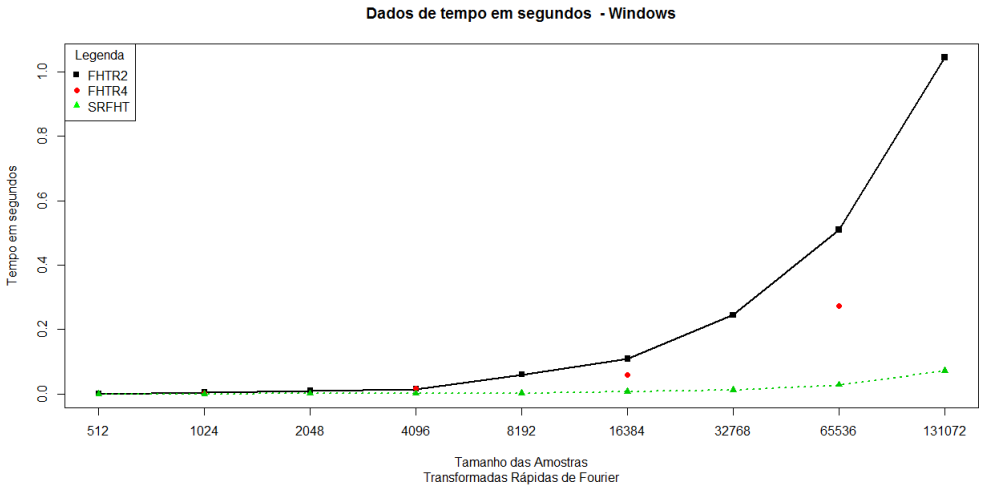

Figura 4.9: Amostras de Tempo de processamento em segundos dos algoritmos rápidos de transformadas de Fourier em sistema operacional Windows

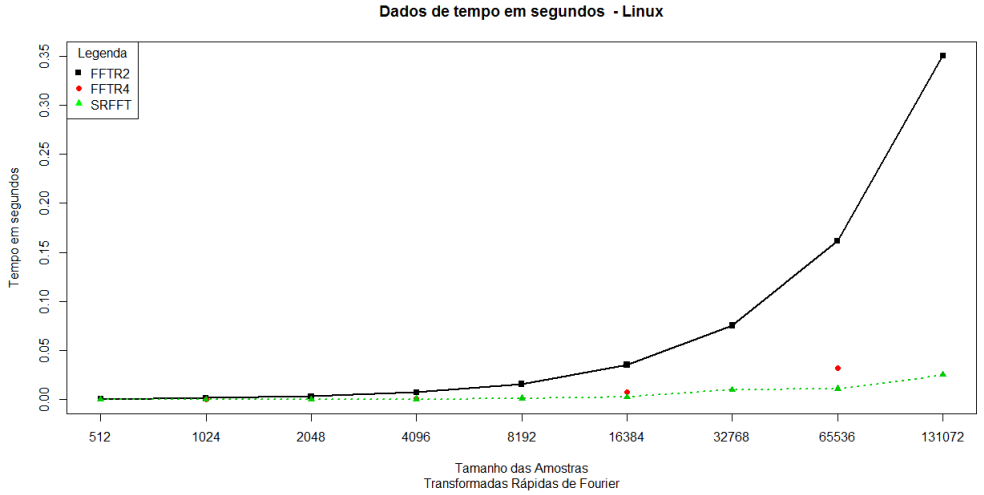

Figura 4.10: Amostras de Tempo de processamento em segundos dos algoritmos rápidos de transformadas de Fourier em sistema operacional Linux

De acordo com os gráficos 4.9 e 4.10, é notável que entre os algoritmos FFTR2 **(do inglês, Fast Fourier Transform Radix 2)**, FFTR4 **(do, inglês, Fast Fourier Transform Radix 4)** e SRFFT **(do inglês, Split Radix Fast Fourier Transform)**. O algoritmo rápido SRFFT mostrou - se com melhor desempenho de tempo em ambos sistemas operacionais. Nota - se também, que as execuções em ambiente Linux obtiveram menores tempos em relação ao windows.

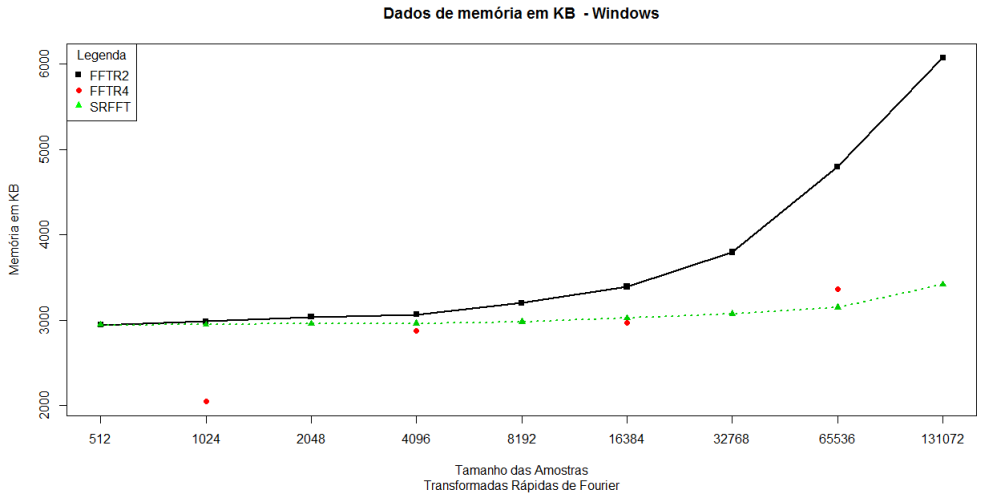

Figura 4.11: Amostras de Memória em KB dos algoritmos rápidos de transformadas de Fourier em sistema operacional Windows

Na figura 4.11, nota - se a semelhança de memória ocupada entre o FFTR4 e SRFFT.

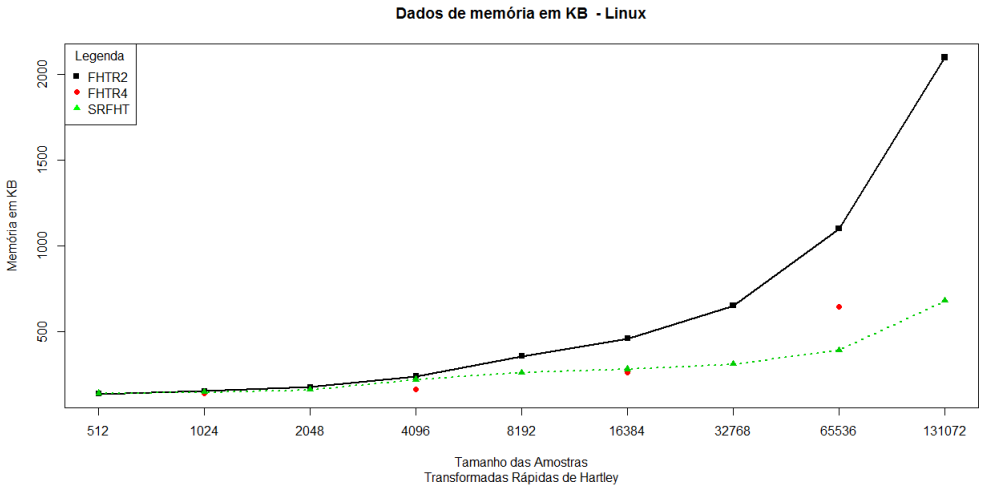

Figura 4.12: Amostras de Memória em KB dos algoritmos rápidos de transformadas de Fourier em sistema operacional Linux

Na figura 4.12, percebe - se que o algoritmo SRFFT comportou - se de forma mais performática, ocupando menos memória do computador no período de processamento. Quanto aos sistemas operacionais o sistema linux superou o sistema windows em relação a ocupação de memória.

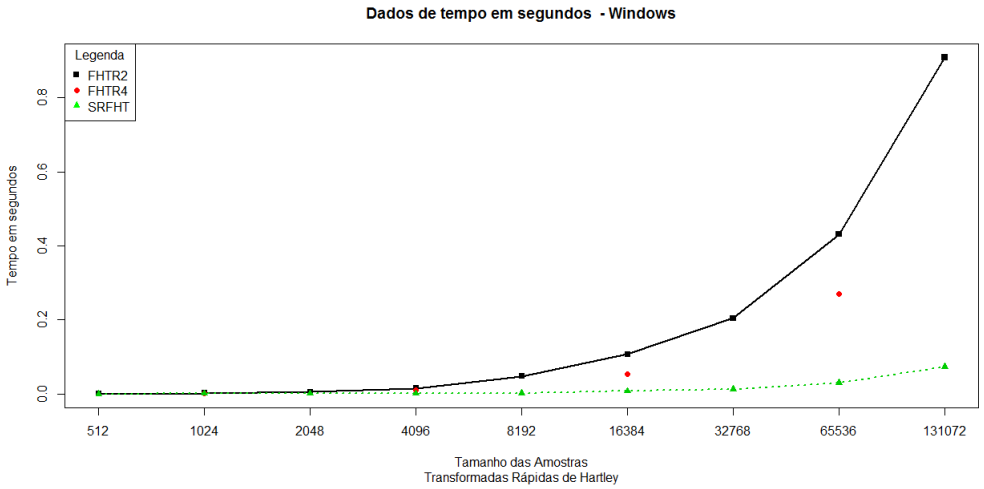

Figura 4.13: Amostras de Tempo de processamento em segundos dos algoritmos rápidos de transformadas de Hartley em sistema operacional Windows

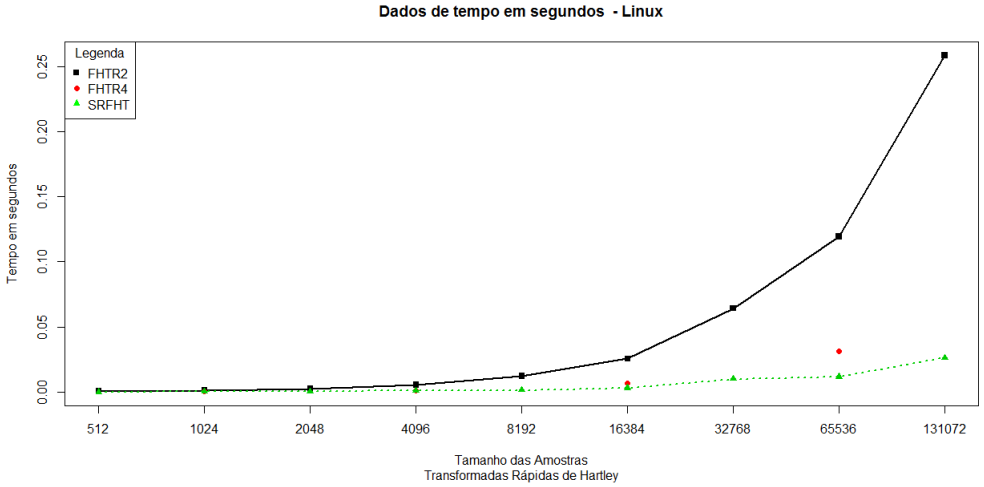

Figura 4.14: Amostras de Tempo de processamento em segundos dos algoritmos rápidos de transformadas de Hartley em sistema operacional Linux

As DHTs executadas utilizando os algoritmos rápidos de Cooley Tukey base 4 e Split Radix, como ilustrados nos gráficos das figuras 4.13 e 4.14, tiveram comportamentos muitos semelhantes, porém as transformadas de Hartley computados com o algoritmo split radix, ainda foram executadas mais rápido que os executados pelos demais. Quanto ao comportamento dos sistemas operacionais, o linux mostrou um melhor desempenho, tendo uma escala de tempo menor que a escala do sistema operacional windows.

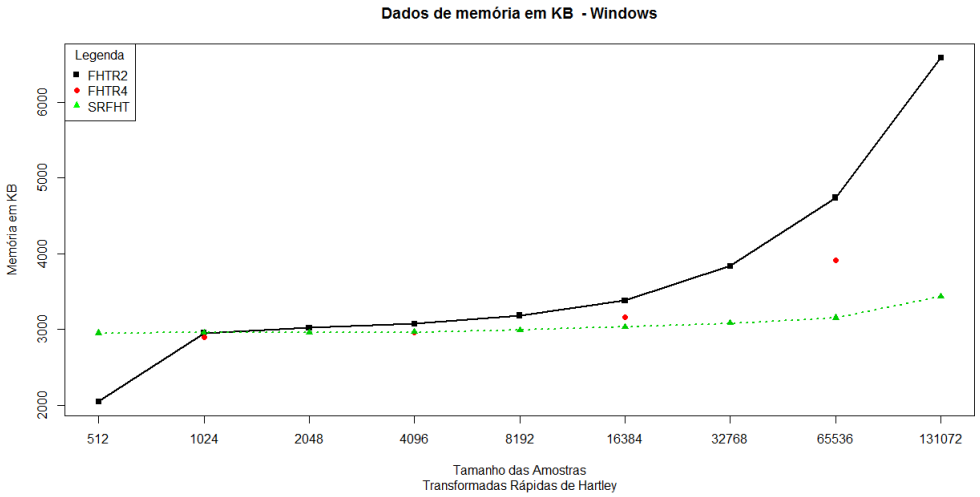

Figura 4.15: Amostras de Memória em KB dos algoritmos rápidos de transformadas de Hartley em sistema operacional Windows

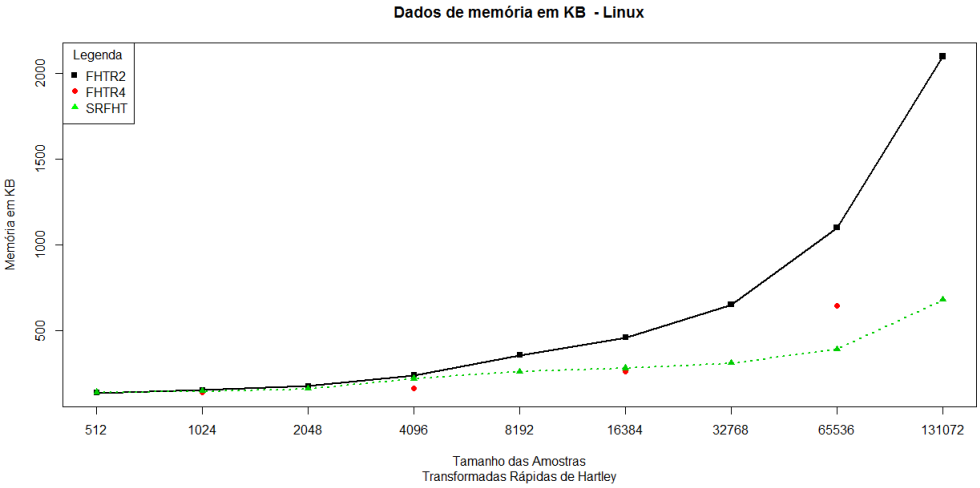

Figura 4.16: Amostras de Memória em KB dos algoritmos rápidos de transformadas de Hartley em sistema operacional Linux

Na computação das DHTs, o sistema operacional linux obteve, mais uma vez vantagem em relação ao sistema operacional windows.Quanto aos algoritmos, o algoritmo rápido split radix, resultou em amostras menores de memória ocupada.

Capítulo 5
Conclusão e Trabalhos Futuros

A proposta deste trabalho foi o projeto e a implementação e análise de algoritmos rápidos que computam transformadas de Fourier e Hartley. Além disso, teve-se por objetivo o estudo dos conceitos dos algoritmos das transformadas de Fourier e Hartley segundo sua definição e os seus algoritmos rápidos, chamados de Cooley tukey base 2, Cooley tukey base 4 e Split Radix e a extração de dados de tempo em segundos e memória ocupadas destes ao serem executados no computador.

Com o desenvolvimento deste trabalho, confirma-se que as transformadas rápidas são realmente mais eficientes que as transformadas de acordo com a definicão.Pois as transformadas como definidas possue complexidade de $O(N^2)$ enquanto as transformadas rápidas possuem complexidade $O(NlogN)$.

Quanto a análise dos dados extraídos nos testes feitos, mesmo que, em alguns momentos, o algoritmo do Split Radix ter um comportamento muito semelhante ao Cooley Tukey base 4, o algoritmo Split Radix possue maior vantagem em relação aos demais, pois mostrou-se com tempos menores, tanto em sistema operacional linux quanto em sistema operacional linux. Quanto nas análises gráficas referentes aos dados de memória, o algoritmo Split Radix também mostrou-se mais eficiente, ocupados tamanhos pequenos da memória em relação aos demais algoritmos estudados. Dessa forma, conclui-se que, o algoritmo split radix, com base nesse estudo, é o mais eficiente. Quanto aos desempenhos dos sistemas operacional é notável nas tabelas e gráficos que o sistema operacional linux possui vantagem em relação ao sistema operacional windows, uma vez que ambos os algoritmos foram mais rápidos e ocuparam menos memória no linux que no Windows.

Como principal contribuição, este trabalho apresenta uma análise comparativa de desempenho computacional utilizando parâmetros de tempo em segundos e memórias em kilo bytes, diferenciando - se de outros trabalhos que fazem a comparação utilizando número de adições e multiplicações dos algoritmos.Assim o trabalho poderá auxiliar pesquisadores de forma mais prática, para aplicações que necessitam de bons desempenhos para cálculos de transformadas.

Através dos resultados, conclui-se que o custo tanto de tempo quanto de memória é menor em sistema operacional linux e que o melhor algoritmo dos três estudos é o algoritmo Split Radix.

Alguns trabalhos futuros são propostos com o objetivo de melhorar o desempenho do projeto e da implementação desenvolvida neste trabalho. A lista a seguir apresenta algumas técnicas que poderão ser incorporadas em uma nova versão.

- Desenvolvimento de algoritmos de transformada rápidas de Fourier e Hartley recursivos para análise comparativas entre algoritmos não-recursivos e recursivos;
- Desenvolver um algoritmo rápido proposto em [Oliveira2013];
- Fazer analise comparativa de desempenho dos três estudados e o proposto em [Oliveira2013];
- Desenvolver sistemas embarcados que utilizem o algoritmo rápido proposto em [Oliveira2013];

Referências Bibliográficas

[Fourier1822] Fourier, J.B. J. Thèorie analytique de la chaleur. France: Acadèmie des Sciences.

[Cooley1967] Cooley, J. W. ,Lewis, Peter A. W. , and Welch, Peter D. ,Historical Notes on the Fast Fourier Transform,IEEE Trans.Audio Electroecust, vol AU-15 pp. 76-79, 1967.

[Oppnheim2013] Oppnheim, A. V. e Schafer, R. W. ,Processamento Em Tempo Discreto De Sinais,3 Edição, 2013

[Bracewell1984] Bracewell, R. N. Fellow,The Fast Hartley Transform ,IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. 72, No.8, 1984.

[Bracewell1983] Bracewell, R. N. Discrete Hartley Transform , J Opt SOc vol. 73, No. 12

[Cooley and Tukey1965] Cooley, J. W. and Tukey J. W. An algorithm for the machine calculation of complex Fourier series, of Computation, vol. 19, no. 90, pp. 297-301.

[Jones2006] Jones, D. L. radix Split FFT Algorithms.

[Johnson and Frigo2007] Johnson, S. G. and Frigo, M. A modified Split-Radix FFT with Fewer Arithmetic Operation, IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. 55.

[Sorrensen1985] Sorrensen, H. V. , Burrus, C. S. and Headman , M. T. On Computing the Discrete Hartley Transform, IEEE Transactions on Acoustics, Speech, and Signal Processing ,Vol assp 33, no.4.

[Sorrensen1986] Sorrensen,H. V.,Jonick , C. L. ,Burrus, C. S. and Headman , M. T. On Computing the split-radix FFT, IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol assp 34, no.1.

[Duhamel1986] Duhamel, P. Implementation of Split-Radix FFT Algorithms for Complex, Real, and Real Symmetric Data, IEEE Transactions on Acoustics, Speech, and Signal Processing.

[Leonidas2000] Leonidas Los eclipses de sol y luna Astronomía digital ,AstroRed, Houston,no 8.

[EclipseJUNO] `http://wiki.eclipse.org/Main_Page`. Página visitada em 31 de outubro de 2013.

[Matlab2012] `http://www.mathworks.com/academia/student_version/`. Página visitada em 31 de outubro de 2013.

[libraryGetTickCount] `http://msdn.microsoft.com/en-us/library/windows/desktop/ms724408(v=vs.85).aspx`. Página visitada em 31 de outubro de 2013.

[librarygettimeofday] `http://linux.die.net/man/2/gettimeofday`. Página visitada em 31 de outubro de 2013.

[MonitorWindows2013] `http://go.microsoft.com/fwlink/?LinkId=241605`. Página visitada em 31 de outubro de 2013.

[Ables1968] J. G. Ables, "Fourier transform photography: a new method for x-ray astronomy,"Proceedings of the Astronomical Society of Australia, Vol. 1, p.172, vol. 1, p. 172, 1968.

[Birkfellner2011] B. Birkfellner, Aplied Medical Image. USA: CRC Press, 2011.

[Fulop2011] S. A. Fulop, Speech Spectrum Analysis. Berlin Heidelberg, Germany: Springer, 2011.

[Kondoz2004] A. M. Kondoz, Digital Speech: Coding for Low Bit Rate Communication Systems. New Jersey, USA: John Wiley e Sons, Ltd., 2004.

[Gonzalez2010] R. C. Gonzalez and R. E. Woods, Processamento Digital de Imagens. BR: Pearson,2010.

[Massey1998] J. L. Massey, "The discrete Fourier transform in coding and cryptography," IEEE Information Theory Workshop, February 1998.

[Campello2009] R. M. Campello de Souza, E. S. V. Freire, and H. M. Oliveira, "Fourier codes," Proceedings of the ISCTA09, vol. 1, pp. 370 – 375, August 2009.

[Kitamura2001] I. Kitamura, S. Kanai, and T. Kishinami, "Copyright protection of vector map using digital watermarking method based on discrete fourier transform," in Geoscience and Remote Sensing Symposium, 2001. IGARSS '01. IEEE 2001 International, vol. 3, 2001,pp. 1191 –1193 vol.3.

[Wysocki2005] T. A. Wysocki, B. Honary, and B. J. Wysocki, Signal Processing for Telecommunicationsand Multimedia. Boston, USA: Springer, 2005.

[Paradis2005] Emmanuel Paradis,R for Beginners,Institut des Sciences de l'Evolution, 2005.

[Oliveira2013] Raimundo C. de Oliveira,Novos Algoritmos Rápidos para Computação de Transformadas Discretas,Recife,2013.

[Graham2008] Graham R. L., Knuth D. E., Patashnik O, Matemática Concreta: fundamentos para ciência da computação, Rio de Janeiro: LTC,2008.

[Brighm1967] E. O Brighm ,R. E Morrow ,The Fast Fourier Transform,LTV Eletrosystems,Inc IEEE spectrum , 1967.

[rproject2013] `http://cran.r-project.org/`. Página visitada em 31 de outubro de 2013.

[Furlani2005] Estudo Furlani,J.R.,Comparativo entre as Transformadas Wavelet e Hartley Usando Imagens Médicas de Cabeça e Pescoço ,Universidade Estadual Paulista Júlio de Mesquita Filho ,São José do Rio Preto(SP).

[Massey1998] Massey,J. L. , The Discrete Fourier Transform in Coding and Cryptography, Information Theory Workshop, ITW 98, San Diego, CA, Feb. 1998.

[Toivonen1979] Toivonen,T. , Heikkilä, J. , Video Filtering with Fermat Number Theoretic

[Blahut1979] Blahut,R.E., Transform Techniques for Error-Control Codes, IBM J. Res. Dev, vol. 23, pp. 299-315, May 1979.

[Cintra2005] Cintra,Renato Jose de Sobral,Aproximação Espectral e Construção de Wavelets com Aplicações em Eletrogastrografia,UFPE/ENGENHARIA ELÉTRICA,2005.