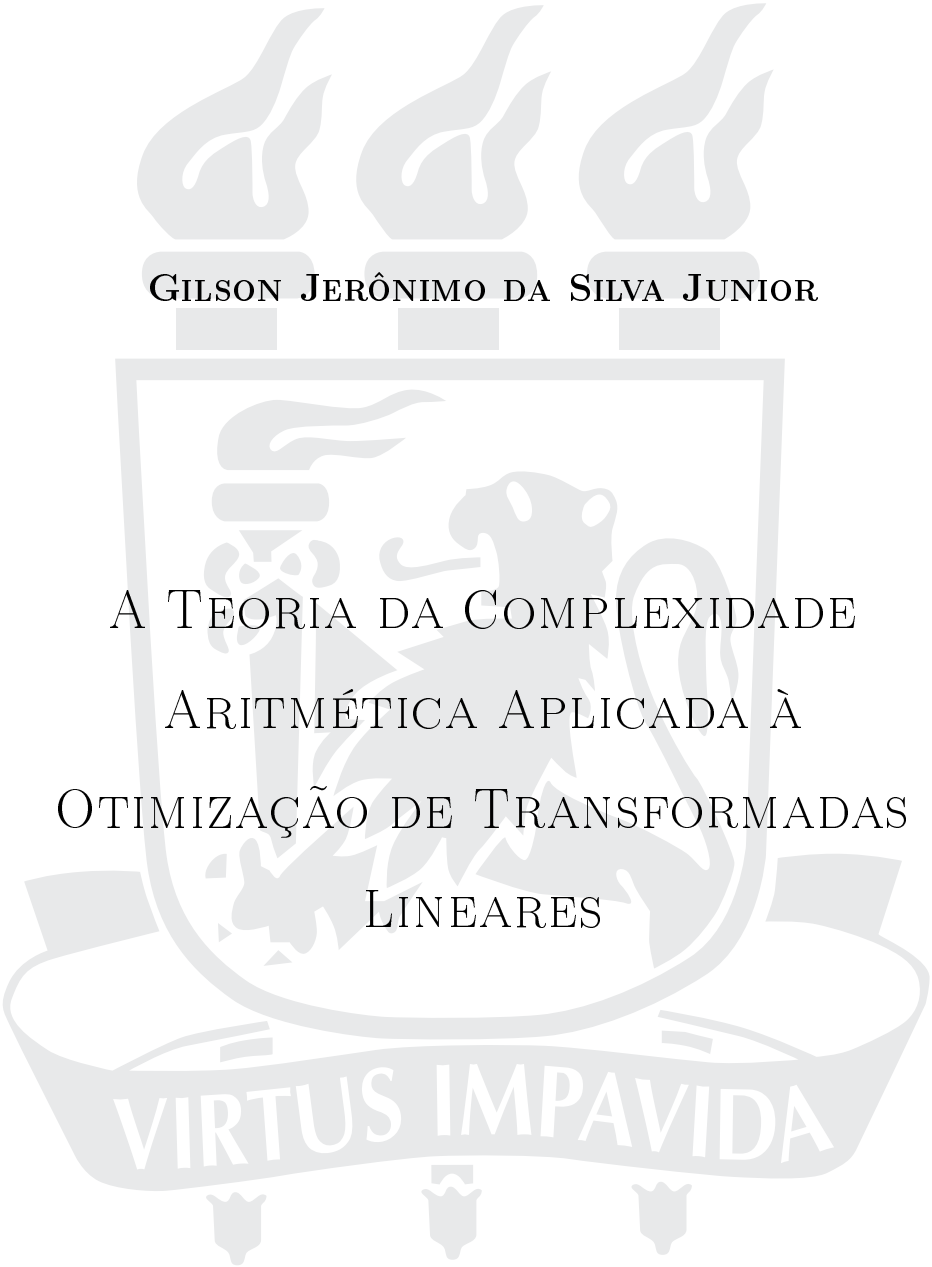
Universidade Federal de Pernambuco
Centro de Tecnologia e Geociências
Programa de Pós-graduação em Engenharia Elétrica



Gilson Jerônimo da Silva Junior

A Teoria da Complexidade Aritmética Aplicada à Otimização de Transformadas Lineares

Recife, Abril de 2012.

Gilson Jerônimo da Silva Junior

A Teoria da Complexidade Aritmética Aplicada à Otimização de Transformadas Lineares

Tese submetida ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Pernambuco como parte dos requisitos para obtenção do grau de **Doutor em Engenharia Elétrica**

Orientador: Prof. Ricardo Menezes Campello de Souza, Ph.D.

Recife, Abril de 2012.

Catalogação na fonte
Bibliotecária: Rosineide Mesquita Gonçalves Luz / CRB4-1361 (BCTG)

S586t Silva Júnior, Gilson Jerônimo da.
A teoria da Complexidade Aritmética aplicada à Otimização de Transformadas Lineares. / Gilson Jerônimo da Silva Júnior – Recife: O Autor, 2012.
132f. il., figs., gráfs., tabs.

Orientador: Prof. Ricardo Menezes Campello de Souza, Ph.D.

Tese (Doutorado) – Universidade Federal de Pernambuco. CTG.
Programa de Pós-Graduação em Engenharia Elétrica, 2012.
Inclui Referências e Apêndice.

1. Engenharia Elátrica. 2. Transformadas Rápidas. *3.* FFT. 4. Complexidade multiplicativa. 4. Complexidade Aditiva. I. Souza, Ricardo Menezes Campello de (Orientador). III. Título.

621.3 CDD (22.ed) UFPE/BCTG-2012 / 179

Universidade Federal de Pernambuco

Centro de Tecnologia e Geociências

Programa de Pós-graduação em Engenharia Elétrica

Gilson Jerônimo da Silva Junior

A Teoria da Complexidade Aritmética Aplicada à Otimização de Transformadas Lineares

'Esta Tese foi julgada adequada para obtenção do Título de Doutor em Engenharia Elétrica, Área de Concentração em Comunicações, e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Pernambuco'.

Prof. Cecilio José Lins Pimentel, Ph.D.
Coordenador do Programa de
Pós-graduação em Engenharia Elétrica

Banca Examinadora:

Prof. Ricardo Menezes Campello de Souza, Ph.D.
Orientador
Universidade Federal de Pernambuco

Profa. Maria das Dores dos Santos Miranda, Doutora
Universidade de São Paulo

Prof. Francisco Madeiro Bernardino Jr., Doutor
Universidade de Pernambuco

Prof. Valdemar Cardoso da Rocha Jr., Ph.D.
Universidade Federal de Pernambuco

Prof. Hélio Magalhães de Oliveira, Docteur
Universidade Federal de Pernambuco

27 de Abril de 2012

Agradecimentos

Aqui expresso meus sinceros agradecimentos às pessoas que contribuíram direta e indiretamente para o desenvolvimento desta tese. Em especial agradeço:

À minha família, todos os Batités e Florêncios, em especial, aos meus pais e meu irmão, pelo apoio e incentivos constantes.

Ao meu orientador, professor Ricardo Campello, por me aceitar e confiar em mim desde o mestrado; pelas contribuições, motivações e correções na redação da tese; por sua disponibilidade e vibração e por ser um excelente professor, na minha opinião, o melhor professor do curso de engenharia eletrônica. Além disso, uma pessoa divertida e agradável.

Ao professor Hélio Magalhães, pelas valorosas contribuições e incentivos. Além disso, por ser um excelente professor.

À minha noiva, Bruna Alves, por me ajudar nas correções dos erros de português desta tese.

A todos os amigos da pós-graduação, que me ajudaram de alguma forma no desenvolvimento desta tese. Em especial ao mestre Caio Barros, e à aluna de mestrado Elda Lizandra.

Gilson Jerônimo da Silva Junior

Universidade Federal de Pernambuco

27 de Abril de 2012

Resumo da Tese apresentada à UFPE como parte dos requisitos necessários para a obtenção do grau de Doutor em Engenharia Elétrica

A Teoria da Complexidade Aritmética Aplicada à Otimização de Transformadas Lineares

Gilson Jerônimo da Silva Junior

Abril/2012

Orientador: Prof. Ricardo Menezes Campello de Souza, Ph.D.
Área de Concentração: Comunicações
Palavras-chaves: Transformadas rápidas, FFT, complexidade multiplicativa, complexidade aditiva
Número de páginas: 132

Encontrar a forma mais eficiente de resolver um problema aritmético e desenvolver algoritmos cada vez melhores é uma grande preocupação dos cientistas, matemáticos e engenheiros projetistas. Economizar operações aritméticas significa diminuir o tamanho do *hardware*, reduzir o consumo de energia e baixar custos de produção. Um algoritmo otimizado minimiza essas três variáveis destacadas. Nesta tese é introduzida a teoria para se obter algoritmos otimizados para qualquer transformada linear. Uma aplicação direta dessa teoria resulta na construção da transformada rápida de Fourier otimizada, a qual atinge o número mínimo possível de multiplicações, sendo mais eficiente do que qualquer algoritmo conhecido na literatura, para computar a transformada discreta de Fourier.

Abstract of Thesis presented to UFPE as a partial fulfillment of the requirements for the degree of Doctor in Electrical Engineering

The Theory of Aritmetical Complexity Applied to the Optimization of Linear Transforms

Gilson Jerônimo da Silva Junior

April/2012

Supervisor: Prof. Ricardo Menezes Campello de Souza, Ph.D.
Area of Concentration: Communications
Keywords: Fast Transforms, FFT, multiplicative complexity, additive complexity
Number of pages: 132

To find the most efficient way to solve an arithmetic problem and to develop efficient algorithms is a major concern of scientists, mathematicians and design engineers. Saving arithmetic operations means reducing the size of the hardware, power consumption and production costs, and an optimized algorithm minimizes these three variables. In this thesis, the theory for constructing optimized algorithms for any linear transform is introduced. A direct application of the theory produced the optimized Fast Fourier transform, which achieves the minimum number of multiplications and, therefore, is the most efficient algorithm known in the literature, for computing the discrete Fourier transform.

Lista de Figuras

Lista de Tabelas

Sumário

capítulo 1

Introdução

Esta tese trata do problema da otimização de algoritmos aritméticos, com uma atenção especial a transformadas lineares[1]. A transformada discreta de Fourier (DFT, do inglês *discrete Fourier Transform*) é uma das transformadas mais populares da engenharia [1, 2], e, neste trabalho, dá-se uma atenção especial à mesma. No século XX, com a evolução dos computadores, os conhecimentos nas áreas de complexidade computacional e algoritmos aumentaram, bem como o interesse em diversos assuntos relacionados a essas áreas.

Entre esses assuntos, um deles é particularmente interessante, a classificação de um problema em tipo P ou NP (definições de P e NP são apresentadas adiante neste capítulo). Esse é um dos assuntos preferidos dos teóricos da computação (talvez por um mero incentivo de um milhão de dólares[2]), entretanto, leva a um comportamento bastante peculiar entre os estudiosos em relação a algoritmos do tipo P. Para ser um pouco mais específico, se existe um algoritmo que computa uma DFT de comprimento n com $1000n \log n$ passos e alguém desenvolve um algoritmo melhor, que resolve o problema em $n \log n$, essa descoberta não foi um grande feito (para um teórico da computação), pois, o segundo algoritmo tem a mesma complexidade computacional do primeiro (embora seja 1000 vezes mais rápido). De fato, na teoria isso não é nenhum avanço, entretanto, tente vender esse produto mil vezes mais lento alegando que tem a mesma complexidade dos concorrentes.

Felizmente, as complexidades aritméticas de transformadas são todas do tipo P e, desde

[1] Para simplificar a escrita desta tese, *transformadas lineares* são chamadas simplesmente de *transformadas*.

[2] Prêmio oferecido pelo milionário americano Landon Clay, através do instituto Clay de matemática, aos que resolverem cada um de sete problemas selecionados por matemáticos de reconhecido saber, sendo "P=NP?" um desses problemas.

Winograd [3], muitos algoritmos de transformada rápida de Fourier (implementação de alto desempenho da DFT) competem, multiplicação a multiplicação, para ser considerado o melhor algoritmo [4]. De fato, cada multiplicação pode representar um multiplicador a mais no *hardware*, o que significa um maior custo e maior tamanho, além de um maior consumo de energia.

Para entendermos um pouco mais sobre o assunto, a próxima seção apresenta, de maneira resumida, os principais conceitos e resultados do estudo de algoritmos. Ainda neste capítulo são apresentados os principais conceitos sobre complexidade multiplicativa e sobre transformadas rápidas de Fourier e convolução cíclica, ambos com diversas aplicações nas áreas de processamento digital de sinais.

1.1 Introdução a Algoritmos

Antes de começar um estudo sobre algoritmos, é importante apresentar uma definição.

Definição 1.1 *Algoritmo é um procedimento que consiste em um conjunto finito de regras, as quais especificam uma sequência finita de operações que fornecem a solução de um problema* [5].

Embora esta definição pareça um pouco vaga, ela contém duas palavras fundamentais para o entendimento de um algoritmo, que são *regra* e *sequência de operações*. Ambas representam dois aspectos distintos e importantes na implementação de um algoritmo. A regra está ligada à descrição do algoritmo e tem a ver com a memória de programa, o número de estados da máquina ou o programa fonte. Já a sequência de operações está relacionada com a sequência de procedimentos realizados até o fim da execução do algoritmo, em relação a uma determinada entrada, e está ligada diretamente à complexidade do algoritmo. O exemplo a seguir esclarece bem esses dois aspectos de um algoritmo.

Exemplo 1.1

Considere um algoritmo para calcular o fatorial de um número inteiro positivo n, descrito pela seguinte regra:

Regra: $f = \text{fat}(n)\{\text{se } (n = 1) f = 1; \text{senão } f = n\text{fat}(n-1);\}$

Esta regra define um algoritmo recursivo para a função fatorial escrito em apenas uma linha,

entretanto, a sequência de operações pode ser bastante longa, dependendo do valor de n, precisamente, n comparações e $n-1$ multiplicações. •

Também é possível ter uma regra complicada e uma sequência de execução rápida. De forma geral, esses dois aspectos estão ligados mas existe uma certa independência sobre eles. Minimizar a regra é minimizar o *hardware*, a memória de programa, enquanto minimizar a sequência de operações é diminuir a complexidade, minimizar o tempo de execução e reduzir o número de operações elementares realizadas, diminuindo a energia consumida no processo.

O objetivo da teoria da complexidade de algoritmos é estudar, quantificar e classificar a sequência de operações de um algoritmo, ou simplesmente a complexidade do algoritmo. O objetivo do estudo de algoritmos rápidos é procurar os algoritmos com menor complexidade. Na próxima seção é apresentado um conceito mais formal sobre complexidade computacional e classes de problemas.

1.1.1 Complexidade de Algoritmos

A complexidade de um algoritmo, como foi visto na seção anterior, está ligada à sequência de operações realizadas pela máquina para executar o algoritmo. Nesse caso, um tipo de máquina deve ser definido, a fim de padronizar a medição da complexidade (vale observar que a complexidade varia de acordo com a máquina). A seguir, uma definição clássica na ciência da computação [6].

Definição 1.2 *Uma máquina de Turing (MT) M é uma tupla $(Q, s, q_a, q_r, \Gamma, \Sigma, \delta)$, em que Q denota um conjunto finito de estados. Os estados (diferentes entre si) $s, q_a, q_r \in Q$, são, respectivamente, o estado inicial, o estado de aceitação e o estado de rejeição da máquina M; Γ é o conjunto finito que compõe o alfabeto de entrada, $\Sigma \supset \Gamma$ é um conjunto finito definido como alfabeto fita, e δ:Q x $\Gamma \rightarrow Q$ x Γ é a função de transferência.*

A MT funciona sobre uma fita que contém a entrada. A MT inicia no estado s e aponta para o início da fita. A função de transferência δ indica o que a MT faz, qual o próximo estado e qual a próxima posição da fita em função do estado atual e da posição atual na fita. O processamento termina quando a máquina chega no estado de aceitação ou rejeição (q_a ou q_r).

A MT apresentada na Definição 1.2 é também chamada de máquina de Turing determinística, porque é possível prever o próximo estado da MT a partir do estado atual e da posição

atual na fita.

O tempo de computação é o número de passos executado pela MT até o processamento terminar. O tempo de computação está ligado com a sequência de operação do algoritmo ou simplesmente com a complexidade computacional. A seguir, uma definição mais abstrata, porém, importante para a classificação de problemas.

Definição 1.3 *Uma máquina de Turing não determinística (MTND) é uma máquina que difere da MT por δ ser uma relação.*

Nesta situação, não há uma função de transferência e sim uma relação de transferência, isto é, pode haver vários estados que a MTND possa seguir dependendo do estado atual. A seguir, mais uma definição que ajudará a compreender a classificação de problemas.

Definição 1.4 *Uma MT (ou MTND) é dita de tempo polinomial se existe um polinômio $p(x)$ tal que, para uma entrada v, o tempo de computação é no máximo $p(|v|)$ ($|v|$ denota o comprimento de v ou o número de posições que v ocupa na fita).*

Com essa definição, temos a definição das classes de problema P e NP.

Definição 1.5 *A classe P de problemas (classe de problemas deterministicamente polinomial) é o conjunto dos problemas que admitem uma solução por uma MT de tempo polinomial.*

Definição 1.6 *A classe NP de problemas (classe de problemas não deterministicamente polinomial) é o conjunto dos problemas que admitem uma solução por uma MTND de tempo polinomial.*

Se existe uma MT de tempo polinomial que resolve um problema, então também existe um algoritmo polinomial e uma máquina que pode resolver este problema. O problema é então classificado como P. Se existe uma MTND de tempo polinomial que resolve um problema, então também existe um algoritmo não determinístico de tempo polinomial, que é um algoritmo determinístico de tempo exponencial, que resolve o problema (o tempo de computação é da ordem de $2^{p(|v|)}$ [6]). Neste caso o problema é dito NP. Problemas NP tornam-se intratáveis quando o comprimento da entrada aumenta. A relação entre as classes de problemas P e NP é um famoso problema em aberto da ciência da computação. Sabe-se que $P \subseteq NP$ mas não se sabe se $P = NP$. A seguir, é apresentada a notação assintótica utilizada pela ciência da computação na classificação e quantificação de problemas e algoritmos.

1.1.2 Notações Assintóticas

Notações assintóticas são úteis na classificação de algoritmos e problemas, embora não sejam de muito uso da área de algoritmos rápidos. Nesta seção temos uma entrada v com comprimento n e um algoritmo A que resolve um problema com exatos $f(n)$ passos (apenas hipoteticamente falando; de forma geral, o número de passos varia com v e n). Pode-se utilizar as definições assintóticas a seguir para qualificar e quantificar o algoritmo A.

Definição 1.7 *A função $f(n)$ é $O(g(n))$ (cota assintótica superior a menos de uma constante) se, e somente se, existe $c \in \mathbb{R}^+$ e $n_0 \in \mathbb{N}$ tal que $f(n) \leq c.g(n)$ para todo $n \geq n_0$.*

A seguir, alguns exemplos.

Exemplo 1.2

Se $f(n) = 3n^2 + n + 1$, $f(n)$ é $O(n^2)$ (lê-se "ordem de n^2"), com $c = 5$ e $n_0 = 1$ por exemplo;
$f(n)$ também é $O(n^3)$, mas não é $O(n)$;
Se $h(n) = n^3 + n$, então $h(n)$ é $O(n^3)$ mas não é $O(n^2)$, com $c = 2$ e $n_0 = 1$ por exemplo. •

Definição 1.8 *A função $f(n)$ é $\Omega(g(n))$ (cota assintótica inferior a menos de uma constante) se, e somente se, existe $d \in \mathbb{R}^+$ e $n_0 \in \mathbb{N}$ tal que $g(n) \leq d.f(n)$ para todo $n \geq n_0$.*

Exemplo 1.3

Se $f(n) = 3n^2 + n + 1$, $f(n)$ é $\Omega(n^2)$;
$f(n)$ é também $\Omega(n)$, mas não é $\Omega(n^3)$;
Se $h(n) = n^3 + n$, então $h(n)$ é $\Omega(n^3)$ e $\Omega(n^2)$. •

Definição 1.9 *A função $f(n)$ é $\Theta(g(n))$ (cota assintótica exata a menos de uma constante) se, e somente se, existem $c, d \in \mathbb{R}^+$ e $n_0 \in \mathbb{N}$ tal que $f(n) \leq c.g(n)$ e $g(n) \leq d.f(n)$ para todo $n \geq n_0$.*

Exemplo 1.4

Se $f(n) = 3n^2 + n + 1$, $f(n)$ é $\Theta(n^2)$;
$f(n)$ não é $\Theta(n^3)$;
Se $h(n) = n^3 + n$, então $h(n)$ é $\Theta(n^3)$ mas não é $\Theta(n^2)$. •

Essas três definições são utilizadas para quantificar um algoritmo A. Quando a função $f(n)$ é $\Theta(g(n))$, diz-se que A tem complexidade $\Theta(g(n))$, ou simplesmente A é $\Theta(g(n))$, se A possui tempo de computação $f(n)$. A seguir são apresentados testes para verificar se a função $f(.)$ é assíntota de $g(.)$.

Proposição 1.1 (Teste da Razão O) *Se*

$$L = \lim_{n\to\infty} \frac{f(n)}{g(n)}$$

converge para um real não negativo, então, a função $f(n)$ é $O(g(n))$.

Demonstração: Se L converge a um real não negativo, então, pela definição de limite, existe um par $N \in \mathbb{N}$ e $\epsilon \in \mathbb{R}$, $\epsilon > 0$, tal que

$$|\frac{f(n)}{g(n)} - L| \leq \epsilon$$

para todo $n \geq N$. Pela desigualdade triangular

$$|\frac{f(n)}{g(n)}| - |L| \leq |L - \frac{f(n)}{g(n)}| \leq \epsilon.$$

Utilizando o fato de que $f(n)$ e $g(n)$ representam tempo de computação, portanto funções positivas, e L é não negativo, então

$$\frac{f(n)}{g(n)} - L \leq \epsilon,$$

de modo que

$$f(n) \leq (L + \epsilon)g(n),$$

e com $n_0 = N$ e $c = (L + \epsilon)$, pela Definição 1.7, $f(n)$ é $O(g(n))$. ■

Vale observar que a condição de L convergir é suficiente mas não necessária para $f(n)$ ser $O(g(n))$ e se $f(n)$ é $O(g(n))$ não significa que o limite L vai existir. Um exemplo disso ocorre quando $f(n) = (10 + (-1)^n)n$ e $g(n) = n$. No caso em que $f(n)$ e $g(n)$ são crescentes, a proposição torna-se suficiente e necessária. A seguir são apresentados os demais testes.

Proposição 1.2 (Teste da Razão Ω) *Se*

$$M = \lim_{n\to\infty} \frac{g(n)}{f(n)}$$

converge para um real não negativo, então, a função $f(n)$ é $\Omega(g(n))$.

Proposição 1.3 (Teste da Razão Θ) *Se*

$$L = \lim_{n\to\infty} \frac{f(n)}{g(n)}$$

ou

$$M = \lim_{n\to\infty} \frac{g(n)}{f(n)}$$

converge para um real positivo, então, a função $f(n)$ *é* $\Theta(g(n))$. *Além disso, se* M *ou* L *converge a zero, então* $f(n)$ *não é* $\Theta(g(n))$.

A prova das proposições apresentadas segue o modelo da prova do Teste da Razão O. Um exemplo de aplicação para esse teste é mostrado a seguir.

Exemplo 1.5

Sabendo que a complexidade multiplicativa (multiplicações complexas) de uma FFT de Cooley-Tukey, para uma entrada v de comprimento n (n potência de dois [7]), é de $\frac{n}{2}\log_2(n)$, verifique que a complexidade multiplicativa da FFT é $O(n^2)$ mas não é $\Theta(n^2)$.

solução: basta calcular L utilizando a regra de L'Hôpital,

$$L = \lim_{n\to\infty} \frac{\frac{n}{2}\log_2(n)}{n^2} = \lim_{n\to\infty} \frac{\log_2(n)}{2n} = \lim_{n\to\infty} \frac{\frac{1}{n\ln 2}}{2} = 0.$$

Portanto, a complexidade é $O(n^2)$ mas não é $\Theta(n^2)$, é $\Theta(n\log_2(n))$. •

A próxima seção trata sobre a teoria da complexidade aritmética proposta por Winograd [3]. O postulado da complexidade aritmética apresentado no próximo capítulo foi inspirado nesta teoria.

1.2 Complexidade Computacional Aritmética e Convolução

De forma direta e simplificada, a proposta de Winograd é contabilizar o número de operações aritméticas envolvidas na solução de um problema computacional [3, 8]. Parece simples e intuitivo supor que as operações aritméticas governam a ordem da complexidade de um tal problema. Uma justificativa para isso ocorre quando as operações aritméticas envolvem ponto flutuante ou pontos fixos formados por vários *bits*. Uma outra ocorre quando comparamos a complexidade física (complexidade da implementação do circuito, envolvendo número de portas ou área utilizada para a integração num *chip*) de um *hardware* que implementa uma mudança de estados a outro que implementa uma multiplicação.

Para medir a complexidade aritmética de um algoritmo, basta contar o número de operações aritméticas envolvidas até o fim da computação. As operações são adição, subtração, multiplicação e divisão.

Winograd propõe que computar a complexidade multiplicativa de um algoritmo consiste em contabilizar apenas o número de multiplicações e divisões necessárias. Além disso, a teoria da complexidade multiplicativa não contabiliza nenhuma multiplicação trivial. Segundo Winograd, a multiplicação $3x$ é considerada trivial, pois pode ser resolvida por $x + x + x$. Para formalizar a teoria, define-se o *corpo das constantes* como um conjunto G, no qual toda multiplicação ou divisão por um elemento de G não é contabilizada. Nos trabalhos de Winograd e Heideman, eles adotaram G como o corpo dos números racionais.

A teoria da complexidade multiplicativa se baseia no fato de que o custo (temporal e/ou espacial) das operações de multiplicação e divisão é muito mais alto que o das demais operações.

1.2.1 Definições Básicas e Teoremas

Para uma breve introdução à teoria da complexidade multiplicativa, deve-se primeiro definir o que é uma multiplicação. Para isso é necessário definir alguns conjuntos.

Definição 1.10 *Todo e qualquer conjunto de procedimentos aritméticos aplicados a uma entrada não inicialmente especificada, que é chamada de variável ou indeterminado, que apresenta um resultado como saída, é definido como operação aritmética.*

Por exemplo, supondo x um indeterminado, então $x + 1$ é uma operação, que tem como saída um valor numérico. Mas 1+1 não é uma operação pois, nesse caso, não existem variáveis. $1 + 1$ é simplesmente considerado como uma constante, o mesmo que um único valor igual a 2.

Definição 1.11 *O corpo de computações, denotado por F, é o corpo ao qual pertence qualquer variável de uma operação aritmética.*

Definição 1.12 *O corpo das constantes é um subcorpo de F, denotado por G, no qual qualquer multiplicação ou divisão, envolvendo um elemento de G, é considerada como trivial, isto é, não é contabilizada.*

No estudo de complexidade multiplicativa, F é adotado como o corpo dos números reais (ou complexos) e G é adotado como o corpo dos números racionais (ou complexos com parte real e imaginária racional, isto é, gaussianos). Por exemplo, sendo x uma variável, uma multiplicação da forma $x(3/4)$ é dita como trivial e portanto não contabilizada. Já a multiplicação $x\sqrt{2}$ é contabilizada pois $\sqrt{2} \notin G$.

Para ilustrar esses conceitos, considere o produto de dois números complexos, $(a + jb)$ e $(c + jd)$. A expressão

$$m = (a + jb)(c + jd)$$

caracteriza um problema aritmético, o qual, para ser resolvido, necessita efetuar a operação $(a + jb)(c + jd)$, que é uma multiplicação entre números complexos. Considerando que a, b, c e d são variáveis, esse problema pode ser posto na forma matricial

$$\begin{bmatrix} m_r \\ m_i \end{bmatrix} = \begin{bmatrix} c & -d \\ d & c \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix}. \tag{1.1}$$

Neste caso são necessárias 4 multiplicações reais e 2 adições reais para resolver o problema. Utiliza-se a seguinte notação para complexidade de algoritmos numéricos: $M_c(.)$ e $A_c(.)$ denotam, respectivamente, o número de multiplicações e adições complexas de um determinado algoritmo, bem como $M_r(.)$ e $A_r(.)$ denotam, respectivamente, o número de multiplicações e adições reais [7]. Assim, para a operação matricial (1.1), tem-se

$$M_r(\text{Algoritmo de multiplicação entre números complexos}) = 4,$$

$$A_r(\text{Algoritmo de multiplicação entre números complexos}) = 2.$$

Entretanto, o mesmo problema pode ser resolvido por outra sequência de operações. Considere agora a seguinte operação aritmética para resolver o problema da multiplicação complexa,

$$\begin{bmatrix} m_r \\ m_i \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \left(\begin{bmatrix} (c-d) & 0 & 0 \\ 0 & (c+d) & 0 \\ 0 & 0 & d \end{bmatrix} \left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} \right) \right).$$

Nesse caso tem-se

$$M_r(\text{Algoritmo de multiplicação entre números complexos}) = 3$$

e

$$A_r(\text{Algoritmo de multiplicação entre números complexos}) = 5,$$

o que significa que o mesmo problema pode ser resolvido por um número diferente de multiplicações e adições reais, dependendo da forma em que a operação é posta a ser resolvida. Considerando que c e d são constantes, isto é, existem apenas a e b como indeterminados, o problema passa a ser a multiplicação de um número complexo $(a + jb)$ por uma constante complexa $(c + jd)$. Assim, como $(c + d), (c - d), d \notin G$, a complexidade do problema é

$$M_r(\text{Algoritmo de multiplicação por cte complexa}) = 3$$

e

$$A_r(\text{Algoritmo de multiplicação por cte complexa}) = 3,$$

como $(c+d)$ e $(c-d)$ não necessitam ser computadas pois essas adições não envolvem variáveis.

Muitos problemas aritméticos podem ser postos na forma matricial

$$s = Hd,$$

em que d, $(N \times 1)$, é uma matriz de indeterminados e H, $(M \times N)$, é uma matriz de indeterminados.

Dois teoremas, que são importantes para esse tipo de problema, estabelecem uma cota superior para a complexidade multiplicativa.

Teorema 1.1 (Teorema do posto linha) *O número de multiplicações necessárias de algum algoritmo que computa $s = Hd$ é, pelo menos, tão grande quanto o posto linha de H.*

Teorema 1.2 (Teorema do posto coluna) *O número de multiplicações necessárias de algum algoritmo que computa $s = Hd$ é, pelo menos, tão grande quanto o posto coluna de H.*

As provas desses teoremas podem ser encontradas em [3, 7, 8].

1.2.2 Complexidade de uma Convolução

A convolução é um problema numérico clássico. Ela é utilizada em processamento de sinais, filtragem digital, algoritmos de interpolação, modelagem de canais de telecomunicações e muitos outros problemas [2, 9, 10]. A convolução discreta se caracteriza conforme descrito a seguir. Considerando duas sequência d_n e g_n, a convolução linear entre elas é a sequência s_n dada por

$$s_n \triangleq \sum_{m=-\infty}^{\infty} d_m g_{n-m} \triangleq d_n * g_n. \tag{1.2}$$

A convolução é comutativa, isto é

$$s_n = d_n * g_n = g_n * d_n.$$

Esse problema pode ser restrito à situação em que d_n e g_n tem comprimentos finitos, respectivamente N e M. Isto significa que $d_n = 0, \forall\ n < 0, n > N-1$ e $g_n = 0, \forall\ n < 0, n > M-1$. Assim a computação da convolução linear se torna

$$s_n = \sum_{m=0}^{N-1} d_m g_{n-m} = \sum_{m=0}^{M-1} g_m d_{n-m}.$$

É possível observar que o comprimento de s_n é $M+N-1$, isto é, $s_n = 0$, $\forall\ n < 0$, $n > M+N-2$. Além disso, o problema pode ser visto como uma multiplicação polinomial; definindo-se a notação polinomial

$$d(x) \triangleq \sum_{n=0}^{N-1} d_n x^n,$$

$$g(x) \triangleq \sum_{n=0}^{M-1} g_n x^n$$

e

$$s(x) \triangleq \sum_{n=0}^{M+N-2} s_n x^n,$$

a convolução linear pode ser expressa por

$$s(x) = g(x)d(x).$$

A complexidade multiplicativa de uma convolução linear, pela operação direta de (1.2), é dada por MN.

Observa-se também que uma convolução linear pode ser escrita como um problema matricial, pela expressão

$$s = \begin{bmatrix} g_0 & 0 & \dots & 0 & 0 \\ g_1 & g_0 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & g_{M-1} & g_{M-2} \\ 0 & 0 & \dots & 0 & g_{M-1} \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \\ \vdots \\ d_{N-1} \end{bmatrix}.$$

como g_n e d_n são variáveis, esse problema é da forma $s = Hd$, em que a matriz H contém todas as linhas linearmente independentes e, portanto, decorre do teorema do posto linha que a complexidade multiplicativa para se calcular s é, pelo menos, $M+N-1$.

Em processamento digital de sinais, além da convolução linear, existe a chamada convolução cíclica ou circular, a qual é definida para sequências de mesmo comprimento. A expressão da convolução circular é dada por

$$s_n = \sum_{m=0}^{N-1} d_m g_{((n-m))_N} = g_n \circ d_n, \tag{1.3}$$

em que $((n))_N$ significa $n \pmod N$, sendo N o comprimento das sequências s_n, d_n e g_n. A convolução cíclica pode também ser representada como uma multiplicação polinomial por

$$s(x) = g(x)d(x) \pmod{x^N - 1}.$$

Pela operação definida em (1.3), são necessárias N^2 multiplicações e $N(N-1)$ adições para se computar um convolução cíclica de comprimento N.

Considerando o mesmo argumento utilizado para a convolução linear, mostra-se que esse número de multiplicações é de, pelos menos, N. Entretanto, existe um teorema que estabelece uma cota inferior maior.

Teorema 1.3 *Seja $p(x)$ um polinômio de grau N, produto de t polinômios relativamente primos. Então, todo algoritmo que computa o produto polinomial*

$$s(x) = g(x)d(x) \ (mod\ p(x))$$

necessita de, pelo menos, $2N - t$ multiplicações.

A prova desse teorema é encontrada em [3, 7].

1.2.3 Algoritmo de Convolução de Winograd

O algoritmo de convolução de Winograd, em relação à complexidade multiplicativa, é um algoritmo ótimo. Este algoritmo é baseado no teorema chinês dos restos para polinômios [7, 11]. Considere a multiplicação polinomial modular

$$s(x) = g(x)d(x) \pmod{m(x)},$$

com

$$m(x) = m_0(x)m_1(x)\ldots m_{k-1}(x),$$

em que todos os polinômios $m_i(x)$ tem coeficientes inteiros e $\mathrm{MDC}(m_i(x), m_j(x)) = 1, \forall\, i \neq j$. Definindo

$$s_i(x) = s(x) \pmod{m_i(x)},$$

o polinômio $s(x)$ pode ser escrito como

$$s(x) = \sum_{i=0}^{k-1} a_i(x)s_i(x) \pmod{m(x)},$$

em que $a_i(x)$ tem coeficientes inteiros. As componentes $s_i(x)$ são computadas por

$$s_i(x) = g(x)d(x) \pmod{m_i(x)},$$

$$s_i(x) = (g(x)(\text{mod } m_i(x))d(x)(\text{mod } m_i(x)))(\text{mod } m_i(x)),$$

$$s_i(x) = g_i(x)d_i(x)(\text{mod } m_i(x)).$$

Observa-se que a computação do resto da divisão por $m_i(x)$ não tem complexidade multiplicativa, pois os coeficientes dos polinômios são todos inteiros (contido no corpo dos racionais). Assim, as multiplicações estão todas no produto $g_i(x)d_i(x)$. O algoritmo de Winograd para a convolução pode ser representado de forma matricial por

$$s = C[(Bg) \odot (Ad)],$$

em que o operador $\odot$ representa o produto de vetores termo a termo e as matrizes C, B e A são matrizes de inteiros.

1.3 Algoritmos Rápidos para a DFT

Neste capítulo são apresentados os principais algoritmos *rápidos* para a computação da transformada discreta de Fourier. Transformadas são ferramentas matemáticas utilizadas em muitas aplicações da Engenharia. Particularmente, nenhuma delas é tão popular quanto a transformada de Fourier [1, 9]. O mesmo vale para a sua versão discreta no tempo e frequência, a transformada discreta de Fourier (DFT, do inglês *Discrete Fourier Transform*) [2, 4, 7]. A seguir é apresentada a transformada discreta de Fourier, bem como algumas notações e definições.

1.3.1 A Transformada Discreta de Fourier

As sequências v e V, com elementos v_n, $n = 0, 1, \ldots, N-1$, e V_k, $k = 0, 1, \ldots, N-1$, respectivamente, em que $v_n \in \mathbb{R}$ e $V_k \in \mathbb{C}$, formam um par da transformada discreta de Fourier, denotado por

$$v \overset{\mathcal{F}}{\longleftrightarrow} V,$$

se

$$V_k \triangleq \sum_{n=0}^{N-1} v_n W_N^{kn}, \tag{1.4}$$

e

$$v_n = \frac{1}{N} \sum_{k=0}^{N-1} V_k W_N^{-kn}, \tag{1.5}$$

em que $W_N \triangleq e^{-j\frac{2\pi}{N}}$ e $j \triangleq \sqrt{-1}$. Diz-se que a sequência complexa V é a transformada discreta de Fourier da sequência real v. O inteiro positivo N é dito ser o comprimento da transformada. De forma geral, a sequência v pode ser complexa também, entretanto, para fins de cálculo de complexidade, é considerado que v é sempre uma sequência real.

O problema computacional a ser analisado é o da expressão (1.4). É possível representar esta expressão por uma notação matricial, em que as sequências v e V são consideradas vetores ou matrizes coluna, através de

$$V = \mathbf{W}v, \tag{1.6}$$

em que

$$V \triangleq \begin{bmatrix} V_0 \\ V_1 \\ \vdots \\ V_{N-1} \end{bmatrix},$$

$$v \triangleq \begin{bmatrix} v_0 \\ v_1 \\ \vdots \\ v_{N-1} \end{bmatrix}$$

e

$$\mathbf{W} \triangleq \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & W_N & W_N^2 & \dots & W_N^{N-1} \\ 1 & W_N^2 & W_N^4 & \dots & W_N^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & W_N^{N-1} & W_N^{2(N-1)} & \dots & W_N^{(N-1)(N-1)} \end{bmatrix}.$$

Sendo $\mathbf{W}$ uma matriz $N \times N$, a complexidade multiplicativa da implementação direta, em multiplicações complexas não triviais, representada pela expressão (1.4) ou pela equação matricial (1.6), é dada por

$$M_c(\text{DFT de comprimento } N) \triangleq M_c(N) = (N-1)(N-1).$$

A complexidade aditiva, em adições complexas, é dada por

$$A_c(\text{DFT de comprimento } N) \stackrel{\Delta}{=} A_c(N) = N(N-1),$$

pois são necessárias $N-1$ adições complexas para cada componente da DFT. A notação utilizada para a complexidade segue o modelo da referência [7]. A seguir, são apresentados os principais algoritmos utilizados para implementar a DFT; esses algoritmos são conhecidos como transformadas rápidas de Fourier (FFT)[3] pois reduzem a complexidade multiplicativa da DFT.

1.3.2 A FFT de Cooley-Tukey

O algoritmo de Cooley-Tukey é um dos mais utilizados para a computação da DFT, tanto que ficou conhecido popularmente como FFT. Embora tenha o nome de Cooley-Tukey, esse algoritmo foi descoberto por Gauss, em 1805, que estava aplicando a DFT para interpolar as rotas de asteróides. Mais tarde, Gauss descobriu um método analítico melhor para resolver o problema e não publicou as notas sobre a FFT, que só apareceram numa coletânea de seus trabalhos [4]. Em 1965 o algoritmo foi redescoberto por Cooley e Tukey, e só em 1977 Goldstine percebeu a relação entre o algoritmo utilizado por Gauss e o algoritmo de Cooley e Tukey [4].

Esse algoritmo considera que $N = N_1 N_2$ é um número composto. Assim, considerando a expressão (1.4), pode-se fazer a mudança de variáveis

$$n = n_1 + N_1 n_2,$$

com $n_1 = 0, 1, \ldots, N_1 - 1$, $n_2 = 0, 1, \ldots, N_2 - 1$ e

$$k = k_1 N_2 + k_2,$$

com $k_1 = 0, 1, \ldots, N_1 - 1$ e $k_2 = 0, 1, \ldots, N_2 - 1$. Com isso a expressão (1.4) se torna

$$V_{k_1 N_2 + k_2} = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} v_{n_1+N_1 n_2} W_N^{(k_1 N_2 + k_2)(n_1 + N_1 n_2)},$$

$$V_{k_1 N_2 + k_2} = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} v_{n_1+N_1 n_2} W_N^{k_1 n_1 N_2 + k_2 n_1 + k_1 n_2 N_1 N_2 + k_2 n_2 N_1}.$$

[3] A sigla FFT vêm do inglês *fast Fourier transform*, um termo questionável pois uma transformada com baixa complexidade multiplicativa não implica, necessariamente, menos tempo de processamento.

Considerando que, se $M|N$, então $W_N^M = W_{N/M}$, tem-se

$$V_{k_1N_2+k_2} = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} v_{n_1+N_1n_2} W_{N_1}^{k_1n_1} W_N^{k_2n_1} W_{N_2}^{k_2n_2}$$

ou

$$V_{k_1N_2+k_2} = \sum_{n_1=0}^{N_1-1} W_{N_1}^{k_1n_1} \left(W_N^{k_2n_1} \sum_{n_2=0}^{N_2-1} v_{n_1+N_1n_2} W_{N_2}^{k_2n_2} \right). \tag{1.7}$$

Colocando os vetores v e V em forma bidimensional, isto é

$$v_{n_1+N_1n_2} \triangleq v_{n_1,n_2} \tag{1.8}$$

e

$$V_{k_1N_2+k_2} \triangleq V_{k_1,k_2}$$

e definindo as matrizes

$$v_{N_1N_2} \triangleq [v_{i,j}]_{(N_1 \times N_2)} \tag{1.9}$$

e

$$V_{N_1N_2} \triangleq [V_{i,j}]_{(N_1 \times N_2)},$$

pode-se escrever que

$$V_{k_1,k_2} = \sum_{n_1=0}^{N_1-1} W_{N_1}^{k_1n_1} \left[W_N^{n_1k_2} \left(\sum_{n_2=0}^{N_2-1} v_{n_1,n_2} W_{N_2}^{k_2n_2} \right) \right]. \tag{1.10}$$

Analisando de forma matricial a expressão (1.10), entre os parênteses, temos uma DFT para as linhas de $v_{N_1N_2}$, seguida de multiplicações termo a termo, do resultado, com os valores W_N^{ij} para cada linha i e coluna j. Finalmente, aplica-se uma DFT para cada coluna do resultado anterior. Isto resulta na matriz $V_{N_1N_2}$.

Exemplo 1.6

Supondo $N = 12$, $N_1 = 3$ e $N_2 = 4$. Assim

$$v = [v_0\ v_1\ v_2\ v_3\ v_4\ v_5\ v_6\ v_7\ v_8\ v_9\ v_{10}\ v_{11}]^T,$$

que, na forma bidimensional, definida pelas expressões (1.8) e (1.9), é

$$v_{N_1N_2} = \begin{bmatrix} v_0 & v_3 & v_6 & v_9 \\ v_1 & v_4 & v_7 & v_{10} \\ v_2 & v_5 & v_8 & v_{11} \end{bmatrix}.$$

Aplicando a DFT em todas as linhas de v_{n_1,n_2}, obtém-se a matriz $V^{(1)}$, isto é

$$V^{(1)} = \mathrm{DFT}_{\mathrm{linhas}}(v_{N_1N_2}).$$

A matriz $V^{(1)}$ é multiplicada termo a termo pela matriz $[W_{12}^{ij}]$ para se obter a matriz $V^{(2)}$, isto é,

$$V^{(2)} = V^{(1)} \odot \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & W_{12} & W_{12}^2 & W_{12}^3 \\ 1 & W_{12}^2 & W_{12}^4 & W_{12}^6 \end{bmatrix}.$$

Finalmente, a matriz $V_{N_1N_2}$ é obtida a partir da DFT de todas as colunas de $V^{(2)}$,

$$V_{N_1N_2} = \mathrm{DFT}_{\mathrm{colunas}}(V^{(2)}),$$

ou

$$v_{N_1N_2} = \begin{bmatrix} V_0 & V_1 & V_2 & V_3 \\ V_4 & V_5 & V_6 & V_7 \\ V_8 & V_9 & V_{10} & V_{11} \end{bmatrix},$$

cujos elementos são as componentes da DFT a serem computados. •

Observa-se, portanto, que a complexidade multiplicativa do algoritmo, utilizando a FFT de Cooley-Tukey, pode ser decomposta por

$$M_c(N_1N_2) = N_1M_c(N_2) + N_2M_c(N_1) + (N_1 - 1)(N_2 - 1), \tag{1.11}$$

bem como a complexidade aditiva é dada por

$$A_c(N_1N_2) = N_1M_c(N_2) + N_2M_c(N_1). \tag{1.12}$$

Um caso particular de bastante interesse, denominado FFT de Cooley-Tukey de base 2, ocorre quando N é uma potência de dois, isto é, $N = 2^m$. Nesse caso, a expressão (1.7), fazendo $N_1 = 2$ e $N_2 = N/2$, torna-se

$$V_k = \sum_{n=0}^{N/2-1} v_{2n} W_{N/2}^{kn} + W_N^k \sum_{n=0}^{N/2-1} v_{2n+1} W_{N/2}^{kn}, \tag{1.13}$$

$$V_{N/2+k} = \sum_{n=0}^{N/2-1} v_{2n} W_{N/2}^{kn} - W_N^k \sum_{n=0}^{N/2-1} v_{2n+1} W_{N/2}^{kn}, \tag{1.14}$$

com $k = 0, 1, \ldots, N/2 - 1$. Esta implementação é conhecida como dizimação no tempo. É possível observar que

$$M_c(N) = 2M_c(N/2) + N/2$$

e

$$A_c(N) = 2M_c(N/2) + N.$$

Quando N é potência de dois, as soluções para essas recursões são

$$M_c(N) = \frac{N}{2}\log_2(N)$$

e

$$A_c(N) = N\log_2(N).$$

Neste caso, a complexidade do algoritmo foi reduzida de $\Theta(N^2)$ para $\Theta(N\log_2(N))$. A Figura 1.1 mostra, para o caso $N = 8$, como o algoritmo é implementado.

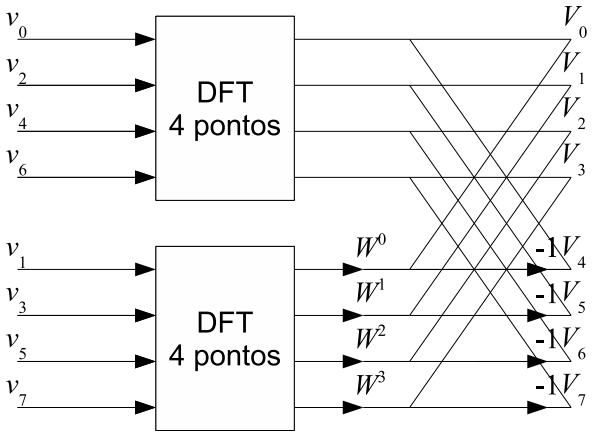


Figura 1.1: Implementação em hardware *do algoritmo Cooley-Tukey, dizimação no tempo, para* $N = 8$.

Existe ainda a implementação do Cooley-Tukey conhecida como dizimação na frequência. Para isso, calculam-se as componentes pares e ímpares da DFT separadamente; assumindo que N é par, pode-se chegar as seguintes expressões [2]

$$V_{2k} = \sum_{n=0}^{N/2-1} (v_n + v_{n+N/2})W_{N/2}^{nk}, \tag{1.15}$$

$$V_{2k+1} = \sum_{n=0}^{N/2-1} W_N^n(v_n - v_{n+N/2})W_{N/2}^{nk}. \tag{1.16}$$

A Figura 1.2 mostra a implementação do Cooley-Tukey, dizimação na frequência, para $N = 8$. A complexidade dessa implementação é a mesma da implementação dizimação no tempo.

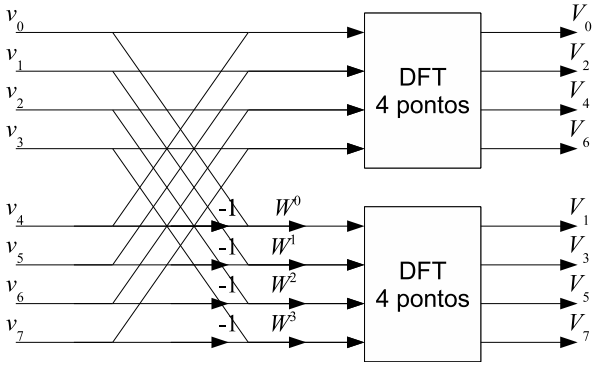


Figura 1.2: Implementação em hardware *do algoritmo Cooley-Tukey, dizimação na frequência, para* $N = 8$.

É possível um melhoramento em termos de complexidade multiplicativa em relação à FFT de Cooley-Tukey, por meio do algoritmo conhecido como FFT de Rader-Brenner. Esta FFT é desenvolvida a seguir utilizando o algoritmo de dizimação na frequência[4]. Para isso, considere as equações (1.15), (1.16) e a sequência a_n, dada por

$$a_n = \begin{cases} 0, & n = 0; \\ (v_n - v_{n+N/2})/[2j\text{sen}(2\pi i/N)], & n = 1, 2, \ldots, N/2 - 1. \end{cases} \tag{1.17}$$

Se o vetor A_k denota a FFT de a_n, isto é,

$$A_k = \sum_{n=0}^{N/2-1} a_n W_{N/2}^{nk}, \tag{1.18}$$

então,

$$A_{k+1} - A_k = \sum_{n=0}^{N/2-1} (a_n W_{N/2}^{n(k+1)} - a_n W_{N/2}^{nk}),$$

$$A_{k+1} - A_k = \sum_{n=1}^{N/2-1} a_n (W_{N/2}^{n} - 1) W_{N/2}^{nk},$$

$$A_{k+1} - A_k = \sum_{n=1}^{N/2-1} a_n (2j\text{sen}(2\pi n/N)) W_N^n W_{N/2}^{nk}$$

e

$$A_{k+1} - A_k = \sum_{n=1}^{N/2-1} W_N^n (v_n - v_{n+N/2}) W_{N/2}^{nk}.$$

Adicionando $(v_0 - v_{N/2})$ em ambos os membros, tem-se

$$A_{k+1} - A_k + (v_0 - v_{N/2}) = \sum_{n=0}^{N/2-1} W_N^n (v_n - v_{n+N/2}) W_{N/2}^{nk},$$

[4] Também pode-se usar o algoritmo sobre a dizimação no tempo [7].

o que leva, por (1.16), a

$$V_{2k+1} = A_{k+1} - A_k + (v_0 - v_{N/2}). \tag{1.19}$$

Nesse caso, para computar as componentes a_n são necessárias $(N/2-1)$ multiplicações reais, pois as constantes $[2j\text{sen}(2\pi n/N)]^{-1}$ são puramente imaginárias. O algoritmo então consiste em computar a_n, computar A_k e obter V_{2k} por (1.15) e V_{2k+1} por (1.19). Assim a complexidade do Rader-Brenner é dada por

$$M_r(N) = 2M_r(N/2) + (N/2-1)$$

e

$$A_r(N) = 2A_r(N/2) + 5N/2.$$

Um exemplo de implementação da FFT de Rader-Brenner, para $N = 8$, é mostrado na Figura 1.3.

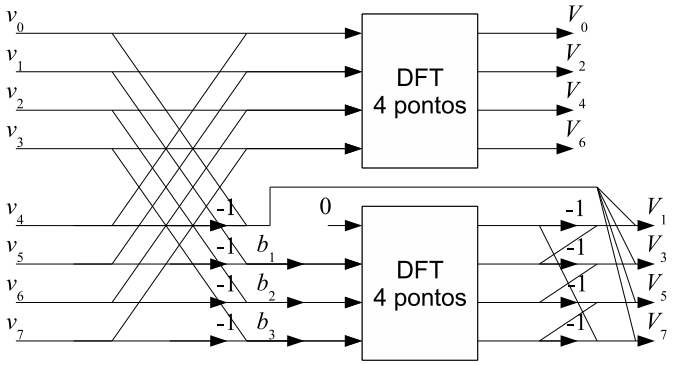


Figura 1.3: Implementação em hardware *do algoritmo Rader-Brenner, dizimação na frequência, para* $N = 8$ *e* $b_n = [2jsen(2\pi n/8)]^{-1}$.

A complexidade da FFT de Cooley-Tukey é dada em número de multiplicações complexas. Isso se deve à sua recursividade, pois um vetor de entrada real produz uma FFT complexa, de forma que as entradas das recursões seguintes são complexas. A expressão da complexidade do algoritmo de Cooley-Tukey ainda conta com algumas multiplicações triviais; para a complexidade exata é necessária a implementação do algoritmo para se verificar quantas multiplicações, de fato, existem (isto é, seguindo a definição de multiplicação de Winograd). A Tabela 1.1 mostra a complexidade da FFT de Cooley-Tukey sem considerar as multiplicações triviais.

Tabela 1.1: Complexidade aritmética da FFT de Cooley-Tukey e Rader-Brenner, assumindo uma entrada complexa, para alguns comprimentos potência de dois.

	Cooley-Tukey Base 2		Rader-Brenner Base 2	
N	Multiplicações reais	Adições reais	Multiplicações reais	Adições reais
8	4	52	4	64
16	24	152	20	192
32	88	408	68	512
64	264	1032	196	1280
128	712	2504	516	3072
256	1800	5896	1284	7168
512	4360	13576	3076	16384
1024	10248	30728	7172	36864
2048	23560	68616	16388	81920
4096	53256	151560	36868	180224

1.3.3 A FFT de Good-Thomas

Esse algoritmo também é conhecido como FFT de fator primo [12] e precede a FFT de Cooley-Tukey; de fato, este algoritmo inspirou o desenvolvimento da FFT de Cooley-Tukey [13]. Nesse caso, $N = N_1 N_2$, em que $\text{MDC}(N_1, N_2) = 1$, isto é, N_1 e N_2 são relativamente primos. Fazendo a mudança de variável

$$n_1 = n (\text{mod } N_1)$$

e

$$n_2 = n (\text{mod } N_2)$$

existe uma solução para n, a partir do teorema chinês dos restos [14], dada por

$$n = n_1 a_1 M_1 + n_2 a_2 M_2 \ (\text{mod } N),$$

em que

$$M_1 = N/N_1 = N_2,$$

$$M_2 = N/N_2 = N_1,$$

$$a_1 N_2 \equiv 1 (\text{mod } N_1)$$

e

$$a_2 N_1 \equiv 1 (\text{mod } N_2).$$

A solução pelo teorema chinês dos restos para essa situação é dada por

$$n = n_1 a_1 N_2 + n_2 a_2 N_1 \pmod{N}.$$

Dessa forma, a equação (1.4) pode ser reescrita por

$$V_k = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} v_{n_1 a_1 N_2 + n_2 a_2 N_1} W_N^{k(n_1 a_1 N_2 + n_2 a_2 N_1)},$$

$$V_k = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} v_{n_1 a_1 N_2 + n_2 a_2 N_1} W_{N_1}^{k n_1 a_1} W_{N_2}^{k n_2 a_2}. \tag{1.20}$$

Agora, pode-se fazer a mudança da variável k da seguinte forma:

$$k_1 = k a_1 \pmod{N_1}$$

e

$$k_2 = k a_2 \pmod{N_2}.$$

Resolvendo o sistema

$$k_1 a_1^{-1} = k \pmod{N_1}$$

e

$$k_2 a_2^{-1} = k \pmod{N_2},$$

o resultado é

$$k = k_1 N_2 + k_2 N_1 \pmod{N},$$

que, substituindo em (1.20), resulta em

$$V_{k_1 N_2 + k_2 N_1} = \sum_{n_1=0}^{N_1-1} \left(\sum_{n_2=0}^{N_2-1} v_{n_1 a_1 N_2 + n_2 a_2 N_1} W_{N_2}^{k_2 n_2} \right) W_{N_1}^{k_1 n_1}. \tag{1.21}$$

Essa expressão representa uma transformada discreta de Fourier bidimensional [7]. Definindo uma nova representação bidimensional (ou matricial),

$$V_{k_1 N_2 + k_2 N_1} \triangleq V_{k_1, k_2}$$

e

$$v_{n_1 a_1 N_2 + n_2 a_2 N_1} \triangleq v_{n_1, n_2},$$

a expressão (1.21) torna-se

$$V_{k_1, k_2} = \sum_{n_1=0}^{N_1-1} \left(\sum_{n_2=0}^{N_2-1} v_{n_1, n_2} W_{N_2}^{k_2 n_2} \right) W_{N_1}^{k_1 n_1}. \tag{1.22}$$

A seguir um exemplo para $N = 12$.

Exemplo 1.7

Supondo $N = 12$, $N_1 = 3$ e $N_2 = 4$, tem-se

$$v = [v_0\ v_1\ v_2\ v_3\ v_4\ v_5\ v_6\ v_7\ v_8\ v_9\ v_{10}\ v_{11}]^T,$$

que, passando para a forma bidimensional do Good-Thomas, leva a

$$[v_{i,j}]_{(N_1 \times N_2)} \triangleq v_{N_1 N_2} = \begin{bmatrix} v_0 & v_9 & v_6 & v_3 \\ v_4 & v_1 & v_{10} & v_7 \\ v_8 & v_5 & v_2 & v_{11} \end{bmatrix}.$$

Então, aplica-se a DFT em todas as linhas de $v_{N_1 N_2}$, e em seguida em todas as colunas do resultado da operação anterior[5], para se obter a matriz $V_{N_1 N_2} \triangleq [V_{i,j}]_{(N_1 \times N_2)}$, isto é

$$V_{N_1 N_2} = \mathrm{DFT}_{\mathrm{colunas}}(\mathrm{DFT}_{\mathrm{linhas}}(v_{N_1 N_2})),$$

ou seja,

$$V_{N_1 N_2} = \begin{bmatrix} V_0 & V_3 & V_6 & V_9 \\ V_4 & V_7 & V_{10} & V_1 \\ V_8 & V_{11} & V_2 & V_5 \end{bmatrix},$$

cujos elementos são as componentes da DFT a ser computada. •

Observa-se, portanto, que a complexidade do algoritmo, utilizando a FFT de Good-Thomas, satisfaz

$$M_c(N_1 N_2) = N_1 M_c(N_2) + N_2 M_c(N_1) \tag{1.23}$$

e

$$A_c(N_1 N_2) = N_1 M_c(N_2) + N_2 M_c(N_1). \tag{1.24}$$

1.3.4 A Permutação de Rader e a FFT de Winograd

A permutação de Rader é um algoritmo que transforma uma DFT, quando N é um número primo, em uma convolução circular de comprimento $N-1$; este algoritmo é também conhecido como Rader primo (*Rader prime*). A grande vantagem da permutação de Rader é que ela pode ser combinada com o algoritmo de Winograd para convolução circular, atingindo baixas

[5] O procedimento de computar a DFT em todas as linhas, seguido da DFT em todas as colunas é conhecido como a DFT bidimensional [7].

complexidades multiplicativas. Essa combinação é conhecida como a FFT de Winograd[6]. A permutação de Rader também pode ser utilizada para potências de números primos.

Considerando a expressão (1.4) quando $N = p$, tem-se

$$V_0 = \sum_{n=0}^{p-1} v_n$$

e

$$V_k = v_0 + \sum_{n=1}^{p-1} v_n W_N^{kn},$$

para $k = 1, 2, \ldots, p-1$. Sendo α uma raiz primitiva de $GF(p)$ [7, 14], é possível fazer a mudança de variável

$$n = \alpha^{r(n)}$$

e

$$k = \alpha^{r(k)},$$

para $n, k = 0, 1, \ldots, p-2$. Assim, tem-se

$$V_{\alpha^{r(k)}} = v_0 + \sum_{n=0}^{p-2} v_{\alpha^{r(n)}} W_N^{\alpha^{r(n)+r(k)}}.$$

Agora faz-se uma nova mudança, $l = r(k)$ e $j = p - 1 - r(n)$; com isso,

$$V_{\alpha^l} = v_0 + \sum_{j=0}^{p-2} v_{\alpha^{p-1-j}} W_N^{\alpha^{p-1-j+l}}.$$

Utilizando o pequeno teorema de Fermat [14], $\alpha^{p-1} \equiv 1 \pmod{p}$,

$$V_{\alpha^l} = v_0 + \sum_{j=0}^{p-2} v_{\alpha^{-j}} W_N^{\alpha^{l-j}}. \tag{1.25}$$

Considerando que

$$V_0 = v_0 + \sum_{j=0}^{p-2} v_{\alpha^{-j}},$$

e subtraindo esta expressão de (1.25), tem-se

$$V_{\alpha^l} - V_0 = \sum_{j=0}^{p-2} v_{\alpha^{-j}} (W_N^{\alpha^{l-j}} - 1). \tag{1.26}$$

[6] A FFT de Winograd pode ser referenciada como pequena FFT (*Winograd Small Fast Fourier Transform*) ou grande FFT (*Winograd Large Fast Fourier Transform*). O algoritmo apresentado aqui é a pequena FFT para comprimentos primos. O caso geral para um comprimento composto bem como a grande FFT pode ser encontrado nas referências [3, 7, 11].

Dessa forma, a DFT V_{α^l} pode ser obtida por meio de

$$V_{\alpha^l} = V_0 + (v_{\alpha^{-l}} \circ (W_N^{\alpha^l} - 1)),$$

em que $\circ$ denota uma convolução circular de comprimento $p-1$.

Para implementar a convolução circular, aplica-se o algoritmo de Winograd para pequenos comprimentos. Assim, definindo

$$s_l = ((W_N^{\alpha^l} - 1) \circ v_{\alpha^{-l}}) = (g_l \circ \hat{v}_l),$$

em que $g_l = W_N^{\alpha^l} - 1$ e $\hat{v}_l = v_{\alpha^{-l}}$, então, pelo algoritmo de Winograd, s_l pode ser representado matricialmente por

$$s = C([Bg \cdot Av]),$$

em que A, B e C são matrizes com elementos inteiros. Como g é conhecido, isto pode ser representado por

$$s = CB'Av,$$

em que C e A são matrizes de inteiros e B' é uma matriz diagonal com $M_r(N)$ elementos que não pertencem aos racionais, os quais determinam a complexidade multiplicativa do algoritmo de Winograd. Obviamente, a expressão pode ser incorporada à equação matricial de V, aumentando um pouco mais a dimensão das matrizes; assim, pode-se escrever, de forma geral, que o algoritmo de Winograd fatora a matriz da DFT na forma

$$W = C_N B_N A_N,$$

em que C_N e A_N são matrizes com inteiros e B_N é uma matriz diagonal com $M_r(N)$ elementos que não pertencem aos gaussianos, totalizando $M_r(N)$ multiplicações reais (multiplicações reais pelo fato de que os elementos de B_N são puramente reais ou puramente imaginários). A FFT de Winograd é implementada por

$$V = C_N[B_N(A_N v)].$$

A Tabela 1.2 mostra a complexidade aritmética da FFT de Winograd para alguns comprimentos. Essa FFT também é conhecida como a pequena FFT de Winograd em alusão aos comprimentos curtos utilizados na obtenção do algoritmo. Para comprimentos maiores, utilizam-se combinações de técnicas [7]. A FFT de Winograd foi a melhor FFT, em termos de complexidade multiplicativa, até a publicação da FFT otimizada em 2011 por Jerônimo da Silva e Campello de Souza [15].

Tabela 1.2: Complexidade aritmética da FFT de Winograd para uma entrada real.

Comprimento N	Multiplicações reais	Adições reais
2	0	2
3	2	6
4	0	8
5	5	17
7	8	36
8	2	26
9	10	44
11	20	84
13	20	94
16	10	74
17	35	157
19	38	186

A seguir são apresentados algoritmos eficientes para computar apenas uma componente da DFT.

1.4 Algoritmos para Computar uma Componente da DFT

Em algumas situações é importante calcular apenas uma determinada componente da DFT. Nesse caso, obter todas as componentes através de uma FFT é, na maioria das vezes, um esforço maior do que computar uma única componente pelo método direto (1.4). Para essa situação, existem algoritmos que reduzem a complexidade de implementação e a complexidade multiplicativa. Esses algoritmos, entretanto, não são considerados FFT pois não reduzem a complexidade multiplicativa assintoticamente. A seguir é apresentado o algoritmo de Goertzel. Ainda existem os algoritmos JCO e JCO-Goertzel[7], que são apresentados na Seção 4.6 do Capítulo 4.

1.4.1 O Algoritmo de Goertzel

O algoritmo de Goertzel data de 1958 [16]; em uma única página de texto, Goertzel desenvolveu uma forma eficiente para computar uma única componente da DFT. Mais tarde, esse trabalho foi melhorado e apresentado segundo as teorias de processamento digital de sinais

[7] Os algoritmos JCO e JCO-Goertzel são apresentados mais adiante pelo fato de que são partes dos trabalhos originais desenvolvidos nesta tese.

atuais [2, 17]. Durante muito tempo, o algoritmo de Goertzel foi o único algoritmo eficiente, em termos de implementação, para a computação de uma única componente da DFT. Para apresentar o algoritmo de Goertzel, considere a representação polinomial para uma sequência v_n, denotada por $v(x)$, dada por

$$v(x) \triangleq \sum_{n=0}^{N-1} v_n x^n.$$

Dessa forma, uma componente da DFT, V_k, pode ser computada avaliando-se $v(x)$ para $x = W_N^k$, isto é

$$V_k = v(W_N^k). \tag{1.27}$$

Considere agora o polinômio $p_k(x)$, dado por

$$p_k(x) = (x - W_N^k)(x - W_N^{-k}) = 1 - 2\cos\left(\frac{2\pi k}{N}\right)x + x^2. \tag{1.28}$$

Nesse caso $v(x)$ pode ser escrito, em notação polinomial, na forma

$$v(x) = p_k(x)q(x) + r(x).$$

Os polinômios $q(x)$ e $r(x)$, obtidos por divisão polinomial, são únicos, de forma que $r(x)$ tem grau um. Como $p_k(W_N^k) = 0$, então

$$V_k = v(W_N^k) = r(W_N^k). \tag{1.29}$$

Para avaliar $r(W_N^k)$ é necessário uma multiplicação complexa e uma adição complexa, entretanto, os coeficientes do polinômio $r(x) = r_0 + r_1 x$ devem ser computados. A Figura 1.4 mostra como é possível computar esses coeficientes; essa implementação é conhecida como *Goertzel ordem inversa*.

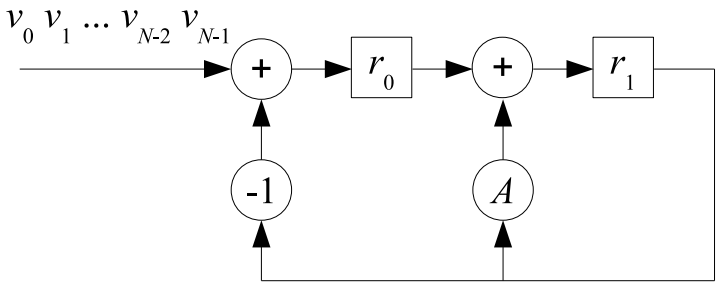


Figura 1.4: Filtro para implementar a divisão polinomial por $p_k(x)$. Essa implementação é conhecida como Goertzel ordem inversa*, pois v_{N-1} é a primeira componente a ser alimentada no filtro, $A = 2\cos(2\pi k/N)$.*

Existem algumas formas de implementar o algoritmo de *Goertzel ordem direta*. A mais conhecida é a que usa o filtro digital $H(z)$, dado por

$$H(z) = \frac{1}{1 - W_N^{-k} z^{-1}},$$

o qual recebe v_n, $n = 0, \ldots, N-1$, como entrada e produz y_n como saída. É possível mostrar [2, 17] que

$$V_k = y_N.$$

Uma outra forma é considerar a sequência $\hat{v}_n$, definida por

$$\hat{v}_n \stackrel{\Delta}{=} v_{((-n-1))_N}. \tag{1.30}$$

Com isso, a DFT de $\hat{v}_n$ é dada por

$$\hat{V}_k = \sum_{n=0}^{N-1} v_{((-n-1))_N} W_N^{nk}.$$

Fazendo a mudança $i = -n - 1 \pmod N$, tem-se

$$\hat{V}_k = \sum_{i=0}^{N-1} v_i W_N^{(-i-1)k},$$

$$\hat{V}_k = W_N^{-k} \sum_{i=0}^{N-1} v_i W_N^{-ik}$$

e, como v contém apenas componentes reais, então

$$\hat{V}_k = W_N^{-k} (V_k)^*,$$

ou

$$V_k = W_N^{-k} (\hat{V}_k)^*. \tag{1.31}$$

A sequência $\hat{V}$ pode ser computada pelo algoritmo de Goertzel ordem inversa e, nesse caso, a entrada é a sequência v na ordem direta. Considerando que $\hat{V}_k = \hat{r}_0 + \hat{r}_1 W_N^k$, tem-se que

$$V_k = W_N^{-k} (\hat{r}_0 + \hat{r}_1 W_N^k)^*,$$

$$V_k = \hat{r}_0 W_N^{-k} + \hat{r}_1 W_N^{-2k} \tag{1.32}$$

e, como W_N^{-k} é zero de $p_k(x)$, pode-se escrever

$$1 - 2\cos\left(\frac{2\pi k}{N}\right) W_N^{-k} + W_N^{-2k} = 0,$$

ou

$$W_N^{-2k} = 2\cos\left(\frac{2\pi k}{N}\right) W_N^{-k} - 1,$$

o que, substituindo em (1.32), resulta em

$$V_k = \left[\hat{r}_0 + 2\cos\left(\frac{2\pi k}{N}\right)\hat{r}_1\right] W_N^{-k} - \hat{r}_1. \tag{1.33}$$

A equação (1.33) é uma nova implementação direta do algoritmo de Goertzel. Embora apresente uma multiplicação real a mais, a implementação direta não precisa armazenar a sequência de entrada v.

A complexidade computacional do Algoritmo de Goertzel é igual à complexidade para computar r_0 e r_1 (ou $\hat{r}_0$ e $\hat{r}_1$), mais a complexidade de se computar V_k. Assim,

$$M_r(N) = N, \tag{1.34}$$

$$A_r(N) = 2N - 3 \tag{1.35}$$

para a implementação ordem inversa, e

$$M_r(N) = N + 1, \tag{1.36}$$

$$A_r(N) = 2N - 2 \tag{1.37}$$

para a implementação ordem direta.

1.5 Descrição do Conteúdo da Tese

Esta tese está organizada em sete capítulos. Este primeiro capítulo apresenta um resumo sobre a teoria de algoritmos e teoria da complexidade aritmética. Ainda neste capítulo, existem descrições dos principais algoritmos utilizados em processamento digital de sinais, dentre os quais se destacam as transformadas rápidas de Fourier (FFT), o algoritmo de convolução e a FFT de Winograd e os algoritmos para computação de uma componente da transformada discreta de Fourier (DFT).

A partir do Capítulo 2, existem apenas contribuições originais da pesquisa. O Capítulo 2 é uma reformulação ou renovação da teoria da complexidade multiplicativa, com o objetivo de introduzir a teoria da complexidade aditiva e, junto com a teoria clássica [3], formar uma teoria geral da complexidade aritmética.

O Capítulo 3 apresenta a teoria da complexidade multiplicativa para transformadas lineares, que, de uma maneira geral, relaciona o problema de se obter um algoritmo ótimo para

se computar uma transformada ao problema de se obter uma base numérica para o núcleo da transformada e, a partir disso, obter uma decomposição em matrizes postunitárias para um conjunto de matrizes. Além disso, mostra-se como aplicar essa teoria na computação de convoluções cíclicas.

O Capítulo 4 apresenta a teoria do Capítulo 3 aplicada à DFT. Neste capítulo, é apresentada a transformada rápida de Fourier otimizada, [15], bem como o caso específico da computação de uma única componente da DFT.

O Capítulo 5 apresenta a teoria da complexidade aditiva para transformadas racionais, e mostra como essa teoria pode ser aplicada para qualquer transformada linear. Alguns resultados apresentados nesse capítulo foram publicados em [18].

O Capítulo 6 apresenta os principais requerimentos de uma representação numérica e de uma arquitetura de processamento para a implementações da transformada discreta de Fourier otimizada para diversos comprimentos.

Finalmente, as conclusões da tese estão no Capítulo 7. A seguir é apresentado o postulado da complexidade aritmética, que é a base da teoria apresentada nesta tese e é inspirado na teoria da complexidade multiplicativa proposta por Winograd [3], a qual foi resumida na Seção 1.2 desta introdução.

capítulo 2

Introdução à Complexidade Aritmética

Este capítulo trata das definições de complexidade aritmética. Seu objetivo é definir o que são operações aritméticas e contabilizá-las em um determinado algoritmo para quantificar a sua complexidade. Embora pareça algo simples, uma definição consistente das operações aritméticas é extremamente necessária para as expressões de complexidade multiplicativa e aditiva, detalhadas adiante. Algoritmos com baixa complexidade aritmética implicam, intuitivamente, algoritmos mais eficientes. De fato, menos operações aritméticas realizadas implica uma menor complexidade de *hardware* e em um menor consumo de energia nos dispositivos eletrônicos utilizados para implementar o algoritmo. Como exemplo de algoritmos aritméticos, pode-se citar: uma avaliação de um polinômio de N-ésimo grau, $p(x)$; a computação de uma rotação de um vetor no espaço; a computação de uma transformada ou de uma convolução.

Considerando que existem quatro tipos de operações aritméticas, a saber, adição, subtração, multiplicação e divisão, e que a complexidade aritmética da adição e subtração são iguais e menores que a complexidade da multiplicação, então é aceitável que os primeiros algoritmos aritméticos eficientes procurassem minimizar o número de multiplicações do processo (considerando que não há divisões). Em 1805, Gauss descobriu uma transformada rápida de Fourier[1], a qual chamou de "Um método de diminuir as tediosas computações dos coeficientes da série finita de Fourier", que seria aplicado para o cálculo da trajetória de corpos celestes. Entretanto, ele encontrou um método analítico melhor para o problema e o trabalho ficou

[1] Esse algoritmo foi redescoberto em 1965 por Cooley e Tukey e é comumente conhecido como FFT.

desconhecido até 1977 [4]. Em 1954, Ostrowski propôs a ideia de se obter um valor mínimo para o número de multiplicações necessárias para a avaliação de um polinômio de grau N. Seu objetivo era demonstrar que o número mínimo de multiplicações para isso é N, entretanto, Motzkin, em 1955, introduziu o conceito de preprocessamento de coeficientes e mostrou um meio de avaliar um polinômio de grau 4 com 3 multiplicações e 5 adições, e um polinômio de grau 7 com 4 multiplicações e 7 adições [5]. Em 1980, Winograd introduziu a teoria da complexidade multiplicativa, na qual se preocupava em diminuir o número de multiplicações de um algoritmo aritmético [3]. Todos os pesquisadores citados objetivaram diminuir o número de multiplicações em algoritmos aritméticos.

Quando Winograd formalizou os primeiros conceitos sobre complexidade multiplicativa, ele também apresentou algoritmos eficientes para computar a convolução cíclica e a transformada discreta de Fourier [3, 11]. Com essas definições, em 1988, Heideman apresentou um valor mínimo para a complexidade multiplicativa (número de multiplicações reais) de uma transformada discreta de Fourier [8]. Curiosamente, a FFT de Winograd não atingia este mínimo para diversos valores do comprimento da transformada, mas, mesmo assim, as FFT de Winograd foram os mais eficientes algoritmos para o cálculo da DFT[2] até 2011, quando foi publicada a FFT otimizada de Jerônimo da Silva e Campello de Souza [15], a qual apresenta algoritmos com a menor complexidade possível proposta por Heideman. Esses algoritmos são apresentados no Capítulo 4.

A seguir são introduzidas as definições referentes à complexidade aritmética; essas definições são compatíveis com as definições de Winograd, de forma que as complexidades multiplicativas não são alteradas. As novidades são a definição de adição e o fato de que as multiplicações triviais são contabilizadas de forma independente, de modo a simplificar a computação das complexidades aditivas.

É importante observar que o postulado da complexidade aritmética, introduzido a seguir, é inspirado nas definições de complexidade multiplicativa de Winograd. Assim como Winograd o fez, são adotados como o corpo das constantes e corpo de computação, o corpo dos gaussianos e o corpo dos complexos, respectivamente; essas noções foram incorporadas diretamente nas definições apresentadas a seguir.

É possível redefinir o postulado da complexidade aritmética para outros corpos, por exemplo, pode-se adotar o corpo GF(p) como o corpo das constantes e GF(p^m) como o corpo de

[2] A FFT de Winograd é aplicada para DFT de pequenos comprimentos, geralmente, comprimentos de até 16 amostras.

computação, para se definir um postulado da complexidade aritmética sobre corpos finitos [19].

2.1 Postulado da Complexidade Aritmética

Operações aritméticas são bem conhecidas, entretanto, é necessário definir uma regra para contabilizar essas operações. Essa regra remove a dificuldade de decidir se "$2x$" é uma multiplicação ou se é uma adição "$x + x$". Isso depende também do que é "x". Para que a teoria seja consistente, o postulado atua sobre algoritmos aritméticos, definidos a seguir.

Definição 2.1 *Uma função que recebe entradas numéricas e computa uma saída (ou resultado) através de um número finito de operações aritméticas (adição, subtração, multiplicação e divisão) entre as entradas e constantes, é chamada de* algoritmo aritmético.

Definição 2.2 *Qualquer elemento numérico não inicialmente especificado, ou função de elementos numéricos não inicialmente especificados, que faz parte da descrição de um algoritmo aritmético é chamado de* variável.

Por exemplo, se um algoritmo computa $y = 2(x_1 + x_2)$, em que x_1 e x_2 são entradas, então a saída, y, é computada através de um algoritmo aritmético, com uma adição de variáveis e uma multiplicação pela constante 2. Os valores x_1, x_2 e $(x_1 + x_2)$ são considerados variáveis.

Definição 2.3 *Considera-se uma multiplicação trivial quando, entre os elementos envolvidos na multiplicação, existe pelo menos um que pertence ao corpo dos números racionais.*

Definição 2.4 *Considera-se uma multiplicação (multiplicação não trivial) quando os elementos envolvidos são duas variáveis ou uma variável e um elemento que não pertence ao corpo dos números racionais.*

Definição 2.5 *As variáveis x_1 e x_2 são ditas dependentes, se x_1 pode ser escrito na forma $x_1 = qx_2$, $q \in \mathbb{Q}$ e são ditas independentes, em caso contrário.*

Lema 2.1 *Considera-se uma adição quando as variáveis envolvidas na operação de adição são independentes.*

Demonstração: Considere a adição $x_1 + x_2$, quando x_1 é dependente de x_2. Por definição, existe um racional q tal que $x_1 = qx_2$; com isso, a soma é $x_1 + x_2 = (q + 1)x_2$; desde que $(q + 1) \in \mathbb{Q}$, então, por definição, isso é uma multiplicação trivial. ■

Definição 2.6 Complexidade multiplicativa *é o número de multiplicações (não triviais) reais realizadas em uma função aritmética.*

Definição 2.7 Complexidade aditiva *é o número de adições reais realizadas em uma função aritmética.*

Observe que a complexidade multiplicativa ou aditiva não depende do algoritmo aritmético, mas sim, de como o mesmo é realizado. Por exemplo, as funções $y = \sqrt{2}(x_1 + x_2)$ e $y = \sqrt{2}x_1 + \sqrt{2}x_2$ são iguais (isto é, apresentam o mesmo resultado, dados x_1 e x_2) mas são realizadas de forma diferente; a primeira é implementada com uma multiplicação e uma adição, enquanto a segunda é implementada com duas multiplicações e uma adição, e, portanto, tem complexidades aritméticas diferentes.

Pode parecer ilógico considerar que qualquer multiplicação que envolve números racionais é considerada trivial, mas o fato é que algumas multiplicações com números racionais podem ser implementadas de forma mais simples. Um exemplo são as multiplicações por 2^i, $i \in \mathbb{Z}$, que podem ser implementadas de forma muito simples, sem o uso de um multiplicador ou somador[3]. A própria definição de complexidade aditiva e multiplicativa está intimamente ligada ao número de somadores e multiplicadores, de forma que mesmo as definições parecendo bem teóricas, elas foram definidas de forma a se ajustar com a implementação de dispositivos eletrônicos.

2.2 Complexidade Aritmética sobre Transformadas

Na seção anterior, definimos a complexidade aditiva e a multiplicativa. Quando estas são consideradas em conjunto, usa-se o termo complexidade aritmética. Este estudo pode ser aplicado a qualquer tipo de algoritmo aritmético, entretanto, o mesmo será focado em transformadas lineares e convoluções. Uma transformada linear de uma sequência v, de componentes v_n, $n = 0, 1, \ldots, N-1$, é a sequência V, de componentes V_k, $k = 0, 1, \ldots, M-1$, dadas por

$$V_k \triangleq \sum_{n=0}^{N-1} v_n \varphi[k, n]. \tag{2.1}$$

[3]No caso em que se utiliza a representação binária, como mostrado no Capítulo 6.

Definindo os vetores

$$V \triangleq \begin{bmatrix} V_0 \\ V_1 \\ \vdots \\ V_{M-1} \end{bmatrix}$$

e

$$v \triangleq \begin{bmatrix} v_0 \\ v_1 \\ \vdots \\ v_{N-1} \end{bmatrix},$$

e a matriz de transformação

$$\Psi \triangleq \begin{bmatrix} \varphi[0,0] & \varphi[0,1] & \cdots & \varphi[0,N-1] \\ \varphi[1,0] & \varphi[1,1] & \cdots & \varphi[1,N-1] \\ \vdots & \vdots & \ddots & \vdots \\ \varphi[M-1,0] & \varphi[M-1,1] & \cdots & \varphi[M-1,N-1] \end{bmatrix},$$

pode-se então representar a transformada pela equação matricial

$$V = \Psi v. \tag{2.2}$$

A seguir é definido um operador que atua sobre uma determinada implementação de uma transformada. Note que cada transformada pode ser implementada de diversas maneiras, cada uma delas com complexidade diferente. Devido a isso, o operador se aplica sobre o algoritmo e não sobre a matriz que o algoritmo representa. Também são definidas formas de representação de um mesmo algoritmo.

Definição 2.8 *A complexidade de um algoritmo que implementa uma transformada Ψ através de $V = \Psi v$, representada por $C\{\Psi\}$, é o número de operações realizadas para a computação de V.*

Essa definição é bem ampla e abstrata, mas se aplica à prática quando se leva em conta a complexidade de uma operação específica, como adição ou multiplicação. Portanto, $C\{\Psi\}$ é a complexidade de se implementar a transformada $V = \Psi v$ pelo método direto, isto é, através de (2.1). Supondo agora que $\Psi = BA$, pode-se implementar Ψ através de $y = Av$, seguido de $v = By$; nesse caso, a complexidade não é necessariamente a mesma.

Definição 2.9 *A complexidade de um algoritmo que implementa uma transformada* $\Psi = \prod_{i=1}^{R} \Psi_i$ *através de* $v_1 = \Psi_1 v$, $v_2 = \Psi_2 v_1$, ..., $v_i = \Psi_i v_{i-1}$, ..., $V = \Psi_R v_{R-1}$ *é representada por* $C\{\prod_{i=1}^{R} \Psi_i\}$.

Segue da definição que

$$C\{\prod_{i=1}^{R} \Psi_i\} = \sum_{i=1}^{R} C\{\Psi_i\}. \tag{2.3}$$

De forma mais simples, isso significa que se $W = BA$, então $C\{BA\} = C\{B\} + C\{A\}$. Note que $W = BA$ não implica, de forma alguma, $C\{BA\} = C\{W\}$. O símbolo de complexidade $C\{BA\}$ representa a complexidade de computar a transformada $V = Wv$ através de $v_1 = Av$, e $V = Bv_1$, enquanto $C\{W\}$ é a complexidade da mesma transformada implementada diretamente por $V = Wv$.

Definição 2.10 *A complexidade de um algoritmo que implementa uma transformada* $\Psi = \sum_{i=1}^{R} \Psi_i$ *através de* $v_1 = \Psi_1 v$, $v_2 = \Psi_2 v$, ..., $v_i = \Psi_i v$, ..., $V = \Psi_R v_R$, *e* $V = \sum_{i=1}^{R} v_i$ *é representada por* $C\{\sum_{i=1}^{R} \Psi_i\}$.

Aqui também vale a obervação anterior, se $W = B + A$, não significa que $C\{W\} = C\{A + B\}$; $C\{W\}$ é a complexidade de se computar $V = Wv$, enquanto $C\{A + B\}$ é a complexidade de se computar $v_A = Av$, $v_B = Bv$ e $V = v_A + v_B$.

Definição 2.11 *As complexidades aditiva e multiplicativa, representadas por* $C_A\{\Psi\}$ *e* $C_M\{\Psi\}$ *respectivamente, representam o número de adições e multiplicações, respectivamente, que envolve a computação de* $V = \Psi v$ *diretamente.*

Note que as definições e representações de complexidades estabelecidas anteriormente valem para ambas, a complexidade aditiva e a multiplicativa.

Teorema 2.1 *Seja* Ψ *uma matriz* $M \times N$, *com* N_r *elementos que não pertencem ao corpo dos reais; então*

$$C_M\{\Psi\} = N_r.$$

Demonstração: *A complexidade* $C_M\{\Psi\}$ *representa o número de multiplicações realizadas para computar*

$$V_k = \sum_{n=0}^{N-1} v_n \varphi[k, n],$$

para $k = 0, \ldots, M-1$. Para cada elemento de Ψ, $\varphi[k,n]$, é necessária uma multiplicação, mas desde que apenas N_r elementos não pertencem ao corpo dos racionais, então, todas as multiplicações, exceto essas N_r, são triviais, por definição. ■

Corolário 2.1 *Seja Ψ uma matriz $M \times N$, com elementos racionais, então*

$$C_M\{\Psi\} = 0.$$

Demonstração: Segue diretamente do Teorema 2.1 quando $N_r = 0$. ■

Corolário 2.2 *Seja Ψ uma matriz diagonal $N \times N$, com elementos que não pertencem ao corpo dos racionais; então*

$$C_M\{\Psi\} = N.$$

Demonstração: Os únicos elementos não nulos estão na diagonal principal e, nesse caso, pelo Teorema 2.1, $N_r = N$. ■

Teorema 2.2 *Seja Ψ uma matriz $M \times N$, sem linhas nulas, com Z elementos nulos; então*

$$C_A\{\Psi\} = M(N-1) - Z. \tag{2.4}$$

Demonstração: A complexidade $C_A\{\Psi\}$ representa o número de adições realizadas para computar

$$V_k = \sum_{n=0}^{N-1} v_n \varphi[k,n],$$

para $k = 0, \ldots, M-1$. Seja Z_k o número de zeros contidos na linha k da matriz Ψ, então existem $N - Z_k$ termos a serem somados para a computação específica de V_k, totalizando $N - 1 - Z_k$ adições. Assim, o total de adições é dado por

$$C_A\{\Psi\} = \sum_{i=0}^{M-1} (N - 1 - Z_k) = (N-1) \sum_{i=0}^{M-1} 1 - \sum_{i=0}^{M-1} Z_k.$$

Desde que

$$\sum_{i=0}^{M-1} Z_k = Z,$$

e

$$\sum_{i=0}^{M-1} 1 = M,$$

o resultado segue. ■

Corolário 2.3 *Seja Ψ uma matriz diagonal $N \times N$, então*

$$C_A\{\Psi\} = 0.$$

Demonstração: Nesse caso, $Z = NN - N = N(N-1)$. Aplicando o Teorema 2.2, tem-se

$$C_A\{\Psi\} = N(N-1) - Z = N(N-1) - N(N-1) = 0,$$

e o corolário está demonstrado. ■

De fato, observa-se que uma matriz diagonal contém apenas multiplicações, desde que a Equação (2.1) se reduz a

$$V_k = v_k \varphi[k,k].$$

Alguns algoritmos rápidos para computar a transformada $V = \Psi v$ consistem em fatorar a matriz de transformação na forma

$$\Psi = CBA,$$

em que A e C são matrizes de gaussianos e B é uma matriz diagonal [3, 7]. Implementar Ψ através de CBA é, de forma geral, mais eficiente. Nesse caso, pelos Corolários 2.1 e 2.3, as complexidades aditivas e multiplicativas são dadas respectivamente por

$$C_A\{CBA\} = C_A\{C\} + C_A\{A\}$$

e

$$C_M\{CBA\} = C_M\{B\}. \tag{2.5}$$

Assim, a complexidade multiplicativa é dada por B, enquanto a complexidade aditiva é dada pelas matrizes A e C. Note que existem muitas maneiras de fatorar uma matriz em $\Psi = CBA$. A complexidade do algoritmo depende, portanto, dessa fatoração. O próximo capítulo mostra como obter uma decomposição do tipo CBA de tal forma que o número de multiplicações seja o menor possível.

capítulo 3

Teoria da Complexidade Multiplicativa Para Transformadas

3.1 Introdução

Esse capítulo trata do estudo da complexidade multiplicativa em uma transformada linear qualquer. De maneira mais específica, os resultados são aplicados à otimização de algoritmos que computam a DFT, levando à chamada transformada rápida de Fourier otimizada (OFFT).

Uma transformada, como visto no capítulo anterior, é um processo que computa um vetor de saída $V = \Psi v$, a partir de um vetor de entrada, v. Desde que Ψ, a princípio, possa representar qualquer transformada, a mesma é chamada simplesmente de "transformada Ψ". Muitos pesquisadores utilizam a fatoração $\Psi = CBA$ para implementar uma transformada[1]; nesse caso, a complexidade multiplicativa passa a ser $C_M\{B\}$. Entretanto, ainda não se pode dizer que a complexidade é exatamente a dimensão de B, já que B pode conter elementos no corpo dos números racionais. O que se pode dizer com certeza é que o número de multiplicações não é maior que a dimensão de B, pela Equação (2.5). O estudo sobre a complexidade multiplicativa começa por introduzir uma nova decomposição, a saber $\Psi = \Psi_0 + CBA$, em que a dimensão de B é exatamente o número de multiplicações necessárias para a computação de Ψv.

Pode-se adiantar que a principal contribuição deste capítulo é o teorema da complexi-

[1] Veja Seção 1.3.4 desta tese e [7].

dade multiplicativa sobre transformadas lineares, que transforma o problema de se obter um algoritmo otimizado em um problema de decomposição de um conjunto de matrizes em matrizes de posto unitário (postunitárias). A ideia do teorema aparece naturalmente quando se tenta escrever uma transformada, que esteja na forma CBA, em uma outra forma dada por $\Psi_0 + CBA$ e é mostrada a seguir[2]. Considere a matriz Ψ dada por

$$\Psi = CBA,$$

em que C e A são matrizes de números racionais e B é uma matriz diagonal. Sabe-se que o número de elementos que não pertencem aos números racionais de B é igual à complexidade multiplicativa. Sendo assim, considere que

$$CBA = \begin{bmatrix} C_1 & \cdots & C_r \end{bmatrix} \begin{bmatrix} b_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & b_r \end{bmatrix} \begin{bmatrix} A_1^T \\ \vdots \\ A_r^T \end{bmatrix}, \tag{3.1}$$

isto é, C_i representa a i-ésima coluna de C, b_i representa os elementos de B e A_i^T representa a i-ésima linha de A. Através de uma simples manipulação matricial, pode-se escrever que

$$B = \begin{bmatrix} b_1 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & b_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} \ldots + \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & b_r \end{bmatrix}. \tag{3.2}$$

Sendo assim,

$$\Psi = C \begin{bmatrix} b_1 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} A + C \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & b_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} A \ldots + C \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & b_r \end{bmatrix} A \tag{3.3}$$

e, portanto,

$$\Psi = C_1 b_1 A_1^T + C_2 b_2 A_2^T + \ldots + C_r b_r A_r^T, \tag{3.4}$$

que pode ser escrita como

$$CBA = \sum_{i=1}^{r} b_i C_i A_i^T. \tag{3.5}$$

[2] As matrizes A, B e C das formas CBA e $\Psi_0 + CBA$ são diferentes. Utilizou-se a mesma notação (ou abuso de notação) por se tratar apenas de uma representação para a forma de decomposição.

As matrizes $C_i A_i^T$ são matrizes postunitárias e de números racionais, isso porque C_i é uma matriz coluna e A_i^T uma matriz linha. Portanto, a Equação (3.5) mostra que qualquer transformada da forma CBA pode ser escrita por uma combinação finita de r matrizes postunitárias. Pode-se, ainda, separar os M_r elementos de B que não pertencem aos números racionais, por

$$\Psi = \sum_{b_i \in \mathbb{Q}} b_i C_i A_i^T + \sum_{b_j \notin \mathbb{Q}} C_j b_j A_j^T. \tag{3.6}$$

Como o primeiro somatório contém apenas matrizes com números racionais, definindo

$$\Psi_0 \triangleq \sum_{b_j \in \mathbb{Q}} b_j C_j A_j^T, \tag{3.7}$$

tem-se

$$\Psi = \Psi_0 + \sum_{b_j \notin \mathbb{Q}} C_j b_j A_j^T. \tag{3.8}$$

O somatório pode ser escrito, utilizando (3.5), na forma CBA (nesse caso, utilizando diferentes matrizes CBA), e portanto

$$\Psi = \Psi_0 + C'B'A', \tag{3.9}$$

em que a matriz B' contém apenas elementos que não pertencem ao corpo dos números racionais. Nesse caso,

$$C_M\{\Psi_0 + C'B'A'\} = C_M\{B'\} = \dim(B'). \tag{3.10}$$

Essas simples manipulações matriciais demonstram dois fatos importantes sobre transformadas na forma CBA: o primeiro é que qualquer matriz da forma CBA pode ser escrita como uma combinação finita de matrizes postunitárias; o segundo é que toda decomposição da forma CBA pode ser posta na forma $\Psi_0 + CBA$, em que a complexidade multiplicativa é igual à dimensão de B. Entretanto, a demonstração de que qualquer matriz de transformação pode ser escrita na forma CBA não é trivial. Para tanto, o conceito de espaço vetorial sobre o corpo dos números racionais, utilizado para gerar um tipo especial de subespaços dos reais, é apresentado na seção a seguir.

3.2 Preliminares

Considere o espaço vetorial, sobre o corpo dos números racionais, de dimensão finita, r, formado pela base de r números reais, $\Lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_r\}$ [20, 21]. Assim, cada vetor x

desse espaço vetorial é um número real dado por

$$x = \sum_{i=1}^{r} x_i \lambda_i, \tag{3.11}$$

em que $x_i \in \mathbb{Q}$. Como o conjunto Λ forma uma base desse espaço vetorial, então a equação

$$\sum_{i=1}^{r} a_i \lambda_i = 0 \tag{3.12}$$

admite como única solução, para $a_i \in \mathbb{Q}$,

$$a_1 = a_2 = \ldots = a_r = 0. \tag{3.13}$$

Nesta tese, adota-se a denominação de *espaço racional numérico* para esse tipo de espaço vetorial. As bases desse espaço são chamadas de *bases numéricas*. Um exemplo de espaço racional numérico é apresentado a seguir.

Exemplo 3.1

A base numérica $\Lambda = \{1, \sqrt{2}, \sqrt{3}\}$ forma um espaço racional numérico que contém o corpo dos números racionais. Entretanto, necessita-se demonstrar que Λ é uma base. Para isso, sabe-se que $\{1, \sqrt{2}\}$ são linearmente independentes num espaço racional numérico. Isso pode ser demonstrado por redução ao absurdo: suponha que exista $a + b\sqrt{2} = 0$, em que $a, b \in \mathbb{Q}$; então, $\sqrt{2} = -a/b$, o que implica que $\sqrt{2}$ é racional, de fato um absurdo.

Para mostrar que Λ é uma base numérica, utiliza-se novamente a redução ao absurdo. Sabe-se que Λ não será uma base se $\sqrt{3}$ for linearmente dependente de $\{1, \sqrt{2}\}$, isto é, se existe

$$a + b\sqrt{2} = \sqrt{3},$$

com $a, b \in \mathbb{Q}$. Nesse caso,

$$(a + b\sqrt{2})^2 = 3$$

$$a^2 + 2b^2 + 2ab\sqrt{2} = 3,$$

então

$$a^2 + 2b^2 = 3$$

e

$$2ab = 0.$$

As soluções são $\{a = 0, b = \sqrt{3/2}\}$ e $\{a = \sqrt{3}, b = 0\}$, que obviamente implicam absurdo, desde que $\sqrt{3}$ e $\sqrt{3/2}$ não são racionais. Assim, Λ é uma base numérica que gera um espaço racional numérico ou, simplesmente, um subespaço do corpo dos reais. Considere agora a seguinte matriz

$$\Psi = \begin{bmatrix} 1 & 1+\sqrt{2} & 1+\sqrt{3} \\ 1 & 1-\sqrt{3} & 1+\sqrt{2} \\ \sqrt{3} & -\sqrt{2} & 1-\sqrt{2}+\sqrt{3} \end{bmatrix}.$$

A matriz Ψ é formada por vetores do espaço racional numérico Λ. Dessa forma, a matriz Ψ pode ser escrita por

$$\Psi = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & -1 \end{bmatrix}\sqrt{2} + \begin{bmatrix} 0 & 0 & 1 \\ 0 & -1 & 0 \\ 1 & 0 & 1 \end{bmatrix}\sqrt{3}.$$

Utilizando uma decomposição em matrizes postunitárias de forma individual sobre essas matrizes, tem-se que

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 \end{bmatrix},$$

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

e

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & -1 & 0 \\ 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}.$$

Utilizando (3.5), pode-se escrever

$$\Psi = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 1 & -1 & -1 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \sqrt{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \sqrt{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sqrt{3} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \sqrt{3} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \sqrt{3} \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

isto é, Ψ está na forma CBA. Note que $C_M\{\Psi\} = 7$ enquanto $C_M\{CBA\} = 5$. Isso significa que o simples fato de se utilizar a decomposição dos elementos da matriz Ψ (do núcleo da transformada Ψ) em uma base numérica mais a decomposição individual das matrizes obtidas em matrizes postunitárias, possibilita à transformada ser escrita na forma CBA, ocasionando uma diminuição na complexidade multiplicativa. Entretanto, essa ainda não é a melhor maneira de se resolver o problema. •

Uma conclusão simples é que qualquer conjunto finito de números reais pode ser decomposto por uma base numérica. Por conseguinte, qualquer matriz de dimensão finita pode ter seus elementos decompostos em uma base numérica, isto é, toda matriz Ψ pode ser escrita como

$$\Psi = \sum_{i=1}^{r} \Psi_i \lambda_i, \tag{3.14}$$

em que $\Lambda = \{\lambda_1, \ldots, \lambda_r\}$ é a base numérica que gera todos os elementos de Ψ e $\Psi_1, \ldots, \Psi_r$ são matrizes de números racionais. Isso implica que qualquer matriz de dimensão finita pode ser posta na forma CBA e, portanto, também na forma $\Psi_0 + CBA$. Em seguida, utilizando todos esses conceitos, é apresentado o Teorema da complexidade multiplicativa sobre transformadas.

3.3 Teorema da Complexidade Multiplicativa para Transformadas

Toda matriz de transformação Ψ de dimensão finita pode ser decomposta como em (3.14). Segue o primeiro teorema cuja demonstração já foi feita na introdução (com ênfase na mudança da forma CBA para combinação linear de matrizes postunitárias), entretanto, pode-se ainda enfatizar como segue.

Teorema 3.1 *Todo algoritmo de transformada $V = \Psi v$ escrito da forma $V = CBAv$, isto é, através da decomposição $\Psi = CBA$, em que C e A são matrizes de números racionais, e B é uma matriz diagonal com M_r componentes que não pertencem aos números racionais, pode ser escrito na forma*

$$V = (\Psi_0 + \sum_{j=1}^{M_r} \beta_j K_j)v,$$

e vice-versa, em que $\beta_j \notin \mathbb{Q}$, Ψ_0 é uma matriz de números racionais e K_j são matrizes de números racionais postunitárias.

Demonstração: Sendo C_i as colunas de C, b_i os elementos de B e A_i^T as linhas de A, então

$$CBA = \begin{bmatrix} C_1 & \cdots & C_N \end{bmatrix} \begin{bmatrix} b_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & b_N \end{bmatrix} \begin{bmatrix} A_1^T \\ \vdots \\ A_N^T \end{bmatrix}, \tag{3.15}$$

$$CBA = \sum_{j=1}^{N} b_j C_j A_j^T. \tag{3.16}$$

As matrizes $C_j A_j^T$ são postunitárias. Chamando os M_r valores de $b_j \notin \mathbb{Q}$ de β_i, pode-se escrever

$$CBA = (\sum_{b_j \in \mathbb{Q}} b_j C_j A_j^T) + \sum_{i=1}^{M_r} \beta_i K_i, \tag{3.17}$$

em que K_i é o respectivo $C_j A_j^T$ para $b_j \notin \mathbb{Q}$. ■

O próximo teorema estabelece uma relação se e somente se entre a complexidade do algoritmo e a decomposição em matrizes postunitárias das matrizes Ψ_i.

Teorema 3.2 (Teorema da complexidade multiplicativa) *Seja*

$$\Psi = \Psi_0 + \sum_{i=1}^{L} \lambda_i \Psi_i \tag{3.18}$$

a matriz de uma transformada em que $\Lambda = \{1, \lambda_1, \ldots, \lambda_L\}$, $\lambda_i \notin \mathbb{Q}$, é o conjunto das bases numéricas que gera o núcleo da transformada, a matriz Ψ_0 é uma matriz de números racionais (ou gaussianos) e as matrizes Ψ_i, $i = 1, \ldots, L$ são matrizes de números racionais. Existe um algoritmo para computar $V = \Psi v$ com M_r multiplicações se, e somente se, existe uma decomposição das matrizes Ψ_i, $i = 1, \ldots, L$, como combinação linear de M_r matrizes postunitárias.

Demonstração: Supondo que existe um algoritmo com M_r multiplicações, então, pelo Teorema 3.1,

$$\Psi = \Psi'_0 + \sum_{j=1}^{M_r} \beta_j K_j.$$

Mas todos os β_j podem ser escritos através de uma combinação linear das bases numéricas Λ por

$$\beta_j = c_{0j} + \sum_{i=1}^{L} c_{ij}\lambda_i,$$

em que $c_{ij} \in \mathbb{Q}$, então

$$\Psi = \Psi'_0 + \sum_{j=1}^{M_r}(c_{0j} + \sum_{i=1}^{L} c_{ij}\lambda_i)K_j = \Psi'_0 + \sum_{j=1}^{M_r} c_{0j}K_j + \sum_{i=1}^{L}\lambda_i \sum_{j=1}^{M_r} c_{ij}K_j.$$

Usando (3.18) no primeiro membro da equação, chega-se a

$$\Psi_0 + \sum_{i=1}^{L}\lambda_i\Psi_i = \Psi'_0 + \sum_{j=1}^{M_r} c_{0j}K_j + \sum_{i=1}^{L}\lambda_i \sum_{j=1}^{M_r} c_{ij}K_j,$$

então

$$(\Psi_0 - \Psi'_0 - \sum_{j=1}^{M_r} c_{0j}K_j) + \sum_{i=1}^{L}\lambda_i(\Psi_i - \sum_{j=1}^{M_r} c_{ij}K_j) = O,$$

em que O é uma matriz nula. Neste caso, como os λ_i são linearmente independentes sobre o corpo dos números racionais, a única solução para a equação matricial é que todas as matrizes devem ser nulas, isto é

$$(\Psi_0 - \Psi'_0 - \sum_{j=1}^{M_r} c_{0j}K_j) = (\Psi_i - \sum_{j=1}^{M_r} c_{ij}K_j) = O,$$

o que implica que existe uma decomposição em matrizes postunitárias para todos os Ψ_i. Por outro lado, suponha que exista uma decomposição de Ψ_i, $i = 1, \ldots, L$, em M_r matrizes postunitárias, isto é

$$\Psi_i = \sum_{j=1}^{M_r} c_{ij}K_j. \tag{3.19}$$

Substituindo a decomposição em (3.18), tem-se

$$\Psi = \Psi_0 + \sum_{j=1}^{M_r} K_j(\sum_{i=1}^{L} c_{ij}\lambda_i),$$

fazendo

$$\beta_j = \sum_{i=1}^{L} c_{ij}\lambda_i, \tag{3.20}$$

desde que $\lambda_i \notin \mathbb{Q}$, então $\beta_j \notin \mathbb{Q}$ e, dessa forma,

$$\Psi = \Psi_0 + \sum_{j=1}^{M_r} \beta_j K_j,$$

que, pelo Teorema 3.1, pode ser escrito na forma CBA com M_r multiplicações. ■

Esse teorema mostra que existe um algoritmo ótimo para se computar $V = \Psi v$ e mostra como esse algoritmo pode ser obtido. Assim, qualquer transformada pode ser otimizada se o núcleo é decomposto em bases numéricas e as matrizes obtidas são decompostas com o menor número possível de matrizes postunitárias. O problema de se obter uma base numérica é relativamente simples. Já o problema de decompor um conjunto de matrizes em matrizes postunitárias da melhor maneira possível requer, até o presente momento (espera-se encontrar um método direto que resolva a decomposição da melhor maneira possível), uma busca; entretanto, um método para simplificar essa busca é apresentado no próximo capítulo. Mas, antes disso, é apresentado como uma convolução cíclica pode ser manipulada de tal forma que é possível aplicar o teorema da complexidade multiplicativa para transformadas. A seguir é apresentado o teorema da complexidade multiplicativa para convoluções cíclicas.

3.4 A Complexidade de uma Convolução Cíclica

Uma convolução cíclica é uma operação que envolve dois vetores de variáveis, descrita pela equação apresentada no Capítulo 1 (1.3). Definindo

$$s \triangleq \begin{bmatrix} s_0 \\ s_1 \\ \vdots \\ s_{N-2} \\ s_{N-1} \end{bmatrix},$$

$$H \triangleq \begin{bmatrix} g_0 & g_{N-1} & \cdots & g_2 & g_1 \\ g_1 & g_0 & \cdots & g_3 & g_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ g_{N-2} & g_{N-3} & \cdots & g_0 & g_{N-1} \\ g_{N-1} & g_{N-2} & \cdots & g_1 & g_0 \end{bmatrix}$$

e

$$d \triangleq \begin{bmatrix} d_0 \\ d_1 \\ \vdots \\ d_{N-1} \end{bmatrix},$$

então

$$s = Hd.$$

Isso é semelhante a uma transformada, exceto pelo fato de que a matriz de transformação contém variáveis. Nesse caso, pode-se decompor a matriz H como se as variáveis fossem as bases numéricas, isto é,

$$H = \begin{bmatrix} 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & \dots & 0 & 1 \end{bmatrix} g_0 + \dots + \begin{bmatrix} 0 & 1 & \dots & 0 & 0 \\ 0 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & 1 \\ 1 & 0 & \dots & 0 & 0 \end{bmatrix} g_{N-1}.$$

Pode-se definir a matriz de deslocamento unitário cíclico de ordem N, $D_N(N \times N)$ por

$$D_N \triangleq \begin{bmatrix} 0 & 0 & \dots & 0 & 1 \\ 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & 0 \end{bmatrix}.$$

Dessa forma, a matriz H pode ser escrita como

$$H = \sum_{n=0}^{N-1} D_N^n g_n,$$

em que a matriz $D_N^0 = D_N^N = I_N$. Portanto,

$$s = \left(\sum_{n=0}^{N-1} D_N^n g_n \right) d. \tag{3.21}$$

Teorema 3.3 *Existe um algoritmo que computa uma convolução cíclica de ordem N com M_r multiplicações se, e somente se, existe uma decomposição para as matrizes D_N^n, $n = 0, \dots, N-1$, em M_r matrizes postunitárias.*

Demonstração: Se existe a decomposição, então

$$D_N^i = \sum_{j=0}^{M_r-1} c_{ij} K_j,$$

em que c_{ij} são números racionais para todo $i = 0, \ldots, N-1$ e $j = 0, \ldots, M_r - 1$, e as matrizes K_j, $j = 1, \ldots, M_r$, são matrizes postunitárias. Então, a Equação (3.21) pode ser escrita por

$$s = \left(\sum_{n=0}^{N-1} \sum_{j=0}^{M_r-1} c_{nj} K_j g_n \right) d,$$

$$s = \left[\sum_{j=0}^{M_r-1} K_j \left(\sum_{n=0}^{N-1} c_{nj} g_n \right) \right] d.$$

Definindo uma nova variável, β_j, por

$$\beta_j \triangleq \sum_{n=0}^{N-1} c_{nj} g_n$$

e separando as matrizes K_j em

$$K_j = C_j A_j^T,$$

em que as matrizes C_j contêm apenas uma coluna e as matrizes A_j^T contêm apenas uma linha, pode-se escrever que

$$s = \sum_{j=0}^{M_r-1} C_j \beta_j A_j^T d,$$

o que, por (3.5), pode ser posto na forma

$$s = \begin{bmatrix} C_0 & \cdots & C_{M_r-1} \end{bmatrix} \begin{bmatrix} \beta_0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \beta_{M_r-1} \end{bmatrix} \begin{bmatrix} A_0^T \\ \vdots \\ A_{M_r-1}^T \end{bmatrix} d, \tag{3.22}$$

ou

$$s = C[(c^T g) \odot (Ad)], \tag{3.23}$$

em que a matriz c é a matriz formada pelos elementos c_{ij}, a matriz g é dada por

$$g \triangleq \begin{bmatrix} g_0 \\ g_1 \\ \vdots \\ g_{N-1} \end{bmatrix}$$

e $\odot$ representa uma multiplicação termo a termo entre as matrizes com apenas uma coluna, que são as únicas multiplicações na implementação da Equação (3.23). Desde que as matrizes $c^T g$ e as matrizes Ad tem M_r linhas, então a implementação utiliza M_r multiplicações.

Agora, supondo que uma equação do tipo (3.23) existe, chega-se a

$$s = \left(\sum_{n=0}^{N-1} \sum_{j=0}^{M_r-1} c_{nj} C_j A_j^T g_n \right) d,$$

o que pode ser igualado a (3.21), levando a

$$\sum_{n=0}^{N-1} \sum_{j=0}^{M_r-1} c_{nj} C_j A_j^T g_n = \sum_{n=0}^{N-1} D_N^n g_n,$$

e, portanto,

$$\sum_{n=0}^{N-1} g_n \left(D_N^n - \sum_{j=0}^{M_r-1} c_{nj} C_j A_j^T \right) = O,$$

em que O é uma matriz nula. Essa relação deve valer para qualquer escolha de g, já que g é uma variável. Então, a única solução possível é dada por

$$D_N^n - \sum_{j=0}^{M_r-1} c_{nj} C_j A_j^T = O,$$

para todo $n = 0, \ldots, N-1$. Dessa forma existe uma decomposição para D_N^n em M_r matrizes postunitárias. ■

Antes de apresentar uma solução simplificada para a decomposição em matrizes postunitárias, é apresentado um exemplo da obtenção de um algoritmo ótimo para convolução cíclica. As matrizes D_N^n estão em uma forma que não podem ser simplificadas, assim a decomposição deve ser feita por busca. Nos exemplos a seguir são apresentadas algumas técnicas de busca.

Exemplo 3.2

Obter um algoritmo para a convolução cíclica de comprimento 2. Pelo Teorema 3.3, é necessário obter uma decomposição em matrizes postunitárias para as matrizes

$$D_2^0 = I_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

e

$$D_2 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.$$

Um maneira de obter decomposições em matrizes de posto menor é tentar combinar as matrizes, toda vez que o resultado é uma matriz de posto menor que os das matrizes a serem decompostas, essas podem ser expressas como combinação linear da nova matriz. Nesse exemplo, é possível obter as seguintes matrizes postunitárias combinando as matrizes I_2 e D_2 da seguinte forma:

$$K_1 = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = I_2 + D_2$$

e

$$K_2 = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} = I_2 - D_2.$$

Pode-se, então, escrever

$$\begin{bmatrix} K_1 \\ K_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} I_2 \\ D_2 \end{bmatrix},$$

de modo que

$$\begin{bmatrix} I_2 \\ D_2 \end{bmatrix} = \begin{bmatrix} -\frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} K_1 \\ K_2 \end{bmatrix}.$$

As matrizes postunitárias podem ser escritas, aplicando a decomposição du, apresentada mais adiante, para cada matriz separadamente, levando a

$$K_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \end{bmatrix}$$

e

$$K_2 = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \begin{bmatrix} 1 & -1 \end{bmatrix}.$$

Dessa forma, utilizando (3.23),

$$s = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \left(\begin{bmatrix} -\frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & \frac{1}{2} \end{bmatrix} g \odot \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} d \right). \quad \bullet$$

A partir do teorema da complexidade multiplicativa, é possível derivar um algoritmo otimizado (em termos da complexidade multiplicativa) para qualquer transformada que se conheça as bases numéricas que forma o núcleo da mesma. O próximo capítulo apresenta uma metodologia para se obter algoritmos otimizados para a DFT.

capítulo 4

A Transformada Rápida de Fourier Otimizada

4.1 Introdução

Ocorreu um grande avanço na área de processamento digital de sinais quando J. W. Cooley e J. W. Tukey apresentaram, em 1965, a transformada rápida de Fourier [13], um algoritmo capaz de computar a transformada discreta de Fourier com uma complexidade aritmética menor em relação ao método direto. Em 1978, S. Winograd publicou o algoritmo que hoje é conhecido como a FFT de Winograd [11], além de publicar diversos trabalhos sobre complexidade multiplicativa [3]. Este trabalho se destaca por apresentar uma complexidade ainda menor que a FFT de Cooley-Tukey. Em 1988, M. T. Heideman publicou um trabalho sobre a complexidade multiplicativa para a computação da DFT [8], mostrando que as complexidades mínimas para os comprimentos $N = 3, 5, 7, 9, 10$ estavam abaixo da complexidade multiplicativa da mais eficiente FFT existente na época, a FFT de Winograd. Até recentemente, não tinha sido apresentado um único algoritmo com complexidade multiplicativa inferior à da FFT de Winograd. Somente em 2010, G. Jerônimo da Silva Jr. e R. M. Campello de Souza publicaram a primeira FFT de comprimento 3 que atinge a complexidade multiplicativa mínima estabelecida por Heideman [22]. O artigo se baseia numa decomposição para o núcleo da transformada de Fourier denominada *decomposição em bases ciclotômicas*, que são as bases numéricas para o núcleo da DFT.

No capítulo anterior, o Teorema 3.2 transforma o problema da construção de um algoritmo ótimo em um problema de decomposição em matrizes postunitárias. O problema de

se obter uma transformada otimizada para qualquer conjunto de componentes de qualquer transformada se resume, portanto, a obter uma base numérica para o núcleo da matriz de transformação e decompor as matrizes obtidas, a partir disso, em matrizes postunitárias.

O problema de obter uma base numérica depende essencialmente da transformada. Já o problema da decomposição de um conjunto de matrizes em matrizes postunitárias é geral e aplicável a todas as otimizações. As bases numéricas para as transformadas discretas de Fourier e Hartley estão bem estabelecidas e são apresentadas na seção a seguir, sendo chamadas de bases ciclotômicas.

4.2 Bases Ciclotômicas

É interessante mencionar que a teoria das bases ciclotômicas foi descoberta antes do Teorema da complexidade multiplicativa [22], entretanto, ela completa perfeitamente uma parte do processo de se obter um algoritmo otimizado para a DFT. Para obter as bases ciclotômicas é necessário apresentar o chamado polinômio ciclotômico.

O polinômio ciclotômico de ordem N sobre $\mathbb{C}$, denotado por $\Phi_N(x)$, é definido como o polinômio mônico cujas raízes são todos os elementos de ordem N no corpo dos números complexos [7, 8]. Esse polinômio pode ser escrito como

$$\Phi_N(x) = \prod_{\mathrm{ord}(\theta)=N} (x - \theta).$$

Sabendo que

$$\prod_{d|N} \Phi_d(x) = (x^N - 1),$$

pode-se mostrar, utilizando a formula de inversão de Möbius [19], que

$$\Phi_N(x) = \prod_{d|N} (x^d - 1)^{\mu(N/d)},$$

em que $\mu(n)$ é a função de Möbius [14], a qual é definida, para a fatoração de n dada por $n = p_1^{e_1} p_2^{e_2} \ldots p_m^{e_m}$, como

$$\mu(n) = \begin{cases} 1, & \text{se } n = 1; \\ 0, & \text{se } \exists e_i \geq 2; \\ (-1)^m, & \text{caso contrário.} \end{cases}$$

O grau do polinômio $\Phi_N(x)$ é dado pela função de Euler $\phi(N)$ (*Euler's totient function*), definida como o número de inteiros positivos menores que N e relativamente primos com N.

Os polinômios ciclotômicos têm a interessante propriedade de possuírem apenas coeficientes inteiros, sendo a maioria deles iguais a 0, 1 ou -1 (para $N < 106$ os coeficientes são apenas 0,1 ou -1, [7]). O polinômio ciclotômico é utilizado para construir bases ciclotômicas por um processo apresentado mais adiante. A seguir, é apresentada a definição de base ciclotômica.

Definição 4.1 *Uma* base ciclotômica *(BC_N) é um conjunto do corpo dos complexos tal que cada elemento do grupo cíclico multiplicativo de ordem N (GCM_N) pode ser representado por meio de uma combinação linear, com coeficientes racionais, dos elementos de BC_N.*

Em outras palavras, uma base ciclotômica é uma base numérica que gera GCM_N, isto é, gera o núcleo da DFT de comprimento N. Pode-se observar que $\alpha = W_N$ é uma raiz de $\Phi_N(x)$. Dessa forma, a partir da equação $\Phi(\alpha) = 0$, pode-se escrever todos os elementos do $\mathrm{GCM}_N = \{\alpha^0, \ldots, \alpha^{N-1}\}$ como uma combinação, com coeficientes inteiros, dos elementos $\{1, \alpha, \alpha^2, \ldots, \alpha^{\phi(N)-1}\}$.

Proposição 4.1 *Seja α um elemento gerador do GCM_N. A* base ciclotômica canônica *(BCC) para GCM_N é a base ciclotômica $BCC_N = \{1, \alpha, \alpha^2, \ldots, \alpha^{\phi(N)-1}\}$.*

Demonstração: Como α possui ordem N, então, α é raiz de $\Phi_N(x)$. Usando divisão polinomial, todo x^i pode ser expresso por

$$x^i = q_i(x)\Phi_N(x) + r_i(x),$$

em que $r_i(x)$ é um polinômio com coeficientes inteiros de grau menor que $\phi(N)$. Substituindo $x = \alpha$, e usando o fato que $r_i(x) = x^i \pmod{\Phi_N(x)}$, então

$$\alpha^i = \alpha^i (\mathrm{mod}\ \Phi_N(\alpha)). \tag{4.1}$$

■

Exemplo 4.1

Para $N = 6$, $\alpha = W_6$ é raiz de $\Phi_6(x) = x^2 - x + 1$, ou $\Phi(\alpha) = \alpha^2 - \alpha + 1 = 0$. O conjunto $\mathrm{BCC}_6 = \{1, \alpha\}$ é a base ciclotômica canônica do grupo multiplicativo $\mathrm{GMC}_6 = \{1, \alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^5\}$. A representação dos elementos de GCM_6 como combinação linear de $\{1, \alpha\}$ é obtida através de

$$\begin{aligned} \alpha^0 &= 1, & \alpha^3 &= -1, \\ \alpha^1 &= \alpha, & \alpha^4 &= -\alpha, \\ \alpha^2 &= -1 + \alpha, & \alpha^5 &= 1 - \alpha; \end{aligned}$$

ou simplesmente

$$\begin{bmatrix} \alpha^0 \\ \alpha^1 \\ \alpha^2 \\ \alpha^3 \\ \alpha^4 \\ \alpha^5 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ -1 & 1 \\ -1 & 0 \\ 0 & -1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} \alpha^0 \\ \alpha^1 \end{bmatrix}.$$

•

Pode-se escrever, para todo GCM_N,

$$\begin{bmatrix} \alpha^0 \\ \vdots \\ \alpha^{N-1} \end{bmatrix} = R_c \begin{bmatrix} \alpha^0 \\ \vdots \\ \alpha^{\phi(N)-1} \end{bmatrix}, \tag{4.2}$$

em que $\alpha = W_N$ e $R_c(N \times \Phi(N))$ é a matriz da decomposição de GCM_N na base ciclotômica canônica, chamada de *matriz canônica de ordem* N, a qual possui apenas elementos inteiros.

Utilizando a decomposição (4.2), é possível fazer uma mudança de base para representar o GCM_N sobre outra base ciclotômica. As bases ciclotômicas são complexas, portanto, é mais conveniente procurar bases com elementos puramente reais ou puramente imaginários. Para fazer a mudança da base ciclotômica para uma outra base, basta expressar a nova base como

$$\begin{bmatrix} \gamma_1 \\ \vdots \\ \gamma_{\phi(N)} \end{bmatrix} = Q \begin{bmatrix} \alpha^0 \\ \vdots \\ \alpha^{\phi(N)-1} \end{bmatrix},$$

em que Q é uma matriz quadrada qualquer. Se Q é não singular, então

$$\begin{bmatrix} \alpha^0 \\ \vdots \\ \alpha^{\phi(N)-1} \end{bmatrix} = Q^{-1} \begin{bmatrix} \gamma_1 \\ \vdots \\ \gamma_{\phi(N)} \end{bmatrix}$$

e o GCM_N pode ser expresso pela nova base

$$\begin{bmatrix} \alpha^0 \\ \vdots \\ \alpha^{N-1} \end{bmatrix} = R_c Q^{-1} \begin{bmatrix} \gamma_1 \\ \vdots \\ \gamma_{\phi(N)} \end{bmatrix},$$

e, nesse caso, a matriz de decomposição é $R = R_c Q^{-1}$. Assim, é possível escolher como base ciclotômica qualquer combinação linear (com coeficentes racionais) dos elementos de GCM_N, desde que Q seja invertível. O próximo exemplo demonstra detalhadamente como uma mudança de base pode ser conveniente.

Exemplo 4.2

Para $N = 8$ tem-se, a partir de (4.2), $\Phi_8(x) = 1 + x^4$. Computando R_c(8x4) através de (4.1) e (4.2) chega-se à seguinte decomposição utilizando $\mathrm{BCC}_8 = \{1, \alpha, \alpha^2, \alpha^3\}$,

$$\begin{bmatrix} \alpha^0 \\ \alpha^1 \\ \alpha^2 \\ \alpha^3 \\ \alpha^4 \\ \alpha^5 \\ \alpha^6 \\ \alpha^7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ \alpha \\ \alpha^2 \\ \alpha^3 \end{bmatrix}.$$

Procura-se uma base ciclotômica com elementos puramente reais ou puramente imaginários a partir de BCC_8. Sabe-se que $(\alpha + \alpha^7)/2 = \cos(\pi/4)$ e $(\alpha - \alpha^7)/2 = -j\mathrm{sen}(\pi/4)$. Faz-se então as mudanças de base para $CB_8 = \{1, -j, \cos(\pi/4), -j\mathrm{sen}(\pi/4)\}$. Usando a matriz R_c, dada por

$$R_c = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix},$$

pode-se escrever

$$\begin{bmatrix} 1 \\ -j \\ \frac{\sqrt{2}}{2} \\ -j\frac{\sqrt{2}}{2} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & \frac{1}{2} & 0 & -\frac{1}{2} \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \end{bmatrix} \begin{bmatrix} 1 \\ \alpha \\ \alpha^2 \\ \alpha^3 \end{bmatrix},$$

em que a terceira linha de Q, referente ao cosseno, pode ser obtida somando-se as linhas 2 e 8 de Rc e dividindo tudo por 2, assim como a quarta linha de Q pode ser obtida subtraindo

a linha 8 da linha 2 de R_c e dividindo o resultado por 2. Desde que a matriz

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & \frac{1}{2} & 0 & -\frac{1}{2} \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \end{bmatrix}$$

seja invertível, pode-se escrever a matriz de decomposição $R = R_c Q^{-1}$, o que resulta em

$$\begin{bmatrix} \alpha^0 \\ \alpha^1 \\ \alpha^2 \\ \alpha^3 \\ \alpha^4 \\ \alpha^5 \\ \alpha^6 \\ \alpha^7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 1 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & -1 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ -j \\ \frac{\sqrt{2}}{2} \\ -j\frac{\sqrt{2}}{2} \end{bmatrix}.$$

•

Esse exemplo inspira as próximas duas definições.

Definição 4.2 *A* base ciclotômica do seno e cosseno *(sen/cos-BC) é a base ciclotômica com os* $\phi(N)$ *elementos,* $CB_N = \{1, (\alpha - \alpha^{N-1})/2, (\alpha + \alpha^{N-1})/2, (\alpha^2 - \alpha^{N-2})/2, \ldots\}$*, ou simplesmente* $CB_N = \{1, -jsen(2\pi/N), cos(2\pi/N), -jsen(4\pi/N), \ldots\}$.

Definição 4.3 *A* base ciclotômica do cosseno *(cos-BC) é a base com os* $\phi(N)$ *elementos,* $N \equiv 0 \ (mod\ 4)$*,* $CB_N = \{1, \alpha^{N/4}, (\alpha + \alpha^{N-1})/2, (\alpha^{1+N/4} + \alpha^{N/4-1})/2, \ldots\}$*, ou simplesmente* $CB_N = \{1, -j, \cos(2\pi/N), -j\cos(2\pi/N), \cos(4\pi/N), -j\cos(4\pi/N), \ldots\}$.

A vantagem de escolher a base ciclotômica sen/cos-BC ou cos-BC, em relação a base ciclotômica canônica, é que os elementos da base são puramente reais ou imaginários. Nesse caso, tem-se

$$\begin{bmatrix} \alpha^0 \\ \vdots \\ \alpha^{N-1} \end{bmatrix} = R \begin{bmatrix} \gamma_1 \\ \vdots \\ \gamma_{\phi(N)} \end{bmatrix}, \tag{4.3}$$

em que R é a matriz de decomposição na base ciclotômica escolhida. Frequentemente, R contém, na maioria de seus elementos, números inteiros. A base ciclotômica formada pelos elementos $\gamma_1, \gamma_2, \ldots, \gamma_{\phi(N)}$ é a cos-BC se N é divisível por quatro ou a sen/cos-BC no caso de

N não ser divisível por quatro. Se r_{ij}, $i = 1, \ldots, N$, $j = 1, \ldots, \phi(N)$, representa os elementos de R, então

$$\alpha^i = \sum_{j=1}^{\phi(N)} r_{(i+1)j}\gamma_j. \tag{4.4}$$

A Equação (4.4) caracteriza a decomposição em bases ciclotômicas. O GCM_N, que é o núcleo de uma DFT de comprimento N, pode ser decomposto por uma base numérica, por exemplo a cos-BC ou a sen/cos-BC. Isso significa que qualquer matriz de DFT, W, pode ser posta na forma

$$W = \sum_{i=1}^{\phi(N)} W_i\gamma_i,$$

em que as matrizes W_i são matrizes de racionais. Com isso, pode-se então aplicar o teorema da complexidade multiplicativa para implementar o melhor algoritmo para uma determinada DFT. Um algoritmo para a computação de uma DFT (computando todas as componentes) obtido através desse processo é chamado de *transformada rápida de Fourier otimizada* (OFFT) e é apresentado mais adiante.

Antes de apresentar o método de construção do algoritmo otimizado, é apresentada uma maneira de resolver a decomposição das matrizes W_i em matrizes postunitárias.

4.3 Decomposição Ortogonal Postunitária e Solução Otimizada

Antes de abordar a decomposição em matrizes postunitárias, é apresentada uma decomposição matricial, a qual será chamada de decomposição *du*, que pode ser vista como uma variante da decomposição *qr* [20].

4.3.1 Decomposição *du* para uma matriz

Definição 4.4 *Seja A uma matriz qualquer $M \times N$. Qualquer decomposição $A = du$, em que u é uma matriz com p linhas ortogonais e p é o posto de A, é chamada de decomposição du de A.*

Esta definição afirma que, em uma decomposição *du*, as linhas da matriz u formam uma base ortogonal para o espaço de todas as linhas de A. Como exemplo, considere a seguinte matriz

$$A = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}.$$

Essa matriz tem posto dois, portanto, qualquer decomposição da forma $A = du$, em que u contém duas linhas ortogonais, é uma decomposição du de A. As linhas de u podem ser obtidas utilizando o processo de Gram-Schmidt nas linhas de A, sem a normalização, [20, 21] (consulte o Apêndice A). Assim, com

$$u = \begin{bmatrix} 1 & 0 & 1 & 1 \\ -\frac{2}{3} & 1 & \frac{1}{3} & \frac{1}{3} \end{bmatrix},$$

tem-se

$$A = \begin{bmatrix} 1 & 0 \\ \frac{2}{3} & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 & 1 \\ -\frac{2}{3} & 1 & \frac{1}{3} & \frac{1}{3} \end{bmatrix},$$

de modo que, sendo u_i a i-ésima linha de u, A é dado por

$$A = \begin{bmatrix} u_1 \\ \frac{2}{3}u_1 + u_2 \end{bmatrix},$$

o que é uma combinação linear das linhas de u descrita pelas linhas da matriz d. Esse resultado em particular é generalizado da seguinte maneira; seja

$$A = \begin{bmatrix} A_1^T \\ A_2^T \\ \vdots \\ A_M^T \end{bmatrix},$$

(considerando que A_i é um vetor, utiliza-se aqui a convenção de que um vetor é uma matriz com uma coluna, assim A_i^T é uma matriz linha ou simplesmente uma linha de A) e seja $A = du$ uma decomposição du de A, sendo

$$u = \begin{bmatrix} u_1^T \\ u_2^T \\ \vdots \\ u_p^T \end{bmatrix},$$

e sendo d_{ij} o elemento da linha i e coluna j de d. Então

$$A_i = \sum_{j=1}^{p} d_{ij} u_j.$$

O fato de que as linhas de u são ortogonais implica que o produto escalar $\langle u_i, u_j \rangle = 0$, para todo $i \neq j$. Assim, pode-se escrever que

$$\langle A_i, u_k \rangle = \sum_{j=1}^{p} d_{ij} \langle u_j, u_k \rangle = d_{ik} \langle u_k, u_k \rangle,$$

e, portanto, pode-se encontrar d_{ij} por meio de

$$d_{ij} = \frac{\langle A_i, u_j \rangle}{\langle u_j, u_j \rangle}. \tag{4.5}$$

Se a matriz u de uma decomposição du é obtida através do processo de ortogonalização de Gram-Schmidt, na ordem exata das linhas de A, a matriz d é triangular. Isso é um processo parecido com a decomposição qr; por exemplo, as matrizes q e r obtidas da decomposição $A^T = qr$, implicam $A = r^T q^T$ e a matriz q^T é uma matriz com todas as linhas ortogonais e, portanto, a decomposição $A = r^T q^T$ é uma decomposição du. Entretanto, a matriz u não precisa ser necessariamente ortogonal, como a matriz q da decomposição qr. O fato de não haver necessidade de normalizar as matrizes permite ao processo de ortogonalização de Gram-Schmidt preservar o corpo da matriz A, isto é, se A é uma matriz de racionais, então existe $A = du$, uma decomposição du, em que as matrizes d e u são matrizes racionais, o que não é necessariamente verdade para a decomposição qr.

Uma outra observação importante é que existem muitas decomposições du para uma mesma matriz A, isto é, a decomposição não é única, ela depende necessariamente da matriz u escolhida. Para padronizar as decomposições du utilizadas nessa tese, considera-se, salvo se devidamente especificado, que todas as decomposições du são obtidas escolhendo as p linhas ortogonais para u através do processo de Gram-Schmidt das linhas de A, na ordem da primeira para a última linha e sem normalização. Dessa forma, são utilizadas as notações a seguir.

Uma matriz u obtida a partir do processo de ortogonalização de Gram-Schmidt (sem normalização) das linhas da matriz A (aplicado na ordem das linhas da matriz) é denotada por

$$u = \mathrm{GS}_L(A).$$

Observe que vale a propriedade

$$\mathrm{posto}(A) = \mathrm{posto}(\mathrm{GS}_L(A)).$$

Uma matriz u obtida a partir do processo de ortogonalização de Gram-Schmidt (sem normalização) das colunas da matriz A (aplicado na ordem das colunas da matriz) é denotada por

$$u = \mathrm{GS}_C(A).$$

Note que

$$\mathrm{posto}(A) = \mathrm{posto}(\mathrm{GS}_C(A))$$

e que

$$\mathrm{GS}_C(A) = \mathrm{GS}_L(A^T)^T.$$

Com isso, representa-se o processo de fatoração du, para uma matriz A, por

$$(d, u) = \Gamma_{du}(A),$$

em que

$$u = \mathrm{GS}_L(A), \tag{4.6}$$

e d é obtida por (4.5). Existe ainda uma outra maneira de expressar d; desde que

$$uu^T = \begin{bmatrix} \langle u_1, u_1 \rangle & 0 & \cdots & 0 \\ 0 & \langle u_2, u_2 \rangle & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \langle u_p, u_p \rangle \end{bmatrix},$$

definindo

$$N_u \triangleq \begin{bmatrix} \langle u_1, u_1 \rangle^{-1} & 0 & \cdots & 0 \\ 0 & \langle u_2, u_2 \rangle^{-1} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \langle u_p, u_p \rangle^{-1} \end{bmatrix},$$

chega-se a

$$d = Au^T N_u, \tag{4.7}$$

já que

$$uu^T N_u = I_p.$$

Dessa forma, o processo de fatoração du, para uma matriz A, representado por $(d, u) = \Gamma_{du}(A)$, é obtido simplesmente através das equações (4.6) e (4.7) ou (4.5).

4.3.2 Decomposição du para um conjunto de matrizes

Nesse caso, necessita-se de uma decomposição para um conjunto de matrizes A_i, $i = 1, \ldots, L$, em que qualquer matriz A_i pode ser escrita como

$$A_i = d_i u,$$

e a matriz u possui linhas ortogonais (sem linhas nulas). A matriz u é a mesma para todas as decomposições. Uma decomposição desse tipo pode ser feita escolhendo

$$u = \mathrm{GS}_L \left(\begin{bmatrix} A_1 \\ \vdots \\ A_L \end{bmatrix} \right),$$

que é o processo de ortogonalização de Gram-Schmidt (sem normalização) aplicado às matrizes A_i concatenadas verticalmente. Então, as matrizes d_i podem ser computadas por

$$d_i = A_i u^T N_u. \tag{4.8}$$

Esse processo é representado por

$$(d_i, u) = \Gamma_{du}(A_i),$$

em que A_i e d_i representam um conjunto de matrizes.

4.3.3 Decomposição *udu* para um conjunto de matrizes

Essa decomposição também é chamada de *decomposição em matrizes postunitárias ortogonais* e é obtida para um conjunto de matrizes A_i, $i = 1, \ldots, L$. A decomposição *udu* é dada por

$$A_i = U_c D_i U_l, \tag{4.9}$$

em que U_c é uma matriz de colunas ortogonais e U_l é uma matriz de linhas ortogonais. Esse processo que gera, a partir das matrizes A_i, as matrizes D_i, U_c e U_l é denotado por

$$(U_c, D_i, U_l) = \Gamma_{udu}(A_i).$$

O processo de fatoração $\Gamma_{udu}(A_i)$ consiste em computar d_i e U_l através de

$$(d_i, U_l) = \Gamma_{du}(A_i),$$

e então, computar D_i e U_c através de

$$(D_i^T, U_c^T) = \Gamma_{du}(d_i^T),$$

isto é, aplicar o processo Γ_{du} sobre as colunas de d_i. O processo de fatoração Γ_{udu} pode ser visto como uma transformada matricial ou simplesmente como uma decomposição em matrizes postunitárias ortogonais. Algumas propriedades podem ser verificadas a partir de (4.9), e. g., a linearidade

$$b_i A_i + b_j A_j = U_c (b_i D_i + b_j D_j) U_l.$$

Se

$$U_c = \left[\begin{array}{cccc} u_{c_1} & u_{c_2} & \ldots & u_{c_r} \end{array} \right],$$

e a componente da linha m e coluna n da matriz D_i, $D_{m,n,i} = 1$, é a única componente não nula, então a matriz A_i é dada por

$$A_i = u_{c_m} u_{l_n}^T,$$

e é uma matriz postunitária. Utilizando a linearidade, pode-se escrever que

$$A_i = \sum_{m=1}^{r} \sum_{n=1}^{r} D_{m,n,i} u_{c_m} u_{l_n}^T,$$

o que é uma decomposição de A_i em matrizes postunitárias ortogonais, visto que $\langle u_{c_m} u_{l_n}^T, u_{c_i} u_{l_j}^T \rangle$ é diferente de zero apenas se $m = i$ e $n = j$, isto é, se as matrizes são as mesmas.

Uma outra propriedade é que

$$\text{posto}(A_i) = \text{posto}(D_i),$$

o que significa que A_i é postunitária se, e somente se, D_i é postunitária.

A vantagem de se utilizar o processo de fatoração Γ_{udu} é que o problema passa a ser a fatoração das matrizes D_i, no lugar das matrizes A_i, em matrizes postunitárias, que geralmente é um problema mais simples. Esse processo de fatoração é em si uma decomposição em matrizes postunitárias ortogonais, entretanto, a solução ótima para o processo nem sempre é composta de matrizes postunitárias ortogonais. Para ilustrar este fato considere o exemplo a seguir.

Exemplo 4.3

Considere o problema de decompor as matrizes

$$A_1 = \left[\begin{array}{cc} 1 & 0 \\ 0 & 1 \end{array} \right]$$

e

$$A_2 = \left[\begin{array}{cc} 0 & 1 \\ -1 & 0 \end{array} \right]$$

em matrizes postunitárias. Neste caso, $U_c = U_l = I_2$, as matrizes podem ser decompostas em

$$A_1 = \left[\begin{array}{c} 1 \\ 0 \end{array} \right] \left[\begin{array}{cc} 1 & 0 \end{array} \right] + \left[\begin{array}{c} 0 \\ 1 \end{array} \right] \left[\begin{array}{cc} 0 & 1 \end{array} \right],$$

$$A_2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \begin{bmatrix} 0 & 1 \end{bmatrix} - \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \end{bmatrix},$$

que é uma solução formada por 4 matrizes postunitárias ortogonais. Entretanto, considere as matrizes

$$K_1 = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix},$$

$$K_2 = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$$

e

$$A_2 + K_1 - K_2 = K_3 = \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}.$$

A matriz K_3 é postunitária. Sendo assim, pode-se escrever

$$\begin{bmatrix} A_1 \\ A_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 \\ -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} K_1 \\ K_2 \\ K_3 \end{bmatrix},$$

que é uma decomposição melhor que a decomposição ortogonal, pois gera as matrizes com apenas 3 matrizes postunitárias. •

4.4 Transformada Rápida de Fourier Otimizada

Para se obter um algoritmo ótimo para computar uma transformada discreta de Fourier de comprimento N, utilizam-se as bases ciclotômicas cos-CB, para N múltiplo de 4, ou sen/cos-CB, para outros valores de N [22]. Seja W a matriz de transformação da DFT de N pontos. Então, para N múltiplo de 4 tem-se

$$W = (W_1 - jW_2) + \sum_{i=3}^{\phi(N)} \gamma_i W_i, \tag{4.10}$$

ou, para valores de N não múltiplos de 4,

$$W = W_1 + \sum_{i=2}^{\phi(N)} \gamma_i W_i. \tag{4.11}$$

O Teorema da complexidade multiplicativa afirma que o algoritmo ótimo consiste em fatorar as matrizes W_i, referentes aos $\gamma_i \notin \mathbb{Q}$ do processo Γ_{udu}, da melhor forma possível,

isto é, obter as matrizes U_c, D_is e U_l, e encontrar a decomposição ótima das matrizes D_is em matrizes postunitárias, na forma

$$\begin{bmatrix} D_2 \\ D_3 \\ \vdots \\ D_L \end{bmatrix} = c \begin{bmatrix} K_1 \\ K_2 \\ \vdots \\ K_{M_r} \end{bmatrix},$$

em que a matriz c, $L \times M_r$, é formada pelos elementos c_{ij} de (3.19). Note que, dependendo da base ciclotômica utilizada, pode-se começar com D_3 na equação anterior. Pela linearidade,

$$\begin{bmatrix} W_2 \\ W_3 \\ \vdots \\ W_L \end{bmatrix} = c \begin{bmatrix} U_c K_1 U_l \\ U_c K_2 U_l \\ \vdots \\ U_c K_{M_r} U_l \end{bmatrix}.$$

A partir de (3.20), pode se escrever

$$\begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_{M_r} \end{bmatrix} = c^T \begin{bmatrix} \gamma_2 \\ \gamma_3 \\ \vdots \\ \gamma_{\phi(N)} \end{bmatrix},$$

e então, finalmente,

$$W = W_0 + \sum_{i=1}^{M_r} \beta_i U_c K_i U_l, \tag{4.12}$$

em que $W_0 = W_1$ ou $W_0 = W_1 - jW_2$ dependendo da base ciclotômica utilizada.

Utilizando o Teorema 3.1, pode-se colocar W na forma $W_0 + CBA$, em que C e A são matrizes de elementos racionais e B uma matriz diagonal. Dessa forma, se

$$C_i A_i^T = U_c K_i U_l,$$

então

$$C = \begin{bmatrix} C_1 & \cdots & C_{M_r} \end{bmatrix},$$

$$B = \begin{bmatrix} \beta_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \beta_{M_r} \end{bmatrix}$$

e

$$A = \begin{bmatrix} A_1^T \\ \vdots \\ A_{M_r}^T \end{bmatrix},$$

o que resulta em

$$W = W_0 + CBA, \tag{4.13}$$

e o algoritmo é implementado por

$$V = W_0 v + CBAv. \tag{4.14}$$

Desde que $W_0 v$ não contém multiplicações, a complexidade multiplicativa do algoritmo está em $CBAv$, que contém M_r multiplicações. A matriz W_0 pode ser ainda anexada às matrizes CBA, como no algoritmo de Winograd [7]. Para isso, se

$$W_0 = du,$$

é uma decomposição du de W_0, em que

$$u = \begin{bmatrix} u_1^T \\ \vdots \\ u_r^T \end{bmatrix},$$

e

$$d = \begin{bmatrix} d_1 & \cdots & d_r \end{bmatrix},$$

com r o posto de W_0, então

$$W = \begin{bmatrix} d_1 & \cdots & d_r & C_1 & \cdots & C_{M_r} \end{bmatrix} \begin{bmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ & & & \beta_1 & & \\ & & & & \ddots & \\ & & & & & \beta_{M_r} \end{bmatrix} \begin{bmatrix} u_1^T \\ \vdots \\ u_r^T \\ A_1^T \\ \vdots \\ A_{M_r}^T \end{bmatrix}. \tag{4.15}$$

Exemplo 4.4

Para obter uma transformada ótima para $N = 5$, utiliza-se a base ciclotômica sen/cos-CB para escrever a matriz de transformação como

$$W = A_1 + \gamma_2 A_2 + \gamma_3 A_3 + \gamma_4 A_4,$$

em que, com $t = 2\pi/5$,

$$\begin{bmatrix} \gamma_2 \\ \gamma_3 \\ \gamma_4 \end{bmatrix} = \begin{bmatrix} -j\text{sen}(t) \\ \cos(t) \\ -j\text{sen}(2t) \end{bmatrix},$$

e

$$A_1 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & -\frac{1}{2} & -\frac{1}{2} & 0 \\ 1 & -\frac{1}{2} & 0 & 0 & -\frac{1}{2} \\ 1 & -\frac{1}{2} & 0 & 0 & -\frac{1}{2} \\ 1 & 0 & -\frac{1}{2} & -\frac{1}{2} & 0 \end{bmatrix},$$

$$A_2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 1 & -1 & 0 \\ 0 & -1 & 0 & 0 & 1 \end{bmatrix},$$

$$A_3 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & -1 & 1 \\ 0 & -1 & 1 & 1 & -1 \\ 0 & -1 & 1 & 1 & -1 \\ 0 & 1 & -1 & -1 & 1 \end{bmatrix}$$

e

$$A_4 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 \\ 0 & 1 & 0 & 0 & -1 \\ 0 & -1 & 0 & 0 & 1 \\ 0 & 0 & -1 & 1 & 0 \end{bmatrix}.$$

Em seguida, utiliza-se o processo Γ_{udu} sobre as matrizes A_2, A_3 e A_4, pois é necessário a decomposição das mesmas em matrizes postunitárias. Com isso, tem-se

$$U_c = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & -1 & -1 \\ 0 & 1 & -1 \\ -1 & 0 & 1 \end{bmatrix},$$

$$U_l = \begin{bmatrix} 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & 1 & -1 & 0 \\ 0 & 1 & -1 & -1 & 1 \end{bmatrix},$$

$$D_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad D_3 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

e

$$D_4 = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

Observa-se que as matrizes D_i são mais simples para a fatoração. Note ainda que D_2 e D_4 podem ser decompostas como no Exemplo 4.3, logo

$$\begin{bmatrix} D_2 \\ D_3 \\ D_4 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} K_1 \\ K_2 \\ K_3 \\ K_4 \end{bmatrix},$$

em que

$$K_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix},$$

$$K_2 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix},$$

$$K_3 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

e

$$K_4 = \begin{bmatrix} 1 & 1 & 0 \\ -1 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

Sendo

$$\begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -j\text{sen}(t) \\ \cos(t) \\ -j\text{sen}(2t) \end{bmatrix},$$

pode-se escrever W como

$$W = A_1 + \sum_{i=1}^{4} \beta_i U_c K_i U_l$$

ou

$$W = A_1 + CBA,$$

em que

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

e

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 1 & -1 & -1 & 1 \\ 0 & 1 & 1 & -1 & -1 \end{bmatrix}.$$

O algoritmo é obtido por

$$V = A_1 v + CBAv.$$

$$V = A_1 v + C \begin{bmatrix} -j(v_1 - v_4)[\text{sen}(t) - \text{sen}(2t)] \\ -j(-v_2 + v_3)[\text{sen}(t) + \text{sen}(2t)] \\ (v_1 - v_2 - v_3 + v_4)\cos(t) \\ -j(v_1 + v_2 - v_3 - v_4)\text{sen}(2t) \end{bmatrix}.$$

Escrevendo a equação para todas as componentes isoladamente, tem-se

$$V_0 = \sum_{n=0}^{4} v_n,$$

$$\begin{aligned} V_1 \quad = \quad & (v_0 - \frac{v_2}{2} - \frac{v_3}{2}) \\ & -j(v_1 - v_4)[\text{sen}(t) - \text{sen}(2t)] \\ & +(v_1 - v_2 - v_3 + v_4)\cos(t) \\ & -j(v_1 + v_2 - v_3 - v_4)\text{sen}(2t), \end{aligned}$$

$$
\begin{aligned}
V_2 &= (v_0 - \frac{v_1}{2} - \frac{v_4}{2}) \\
&\quad -j(-v_2 + v_3)[\text{sen}(t) + \text{sen}(2t)] \\
&\quad -(v_1 - v_2 - v_3 + v_4)\cos(t) \\
&\quad -j(v_1 + v_2 - v_3 - v_4)\text{sen}(2t),
\end{aligned}
$$

$$
\begin{aligned}
V_3 &= (v_0 - \frac{v_1}{2} - \frac{v_4}{2}) \\
&\quad +j(-v_2 + v_3)[\text{sen}(t) + \text{sen}(2t)] \\
&\quad -(v_1 - v_2 - v_3 + v_4)\cos(t) \\
&\quad +j(v_1 + v_2 - v_3 - v_4)\text{sen}(2t),
\end{aligned}
$$

$$
\begin{aligned}
V_4 &= (v_0 - \frac{v_2}{2} - \frac{v_3}{2}) \\
&\quad +j(v_1 - v_4)[\text{sen}(t) - \text{sen}(2t)] \\
&\quad +(v_1 - v_2 - v_3 + v_4)\cos(t) \\
&\quad +j(v_1 + v_2 - v_3 - v_4)\text{sen}(2t).
\end{aligned}
$$

O algoritmo também pode ser deixado na forma CBA; para isso basta inserir a matriz A_1 dentro das matrizes CBA, como mostrado em (4.15). Com a decomposição *du* de A_1, pode-se escrever

$$
A_1 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & -\frac{1}{2} \\ 1 & -\frac{1}{2} & 0 \\ 1 & -\frac{1}{2} & 0 \\ 1 & 0 & -\frac{1}{2} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix},
$$

e a matriz W pode ser escrita na forma

$$
W = C'B'A',
$$

em que

$$
C' = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & -\frac{1}{2} & 1 & 0 & 1 & 1 \\ 1 & -\frac{1}{2} & 0 & 0 & 1 & -1 & 1 \\ 1 & -\frac{1}{2} & 0 & 0 & -1 & -1 & -1 \\ 1 & 0 & -\frac{1}{2} & -1 & 0 & 1 & -1 \end{bmatrix},
$$

$$B' = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \beta_1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \beta_2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \beta_3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \beta_4 \end{bmatrix}$$

e

$$A' = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 1 & -1 & -1 & 1 \\ 0 & 1 & 1 & -1 & -1 \end{bmatrix}.$$

•

De forma semelhante, obtêm-se os algoritmos ótimos para os comprimentos $N = 3, 4, 5, 6, 7, 8, 9, 10, 12, 16, 24$. Os algoritmos para $N = 5, 7, 9, 10$ superam os melhores algoritmos atuais em termos de complexidade multiplicativa. A Tabela 4.1 apresenta a complexidade de alguns algoritmos bem conhecidos na literatura, para comparações.

Tabela 4.1: Complexidade multiplicativa real da: OFFT - A transformada de Fourier otimizada; HMMC - Complexidade multiplicativa mínima (fórmula de Heideman); CT/GT - Combinação da FFT de Cooley-Tukey e Good-Thomas otimizada; SW-FFT - A FFT de Winograd

N	(OFFT)	(HMMC)	(CT/GT)	(SW-FFT)
3	1	1	-	2
4	0	0	0	0
5	4	4	-	5
6	2	2	16	-
7	7	7	-	8
8	2	2	4	2
9	8	8	60	10
10	8	8	64	-
12	4	4	32	-
16	10	10	20	10
24	12	12	108	-

4.5 Computando uma Componente da DFT de Forma Otimizada

Para derivar um algoritmo otimizado para uma única componente V_k da DFT, precisa-se computar

$$V_k = \sum_{n=0}^{N-1} v_n (W_N^k)^n.$$

Desde que W_N^k tem ordem $L = N/\gcd(N,k)$, existe uma decomposição em bases ciclotômicas para $W_N^k = \alpha$ e a componente V_k da DFT pode ser escrita como

$$V_k = \sum_{n=0}^{N-1} v_n \alpha^n.$$

Fazendo a mudança de variável $n = l + mL$, com $l = 0, \ldots, L-1$ e $m = 0, \ldots, N/L-1$, tem-se

$$V_k = \sum_{l=0}^{L-1} \sum_{m=0}^{N/L-1} v_{l+mL} \alpha^{l+mL} = \sum_{l=0}^{L-1} (\sum_{m=0}^{N/L-1} v_{l+mL}) \alpha^l. \tag{4.16}$$

Definindo a matriz coluna v_s, que pode ser vista como o vetor de entrada sobreposto para que seu comprimento seja L, por

$$v_s \triangleq \begin{bmatrix} \sum_{m=0}^{N/L-1} v_{0+mL} \\ \vdots \\ \sum_{m=0}^{N/L-1} v_{L-1+mL} \end{bmatrix},$$

a Equação (4.16) pode ser expressa pelo produto escalar

$$V_k = v_s^T \begin{bmatrix} \alpha^0 \\ \vdots \\ \alpha^{L-1} \end{bmatrix}.$$

Usando a decomposição nas bases ciclotômicas cos-BC ou sen/cos-BC (4.3), pode-se escrever

$$V_k = (v_s^T R) \begin{bmatrix} \gamma_1 \\ \vdots \\ \gamma_{\phi(L)} \end{bmatrix}. \tag{4.17}$$

As multiplicações contidas na operação $({v_s}^T R)$ são triviais. Se $4|N$ a base ciclotômica cos-CB é usada e a complexidade multiplicativa é $(\phi(L) - 2)$; caso contrário, a base sen/cos-BC é utilizada e a complexidade multiplicativa é $(\phi(L) - 1)$. Comparando a complexidade

desse algoritmo com a complexidade mínima para a computação de uma única componente, estabelecida por Heideman [8], dada por

$$\mu(\mathrm{DFT}(N,k)) = \phi\left(\frac{N}{\mathrm{MDC}(N,k)}\right) - \phi\left(\mathrm{MDC}\left(\frac{N}{\gcd(N,k)},4\right)\right),$$

observa-se que o algoritmo é ótimo, em relação à complexidade multiplicativa, desde que $\phi(\mathrm{MDC}(L,4))$ é igual a 2, se 4 divide L, ou igual a 1, caso contrário.

Exemplo 4.5

Obter o algoritmo otimizado para computar V_1 de uma DFT de 8 pontos. Nesse caso, utilizando cos-BC,

$$\begin{bmatrix} \alpha^0 \\ \alpha^1 \\ \alpha^2 \\ \alpha^3 \\ \alpha^4 \\ \alpha^5 \\ \alpha^6 \\ \alpha^7 \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 1 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & -1 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix}}_{R} \begin{bmatrix} 1 \\ -j \\ \frac{\sqrt{2}}{2} \\ -j\frac{\sqrt{2}}{2} \end{bmatrix},$$

e, por (4.17),

$$V_1 = \mathbf{v}^T R \begin{bmatrix} 1 \\ -j \\ \frac{\sqrt{2}}{2} \\ -j\frac{\sqrt{2}}{2} \end{bmatrix}.$$

Portanto,

$$V_1 = (v_0 - v_4) - j(v_2 - v_6) + \frac{\sqrt{2}}{2}(v_1 - v_3 - v_5 + v_7) - j\frac{\sqrt{2}}{2}(v_1 + v_3 - v_5 - v_7),$$

é o algoritmo ótimo para computar V_1, com complexidade multiplicativa igual a dois. •

4.6 O Algoritmo JCO e JCO-Goertzel

O algoritmo JCO[1] foi desenvolvido pelo grupo de processamento de sinais da UFPE e foi apresentado em 2009 no ISCTA [23], aproximadamente 50 anos após a publicação do algoritmo

[1] O nome do algoritmo (JCO) surgiu a partir dos nomes de seus autores do algoritmo: G. Jerônimo da Silva Jr., R. M. Campello de Souza e H. M. de Oliveira.

de Goertel [16] (Seção 1.4 do Capítulo 1). Este algoritmo é uma alternativa ao algoritmo de Goertzel, no sentido de que reduz a complexidade multiplicativa ao custo de um aumento no número de registradores.

O algoritmo JCO consiste em utilizar outro polinômio para a divisão polinomial no lugar de $p_k(x)$. Seja L a ordem de W_N^k, dada por

$$L \triangleq \text{ord}(W_N^k) = \frac{N}{\text{MDC}(N,k)}, \tag{4.18}$$

e $\Phi_L(x)$ o polinômio ciclotômico de grau $l = \phi(L)$, em que $\phi(.)$ é a função aritmética de Euler [7, 14, 19]. O polinômio $\Phi_L(x)$, por definição, tem um zero em W_N^k, assim, escrevendo o polinômio de entrada na forma

$$v(x) = \Phi_L(x)Q(x) + R(x),$$

chega-se a

$$V_k = R(W_N^k). \tag{4.19}$$

A vantagem desse algoritmo, em relação ao algoritmo de Goertzel, em relação ao número de multiplicações, reside no fato de que polinômios ciclotômicos possuem coeficientes inteiros [7], e assim não são necessárias multiplicações para computar os coeficientes de $R(x)$. A complexidade multiplicativa do algoritmo está na avaliação de V_k dada pela expressão (4.19). Desde que $R(x)$ tem grau $l-1$, são necessárias $l-1$ multiplicações complexas para se computar V_k, isto é,

$$M_c(N,k) = \phi\left(\frac{N}{\text{MDC}(N,k)}\right) - 1. \tag{4.20}$$

Nesse caso também é possível uma implementação ordem direta. Considerando a mesma sequência $(\hat{v}_n)$ de (1.30), pode-se escrever

$$\hat{v}(x) = \Phi_L(x)\hat{Q}(x) + \hat{R}(x).$$

Utilizando a equação (1.31) e considerando que $\hat{V}_k = \hat{R}(W_N^k)$, tem-se

$$V_k = W_N^{-k}\hat{R}(W_N^{-k}). \tag{4.21}$$

O algoritmo JCO tem a vantagem de fazer com que a complexidade multiplicativa seja função de $\phi(L)$ e não de N, assim é possível obter complexidades multiplicativas muito menores que a do algoritmo de Goertzel para aplicações específicas. Um exemplo de aplicação, na detecção de tons DTMF, pode ser encontrado em [24]. Além disso, o algoritmo de Goertzel

pode ser utilizado em conjunto com o algoritmo JCO para computar $\hat{R}(W_N^{-k})$, o que torna as multiplicações complexas em multiplicações reais, mas necessita de $N+\phi(L)$ passos para computar a componente desejada. Essa combinação é conhecida como Algoritmo JCO-Goertzel [23]. Nesse caso, considerando que

$$\hat{R}(W_N^{-k}) = \hat{r}_0 + \hat{r}_1 W_N^{-k}$$

e substituindo em (4.21), resulta em

$$V_k = \left[\hat{r}_0 + 2\cos\left(\frac{2\pi k}{N}\right)\hat{r}_1\right] W_N^{-k} - \hat{r}_1, \tag{4.22}$$

em que $\hat{r}_0$ e $\hat{r}_1$ são obtidos aplicando-se o algoritmo de Goertzel ordem inversa sobre $\hat{R}(x)$. A complexidade multiplicativa do JCO-Goertzel depende das variáveis N e k e é sempre menor que a complexidade computacional do algoritmo de Goertzel. A complexidade multiplicativa do JCO-Goertzel ordem inversa é dada por

$$M_r(N,k) = \phi\left(\frac{N}{\mathrm{MDC}(N,k)}\right), \tag{4.23}$$

enquanto a complexidade multiplicativa do JCO-Goertzel ordem direta é dada por

$$M_r(N,k) = \phi\left(\frac{N}{\mathrm{MDC}(N,k)}\right) + 1. \tag{4.24}$$

A seguir, é apresentado um resumo do algoritmo JCO-Goertzel ordem direta. Primeiro computa-se $\hat{R}(x)$ por

$$\hat{R}(x) = \hat{v}(x) \pmod{\Phi_L(x)},$$

operação livre de multiplicações. Em seguida, computa-se $\hat{r}(x)$ com o algoritmo de Goertzel ordem inversa sobre $\hat{R}(x)$, isto é

$$\hat{r}(x) = \hat{R}(x) \pmod{p_k(x)},$$

operação que possui $\phi(L)-2$ multiplicações reais. Finalmente, utiliza-se (4.22) para computar V_k, operação que custa mais uma multiplicação real e uma multiplicação de um real por complexo, totalizando $\phi(L)+1$ multiplicações reais, conforme mostrado na Tabela 4.2. É interessante mencionar que a complexidade mínima para a computação de uma componente, apresentada por Heideman em [8], é dada por $\phi(L)-2$, para L múltiplo de 4, e $\phi(L)-1$, caso contrário, isto é, o algoritmo JCO-Goertzel é um algoritmo quase ótimo (por apenas duas multiplicações não se iguala ao algoritmo apresentado na Seção 4.5).

Tabela 4.2: Complexidade multiplicativa dos Algoritmos de Goertzel, JCO e JCO-Goertzel em número de multiplicações reais, considerando entradas reais com implementação na ordem inversa.

N	k	Goertzel	JCO	JCO-Goertzel
8	1	8	6	4
	2	8	2	2
12	1	12	6	4
	2	12	2	2
16	1	16	14	8
	2	16	6	4
24	1	24	14	8
	2	24	6	4
32	1	32	30	16
	2	32	14	8

O próximo capítulo desta tese apresenta um método para minimizar a complexidade aritmética aditiva. A partir dessa metodologia, é possível obter um número reduzido de adições para um número específico de multiplicação de uma transformada linear qualquer.

capítulo 5

Teoria da Complexidade Aditiva para Transformadas

Em 1978, S. Winograd apresentou uma nova FFT baseada num algoritmo de convolução cíclica [11] e, posteriormente, publicou um livro sobre complexidade aritmética [3]. Esses trabalhos estabeleceram uma maneira de contar o número de multiplicações e adições necessárias para se implementar um determinado algoritmo; grande parte dos esforços, entretanto, se concentrou em minimizar a complexidade multiplicativa. Teoremas sobre a complexidade multiplicativa foram propostos enquanto a complexidade aditiva era obtida a partir do algoritmo final de forma heurística [7, 8, 22, 23].

A motivação que levou à investigação relatada neste capítulo foi a perspectiva de se minimizar a complexidade aditiva de um determinado algoritmo que requer um número estabelecido de multiplicações. Para isso, uma nova forma de se contabilizar a complexidade é introduzida, a qual considera multiplicações triviais independentemente das adições.

5.1 Introdução

A complexidade aditiva de um algoritmo ou transformada é o número de adições utilizadas pelo mesmo para se obter o resultado desejado. Por exemplo, considere um algoritmo somador dado por

$$X = x_0 + x_1 + x_2 + x_3,$$

em que $x_n \in \mathbb{R}$, $n = 0, 1, 2, 3$, são variáveis de entrada. Em forma matricial

$$X = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}.$$

Neste caso, observa-se que a computação pode ser feita via somador de duas entradas de diversas maneiras, como por exemplo

$$X = \{[(x_0 + x_1) + x_2] + x_3\}, \tag{5.1}$$

ou

$$X = [(x_0 + x_1) + (x_2 + x_3)]. \tag{5.2}$$

Observa-se que, não importando a ordem ou arranjo das adições, o número de adições para se computar X não muda e é dado por

$$A_r = \text{Números de elementos} - 1. \tag{5.3}$$

Neste caso, $A_r = 3$ é a complexidade aditiva, entretanto, existe uma diferença sutil entre as duas implementações. A implementação decorrente de (5.1) apresenta três passos, mas possibilita a implementação com apenas um somador, enquanto a implementação decorrente de (5.2) possibilita a computação em dois passos, considerando que $(x_0 + x_1)$ e $(x_2 + x_3)$ podem ser computados em paralelo, contudo, isto demanda dois somadores no processador. Considere agora o seguinte algoritmo com somadores

$$y_0 = x_0 + x_1 + x_2 + x_3 \tag{5.4}$$

$$y_1 = x_0 + x_3, \tag{5.5}$$

ou, em forma matricial,

$$\begin{bmatrix} y_0 \\ y_1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}.$$

Neste caso, utilizando a Equação (5.3) para cada variável, obtém-se três adições devido a y_0 e uma adição para y_1, totalizando $A_r = 4$. Entretanto, o algoritmo pode ser implementado

por

$$a_1 = x_0 + x_3 \tag{5.6}$$

$$a_2 = x_1 + x_2 \tag{5.7}$$

$$y_0 = a_1 + a_2 \tag{5.8}$$

$$y_1 = a_1, \tag{5.9}$$

com $A_r = 3$. Logo, observa-se que existem maneiras eficientes de se implementar algoritmos de forma a minimizar a complexidade aditiva.

5.2 Minimizando a Complexidade Aditiva de Transformadas Racionais

Uma transformada pode ser representada de forma matricial por

$$V = \Psi v,$$

em que

$$V \triangleq \begin{bmatrix} V_0 \\ V_1 \\ \vdots \\ V_{M-1} \end{bmatrix},$$

$$\Psi \triangleq \begin{bmatrix} \varphi_{0,0} & \varphi_{0,1} & \cdots & \varphi_{0,N-1} \\ \varphi_{1,0} & \varphi_{1,1} & \cdots & \varphi_{1,N-1} \\ \vdots & \vdots & \ddots & \vdots \\ \varphi_{M-1,0} & \varphi_{M-1,1} & \cdots & \varphi_{M-1,N-1} \end{bmatrix}$$

e

$$v \triangleq \begin{bmatrix} v_0 \\ v_1 \\ \vdots \\ v_{N-1} \end{bmatrix}.$$

Na condição particular em que $\varphi_{i,j} \in \mathbb{Q}$, isto é, a matriz de transformação Ψ contém apenas elementos racionais, a transformada não contém multiplicações. Neste caso, tem-se apenas multiplicações triviais e adições. Sobre essa condição é possível aplicar todas as relações de complexidade aditiva apresentas no Capítulo 2.

Para implementar transformações racionais de maneira eficiente, no que diz respeito à complexidade aditiva, vamos definir um tipo simples de matriz, tal que estruturas e transformadas mais complexas podem ser obtidas por cascateamento das mesmas.

Definição 5.1 *Define-se como* matriz bielementar *uma matriz, com elementos racionais, em que cada uma das suas linhas tem, no máximo, dois elementos não nulos; além disso, as linhas com dois elementos não nulos, consideradas como vetores no R^N, devem apresentar direções distintas.*

Toda matriz racional Ψ pode ser fatorada na forma

$$\Psi = \Psi_1 \Phi,$$

em que a matriz Φ é bielementar. Essa fatoração não é única. Uma maneira possível de se obter uma fatoração dessa forma é escolher pares de colunas de Ψ e, para cada par, escolher vetores em direções distintas que completem todas as direções contidas no par de colunas de Ψ. Com este procedimento, a matriz Φ é bielementar. Escolhendo-se a matriz bielementar Φ, a matriz Ψ_1 pode ser obtida, desde que cada linha i de Ψ é dada por

$$v_i^T = \sum_{j=1}^{m} c_{i,j} u_j^T,$$

em que $c_{i,j}$ são os coeficientes da linha i e coluna j de Ψ_1 e u_j^T é a linha j de Φ. Considerando essa decomposição das linhas de Ψ num determinado par de colunas específico, tem-se que

$$\hat{v}_i^T = \sum_{j=j_1}^{j_2} c_{i,j} u_j^T,$$

em que $\hat{v}_i^T$ é o vetor v_i^T com zeros nas posições não referentes ao par de colunas específico. Desde que todas as direções estão contidas em Φ, o elemento $c_{i,j}$ pode ser obtido por

$$c_{i,j} = \begin{cases} q, & \text{se } \hat{v}_i^T = q u_j^T, \\ 0, & \text{caso contrário,} \end{cases} \tag{5.10}$$

para $j = j_1, \ldots, j_2$. Fazendo isso para todos os pares, é possível obter a matriz Ψ_1.

Exemplo 5.1

Fatorar a matriz de Hadamard de ordem quatro na forma $\Psi_1\Phi$. Essa matriz é dada por

$$H_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}.$$

Escolhendo como pares de colunas, $(1,2)$ e $(3,4)$, pode-se escolher a matriz Φ por

$$\Phi = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix},$$

e com isso, a matriz Ψ_1 é computada por (5.10) ou, por simples observação,

$$\Psi_1 = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}.$$

Observa-se que a implementação da transformação $V = H_4 v$, pela forma direta (a qual a complexidade aditiva é dada pela Equação (2.4)), tem complexidade aditiva dada por

$$\mathrm{C_A}(H_4) = 12,$$

enquanto que esta implementação na forma $V = \Psi_1\Phi v$ apresenta

$$\mathrm{C_A}(\Psi_1\Phi) = 8,$$

sendo, portanto, mais eficiente que a primeira.

Outra vantagem de se utilizar a fatoração em matrizes bielementares é que a implementação, o número de passos e a quantidade mínima de somadores podem ser derivadas a partir da mesma. O número de passos é dado pelo número de matrizes da fatoração, neste caso dois. A quantidade mínima de somadores é dada pelo número de adições da matriz da fatoração que apresenta maior número de adições, neste caso quatro. A implementação pode ser facilmente obtida das matrizes Φ e Ψ_1; a Figura 5.1 mostra a implementação da transformada de Hadamard de ordem 4 utilizando a fatoração em matrizes bielementares.

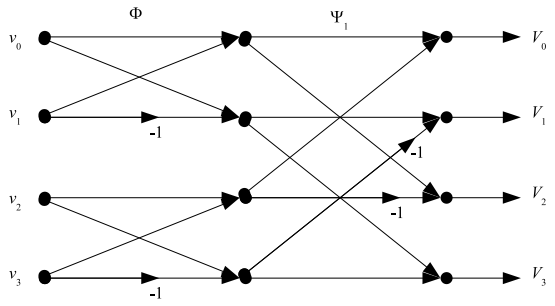


Figura 5.1: Implementação da transformada de Hadamard de ordem 4 utilizando a fatoração em matrizes bielementares

•

O fato principal é que as matrizes bielementares definidas anteriormente aparentemente não podem ser minimizadas. Esse fato é caracterizado pela conjectura a seguir.

Conjectura 5.1 *A complexidade aditiva de uma matriz bielementar é mínima, isto é, se* Φ *é bielementar e* $\Phi = \Phi_2\Phi_1$*, então*

$$C_A(\Psi) \leq C_A(\Phi_2) + C_A(\Phi_1).$$

Isto sugere que, se uma transformada qualquer for fatorada em matrizes bielementares, a implementação através da concatenação das matrizes bielementares apresenta uma complexidade aditiva menor. Entretanto, como mencionado anteriormente, a fatoração em matrizes bielementares não é única, o que faz necessário um mecanismo de controle para que a fatoração em matrizes bielementares resulte na melhor implementação. Tal controle é feito pela proposição a seguir.

Proposição 5.1 *Um método para minimizar a complexidade aditiva da transformação racional* $V = \Psi v$*, fatorando* Ψ *em matrizes bielementares:*

- *Escolher a fatoração* $\Psi = \Psi_2\Phi_1$ *que maximize*

$$G_1 = C_A(\Psi) - C_A(\Psi_2\Phi_1), \tag{5.11}$$

  para $G_1 > 0$*;*

- *Repetir o item anterior para fatorar* $\Psi_i = \Psi_{i+1}\Phi_i$ *até que o máximo valor de* G_i*, obtido por*

$$G_i = C_A(\Psi_i) - C_A(\Psi_{i+1}\Phi_i), \tag{5.12}$$

  seja igual a zero;

- *Implementar* Ψ *através de*

$$\Psi = \Psi_m \Phi_{m-1} \ldots \Phi_2 \Phi_1. \tag{5.13}$$

Sobre esta fatoração, se não existe $G_1 > 0$ que satisfaça o primeiro passo, diz-se então que Ψ é irredutível. A variável G_i pode ser vista como o ganho em complexidade aditiva (o número de adições economizadas) por se implementar Ψ_i através de $\Psi_{i+1}\Phi_i$. Se Ψ_m é bielementar, pode-se dizer que o número de passos do algoritmo, n_P, é dado por

$$n_P = m, \tag{5.14}$$

e o número mínimo de somadores necessários para implementar o algoritmo em m passos, n_S, é dado por

$$n_S = \max_i(\mathrm{C_A}(\Phi_i)). \tag{5.15}$$

Teorema 5.1 *Seja* Ψ *uma matriz racional que foi fatorada em matrizes bielementares, pela Equação* (5.13)*, utilizando a Proposição 5.1. Então, a complexidade aditiva da fatoração é dada por*

$$C_A(\Psi_m \Phi_{m-1} \ldots \Phi_2 \Phi_1) = C_A(\Psi) - \sum_{i=1}^{m-1} G_i.$$

Demonstração: Pode-se escrever, utilizando (2.3) em (5.12),

$$\mathrm{C_A}(\Psi_m \Phi_{m-1}) = \mathrm{C_A}(\Psi_{m-1}) - G_{m-1} \tag{5.16}$$

$$\mathrm{C_A}(\Phi_{m-2}) = \mathrm{C_A}(\Psi_{m-2}) - \mathrm{C_A}(\Psi_{m-1}) - G_{m-2} \tag{5.17}$$

$$\mathrm{C_A}(\Phi_{m-3}) = \mathrm{C_A}(\Psi_{m-3}) - \mathrm{C_A}(\Psi_{m-2}) - G_{m-3} \tag{5.18}$$

$$\vdots \qquad\qquad \vdots$$

$$\mathrm{C_A}(\Phi_2) = \mathrm{C_A}(\Psi_2) - \mathrm{C_A}(\Psi_3) - G_2 \tag{5.19}$$

$$\mathrm{C_A}(\Phi_1) = \mathrm{C_A}(\Psi) - \mathrm{C_A}(\Psi_2) - G_1. \tag{5.20}$$

Somando-se membro a membro estas expressões resulta, no primeiro membro, a complexidade aditiva da fatoração. No segundo membro, todas os termos $\mathrm{C_A}(\Psi_i)$ se cancelam, restando apenas $\mathrm{C_A}(\Psi)$ menos o somatório de G_i. ■

Isso justifica que uma boa implementação para $V = \Psi v$, dado que Ψ contém elementos racionais, é obtida através da Proposição 5.1. Entretanto, quando a matriz Ψ é grande, o problema de maximizar G_i torna-se inviável. Esse problema se resume em escolher os melhores pares de colunas para Ψ. Uma regra prática é escolher os pares de colunas que

apresentem o menor número de direções possíveis; os próximos exemplos mostram como isso pode ser feito.

Exemplo 5.2

Fatorar a matriz

$$\Psi = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 1 & -1 & -1 & 1 \\ 0 & 1 & 1 & -1 & -1 \end{bmatrix},$$

como produto de matrizes bielementares. O primeiro passo seria escolher os pares de colunas para efetuar a primeira fatoração, $\Psi = \Psi_2\Phi_1$. Observa-se que, escolhendo-se as colunas 2 e 5 como primeiro par e as colunas 3 e 4 como segundo par, existem apenas 4 direções distintas para esses dois pares. Uma boa correlação de elementos é um indicativo de que o par de colunas é uma boa escolha para a fatoração. Assim, os pares de colunas escolhidos são $(2,5)$ e $(3,4)$, a coluna 1 fica isolada. Escolhe-se então

$$\Phi_1 = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & -1 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix},$$

e computa-se Ψ_2 através de (5.10) ou por simples observação, o que resulta em

$$\Psi_2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 \\ 1 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \end{bmatrix}.$$

Desde que Ψ_2 é bielementar a fatoração está concluída e é dada por

$$\Psi = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 \\ 1 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & -1 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

•

Exemplo 5.3

Fatorar a matriz

$$\Psi = \begin{bmatrix} 1 & 1 & 1 & 0 \\ -\frac{1}{2} & 0 & 1 & 1 \\ 0 & -\frac{1}{2} & 1 & -1 \end{bmatrix},$$

como produto de matrizes bielementares. Neste caso, independente dos pares de colunas escolhidos, obtém-se sempre 3 vetores bidimensionais com direções diferentes. Escolhendo-se

$$\Phi_1 = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 1 & 0 \end{bmatrix},$$

existem varias soluções para Ψ_2; escolhe-se a solução (5.10), ou seja,

$$\Psi_2 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & -\frac{1}{2} & 0 & 1 & 0 & 0 \\ 0 & 0 & -\frac{1}{2} & 0 & 1 & 0 \end{bmatrix}.$$

Observa-se que $G_1 = 0$, entretanto a matriz ainda pode ser fatorada em matrizes bielementares.

•

Na maioria das transformações práticas[1], as matrizes de transformações não são racionais; neste caso, uma implementação ideal preocupa-se em minimizar o número de multiplicações

[1] Transformada discreta de Fourier, Hartley, cosseno e wavelets, Filtros digitais, e muitos outros operadores lineares utilizados comumente na engenharia.

da transformação. Neste contexto, existem algoritmos que implementam uma transformação qualquer Ψ através da fatoração

$$\Psi = CBA,$$

em que C e A são matrizes com elementos racionais e a matriz B é uma matriz diagonal [7]. Essa fatoração é feita de modo a minimizar o número de multiplicações, entretanto, A e C são transformações racionais. Portanto, para minimizar o número de adições sobre esses algoritmos, utiliza-se a Proposição 5.1 para fatorar as matrizes A e C em matrizes bielementares.

Exemplo 5.4

Considere a matriz $W = CBA$ da transformada rápida de Fourier (FFT) para $N = 3$, fatorada pelo algoritmo de decomposição do núcleo em bases ciclotômicas [22],

$$W = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & j \\ 0 & 1 & -j \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \gamma_2 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & 1 & -1 \end{bmatrix},$$

em que $\gamma_2 = -\text{sen}(2\pi/3)$. Iniciando pela matriz A, a qual pode ser fatorada, e escolhendo-se o par de colunas $(2, 3)$, tem-se

$$A = \begin{bmatrix} 1 & 0 & 1 \\ -\frac{1}{2} & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 \\ 0 & 1 & -1 \\ 1 & 0 & 0 \end{bmatrix},$$

isto é, $A = A_1\Phi$. A matriz C, após separada em parte real e parte imaginária, não contém adições. Portanto, a implementação da FFT através de

$$V = CBA_1\Phi v,$$

contém 1 multiplicação, 1 multiplicação trivial e 4 adições. Para comparação, a FFT de Winograd implementa o mesmo algoritmo com 2 multiplicações e 4 adições [11]. •

Exemplo 5.5

Considere a FFT otimizada para $N = 5$ apresentada no Exemplo 4.4, que faz uso da fatoração $W = CBA$, com apenas quatro multiplicações. Para determinar a complexidade aditiva desse

algoritmo, dado que A é racional, B é diagonal e C é racional complexa, basta avaliar a complexidade aditiva de A e das partes real e imaginária de C. A matriz A é dada por

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 1 & -1 & -1 & 1 \\ 0 & 1 & 1 & -1 & -1 \end{bmatrix},$$

a qual já foi fatorada no Exemplo 5.2 e contém 6 adições. A parte real da matriz C é dada por

$$C_r = \begin{bmatrix} 1 & 1 & 1 & 0 \\ -\frac{1}{2} & 0 & 1 & 1 \\ 0 & -\frac{1}{2} & 1 & -1 \\ 0 & -\frac{1}{2} & 1 & -1 \\ -\frac{1}{2} & 0 & 1 & 1 \end{bmatrix},$$

a qual pode ser expressa por

$$C_r = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \Psi,$$

em que a matriz Ψ é a matriz do Exemplo 5.3 e, portanto, apresenta 6 adições. A parte imaginária da matriz C é dada por

$$C_i = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & -1 & -1 \\ -1 & 0 & -1 \end{bmatrix},$$

a qual pode ser decomposta em

$$C_i = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & -1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix},$$

com complexidade aditiva igual a 2. Portanto, a complexidade aditiva da FFT otimizada de comprimento cinco é 14. Para comparação, a FFT de Winograd implementa o mesmo algoritmo com 5 multiplicações e 13 adições [11]. •

O exemplo anterior, junto com a teoria da complexidade multiplicativa, mostra que qualquer transformada linear pode ser posta na forma CBA, em que a matriz diagonal B tem um número reduzido de multiplicações (não precisa ser necessariamente o mínimo). Pode-se, então, aplicar as técnicas para redução da complexidade aditiva apresentadas neste capítulo. Uma vez que as matrizes C e A estão fatoradas em matrizes bielementares, é simples informar a complexidade aritmética do algoritmo, o número de somadores e multiplicadores necessários no processo e o número de passos necessários para a computação da transformada. A Tabela 5.1 apresenta as complexidades aritméticas da FFT otimizada e da FFT de Winograd, para os comprimentos em que há diferença entre as complexidades multiplicativas (para $N < 16$).

Tabela 5.1: Complexidade aritmética da OFFT e da FFT de Winograd

N	OFFT		FFT de Winograd	
	Multiplicações	Adições	Multiplicações	Adições
3	1	6	2	4
5	4	14	5	13
7	7	35	8	30
9	8	64	10	36
10	8	66	-	-

O número de passos, n_p, está intimamente ligado com o número de ciclos de máquina (*clock*) e o atraso da saída da computação mediante a uma entrada. Dessa forma, o parâmetro n_P indica se o algoritmo é rápido ou lento. Algoritmos rápidos devem ter um n_P baixo em relação aos demais algoritmos. A influência de n_P, bem como a influência da representação numérica, são fatores importantes sobre projetos práticos de transformadas e são abordadas no próximo capítulo.

capítulo 6

Projeto de Transformadas Otimizadas

A teoria da complexidade aritmética sobre transformadas apresentada nos capítulos anteriores foi elaborada para ser aplicada no projeto de transformadas lineares, filtros e qualquer tipo de processamento aritmético de sinais digitais. Como consequência desse processamento, os sinais são discretizados e digitalizados. O processo de digitalização representa o valor de um sinal real por um valor aproximado que pode ser representado por uma sequência de *bits*. A Figura 6.1 mostra um exemplo de um sinal analógico sendo discretizado.

A codificação de cada valor discretizado em uma determinada sequência de *bits* é chamada de representação numérica. Existem várias representações numéricas utilizadas na engenharia. Elas se dividem em dois tipos principais: o ponto fixo e o ponto flutuante. A seguir são discutidas as representações compatíveis com a teoria apresentada nesta tese.

6.1 Escolha da Representação Numérica

A escolha da representação numérica é importante, uma vez que a mesma pode simplificar ou dificultar a implementação de blocos aritméticos. Para este caso específico, algumas características são essenciais na representação numérica. A primeira característica importante é que a representação numérica deve facilitar as operações aritméticas, sendo a adição mais simples que a multiplicação. Uma outra característica desejável é que multiplicações triviais

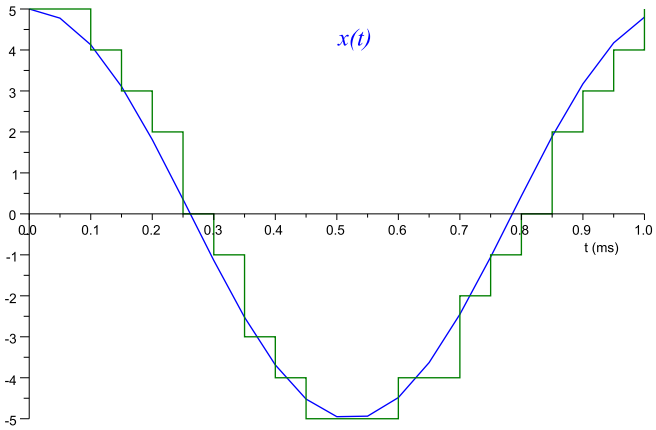


Figura 6.1: Um exemplo de digitalização: o sinal analógico está em azul e o sinal discretizado em verde.

sejam mais simples de ser implementadas que outras multiplicações usuais.

Neste ponto, recorre-se à definição formal de multiplicação trivial. Uma vez que números irracionais não podem ser representados em máquinas digitais, as multiplicações passam a não existir e tem-se apenas multiplicações triviais. Entretanto, números irracionais necessitam de uma grande quantidade de *bits* para serem representados, enquanto pequenos inteiros e números que são potência de dois são representados por poucos *bits* na maioria das representações digitais. Como na teoria, a maioria das multiplicações triviais envolve inteiros pequenos e potências de dois; para esses deve haver uma forma mais simples de implementar uma multiplicação utilizando vantagens da representação numérica.

Por essas e outras razões que são mostradas a seguir, foi escolhida a representação de ponto fixo sinal módulo (PFSM) de 32 *bits* para a implementação das transformadas. Dentre as principais vantagens dessa representação, pode-se citar a facilidade de implementar alguns tipos de multiplicações triviais sem a necessidade de blocos somadores ou multiplicadores, além da possibilidade de construir uma arquitetura que compute uma transformada por pulso do sinal de sincronismo (*clock*).

6.1.1 Representação ponto fixo sinal módulo de 32 *bits*

Essa representação é feita por 32 *bits*, sendo o primeiro *bit* uma informação sobre o sinal, e os 31 *bits* restantes informam sobre o módulo do número representado. Um número, A, é

representado por uma sequência de *bits*, $(a_s, a_{14}, a_{13} \ldots, a_{-16})$, de tal forma que

$$A = (-1)^{a_s} \sum_{i=-16}^{14} a_i 2^i, \tag{6.1}$$

o que pode ser representado por

$$A \overset{FP32}{\rightarrow} (a_s, a_{14}, a_{13} \ldots, a_{-16}). \tag{6.2}$$

É fácil notar que diferentes números recebem representações diferentes, por exemplo, o número 1 é representado pela sequência que o $a_0 = 1$ e todos os outros bits são zeros. Isso é representado como anteriormente (excluindo as vírgulas) por

$$1 \overset{FP32}{\rightarrow} (00000000000000010000000000000000).$$

Essa é a sequência de *bits* exata que representa o número 1 na representação PFSM. O número (-1) é representado por

$$-1 \overset{FP32}{\rightarrow} (10000000000000010000000000000000),$$

que diferiu apenas no primeiro *bit* da representação do 1. De fato, nessa representação, multiplicar por (-1) significa inverter o primeiro *bit*. Para facilitar a notação nesta tese, utiliza-se a notação em hexadecimal, na qual cada 4 *bits* é representado por um símbolo em hexadecimal; assim, pode-se simplificar as notações anteriores por

$$1 \overset{FP32}{\rightarrow} (00010000)_H$$

e

$$-1 \overset{FP32}{\rightarrow} (80010000)_H.$$

6.1.2 Multiplicações triviais na representação ponto fixo sinal módulo

A facilidade da implementação da multiplicação por (-1) para a representação PFSM já foi observada, entretanto, outros tipos de multiplicações podem ser implementadas sem a necessidade de um multiplicador (O bloco multiplicador é apresentado mais adiante). Essas são as multiplicações consideradas triviais na prática. Considere um número A que deve ser multiplicado por outro número, T, da forma

$$T = \pm 2^m,$$

sendo P o produto desta multiplicação. Utilizando (6.1), o produto é

$$P = TA = \pm 2^m (-1)^{a_s} \sum_{i=-16}^{14} a_i 2^i = \pm (-1)^{a_s} \sum_{i=-16}^{14} a_i 2^{i+m}.$$

Fazendo a mudança de variável $n = i + m$, tem-se

$$P = \pm (-1)^{a_s} \sum_{n=-16+m}^{14+m} a_{n-m} 2^n.$$

Supondo que $|P| < 2^{15}$ (isto é, não ocorre *overflow*), ou desprezando os coeficientes das exponenciais menores que 2^{-16}, pode-se escrever

$$P = \pm (-1)^{a_s} \sum_{n=-16}^{14} a_{n-m} 2^n. \tag{6.3}$$

Isso significa que a implementação desse tipo de multiplicação pode ser feita sem um bloco de multiplicação, simplesmente, fazendo um deslocamento nos bits da representação do número A.

Exemplo 6.1

Implementar a multiplicação de um número x que está na representação PFSM pela constante $(-2^{-2}) = -1/4$. Sendo a representação de x dada por

$$x \overset{FP32}{\rightarrow} (x_s, x_{14}, x_{13}, x_{12}, x_{11}, \ldots, x_{-14}, x_{-15}, x_{-16}),$$

então, através de (6.3), o resultado da multiplicação é representado por

$$-x/4 \overset{FP32}{\rightarrow} (\bar{x}_s, 0, 0, x_{14}, x_{13}, \ldots, x_{-12}, x_{-13}, x_{-14}),$$

em que $\bar{x}_s$ é o inverso binário de x_s, o que pode ser implementado com um registrador de deslocamento e um inversor, ou simplesmente por uma reorganização no barramento de saída, como mostra a Figura 6.2. A reorganização do barramento é a forma mais indicada para a implementação de multiplicações triviais e é utilizada no decorrer desta tese. •

A implementação de *pequenos* inteiros também pode ser feita utilizando-se multiplicações triviais e adições, por exemplo, pode-se multiplicar A por 5 com uma única adição, desde que $5 = 2^2 + 1$, então, $5A = [(2^2 A) + A]$. A multiplicação $2^2 A$ é trivial e o produto é obtido com apenas uma adição. Neste ponto, pode-se definir como pequeno inteiro (pode ser um racional) um número, α, no qual a implementação de αA utilizando somadores tem baixa complexidade de implementação em relação à implementação com o multiplicador. Dessa forma, a teoria da complexidade aritmética se adapta perfeitamente ao projeto das transformadas.

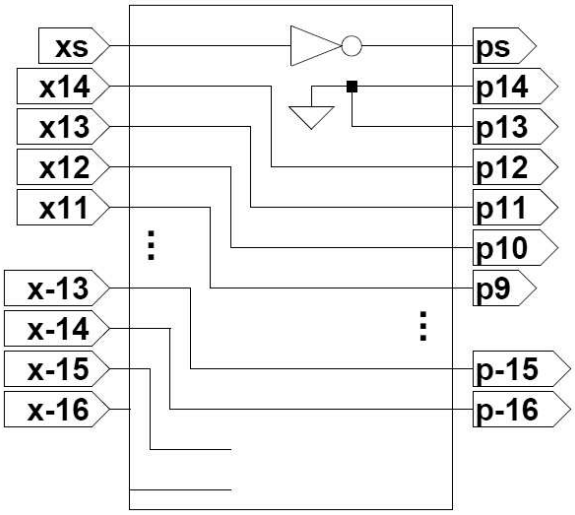


Figura 6.2: Implementação de uma multiplicação trivial por (-2^{-2}) *para um número na representação PFSM de 32 bits.*

6.2 Arquitetura, somadores e multiplicadores

O objetivo deste capítulo é a implementação em linguagem de descrição de *hardware* das transformadas discretas de Fourier otimizadas para diversos comprimentos. A definição da arquitetura é um ponto fundamental do processo e normalmente depende da aplicação. É na arquitetura que se define o tempo de processamento, o número de entradas, se o sistema é síncrono ou assíncrono, as entradas e saídas do dispositivo digital e muitas outras características do projeto.

Uma possível escolha para a arquitetura é um sistema que compute uma transformada por pulso de *clock*. Para isso, o processador deve transferir os resultados de cada computação para o bloco seguinte, de forma que a saída do sistema forneça os resultados esperados após algum atraso, que está relacionado diretamente com o número de passos da transformada a ser implementada.

As entradas do sistema digital são: o vetor v, cuja DFT se deseja computar; um sinal binário, *clr*, para limpar os dados dos registradores quando necessário; o sinal binário, *enable*, para habilitar a computação (pois do contrário o sistema ficaria sempre computando e consumindo energia); e o sinal do sincronismo (*clock*), *clk*, o que significa que as implementações são síncronas.

As saídas do sistema são: o vetor V, que é a transformada discreta de Fourier do vetor de entrada, v, e uma saída binária, *ovrf*, para sinalização de sobrecarga, isto é, quando ocorrer

um resultado que não pode ser representado em PFSM 32 *bits*. O diagrama da arquitetura do sistema que implementa uma transformada discreta de Fourier otimizada, dado que a matriz de transformação foi otimizada em $W = CBA$, é mostrada na Figura 6.3.

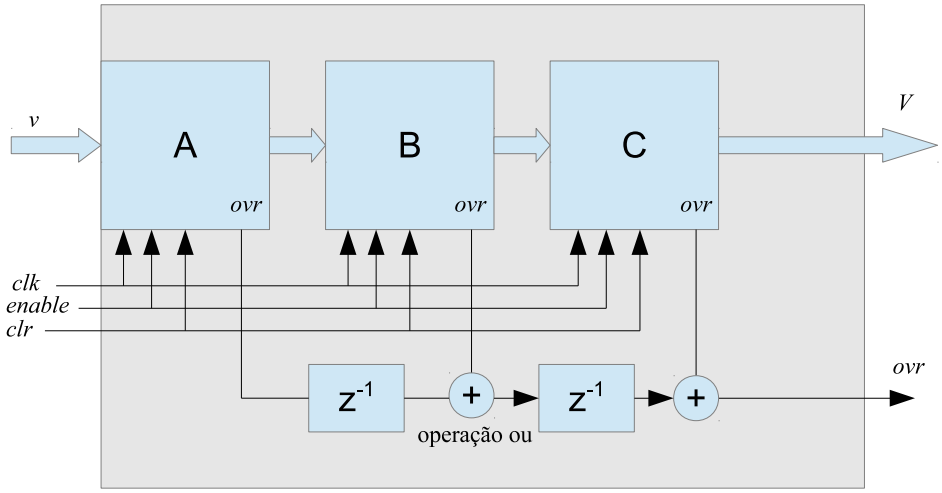


Figura 6.3: Arquitetura para a implementação da transformada discreta de Fourier otimizada.

Os blocos A e C são blocos de somadores, enquanto o bloco B é um bloco de multiplicadores. Pode ser observado que cada bloco deve conter uma saída de sinalização de sobrecarga, $ovrf$. As Figuras 6.4 e 6.5 mostram uma representação, em nível de transferência de registradores (RTL, do inglês *Register Transfer Level*) [25], para o somador e para o multiplicador, respectivamente.

Figura 6.4: Representação do somador em RTL, com todas as entradas e saídas.

Alguns resultados podem ser observados a partir da sintetização do somador e do multiplicador, neste caso específico, utilizando linguagem Verilog, sintetizado no *Quartus II 64-bits Version Build 153 11/29/2010 SJ Web Edition* para dispositivos da família *Cyclone IV GX*[1].

[1] Quartus II é uma ferramenta de síntese disponibilizada pela *Altera Corporation*, a qual sintetiza programas, escritos em algumas linguagens de descrição de *hardware*, para funcionar em dispositivos FPGAs e SoCs vendidos pela mesma. Para mais informações, pode-se consultar www.altera.com.

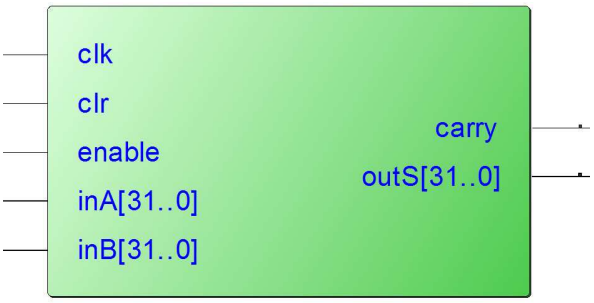


Figura 6.5: Representação do multiplicador em RTL, com todas as entradas e saídas.

A sintetização do somador resultou em: 192 funções combinacionais; 33 registradores lógicos dedicados; frequência máxima de 1416,43 MHz, restringido a frequência máxima do dispositivo de 250 MHz.

A sintetização do multiplicador resultou em: 1313 funções combinacionais; 47 registradores lógicos dedicados; frequência máxima não informada.

Pode ser observado que, na prática, a complexidade de funções combinacionais do multiplicador é bem superior à do somador. Entretanto, isso depende fortemente da implementação de cada dispositivo. Neste caso, ambos os blocos foram implementados utilizando-se a representação PFSM. Essa complexidade varia bastante se o dispositivo é projetado para trabalhar com representação em ponto flutuante.

A seguir, são apresentadas implementações da transformada discreta de Fourier otimizada (em relação à complexidade multiplicativa) para comprimentos específicos, os quais são apresentados pela primeira vez na literatura.

6.3 Transformada Discreta de Fourier Otimizada de Comprimento 5

A transformada rápida de Fourier Otimizada (OFFT) de comprimento 5 foi obtida nos capítulos anteriores e sua implementação, após a fatoração em matrizes bielementares de A e C, resultou no seguinte procedimento para computar $V = Wv$: primeiro se computa o vetor

D através de

$$D = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \beta_1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \beta_2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \beta_3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \beta_4 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 \\ 1 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & -1 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix},$$

em que as constantes β_i são calculadas no Exemplo 4.4 e são dadas por

$$\beta_1 = -j[\text{sen}(2\pi/5) - \text{sen}(4\pi/5)],$$

$$\beta_2 = -j[\text{sen}(2\pi/5) + \text{sen}(4\pi/5)],$$

$$\beta_3 = \cos(2\pi/5)$$

e

$$\beta_4 = -j\text{sen}(4\pi/5),$$

e o vetor D é

$$D = \begin{bmatrix} D_1 \\ D_2 \\ D_3 \\ D_4 \\ D_5 \\ D_6 \\ D_7 \end{bmatrix}.$$

A parte real da transformada é então computada através de

$$\Re(V) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & -\frac{1}{2} & 0 & 1 & 0 & 0 \\ 0 & 0 & -\frac{1}{2} & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} D_1 \\ D_2 \\ D_3 \\ D_6 \end{bmatrix},$$

e a parte imaginária é computada através de

$$\Im(V) = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & -1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} D_4 \\ D_5 \\ D_7 \end{bmatrix}.$$

Pode ser observado que a computação da FFT envolve 5 passos de computação, considerando cada matriz que contenha alguma computação como um passo. Também pode ser observado que a computação das partes real e imaginária pode ser feita ao mesmo tempo. Têm-se no total 4 multiplicações e 14 adições.

A implementação da OFFT de comprimento 5 pode ser visualizada em RTL na Figura 6.6, a sintetização resultou em: 4502 funções combinacionais; 1044 registradores lógicos dedicados; frequência máxima de 85,65 MHz. Os arquivos em *Verilog* desta implementação estão disponíveis no Apêndice B.

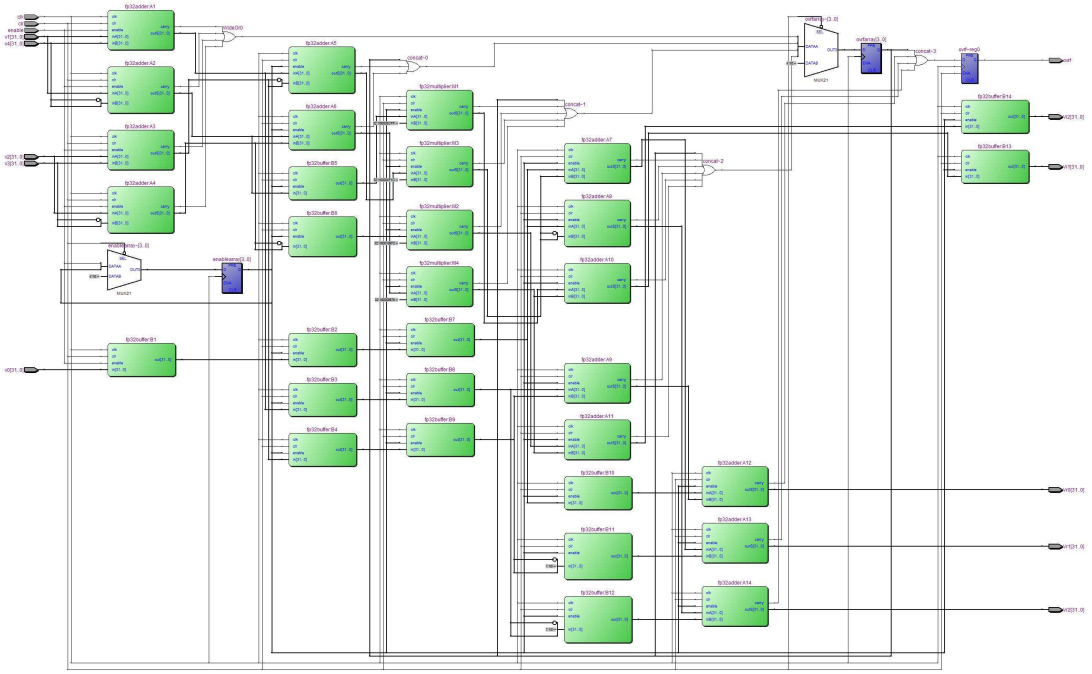


Figura 6.6: Diagrama em RTL da implementação da OFFT de comprimento 5.

6.4 Transformada Discreta de Fourier Otimizada de Comprimento 7

A OFFT de comprimento 7 é obtida utilizando-se as teorias apresentadas nos Capítulos 4 e 5. As matrizes A, B e C, obtidas para a implementação do algoritmo por $W = CBA$, são dadas por

$$A = \begin{bmatrix} 0 & 1 & -1 & 0 & 0 & 1 & -1 \\ 0 & 0 & -1 & -1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & -1 & 0 & -1 \\ 0 & 1 & 1 & -1 & 1 & -1 & -1 \\ 0 & 1 & -\frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & 1 \\ 0 & 0 & \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & 0 \\ 0 & \frac{1}{2} & -\frac{1}{8} & -\frac{5}{8} & -\frac{5}{8} & \frac{1}{8} & \frac{1}{2} \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix},$$

$$B = \begin{bmatrix} \beta_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \beta_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \beta_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \beta_4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \beta_5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \beta_6 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \beta_7 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

e

$$C = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & -\frac{1}{2} \\ -1 & 1 & 0 & 1 & 0 & -2 & 2 & 1 & 0 & -\frac{1}{2} & 0 \\ 0 & 1 & 1 & -1 & -1 & 1 & -3 & 1 & -\frac{1}{2} & 0 & 0 \\ 0 & -1 & -1 & 1 & -1 & 1 & -3 & 1 & -\frac{1}{2} & 0 & 0 \\ 1 & -1 & 0 & -1 & 0 & -2 & 2 & 1 & 0 & -\frac{1}{2} & 0 \\ -1 & 0 & -1 & -1 & 1 & 1 & 1 & 1 & 0 & 0 & -\frac{1}{2} \end{bmatrix},$$

em que todas as constantes β_i são pré-calculadas e dadas por

$$\beta_1 = -j\frac{1}{3}[\text{sen}(2\pi/7) - 2\text{sen}(4\pi/7) - \text{sen}(6\pi/7)],$$

$$\beta_2 = -j\frac{1}{3}[2\text{sen}(2\pi/7) - \text{sen}(4\pi/7) + \text{sen}(6\pi/7)],$$

$$\beta_3 = -j\frac{1}{3}[\text{sen}(2\pi/7) + \text{sen}(4\pi/7) + 2\text{sen}(6\pi/7)],$$

$$\beta_4 = -j\frac{1}{3}[\text{sen}(2\pi/7) + \text{sen}(4\pi/7) - \text{sen}(6\pi/7)],$$

$$\beta_5 = \cos(2\pi/7) - \frac{1}{2}\cos(4\pi/7),$$

$$\beta_6 = \cos(2\pi/7) + \frac{5}{4}\cos(4\pi/7)$$

e

$$\beta_7 = \cos(4\pi/7).$$

Pode-se verificar que a maioria das multiplicações triviais da teoria são triviais também na prática. Entretanto, multiplicações por -3 e por $-5/8$ podem não ser triviais na prática. De fato, essas multiplicações podem ser implementadas com uma adição, e podem ser feitas por um circuito combinacional dedicado, mais simples que o bloco somador, porém, um pouco mais complexo que o circuito que implementa a multiplicação por $\pm 2^m$ apresentado na Seção 6.1.2.

Após a fatoração de A e C em matrizes bielementares, chega-se à conclusão que a complexidade aritmética do algoritmo é de 7 multiplicações e 35 adições. A seguir são apresentados os algoritmos obtidos para comprimento 9 e 10 na forma CBA.

6.5 Transformada Discreta de Fourier Otimizada de Comprimento 9

As matrizes A, B e C, obtidas para a implementação da OFFT de comprimento 9 por $W = CBA$, são dadas por

$$A = \begin{bmatrix} 0 & 2 & 1 & 0 & -1 & 1 & 0 & -1 & -2 \\ 0 & 0 & -1 & 0 & -1 & 1 & 0 & 1 & 0 \\ 0 & 2 & -1 & 0 & -1 & -1 & 0 & -1 & 2 \\ 0 & 0 & 1 & 0 & -1 & -1 & 0 & 1 & 0 \\ 0 & -1 & \frac{1}{4} & 0 & \frac{5}{4} & -\frac{5}{4} & 0 & -\frac{1}{4} & 1 \\ 0 & -3 & \frac{15}{4} & 0 & -\frac{3}{4} & -\frac{3}{4} & 0 & \frac{15}{4} & -3 \\ 0 & 2 & -2 & 0 & 2 & -2 & 0 & 2 & -2 \\ 0 & 0 & 0 & 2 & 0 & 0 & -2 & 0 & 0 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 2 & 0 & 0 & -1 & 0 & 0 & -1 & 0 & 0 \\ 2 & -1 & -1 & 2 & -1 & -1 & 2 & -1 & -1 \end{bmatrix},$$

$$B = \begin{bmatrix} \beta_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \beta_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \beta_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \beta_4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \beta_5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \beta_6 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \beta_7 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \beta_8 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} \end{bmatrix}$$

e

$$C = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 2 & 0 & -2 & -2 & -\frac{2}{3} & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ -1 & 1 & -1 & 1 & -3 & -\frac{1}{3} & 0 & 1 & 0 & 1 & 0 \\ 1 & -1 & -1 & 1 & 3 & -\frac{1}{3} & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 \\ 0 & -2 & 0 & -2 & 2 & -\frac{2}{3} & 0 & 1 & 0 & 1 & 0 \\ -1 & -1 & 1 & 1 & -1 & 1 & 0 & -1 & 0 & 1 & 0 \end{bmatrix},$$

em que todas as constantes β_i são pré-calculadas e dadas por

$$\beta_1 = -j\frac{1}{2}[\text{sen}(2\pi/9) + \frac{1}{2}\text{sen}(4\pi/9)],$$

$$\beta_2 = -j\frac{1}{2}[\text{sen}(2\pi/9) - \frac{5}{4}\text{sen}(4\pi/9)],$$

$$\beta_3 = \frac{1}{2}[\cos(2\pi/9) + \frac{3}{2}\cos(4\pi/9)],$$

$$\beta_4 = \frac{1}{2}[\cos(2\pi/9) - \frac{1}{4}\cos(4\pi/9)],$$

$$\beta_5 = -j\frac{1}{2}\text{sen}(4\pi/9),$$

$$\beta_6 = \frac{1}{2}\cos(4\pi/9)$$

e

$$\beta_7 = \beta_8 = -j\frac{1}{2}\text{sen}(6\pi/9).$$

Após a fatoração de A e C em matrizes bielementares, chega-se à conclusão que a complexidade aritmética do algoritmo é de 8 multiplicações e 64 adições.

6.6 Transformada Discreta de Fourier Otimizada de Comprimento 10

As matrizes A, B e C, obtidas para a implementação da OFFT de comprimento 9 por $W = CBA$, são dadas por

$$A = \begin{bmatrix}
0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\
0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & -1 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 1 \\
0 & 0 & 1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 \\
0 & 1 & 0 & -1 & 0 & 0 & 0 & -1 & 0 & 1 \\
0 & 0 & 1 & 0 & -1 & 0 & -1 & 0 & 1 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 0 & -\frac{1}{2} & \frac{1}{2} & 0 & -1 & 0 & \frac{1}{2} & -\frac{1}{2} & 0 \\
1 & -\frac{1}{2} & 0 & 0 & -\frac{1}{2} & 1 & -\frac{1}{2} & 0 & 0 & -\frac{1}{2} \\
1 & \frac{1}{2} & 0 & 0 & -\frac{1}{2} & -1 & -\frac{1}{2} & 0 & 0 & \frac{1}{2} \\
1 & 0 & -\frac{1}{2} & -\frac{1}{2} & 0 & 1 & 0 & -\frac{1}{2} & -\frac{1}{2} & 0 \\
1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1
\end{bmatrix},$$

$$B = \begin{bmatrix}
\beta_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \beta_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \beta_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & \beta_4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \beta_5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \beta_6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \beta_7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \beta_8 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}$$

e

$$C = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & -1 & 1 & 1 & -1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & -1 & 1 & -1 & -1 & -1 & -1 & 0 & 0 & 0 & 1 & 0 & 0 \\
-1 & 1 & 0 & 0 & 1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
1 & -1 & 0 & 0 & -1 & 1 & -1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & -1 & 1 & 1 & -1 & -1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & -1 & -1 & 1 & -1 & 1 & -1 & 0 & 0 & 1 & 0 & 0 & 0 \\
-1 & -1 & 0 & 0 & -1 & -1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0
\end{bmatrix},$$

em que todas as constantes β_i são pré-calculadas e dadas por

$$\beta_1 = \beta_2 = -j[\text{sen}(2\pi/10) + \text{sen}(4\pi/10)],$$

$$\beta_3 = \beta_4 = -j[\text{sen}(2\pi/10) - \text{sen}(4\pi/10)],$$

$$\beta_5 = \beta_6 = -j\text{sen}(4\pi/10),$$

e

$$\beta_7 = \beta_8 = \cos(2\pi/10).$$

Após a fatoração de A e C em matrizes bielementares, chega-se à conclusão que a complexidade aritmética do algoritmo é de 8 multiplicações e 66 adições.

capítulo 7

Conclusões e Trabalhos Futuros

Este capítulo sumariza as principais contribuições e resultados obtidos e comenta sobre possíveis trabalhos futuros.

O Capítulo 1 apresenta os principais resultados da teoria da complexidade aritmética e os princípios que motivaram o estudo descrito nesta tese. Nos capítulos seguintes estão as contribuições originais e trabalhos realizados no doutorado. Pode-se dizer, de forma resumida, que a tese apresenta uma proposta de teoria da complexidade aritmética e utiliza essa teoria para derivar algoritmos otimizados, em termos da complexidade multiplicativa e aditiva, priorizando a complexidade multiplicativa.

Além disso, a teoria apresentada se aplica a qualquer algoritmo que computa uma transformada linear (em alguns casos, componentes ou uma única componente da transformada), bem como a algoritmos de convolução linear e cíclica. Também foi apresentada uma metodologia para minimizar o número de adições de transformadas racionais.

Essa teoria foi utilizada para a computação da transformada discreta de Fourier, a qual resultou em um novo algoritmo chamado de transformada rápida de Fourier otimizada (OFFT), o qual implementa a DFT com o número mínimo de multiplicações possível. Essas transformadas foram implementadas em linguagem de descrição de *hardware* e apresentadas no capítulo 6. As transformadas apresentadas nesse capítulo são inéditas na literatura.

7.1 Contribuições da Pesquisa

As primeiras contribuições originais da pesquisa relatada nesta tese foram as publicações do algoritmo JCO [23], dos autores G. Jerônimo da Silva Jr., R. M. Campello de Souza e H. M. de Oliveira, e a aplicação do mesmo na detecção de sinais DTMF [24]. O algoritmo JCO é utilizado para a computação de uma única componente da DFT e pode ser usado no lugar do algoritmo de Goertzel[16].

As contribuições seguintes surgiram na tentativa de se obter um algoritmo ótimo para a DFT, que atingisse a complexidade mínima. As bases ciclotômicas foram descobertas nessa tentativa. Uma FFT baseada na decomposição do núcleo da DFT em bases ciclotômicas foi apresentada em 2010 [22], mas esse algoritmo encontrava a mínima complexidade apenas para $N = 3, 4, 6, 8, 12$ e 24. Entretanto, o resultado já era bastante expressivo; a FFT ótima para $N = 3$ já superava a FFT de Winograd, que nessa época era a FFT mais eficiente[1].

As contribuições finais foram a FFT otimizada [15], a qual está apresentada no Capítulo 4, e a teoria da complexidade aditiva para transformadas [18], apresentada no Capítulo 5. O Capítulo 6 apresenta implementações práticas de algumas transformadas otimizadas, implementadas na linguagem de descrição de *hardware Verilog*. Os arquivos das implementações em *Verilog* estão disponibilizados no Apêndice B.

7.2 Trabalhos Futuros

Algumas questões referentes a esta tese permanecem em aberto. Existem alguns temas importantes que podem ser aprofundados. Uma primeira questão é determinar se existe ou não um meio que possibilite encontrar as bases numéricas para qualquer transformada, dada a matriz de transformação. No caso específico da transformada discreta de Fourier, as bases ciclotômicas são a base numérica da transformada. No caso da transformada de Hartley, também é simples derivar as bases numéricas através das bases ciclotômicas, entretanto, o que se pode dizer sobre as bases numéricas das transformadas do cosseno e outras?

Uma segunda questão diz respeito ao processo de decompor um conjunto qualquer de matrizes no menor número de matrizes postunitárias. Um método foi apresentado mas esse ainda exige uma busca, e mostra-se que o algoritmo para obter a melhor transformada é NP. Existe um método direto para se obter a decomposição de um conjunto de matrizes em

[1] É surpreendente observar como a DFT otimizada com comprimento $N = 3$ só foi obtida em 2010 [22], mesmo sendo prevista por Heideman desde 1988[8].

matrizes postunitárias?

Um outro problema é encontrar uma prova ou refutação de que a Proposição 5.1, apresentada no Capítulo 5, encontra o melhor algoritmo em termos da complexidade aditiva. Aparentemente, esta proposição sempre leva ao melhor resultado, pelo menos quando comparado com alguns métodos heurísticos, entretanto a prova disso parece estar além do escopo desta tese.

Uma outra proposta interessante é definir um novo parâmetro chamado de complexidade aritmética, o qual seria uma combinação linear ponderada da complexidade aritmética multiplicativa e aditiva. Alguns autores utilizam o parâmetro *operações com ponto flutuante* (flops), que é simplesmente a soma direta do número de adições e multiplicações [26, 27]. Também é possível definir uma ponderação baseada em alguma outra regra.

Seja como for, a escolha da regra de ponderação para esse parâmetro de complexidade aritmética pode beneficiar interesses específicos e gerar comparações questionáveis. Por exemplo, suponha duas implementações distintas de um algoritmo A, I_1 e I_2. A implementação I_1 utiliza 3 multiplicações e 5 adições, enquanto a implementação I_2 utiliza 4 multiplicações e 3 adições. Se o parâmetro complexidade aritmética, CA, é dado por

$$\text{CA} = (\text{número de adições}) + (\text{número de multiplicações}),$$

então o algoritmo I_2 tem uma complexidade menor que o algoritmo I_1. Porém, se a ponderação do parâmetro CA muda, o resultado pode mudar bastante. Se agora

$$\text{CA} = (\text{número de adições}) + 4(\text{número de multiplicações}),$$

então o algoritmo I_1 passa a ter uma complexidade menor que o algoritmo I_2. Ainda que essa ponderação fosse baseada na implementação em *hardware* do somador e do multiplicador, poderia-se forçar uma implementação de forma a favorecer um determinado algoritmo, o que também resulta em questionamentos.

Uma possível ideia para contornar esse desafio é propor uma regra de ponderação baseada no estado da arte da implementação de somadores e multiplicadores. Por exemplo, ponderar a complexidade baseado no consumo de energia realizada pelo somador e multiplicador, quando os mesmos realizam suas respectivas operações. Essa definição seria um novo tema de estudo e pode ser utilizada para comparar algoritmos com números de multiplicações e adições diferentes.

Essas são as questões principais que ainda não estão resolvidas nesta tese, e podem ser-

vir como tema de estudos para outros pesquisadores interessados na área de otimização de algoritmos para processamento digital de sinais.

Referências

[1] H. M. de Oliveira, *Análise de Fourier e Wavelets: Sinais Estacionários e não Estacionários*. Editora Universitária UFPE, 2007.

[2] A. V. Oppenheim, R. W. Schafer, and J. R. Buck, *Discrete-Time Signal Processing*, 3rd ed. Prentice Hall, 2010.

[3] S. Winograd, *Arithmetic Complexity of Computations*. SIAM Publications, 1980.

[4] M. Heideman, D. Johnson, and C. Burrus, "Gauss and the history of the fast Fourier transform," *ASSP Magazine, IEEE*, vol. 1, no. 4, pp. 14–21, Oct 1984.

[5] L. I. Kronsjö, *Algorithms: Their Complexity and Efficiency*. John Wiley & Sons, 1979.

[6] L. V. Toscani and P. A. S. Veloso, *Complexidade de Algoritmos: análise, projeto e métodos*. Sagra Luzzato, 2005.

[7] R. E. Blahut, *Fast Algorithms for Signal Processing*, 2nd ed. Cambridge University Press, 2010.

[8] M. T. Heideman, *Multiplicative Complexity, Convolution, and the DFT*, 2nd ed. Springer-Verlag, 1988.

[9] A. V. Oppenheim, A. S. Willsky, and S. H. Nawab, *Signals and Systems*, 2nd ed. Prentice Hall, 1996.

[10] C. J. L. Pimentel, *Comunicação Digital*. Brasport, 2007.

[11] S. Winograd, "On computing the discrete Fourier transform," *Mathematics of Computation*, vol. 32, pp. 175–199, Jan. 1978.

[12] I. J. Good, "The interaction algorithm and practical Fourier analysis," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 20, pp. 361–372, August 1958.

[13] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Mathematics of Computation*, vol. 19, no. 90, pp. 297–301, 1965. [Online]. Available: http://www.jstor.org/stable/2003354

[14] D. M. Burton, *Elementary Number Theory*, 6th ed. McGraw-Hill, 2007.

[15] G. J. Jr. and R. M. Campello de Souza, "A transformada de Fourier otimizada," *Simpósio Brasileiro de Telecomunicações, SBrT*, vol. 28, pp. 1–5, Outubro 2011.

[16] G. Goertzel, "An algorithm for the evaluation of finite trigonometric series," *The American Mathematical Monthly*, vol. 65, no. 1, pp. 34–35, 1958.

[17] J. Beraldin and W. Steenaart, "Overflow analysis of a fixed-point implementation of the Goertzel algorithm," *Circuits and Systems, IEEE Transactions on*, vol. 36, no. 2, pp. 322–324, Feb 1989.

[18] G. J. Jr. and R. M. Campello de Souza, "Teoria da complexidade aditiva para transformadas," *Simpósio Brasileiro de Telecomunicações, SBrT*, vol. 28, pp. 1–5, Outubro 2011.

[19] R. J. McEliece, *Finite Fields for Computer Scientists and Engineers.* Kluwer Academic Publishers, 1987.

[20] G. Strang, *Linear Algebra and Its Aplications*, 4th ed. Thomson Brooks/Cole, 2006.

[21] E. L. Lima, *Álgebra Linear*, 5th ed. IMPA, 2001.

[22] G. J. Jr. and R. M. Campello de Souza, "Cyclotomic basis for computing the discrete Fourier transform," *International Telecommunications Symposium, ITS*, vol. 7, pp. 1–5, September 2010.

[23] G. J. Jr., R. M. Campello de Souza, and H. M. de Oliveira, "New algorithms for computing a single component of the discrete Fourier transform," *International Symposium on Communication Theory and Applications, ISCTA*, vol. 10, pp. 151–154, Jul. 2009.

[24] G. J. Jr. and R. M. Campello de Souza, "Implementação de um detector DTMF utilizando o algoritmo JCO," *Simpósio Brasileiro de Telecomunicações, SBrT*, vol. 27, pp. 1–5, September 2009.

[25] M. D. Ercegovac, T. Lang, and J. H. Moreno, *Introdução aos Sistemas Digitais*. Bookman, 2000.

[26] P. Duhamel and M. Veterlli, "Fast Fourier transform: a tutorial review and a state of the art," *Signal Processing*, vol. 19, pp. 259–299, Apr 1990.

[27] S. G. Johnson and M. Frigo, "A modified split-radiz FFT with fewer arithmetic operations," *IEEE Trans. Signal Processing*, vol. 55, pp. 111–119, Jan 2007.

APÊNDICE A

PROCESSO DE ORTOGONALIZAÇÃO DE GRAM-SCHMIDT

O processo de ortogonalização de Gram-Schmidt é apresentado nas principais literaturas sobre álgebra linear[1]. Este processo obtém um conjunto de vetores ortogonais, $\vec{u}_i$, $i = 1, 2, \ldots, M$, a partir de N vetores quaisquer, $\vec{v}_i$, $i = 1, 2, \ldots, N$, $M \leq N$, em que qualquer vetor $\vec{v}_i$ pode ser expresso como

$$\vec{v}_i = \sum_{j=1}^{M} c_{ij} \vec{u}_j,$$

com $c_{ij} \in \mathbb{C}$. Este processo pode ser feito desde que o espaço vetorial seja munido de um produto interno.

O produto interno é um funcional bilinear simétrico e positivo sobre o espaço vetorial. Sejam dois vetores quaisquer, $\vec{v}_1$ e $\vec{v}_2$, cujo produto interno é representado por $\langle \vec{v}_1, \vec{v}_2 \rangle$. A propriedade bilinear do produto interno significa que

$$\langle \vec{v}_1 + \vec{v}_2, \vec{v}_3 \rangle = \langle \vec{v}_1, \vec{v}_3 \rangle + \langle \vec{v}_2, \vec{v}_3 \rangle ,$$

$$\langle \vec{v}_1, \vec{v}_2 + \vec{v}_3 \rangle = \langle \vec{v}_1, \vec{v}_2 \rangle + \langle \vec{v}_1, \vec{v}_3 \rangle ,$$

$$\langle a\vec{v}_1, \vec{v}_2 \rangle = a \langle \vec{v}_1, \vec{v}_2 \rangle$$

e

$$\langle \vec{v}_1, a\vec{v}_2 \rangle = a \langle \vec{v}_1, \vec{v}_2 \rangle ,$$

[1] Para informações mais detalhadas, sugere-se as referências [20] e [21].

em que a pertence ao corpo dos complexos. A propriedade da simetria ou comutatividade significa que

$$\langle \vec{v}_1, \vec{v}_2 \rangle = \langle \vec{v}_2, \vec{v}_1 \rangle .$$

A propriedade da positividade significa que $\langle \vec{v}, \vec{v} \rangle$ é real e maior que zero, se $\vec{v} \neq \vec{0}$, em que $\vec{0}$ é o vetor nulo. Se $\vec{v} = \vec{0}$, então $\langle \vec{v}, \vec{v} \rangle = 0$.

Dois vetores, $\vec{v}_1$ e $\vec{v}_2$, diferentes do vetor nulo, são ditos ortogonais se $\langle \vec{v}_1, \vec{v}_2 \rangle = 0$. A seguir é apresentado o processo de Gram-Schmidt.

A.1 Algoritmo de Ortogonalização

Dado o conjunto de vetores não nulos $\vec{v}_i$, $i = 1, \ldots, N$, toma-se o primeiro vetor como $\vec{u}_1$, isto é, $\vec{u}_1 = \vec{v}_1$, e computa-se, para o próximo vetor $\vec{v}_n$ (dado que existem, até o momento, m vetores ortogonais $\vec{u}_i$),

$$\vec{u}_{m+1} = \vec{v}_n - \sum_{i=1}^{m} \frac{\langle \vec{v}_n, \vec{u}_i \rangle}{\langle \vec{u}_i, \vec{u}_i \rangle} \vec{u}_i. \tag{A.1}$$

Se $\vec{u}_{m+1} \neq \vec{0}$, então adiciona-se $\vec{u}_{m+1}$ para o conjunto de vetores ortogonais, incrementa-se o valor de m e repete-se o processo para o resto dos vetores $\vec{v}_n$.

Pode-se mostrar que o conjunto $\vec{u}_i$ é ortogonal por indução. Supondo que o conjunto é ortogonal até os m primeiros vetores. Então a Equação (A.1) é válida para algum vetor $\vec{v}_n \neq \vec{0}$. Calculando-se o produto interno de $\vec{u}_{m+1}$ por outro vetor $\vec{u}_j$, $j \leq m$, tem-se

$$\langle \vec{u}_{m+1}, \vec{u}_j \rangle = \langle \vec{v}_n, \vec{u}_j \rangle - \sum_{i=1}^{m} \frac{\langle \vec{v}_n, \vec{u}_i \rangle}{\langle \vec{u}_i, \vec{u}_i \rangle} \langle \vec{u}_i, \vec{u}_j \rangle ,$$

$$\langle \vec{u}_{m+1}, \vec{u}_j \rangle = \langle \vec{v}_n, \vec{u}_j \rangle - \frac{\langle \vec{v}_n, \vec{u}_j \rangle}{\langle \vec{u}_j, \vec{u}_j \rangle} \langle \vec{u}_j, \vec{u}_j \rangle = \vec{0},$$

o que significa que o vetor $\vec{u}_{m+1}$ é ortogonal a todos os vetores $\vec{u}_j$, $j \leq m$, do conjunto. Portanto, pode-se concluir, por indução, que o conjunto de vetores $\vec{u}_i$, $i = 1, \ldots, M$, é um conjunto ortogonal.

Se o espaço vetorial for o $\mathbb{R}^n$, um possível produto escalar é a soma dos produtos termo a termo das componentes dos vetores. Se esse é o caso, e se os elementos dos vetores pertencem ao corpo dos racionais, então, o conjunto de vetores ortogonais obtidos a partir do processo de ortogonalização de Gram-Schmidt contém elementos racionais.

Existe ainda o processo de ortonormalização de Gram-Schmidt, que consiste em dividir os vetores obtidos no processo de ortogonalização, $\vec{u}_i$, $i = 1, \ldots, M$, por $\sqrt{\langle \vec{u}_i, \vec{u}_i \rangle}$. O processo

de ortonormalização não é interessante para aplicações em algoritmos rápidos, uma vez que $\langle \vec{u}_i, \vec{u}_i \rangle^{-\frac{1}{2}}$ nem sempre pertence ao corpo dos racionais.

APÊNDICE B

Arquivos em Verilog para a Implementação da OFFT de comprimento 5

```
// UFPE
// Arquivo: fp32adder.v
// Modulo: fp32adder
// Descricao do Modulo: Somador
// Biblioteca: Ponto Fixo sinal modulo 32 bits
// Autor: Gilson Jr

module fp32adder
(
        input [31:0] inA,
        input [31:0] inB,
        input clk,
        input clr,
        input enable,
        output reg [31:0] outS,
        output reg carry
);

        always @ (posedge clk) begin
                if (clr == 0) begin
                        carry <= 0;
                        outS <= 0;
                end
                else begin
                        if (enable == 1) begin
                                if (inA[30:0]>=inB[30:0]) begin
                                        if(inA[31]^inB[31]==1) begin
                                                outS[31]<=inA[31];
                                                outS[30:0]<= inA[30:0]-inB[30:0];
                                                carry <=0;
                                        end
                                        else begin
                                                outS[31]<=inA[31];
                                                {carry,outS[30:0]}<= inA[30:0]+inB[30:0];
```

```
                                        end
                                    end
                                    else begin
                                        if(inA[31]^inB[31]==1) begin
                                            outS[31]<=inB[31];
                                            outS[30:0]<= inB[30:0]-inA[30:0];
                                            carry <=0;
                                        end
                                        else begin
                                            outS[31]<=inB[31];
                                            {carry,outS[30:0]}<= inA[30:0]+inB[30:0];
                                        end

                                    end
                            end
                    end
            end
endmodule

// UFPE
// Arquivo: fp32multiplier.v
// Modulo: fp32multiplier
// Descricao do Modulo: Multiplicador
// Biblioteca: Ponto Fixo sinal modulo 32 bits
// Autor: Gilson Jr

module fp32multiplier
(
        input [31:0] inA,
        input [31:0] inB,
        input clk,
        input clr,
        input enable,
        output reg [31:0] outS,
        output wire carry
);

        reg     [15:0]  underflow16;
        reg     [14:0]  overflow15;

        always @ (posedge clk) begin // Variaveis de estado 1
                if (clr == 0) begin
                        outS <= 0;
                        overflow15 <= 0;
                end
                else begin
                        if (enable == 1) begin
                                {overflow15,outS[30:0],underflow16} <= inA[30:0] * inB[30:0];
                                outS[31]<= inA[31]^inB[31];
                        end
                end
        end

        assign carry = |overflow15;

endmodule

// UFPE
// Arquivo: fp32buffer.v
// Modulo: fp32buffer
// Descricao do Modulo: Buffer (guarda um valor para o proximo estado)
// Biblioteca: Ponto Fixo sinal modulo 32 bits
// Autor: Gilson Jr
```

```
module fp32buffer
(
        input [31:0] in,
        input clk,
        input clr,
        input enable,
        output reg [31:0] out
);

        always @ (posedge clk) begin // Variaveis de estado 1
                if (clr == 0) begin
                        out <= 0;
                end
                else begin
                        if (enable == 1) begin
                                out <= in;
                        end
                end
        end
endmodule

// UFPE
// Arquivo: fp32fft5.v
// Modulo: fp32fft5
// Descricao do Modulo: Implementacao da OFFT de 5 pontos
// Biblioteca: Ponto Fixo sinal modulo 32 bits
// Autor: Gilson Jr

`include "fp32adder.v"
`include "fp32multiplier.v"
`include "fp32buffer.v"

module fp32fft5
(
        input [31:0] v0,v1,v2,v3,v4,
        input clk,
        input clr,
        input enable,
        output wire [31:0] Vr0,Vr1,Vr2,Vi1,Vi2,
        output reg ovrf
);

        wire s1OVRout;
        wire s2OVRout;
        wire s3OVRout;
        wire s4OVRout;
        wire s5OVRout;
        wire [31:0] e1[0:4];
        wire [31:0] e2[0:6];
        wire [31:0] e3[0:6];
        wire [31:0] e4[0:7];
        wire [17:0]     carryarray;
        wire v4signal;
        wire v3signal;
        wire e14signal;
        wire e13signal;
        wire e35signal;
        wire e32signal;
        wire e31signal;
        reg [3:0] enablearray;
        reg [3:0] ovrfarray;
```

```
        always @ (posedge clk) begin //controle de estagios
                if (clr == 0) begin // clear
                        enablearray <= 0;
                        ovrfarray <= 0;
                end
                else begin
                        enablearray <= (enablearray << 1)|enable;
                        {ovrf,ovrfarray} <= ({1'b0,ovrfarray} << 1)|
                        {s5OVRout,s4OVRout,s3OVRout,s2OVRout,s1OVRout};
                end
        end

// Estagio 1

        fp32buffer      B1
(
        .in(v0),
        .clk(clk),
        .clr(clr),
        .enable(enable),
        .out(e1[0])
);

        fp32adder A1
(
        .inA(v1),
        .inB(v4),
        .clk(clk),
        .clr(clr),
        .enable(enable),
        .outS(e1[1]),
        .carry(carryarray[0])
);

        assign v4signal = ~v4[31];

        fp32adder A2
(
        .inA(v1),
        .inB({v4signal,v4[30:0]}),
        .clk(clk),
        .clr(clr),
        .enable(enable),
        .outS(e1[2]),
        .carry(carryarray[1])
);

        fp32adder A3
(
        .inA(v2),
        .inB(v3),
        .clk(clk),
        .clr(clr),
        .enable(enable),
        .outS(e1[3]),
        .carry(carryarray[2])
);

        assign v3signal = ~v3[31];

        fp32adder A4
(
```

```
        .inA(v2),
        .inB({v3signal,v3[30:0]}),
        .clk(clk),
        .clr(clr),
        .enable(enable),
        .outS(e1[4]),
        .carry(carryarray[3])
);

// Estagio 2

        fp32buffer      B2
(
        .in(e1[0]),
        .clk(clk),
        .clr(clr),
        .enable(enablearray[0]),
        .out(e2[0])
);

        fp32buffer      B3
(
        .in(e1[1]),
        .clk(clk),
        .clr(clr),
        .enable(enablearray[0]),
        .out(e2[1])
);

        fp32buffer      B4
(
        .in(e1[3]),
        .clk(clk),
        .clr(clr),
        .enable(enablearray[0]),
        .out(e2[2])
);

        fp32buffer      B5
(
        .in(e1[2]),
        .clk(clk),
        .clr(clr),
        .enable(enablearray[0]),
        .out(e2[3])
);

        assign e14signal = ~e1[4][31];

        fp32buffer      B6
(
        .in({e14signal,e1[4][30:0]}),
        .clk(clk),
        .clr(clr),
        .enable(enablearray[0]),
        .out(e2[4])
);

        assign e13signal = ~e1[3][31];

        fp32adder A5
(
```

```
        .inA(e1[1]),
        .inB({e13signal,e1[3][30:0]}),
        .clk(clk),
        .clr(clr),
        .enable(enablearray[0]),
        .outS(e2[5]),
        .carry(carryarray[4])
);

        fp32adder A6
(
        .inA(e1[2]),
        .inB(e1[4]),
        .clk(clk),
        .clr(clr),
        .enable(enablearray[0]),
        .outS(e2[6]),
        .carry(carryarray[5])
);

// Estagio 3

        fp32buffer      B7
(
        .in(e2[0]),
        .clk(clk),
        .clr(clr),
        .enable(enablearray[1]),
        .out(e3[0])
);

        fp32buffer      B8
(
        .in(e2[1]),
        .clk(clk),
        .clr(clr),
        .enable(enablearray[1]),
        .out(e3[1])
);

        fp32buffer      B9
(
        .in(e2[2]),
        .clk(clk),
        .clr(clr),
        .enable(enablearray[1]),
        .out(e3[2])
);

        fp32multiplier M1
(
        .inA(e2[3]),
        .inB(32'h80005cff),
        .clk(clk),
        .clr(clr),
        .enable(enablearray[1]),
        .outS(e3[3]),
        .carry(carryarray[6])
);

        fp32multiplier M2
(
```

```
        .inA(e2[4]),
        .inB(32'h800189f2),
        .clk(clk),
        .clr(clr),
        .enable(enablearray[1]),
        .outS(e3[4]),
        .carry(carryarray[7])
);

        fp32multiplier M3
(
        .inA(e2[5]),
        .inB(32'h00004f1c),
        .clk(clk),
        .clr(clr),
        .enable(enablearray[1]),
        .outS(e3[5]),
        .carry(carryarray[8])
);

        fp32multiplier M4
(
        .inA(e2[6]),
        .inB(32'h80009679),
        .clk(clk),
        .clr(clr),
        .enable(enablearray[1]),
        .outS(e3[6]),
        .carry(carryarray[9])
);

// Estagio 4

//parte real

        fp32buffer      B10
(
        .in(e3[0]),
        .clk(clk),
        .clr(clr),
        .enable(enablearray[2]),
        .out(e4[0])
);

        fp32adder A7
(
        .inA(e3[0]),
        .inB(e3[5]),
        .clk(clk),
        .clr(clr),
        .enable(enablearray[2]),
        .outS(e4[1]),
        .carry(carryarray[10])
);

        assign e35signal = ~e3[5][31];

        fp32adder A8
(
        .inA(e3[0]),
        .inB({e35signal,e3[5][30:0]}),
        .clk(clk),
```

```
        .clr(clr),
        .enable(enablearray[2]),
        .outS(e4[2]),
        .carry(carryarray[11])
);

        fp32adder A9
(
        .inA(e3[1]),
        .inB(e3[2]),
        .clk(clk),
        .clr(clr),
        .enable(enablearray[2]),
        .outS(e4[3]),
        .carry(carryarray[12])
);

        assign e32signal = ~e3[2][31];

        fp32buffer      B11
(
        .in({e32signal,1'b0,e3[2][30:1]}),
        .clk(clk),
        .clr(clr),
        .enable(enablearray[2]),
        .out(e4[4])
);

        assign e31signal = ~e3[1][31];

        fp32buffer      B12
(
        .in({e31signal,1'b0,e3[1][30:1]}),
        .clk(clk),
        .clr(clr),
        .enable(enablearray[2]),
        .out(e4[5])
);

//parte imaginaria

        fp32adder A10
(
        .inA(e3[3]),
        .inB(e3[6]),
        .clk(clk),
        .clr(clr),
        .enable(enablearray[2]),
        .outS(e4[6]),
        .carry(carryarray[13])
);

        fp32adder A11
(
        .inA(e3[4]),
        .inB(e3[6]),
        .clk(clk),
        .clr(clr),
        .enable(enablearray[2]),
        .outS(e4[7]),
        .carry(carryarray[14])
);
```

```
// Estagio 5

        fp32adder A12
(
        .inA(e4[0]),
        .inB(e4[3]),
        .clk(clk),
        .clr(clr),
        .enable(enablearray[3]),
        .outS(Vr0),
        .carry(carryarray[15])
);

        fp32adder A13
(
        .inA(e4[1]),
        .inB(e4[4]),
        .clk(clk),
        .clr(clr),
        .enable(enablearray[3]),
        .outS(Vr1),
        .carry(carryarray[16])
);

        fp32adder A14
(
        .inA(e4[2]),
        .inB(e4[5]),
        .clk(clk),
        .clr(clr),
        .enable(enablearray[3]),
        .outS(Vr2),
        .carry(carryarray[17])
);

        fp32buffer      B13
(
        .in(e4[6]),
        .clk(clk),
        .clr(clr),
        .enable(enablearray[3]),
        .out(Vi1)
);

        fp32buffer      B14
(
        .in(e4[7]),
        .clk(clk),
        .clr(clr),
        .enable(enablearray[3]),
        .out(Vi2)
);

        assign s1OVRout = |carryarray[3:0];
        assign s2OVRout = |carryarray[5:4];
        assign s3OVRout = |carryarray[9:6];
        assign s4OVRout = |carryarray[14:10];
        assign s5OVRout = |carryarray[17:15];

endmodule
```