UNIVERSIDADE DE SÃO PAULO

INSTITUTO DE FÍSICA E QUÍMICA DE SÃO CARLOS

"IMPLEMENTAÇÃO DO ALGORITMO DA FFT-2D PARA REDE DE TRANSPUTERS"

Regina Fumie Eto

Ok

USP / IFQSC / SBI

[redacted]

8-2-[redacted]

Dissertação apresentada ao Instituto de Física e Química de São Carlos, Universidade de São Paulo, para obtenção do Título de Mestre em Física Aplicada

ORIENTADOR: *Prof. Dr. Jan Frans Willem Slaets*

DEPARTAMENTO DE FÍSICA E CIÊNCIA DOS MATERIAIS

SÃO CARLOS

1993

SERVIÇO DE BIBLIOTECA E INFORMAÇÃO - IFQSC
FÍSICA

UNIVERSIDADE DE SÃO PAULO
Instituto de Física e Química de São Carlos

Fone (0162) 72-6222
Fax (0162) 72-2218

Av. Dr. Carlos Botelho, 1465
Caixa Postal 369
CEP 13560.970 - São Carlos - SP
Brasil

MEMBROS DA COMISSÃO JULGADORA DA DISSERTAÇÃO DE MESTRADO DE REGINA FUMIE ITO APRESENTADA AO INSTITUTO DE FISICA E QUIMICA DE SAO CARLOS, DA UNIVERSIDADE DE SAO PAULO, EM 16/02/1993

COMISSAO JULGADORA:

Prof.Dr.Jan Frans Willem Slaets

Prof.Dr.Luciano da Fontoura Costa

Profa.Dra.Agma Juci Machado Traina

Dedico aos,

Meus queridos pais,
Massafumi e Akiko Etto
com amor e carinho.

Agradecimentos

Ao Prof. Dr. Jan Frans Willem Slaets, pela orientação e paciência dispensadas na realização desse trabalho, bem como os ensinamentos transmitidos que muito contribuíram para a minha formação profissional.

Ao Prof. Gonzalo Travieso, pela ajuda e ensinamentos sobre a linguagem OCCAM e das proveitosas discussões sobre o transputer.

Aos professores Valentin, Carlos Antônio pelos ensinamentos que ajudaram no meu aprendizado tanto na graduação como na pós-graduação.

Ao Mateus pela ajuda na escolha do algoritmo utilizado nesse trabalho e pelos conselhos computacionais.

Ao CNPq pelo apoio financeiro.

Aos técnicos do laboratório do GII, Lírio, Ivanilda, Marquinhos e Ailton pela ajuda e serviços prestados, principalmente na confecção dos capítulos da dissertação.

À Benê pela grande amizade, paciência, dedicação e competência na confecção dos desenhos contidos nesta dissertação.

À Lia, secretária do GII, e ao Dante, ambos pela amizade, pela ajuda na manipulação dos recursos do editor WordPerfect e DrawPerfect e também pela ajuda tanto de carater pessoal quanto profissional.

Ao velhos amigos da sala 27, Márcio, Patrícia, João F. que muito me apoiaram nos estudos e também pela amizade, principalmente ao Márcio devido as conversas e os conselhos além das correções ortográficas desse trabalho.

Aos novos amigos João Angelo, João Martins, Marcos, Oswaldo, Bete, Julinha, Márcia, os rapazes de Goiânia, e tantos outros, pelas conversas nas noites de estudo e das companhias nos finais de semana.

Às bibliotecárias especialmente a Sibely, Mara, Neusa, Célia e Ana pelos seus serviços, mas principalmente pela amizade e ajuda pessoal.

Aos meus pais, Massafumi e Akiko e irmãos Edson, Cristina e Eliza, que me apoiaram com carinho e amor em todas as fases da minha vida.

As minhas melhores amigas Fumika Ono e Massami Sakurai pela força e carinho que sempre deram nos meus momentos mais difíceis.

A todos os meus amigos e colegas, que direta ou indiretamente contribuíram com amizade, apoio e auxílio.

SUMÁRIO

Lista de Figuras . . . . . - i -
Lista de Tabelas . . . . . - iii -
Resumo . . . . . - iv -
Abstract . . . . . - v -

Introdução . . . . . - 1 -

Capítulo 1 - Transformada Rápida de Fourier . . . . . - 4 -
1.1 - Introdução . . . . . - 4 -
1.1.1 - Breve histórico da FFT . . . . . - 4 -
1.2 - Algoritmo da FFT . . . . . - 5 -
1.2.1 - Cálculo da DFT . . . . . - 5 -
1.2.2 - Cálculo da FFT . . . . . - 6 -
1.3 - Implementação do algoritmo da FFT . . . . . - 15 -
1.3.1 - "Radix-4" . . . . . - 15 -
1.3.2 - Implementação do algoritmo FFT-2D . . . . . - 19 -

Capítulo 2 - Transputer e Linguagem OCCAM . . . . . - 21 -
2.1 - Introdução . . . . . - 21 -
2.2 - Transputer T800 . . . . . - 21 -
2.2.1 - Unidade de processamento inteiro . . . . . - 22 -
2.2.2 - As instruções de máquina . . . . . - 24 -
2.2.3 - Estrutura da Unidade de processamento de ponto flutuante . . . . . - 25 -
2.3 - Linguagem OCCAM . . . . . - 27 -
2.3.1 - Processos básicos . . . . . - 27 -
2.3.2 - Tipos de Dados Primitivos . . . . . - 28 -
2.3.3 - Processos em OCCAM . . . . . - 31 -
2.3.3.1 - Replicadores . . . . . - 34 -
2.3.3.2 - Prioridades . . . . . - 35 -

2.3.3.3 - Protocolos . . . . . - 36 -
2.3.4 - Paralelização em transputers . . . . . - 37 -
2.3.5 - Rotinas e Funções . . . . . - 38 -

Capítulo 3 - FFT-2D em Transputer: Processamento Sequencial . . . . . - 39 -
3.1 - Introdução . . . . . - 39 -
3.2 - Implementação da FFT-2D sequencial . . . . . - 39 -
3.3 - Implementação da FFT-2D em linguagem montadora . . . . . - 43 -
3.4 - Performance da FFT-2D sequencial . . . . . - 46 -
3.4.1 - Performance para implementação na linguagem OCCAM . . . . . - 47 -
3.4.2 - Performance para implementação com rotinas em linguagem montadora . . . . . - 47 -
3.5 - Resultados . . . . . - 48 -

Capítulo 4 - FFT-2D paralelo em rede de transputer . . . . . - 50 -
4.1 - Introdução . . . . . - 50 -
4.2 - FFT-2D paralela . . . . . - 51 -
4.3 - Implementação da FFT-2D em uma rede de transputers . . . . . - 53 -
4.4 - Desempenho da implementação . . . . . - 56 -
4.4.1 - Utilizando apenas um transputer . . . . . - 57 -
4.4.2 - Utilizando dois transputers . . . . . - 58 -
4.4.3 - Utilizando quatro transputers . . . . . - 59 -
4.5 - Resultados . . . . . - 61 -

Capítulo 5 - Resultados e Conclusões . . . . . - 63 -
5.1 - Resultados . . . . . - 63 -
5.1.1 - Análise do desempenho da FFT-2D sequencial em um transputer . . - 63 -
5.1.2 - Análise do desempenho entre implementação sequencial e paralelo em um transputer . . . . . - 64 -
5.1.3 - Análise do desempenho da FFT-2D em rede de transputer . . . . . - 65 -
5.2 - Conclusão . . . . . - 71 -

Referências Bibliográficas . . . . . - 73 -

Apêndice A - Listagem do programa sequencial A-1

Apêndice B - Rotinas das "butterflys" em linguagem montadora B-1

Apêndice C - Listagem do programa paralelo C-1

SERVIÇO DE BIBLIOTECA E DOCUMENTAÇÃO - IFQSC
FÍSICA

Lista de Figuras

Figura 1.1 - Diagrama de Fluxo para N = 8 . . . . . - 10 -

Figura 1.2 - Representação de ω para N=8 . . . . . - 11 -

Figura 1.3 - Diagrama para operação "bit-reverse" para N = 8. . . . . - 13 -

Figura 1.4 -Representação de uma operacao "butterfly" . . . . . - 14 -

Figura 1.5 - Diagrama de Fluxo simplificado para N = 16 . . . . . - 18 -

Figura 1.6 - Imagem bidimensional (linhas X colunas) . . . . . - 19 -

Figura 2.1 - Arquitetura do T800 . . . . . - 22 -

Figura 2.2 - Funcionamento da pilha de avaliação . . . . . - 23 -

Figura 2.3 - Funcionamento da pilha de avaliação em ponto-flutuante. . . . . - 26 -

Figura 3.1 - Esquema da implementação do algoritmo da FFT. . . . . - 40 -

Figura 3.2 - Imagem de um corte de uma laranja obtida por RMN. . . . . - 40 -

Figura 3.3 - Diagrama de Blocos do método Radix-4 unidimensional. . . . . - 41 -

Figura 3.4 - Controlador e FAZFFT. . . . . - 46 -

Figura 3.5 - Tempo de processamento da FFT-2D sequencial . . . . . - 49 -

Figura 4.1 - Os quatro "links" de um transputer. . . . . - 52 -

Figura 4.2 - Topologias possíveis com transputers. . . . . - 52 -

Figura 4.3 - Implementação da FFT-2D numa rede de transputers. . . . . - 54 -

Figura 4.4 - Topologia utilizada na implementação. . . . . - 55 -

Figura 4.5 - Atribuição de tarefas para cada transputer. . . . . - 56 -

Figura 4.6 - Configuração para um transputer . . . . . - 57 -

Figura 4.7 - Configuração para dois transputer . . . . . - 58 -

Figura 4.8 - Configuração para quatro transputers . . . . . - 60 -

Figura 4.9 - Tempo de processamento para um, dois e quatro transputers. . . . . - 62 -

Figura 5.1 - Tempo de processamento da FFT-2D para um transputer. . . . . - 65 -

Figura 5.2 - Tempo de processamento paralelo da FFT-2D em um, dois e quatro transputers. . . . . - 67 -

Figura 5.3 - "Speedup" para dois e quatro transputers. . . . . - 69 -

Figura 5.4 - Eficiência para a rede com dois e quatro transputers. . . . . - 71 -

Lista de Tabelas

Tabela 1.1 - Número de computações requeridas. . . . . - 9 -

Tabela 2.1 - Registradores inteiros de 32 bits . . . . . - 23 -

Tabela 2.2 - Operadores básicos do T800 . . . . . - 25 -

Tabela 2.3 - Registradores adicionais para Unidade de Ponto flutuante. . . . . - 26 -

Tabela 2.4 - Processos primitivos . . . . . - 28 -

Tabela 2.5 - Tipos de dados primitivos . . . . . - 29 -

Tabela 2.6 - Classes de construção de processos. . . . . - 32 -

Tabela 3.1 - Tempo de processamento para o algoritmo puramente em linguagem OCCAM. . . . . - 47 -

Tabela 3.2 - Tempo de processamento para o algoritmo com rotinas implementadas em linguagem montadora. . . . . - 48 -

Tabela 4.1 - Tempo de processamento da FFT-2D para 1 transputer. . . . . - 58 -

Tabela 4.2 - Tempo de processamento da FFT-2D para 2 transputers. . . . . - 59 -

Tabela 4.3 - Tempo de processamento da FFT-2D para 4 transputers. . . . . - 61 -

Tabela 5.1 - Redução no tempo de processamento em % devido a adoção de rotinas em linguagem montadora numa implementação sequencial em linguagem OCCAM. . . . . - 64 -

Tabela 5.2 - Tempo de processamento da FFT-2D sequencial e paralelo . . . . . - 64 -

Tabela 5.3 - Tempo de processamento paralelo da FFT-2D. . . . . - 66 -

Tabela 5.4 - "Speedup" para 2 e 4 transputers. . . . . - 68 -

Tabela 5.5 - Eficiência para 2 e 4 transputers. . . . . - 70 -

Resumo

O presente trabalho descreve a implementação do algoritmo discreto da FFT-2D, numa rede de transputers.

Primeiramente a implementação sequencial do algoritmo é analisado, em seguida são apresentados algumas técnicas de paralelização, bem como sua aplicação no algoritmo da FFT-2D.

Finalmente são apresentados os resultados do desempenho obtido por redes compostas de um, dois e quatro transputers.

Abstract

The present work describes the implementation of the discrete FFT-2D algorithm in a distributed transputer network.

First a sequential implementation of the algorithm is presented.

Then some parallelization techniques are analyzed and applied to the FFT-2D algorithm.

Finally the obtained performance is presented for networks containing one, two and four transputers.

Introdução

A Transformada de Fourier tem sido amplamente utilizada em diversas áreas, principalmente na física, em estudo teórico de simulações e métodos numéricos, e também para pesquisas, principalmente na área de processamento de sinais que é utilizada para as mais variadas aplicações como na medicina , astronomia entre outras áreas.

Atualmente o avanço tecnológico tem contribuído no aperfeiçoamento das técnicas computacionais e na tecnologia de construção dos circuitos integráveis. Na construção desses circuitos, (os quais estão se tornando cada vez menores e mais velozes ao longo do tempo), destaca-se principalmete a construção de processadores e de memórias mais eficientes, permitindo uma ampla utilização desses componentes na Instrumentação Eletrônica, principalmente na área de processamento digitais de sinais - Digital Signal Processing - (DSP) [OPP-75][DUD-84], que tornou a solução para muitos problemas de implementação de sistemas em tempo-real, fazendo com que processadores dedicados ao cálculo da FFT nesses instrumentos se tornem cada vez mais corriqueiros a baixos custos e mais velozes também.

Mas com a velocidade de processamento de certos algoritmos nesses processadores vem chegando ao seu limite, devido as restrições físicas, onde a maior velocidade é sempre inferior ao da velocidade da luz, deste modo houve uma necessidade crescente por uma maior velocidade nos cálculos para superar tanto as restrições no tempo quanto para o tamanho do conjunto de dados analisados e executados por sistemas convencionais, possuindo um único processador.

Assim pesquisadores vem colocando seus esforços na área de exploração de paralelismo, tanto a nível de hardware, quanto de software, surgindo dessas pesquisas várias máquinas paralelas dedicadas e também de propósitos gerais, tal como o hipercubo [SEI-85], a estrutura dataflow [GUR-85] e outros [HOC-88], visando explorar o máximo paralelismo em muitos algoritmos.

No Grupo de Instrumentação e Informática (GII) do Instituto de Física e Química de São

SERVIÇO DE BIBLIOTECA E INFORMAÇÃO - IFQSC
FÍSICA

Carlos (IFQSC-USP), foram desenvolvidos vários trabalhos, onde explorou-se tanto a parte de desenvolvimento de hardware, utilizando técnicas de DSP em processamento digitais de imagens, quanto para o desenvolvimento e implementação de software dedicados.

Dentre estes trabalhos o precursor foi o trabalho de [COS-86], onde foi explorado o processador TMS32010, da Texas Instruments, para efetuar os cálculos da Transformada de Fourier para um espectrofotômetro infravermelho desenvolvido pelo Grupo de Ótica do IFQSC-USP, utilizando conceitos e técnicas de DSP.

Após este trabalho, o GII em conjunto com o Grupo de Ressonância do IFQSC-USP realizaram vários trabalhos visando utilizar as técnicas de digitalização e visualização de imagens, trabalhando no desenvolvimento de arquiteturas e softwares dedicados.

O primeiro trabalho foi o de [TOR-88], onde foi desenvolvido um digitalizador e premedidor de dois canais por Tomografia de Ressonância Magnética Nuclear (ToRM), utilizando o método da Transformada de Fourier Bidimensional (FFT-2D), escolhido para servir de base na definição das características do digitalizador por ser mais frequentemente utilizado em ToRM.

Outros dois trabalhos que seguiram e praticamente realizados conjuntamente são os trabalhos de [PAI-90] e [TRA-91], onde [PAI-90] consistiu de um projeto de uma arquitetura de hardware para a visualização de imagens digitais, onde foi explorado o processador TMS34010, ressaltando que no processador digital de imagens as operações como filtragem espacial, detecção de borda e aplicação de algoritmos complexos para a modificação da imagem são realizados por um sistema de processamento dedicado e não pelo sistema gráfico, este último a base do trabalho de [TRA-91], onde foi realizado um estudo e uma implementação de software dedicado para um sistema de visualização de imagens, onde além de possibilitar a visualização da imagem, são realizados as operações gráficas, como operações lógicas e aritméticas entre imagens, zoom, operações com janelas, alterações de cores da imagem, animação, etc.

Um específico problema enfrentado pelo Grupo de Ressonância do IFQSC-USP na reconstrução de imagens, é o tempo de processamento que um único processador apresenta, principalmente nas técnicas de aquisição rápida de dados, o fator limitante para a visualização imediata da imagem analisada.

Assim, o presente trabalho visa explorar o paralelismo tanto a nível de hardware quanto de software, a fim de avaliar uma futura implementação para o tomógrafo de RMN desenvolvido

em nosso Instituto, para tanto foi utilizado um algoritmo paralelo para a FFT-2D, de tal forma que este possa ser executado por uma máquina paralela do tipo "Multiple Instruction stream-Multiple Data stream", MIMD, para tentar sanar o problema da visualização rápida da imagem.

Na parte implementada para hardware foi utilizada uma interface da MICROWAY QUADPUTER disponível no Grupo de Instrumentação e Informática (GII) do Instituto de Física e Química de São Carlos, onde sua arquitetura é baseada em transputers do tipo T800, escolhido devido às facilidades intrínsecas deste circuito integrado em implementar as comunicações entre os processadores e também pela possibilidade de ter como computador hospedeiro um micro tipo PC-XT. E na implementação de software paralelo dedicado ao algoritmo da FFT-2D foi utilizado a linguagem OCCAM2 que é uma linguagem de alto nível especialmente adaptada para explorar o paralelismo do transputer, baseado nos conceitos de processos comunicantes introduzido por [HOA-78].

Essa dissertação está dividida em 5 capítulos, sendo que no:

• Capítulo 1 - É apresentado o algoritmo da FFT, partindo da Transformada de Fourier sendo discretizada para obter a DFT, que representa o algoritmo computacional para números discretos de amostragem. Apresentamos também as variações do algoritmo e as razões para adotar o método "Radix-4" para FFT-2D utilizado nesse trabalho.

• Capítulo 2 - Apresenta a arquitetura do Transputer, quanto a sua estrutura e também uma breve apresentação da linguagem OCCAM, que foram utilizados para as implementações das redes de transputers.

• Capítulo 3 - É mostrado a implemetação da FFT-2D realizada para o processamento sequencial, utilizando a linguagem OCCAM e uma outra implementação com apenas algumas rotinas do cálculo das "butterflys" em linguagem montadora.

• Capítulo 4 - É apresentado a implementação para as redes de transputers do algoritmo da FFT-2D, utilizando os "links" (canais externos) de comunicação entre transputers, para tamanhos diferentes da matriz de dados e também para diferentes números de transputers na rede.

• Capítulo 5 - São apresentados os resultados obtidos dos tempos de processamento da FFT-2D na execução sequencial e das topologias implementadas para formação da rede de transputers, bem como seu desempenho em função do número de transputer na rede.

Capítulo 1
Transformada Rápida de Fourier

1.1 - Introdução.

A Transformada Rápida de Fourier, no inglês "Fast Fourier Transform" (FFT), desde a sua descoberta tem sido aplicada em vários campos, especialmente em processamento de sinais e de imagens em tempo real.

Nesse capítulo é abordado a FFT, quanto a sua história, algoritmos e demandas computacionais, tanto para o caso da transformada unidimensional quanto para o caso bidimensional, sendo esta última a base desse trabalho.

No decorrer desse trabalho, são utilizadas algumas palavras conhecidas pela comunidade computacional, bem como os da comunidade física que trabalham com transformadas, principalmente com a transformada de Fourier.

1.1.1 - Breve histórico da FFT.

Em 1965, Cooley e Tuckey publicaram o artigo "An Algorithm for the Machine Calculation of Complex Fourier Séries" [COO-65], que revolucionou os conceitos de cálculo numérico da época, pois reduzia a quantidade de operações aritméticas para a "Discrete Fourier Transform" (DFT).

Segundo [COO-69] a idéia de agilizar os cálculos da DFT partiu do pesquisador Garwin, que necessitava de um método mais rápido para o processamento da DFT, assim Garwin uniu Tuckey, um "expert" em técnicas de cálculo da FT, e Cooley um membro do alto escalão da IBM na época, surgindo portanto desse encontro o primeiro algoritmo computacional (algoritmo CT), que reduziu os cálculos da DFT até então conhecidos de $O(N^2)$

para $O(N\log_2 N)$ operações aritméticas.

Uma obsevação à parte, é que não foi encontrado nos artigos de [COO-65], nem em [BRI-74], qualquer referência da origem das idéias de Tuckey apresentadas a Garwin, mas vale ressaltar que outros trabalhos sobre FT estavam sendo realizados na mesma época, e alguns, com métodos semelhantes ao de CT, como o de Rudnick [RUD-66], o qual relata que se baseou nas publicações de Danielson [DAN-42] e este por sua vez referia-se a uma publicação de Runge [RUN-64].

Segundo [BRI-74], essas duas últimas referências apresentavam as vantagens computacionais do algoritmo da FFT como o conhecemos atualmente. Outros pesquisadores que também podemos citar são : Thomas [THO-63] que utilizou técnica semelhante ao de CT e referenciava Stumpff [STU-39] para o cálculo da série de Fourier; Yates [YAT-37] que calculou a interação de 2^n fatorial, e vários outros não muito conhecidos que são apresentados em [HOC-88].

Posteriormente, os pesquisadores que trabalham com a FFT, visando obter maior produtividade e eficiência, têm colocado seus esforços em dois aspectos:

- no projeto de algoritmos efetivos
- no desenvolvimento de arquiteturas mais velozes para o cálculo da FFT

Atualmente a velocidade de processamento da FFT, em vários tipos de arquiteturas tem sido utilizada como um importante parâmetro na determinação da performance das mesmas.

1.2 - Algoritmo da FFT.

Antes de introduzir o algoritmo da FFT, será explicado rapidamente como se realiza os cálculos da DFT, para melhor compreensão do algoritmo.

1.2.1 - Cálculo da DFT.

Seja f(k,l) uma função definida sobre um plano infinito então a sua FT é:

$$F(u,v) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(k,l)\, e^{-1(uk+lv)}\, dk\, dl \qquad (1.1)$$

$$onde \quad 0 \le u \quad e \quad v < \infty$$

E se agora assumindo que f(k,l) é uma função de N dados complexos a serem transformados e F(u,v) um conjunto de N amplitudes de complexos harmônicos resultantes da transformação, então sua definição fica:

$$F(u,v) = \sum_{k=0}^{N-1}\sum_{l=0}^{N-1} f(k,l)\, w_N^{(uk-vl)} \qquad (1.2)$$

$$onde 0 \le k \le N-1$$

E a Transformada inversa de Fourier é definida por:

$$f(k,l) = \frac{1}{N^2}\sum_{u=0}^{N-1}\sum_{v=0}^{N-1} F(u,v)\, e^{j\frac{2\pi}{N}(uk+vl)} \qquad (1.3)$$

$$onde 0 \le u \le N-1$$

Pode-se verificar que para o processamento de DFT, dada pela Equação(1.2), este requer um número de multiplicações e de adições complexas proporcional a N^2. Isto é facilmente observado, notando-se que para cada N valores de k, a expansão da somatória requer N multiplicações complexas de f(k,l) por ω_N^{uk} e (N-1) adições para o resultado.

1.2.2 - Cálculo da FFT.

A FFT é simplesmente um algoritmo que calcula a DFT para um conjunto de N dados discretos e que está baseado no método da "duplicações sucessivas", onde a DFT é desmembrada em várias DFT menores, para simplificar os cálculos.

Então partindo da Equação(1.2) e chamando

$$W = e^{-i\frac{2\pi}{N}} \tag{1.4}$$

e

$$W = \begin{matrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega & \omega^2 & \dots & \omega^{N-1} \\ . & . & . & \dots & . \\ 1 & \omega^{N-1} & \omega^{2(N-1)} & \dots & \omega^{(N-1)(N-1)} \end{matrix} \tag{1.5}$$

onde nota-se que:

$$W_{uk} = \omega^{uk} \tag{1.6}$$

Desta forma a Equação(1.2) pode ser escrita na forma de matriz por:

$$F = WfW^{-1} \tag{1.7}$$

onde W^{-1} é a matriz transposta de W. Assim a transformada é obtida pela multiplicação de todas as linhas de W por f(k,l) e depois todas as colunas por W^{-1}.

Para simplificar os cálculos, será utilizado a FFT-1D(unidimensional), sendo que todas as considerações serão válidas também para o caso bidimensional.

Como o algoritmo de CT está estruturado na base 2 para obter a máxima eficiência computacional sobre um conjunto de N pontos, onde N é expresso como uma potência inteira de dois, pode-se assim definir um número inteiro n como:

$$n = \log_2 N \qquad ou \qquad N = 2^n \tag{1.8}$$

Desta forma a Equação(1.2) pode ser separada em outras duas funções menores, onde uma função conterá apenas os pontos pares e a outra apenas pontos ímpares, assim o número de pontos de cada função terá N/2 a serem calculados e redefinindo os valores de k tem-se que:

$$\begin{cases} k=2m & para\ k\ par \\ k=2m+1 & para\ k\ impar \end{cases} \tag{1.9}$$

Substituindo a Equação(1.9) em (1.2) obtem-se

$$F(u) = \sum_{m=0}^{\frac{N}{2}-1} f(2m)\, W_N^{2mu} + W_N^{u} \sum_{m=0}^{\frac{N}{2}-1} f(2m+1)\, W_N^{2mu} \qquad (1.10)$$

mas

$$W_N^2 = W_{\frac{N}{2}} \qquad (1.11)$$

Consequentemente a Equação(1.10) fica:

$$F(u) = \sum_{m=0}^{\frac{N}{2}-1} f(2m)\, W_{\frac{N}{2}}^{um} + W_N^{u} \sum_{m=0}^{\frac{N}{2}-1} f(2m+1)\, W_{\frac{N}{2}}^{um} \qquad (1.12)$$

Podemos definir que a primeira somatória compreende apenas os pontos pares e a segunda somatória apenas os pontos ímpares, onde:

$$\begin{aligned} F_{par}(u) &= \sum_{m=0}^{\frac{N}{2}-1} f(2m)\, W_{\frac{N}{2}}^{um} \\ F_{impar}(u) &= \sum_{m=0}^{\frac{N}{2}-1} f(2m+1)\, W_{\frac{N}{2}}^{um} \end{aligned} \qquad (1.13)$$

então a Equação(1.12) fica:

$$F(u) = F_{par}(u) + W_N^{u}\, F_{impar}(u) \qquad (1.14)$$

que é a equação fundamental do algoritmo de CT.

Como foi dito anteriormente a FFT é baseada no método das "duplicações sucessivas", então a Equação(1.14) é quebrada em várias outras funções pares e ímpares, isto é, F_{par} é quebrada em outra função par e outra função ímpar, e o mesmo procedimento para a F_{impar}, e assim sucessivamente até que não seja mais possível quebrar as funções, ou seja até chegarmos em uma potência menor que dois.

Então o custo operacional é de uma adição complexa e de uma multiplicação complexa para cada processamento menor da transformada e tem valor de $\log_2 N$.

Assim há N/2 componentes intermediários com N adições complexas e N multiplicações complexas, para F_{par} e para F_{impar}. Então, a primeira vista, o número de operações é de $(N/2)\log_2 N$ adições complexas mais $(N/2)\log_2 N$ multiplicações complexas, porém pela simetria e pelo fato de que os termos ω^{uk} serem únicos, o número de cálculos complexos pode ser reduzido pela metade ou até mais.

Pode-se então dizer que a quantidade dos cálculos requeridos no algoritmo de CT é proporcional a $N\log_2 N$.

Uma proporção aproximada do número de cálculos computacionais requeridos pelo algoritmo da FFT para o número de cálculo necessário na forma DFT é dado pela Tabela 1.1.

Tabela 1.1 - Número de computações requeridas.

N	$n(DFT)=N^2$	$n(FFT)=N\log_2 N$	n(FFT)/n(DFT)
16	256	64	0.250
32	1024	160	0.156
64	4096	384	0.0938
128	16.384	896	0.0547
256	65.536	2048	0.0312
512	262.144	4608	0.0176
1024	1.048.576	10.240	0.0098
2048	4.194.304	22.528	0.0054
4096	16.777.216	49.152	0.0029

Assim verifica-se que o algoritmo de FFT tem uma boa performance e que agiliza os cálculos da DFT, principalmente para conjuntos maiores de dados.

Existem algumas variações no algoritmo da FFT, como descrita em [OPP-75], onde esses algoritmos podem ser classificados como :

- "in-place"
- "i/o natural"

Em um algoritmo "in-place" um dado é armazenado no mesmo local anteriormente lido, o que reduz o armazenamento total requerido para a memória interna. Infelizmente o uso do algoritmo "in-place" resulta numa ordenação dos dados de saída que não a normal, isto é, é necessário um rearranjo nos dados de entrada antes destes serem armazenados na memória

interna.

E para um algoritmo "i/o natural", onde tanto o programa quanto os cálculos dos dados realizados são armazenados todos em vetores distintos, isso requer muita memória interna da máquina, o que torna crítico o espaço de memória.

Como nesse trabalho está baseado no processamento da FFT-2D onde a imagem analisada requer um número excessivo de cálculos, temos que a memória disponível na máquina é limitada, então é adotado o algoritmo "in-place", sendo que para o rearranjo dos dados de entrada faremos a operação de "bit-reverse" o qual será explicado mais detalhadamente no decorrer dessa seção.

Devido a complexidade do processamento da Equação(1.12), utilizaremos a representação do mesmo segundo o diagrama de fluxo, o qual facilita a compreensão do algoritmo da FFT, como mostrado na Figura 1.1.

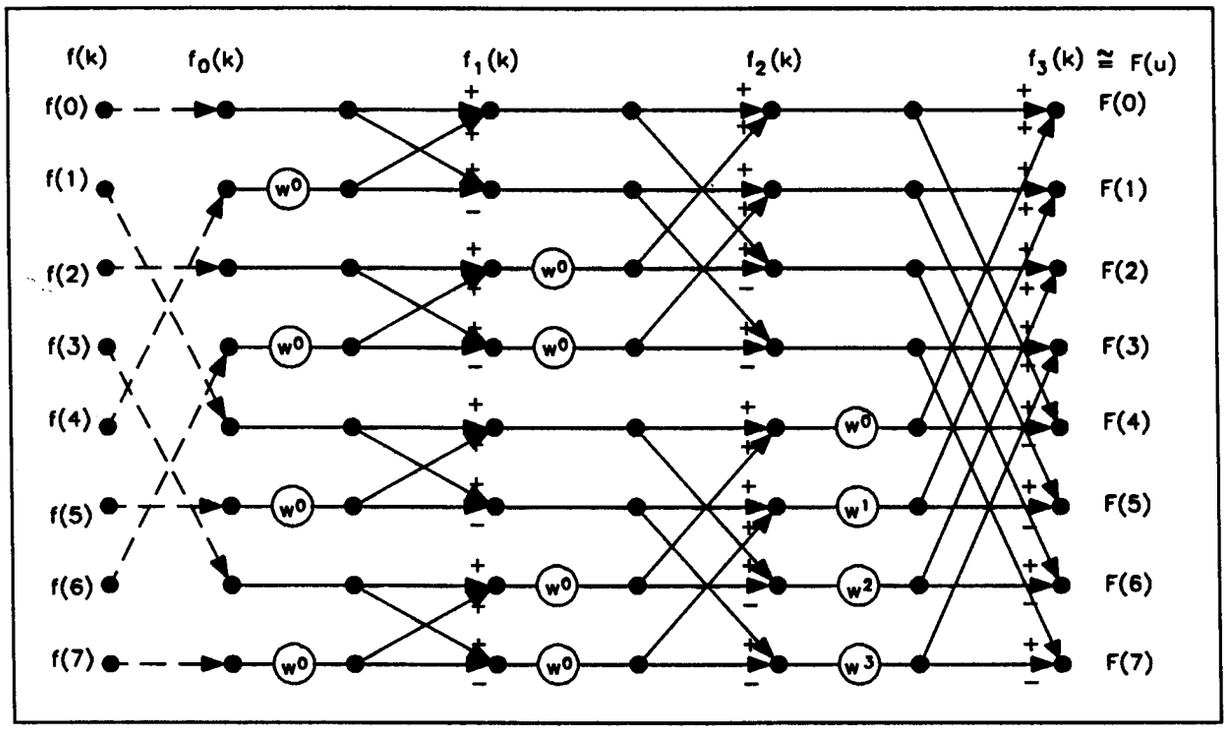

Figura 1.1 - Diagrama de Fluxo para N = 8

Assim, a Figura 1.1 apresenta o método de CT para N=8, com fatores de ω já calculados, inscritos nos círculos da figura.

Esses fatores ω são casos especiais onde:

SERVIÇO DE BIBLIOTECA E DOCUMENTAÇÃO - IFGSC
FISICA

$$W^{k}=\begin{cases} e^{(-2\pi i/N)k} & \textit{para transformada direta} \\ e^{(2\pi i/N)k} & \textit{para transformada inversa} \end{cases} \qquad (1.15)$$

Assim, ao fazer a decomposição do exponente da Equação(1.15) em função de senos e cos-senos obtem-se:

$$w^{k}=\cos\theta - i\sin\theta \qquad \textit{onde} \quad \theta=(2\pi\frac{k}{N}) \qquad (1.16)$$

Verifica-se que a representação dos valores ω^k no plano dos complexo correspondem aos números complexos de módulo 1 e de fase $2\pi k/N$ radianos, ou seja correspondem a N_ésimas raízes complexas da unidade.

A Figura 1.2 ilustra os pontos de ω^k para N=8

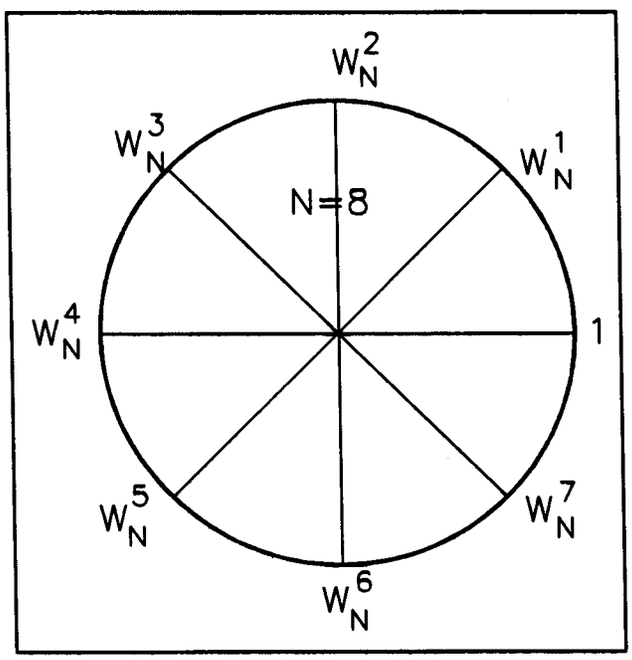

Figura 1.2 - Representação de ω para N=8

Pode-se verificar da Figura 1.2 que

$$\begin{aligned} W^{k+N} &= W^{k} \\ W^{k+\frac{N}{2}} &= -\,W^{k} \end{aligned} \qquad (1.17)$$

Assim no caso da Figura 1.1, quando k=0 então $\omega^4 = -\omega^0$ e assim por diante.

Da Figura 1.1, há cincos colunas denominadas de $f_0(k)$, $f_1(k)$, $f_2(k)$, $f_3(k)$ e F(u) onde k=0..7 e u=0..7, sendo que a primeira coluna representa os f(k) pontos complexos a serem transformados, e $f_0(k)$ é obtida pela troca entre os elementos de f(k), segundo a operação "bit-reverse".

As colunas $f_1(k)$, $f_2(k)$, $f_3(k)$ consistem do resultado intermediário do método de CT e tem um custo operacional de $\log_2 N$ (nosso exemplo N=8 temos três fases porque $\log_2 8=3$) e a última coluna F(u) o resultado da transformação.

A operação "bit-reverse" consiste em rearranjar os dados de tal forma que os dados são separados em índices pares e ímpares, para tanto, os índices são colocados em notação binária.

Para realizar a operação "bit-reverse", deve-se simplesmente mudar os seus bits em binário, onde o bit menos significativo ocupará o local do bit mais significativo, seguindo o mesmo método para os bits intermediários.

Para melhor compreensão é dada uma representação gráfica na Figura 1.3 do procedimento realizado.

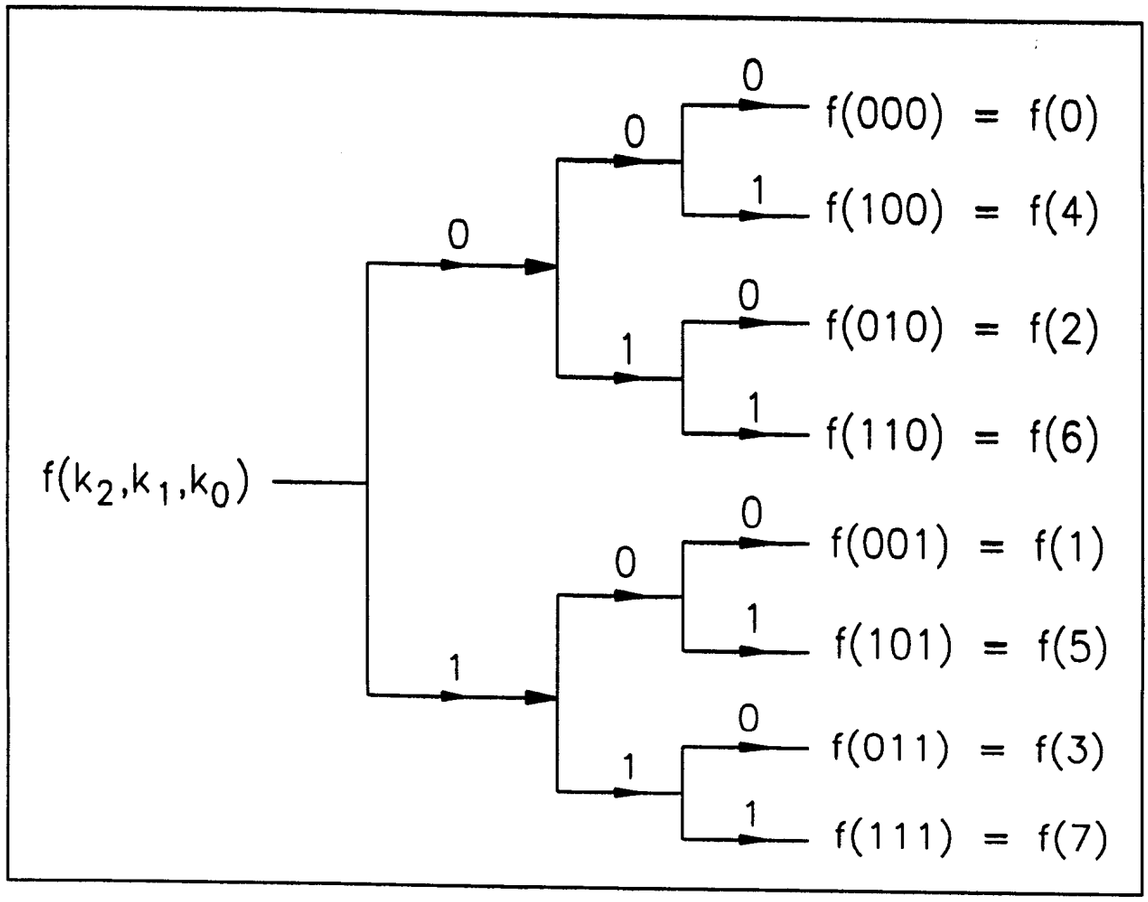

Figura 1.3 - Diagrama para operação "bit-reverse" para N = 8.

Um simples exemplo seria: para k=1 e N=8 então n=3.

k = 1 ===> k = 0 0 1 em binário

o valor que estiver em a_0 vai para a_2 e o valor de a_1 não é alterado. Assim k=1 passa a ser k=4.

a_2	a_1	a_0
0	0	1

a_2	a_1	a_0
1	0	0

E para o processamento de cada fase intermediária, mostrada na Figura 1.1, é aplicado uma operação de combinação sobre dois pontos de cada coluna $f_n(k)$, obtendo outros dois pontos da coluna seguinte $f_{n+1}(k)$ conhecida como operação "butterfly", também conhecido com o nome de "butterfly" base2, como mostra a Figura 1.4.

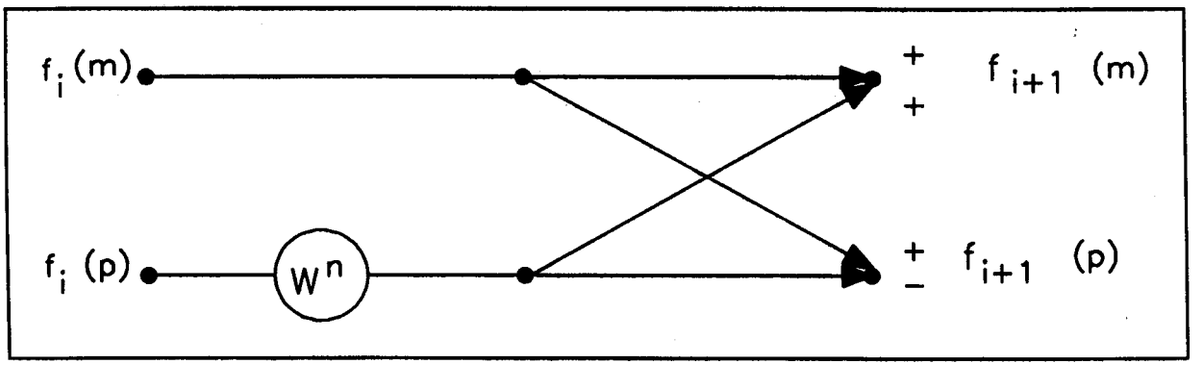

Figura 1.4 -Representação de uma operacao "butterfly"

E em cada coluna intermediária aplica-se N/2 vezes o operador "butterfly".

Para entender melhor o significado do operador "butterfly" é dada a sua forma aritmética complexa:

$$\begin{aligned} f_{i+1}(m) &= f_i(m) + f_i(p)\, W^n \\ f_{i+1}(p) &= f_i(m) - f_i(p)\, W^n \end{aligned} \qquad (1.18)$$

Onde os valores de ω^n de cada "butterfly" são calculados para cada coluna intermediária $f_n(k)$, considerando-se que k esteja sendo representado em binário com $\log_2 N$ bits.

Para o cálculo de n, realiza-se os seguintes passos:

(i) desloca-se k bits para a direita de $\log_2 N$ - i bits

(ii) reverte o resultado encontrado em (i), ou seja, colocamos seus bits de trás para frente.

(iii) desloca-se o resultado encontrado em (ii) para a direita de 1 bit.

Para melhor compreensão desses passos é dado um pequeno exemplo, onde para encontrar o valor de n de $f_2(3)$ segue-se os seguintes passos:

Sendo i=2 e k = 3 ou seja k = 0 1 1 (em binário)

(i) $\log_2 8 = 3$

$\log_2 8 - 2 = 1$ - desloca-se k de 1 bit à direita

k = 0 0 1

(ii) k = 1 0 0 - revertendo o resultado de k

(iii) k = 0 1 0 - deslocando 1 bit à direita

Assim para n = 2 , $f_2(3)$ terá ω^2 e o mesmo procedimento é realizado para o restante dos cálculos de n.

1.3 - Implementação do algoritmo da FFT.

O método de CT apresentado na seção 1.2 é atualmente conhecido como "Radix-2". Outros métodos "Radix" surgiram, tais como "Radix-4", "Radix-8" e "Radix-16" e são utilizados principalmente para processos específicos e são apresentados com mais detalhes em [BRI-74] e em [DUD-84].

Para esse trabalho foi utilizado o método "Radix-4", pois obteve um custo operacional menor que o método "Radix-2" para a matriz de dados analisadas, onde para demonstrar esse fato é dado uma simples análise do número de operações requeridas por cada um.

Assim para o método "Radix-2" cada "butterfly" tem um custo operacional de:

$$\frac{N}{2}\log_2 N(2\ somas\ +\ 1\ produto)$$

onde ao se considerar N=256 o custo operacional de uma "butterfly" será de 2.048 somas mais 1024 produtos.

Já com o método de "Radix-4" o custo de uma "butterfly" é dada por:

$$\frac{N}{4}\log_2 N(8\ somas\ +\ 3\ produtos)$$

tal que para N=256 o custo operacional será de 2.048 somas mais 768 produtos.

Assim há uma redução nos cálculos das multiplicações complexas, portanto obtem-se também uma redução no tempo dos cálculos da transformada.

1.3.1 - "Radix-4".

É apresentado a seguir o algoritmo "Radix-4" (algoritmo da FFT na base 4). Para tanto é desejável que u e k sejam representados em notação binária, assim os índices n e k são representados na base 4 ou no sistema numérico quaternário:

$$\begin{aligned} u &= 4u_1 + u_0 \qquad & u_1, u_0 = 0,1,2,3 \\ k &= 4k_1 + k_0 \qquad & k_1, k_0 = 0,1,2,3 \end{aligned} \tag{1.19}$$

Como exemplo, é adotado o valor de N igual a 16, então:

$$F(u_1, u_0) = \sum_{k_0=0}^{3} \left[\sum_{k_1=0}^{3} f_0(k_1, k_0)\, W^{4uk_1} \right] W^{uk_0} \tag{1.20}$$

Reescrevendo os valores de ω^{4uk_1}

$$\begin{aligned} W^{4uk_1} &= W^{4(4u_1+u_0)k_1} \\ &= [W^{16}]^{u_1k_1} * W^{4u_0k_1} \\ &= W^{4u_0k_1} \end{aligned} \tag{1.21}$$

desde que $\omega_{16}^{16} = 1$

O símbolo " * " da Equação(1.21) é definido como convolução, para maiores detalhes sobre esta operação recomenda-se a leitura de [BRI-74].

Assim, se chamar

$$f_1(u_0, k_1) = \sum_{k_1=0}^{3} f_0(k_1, k_0)\, W^{4u_0k_1} \tag{1.22}$$

o lado esquerdo da Equação(1.20) pode ser expressado por:

$$f_2(u_0, u_1) = \sum_{k_0=0}^{3} f_1(u_0, k_0)\, W^{(4u_1+u_0)k_0} \tag{1.23}$$

Então a transformada na base 4 dará o seguinte resultado:

$$F(u_1, u_0) = f_2(u_0, u_1) \tag{1.24}$$

Portanto as Equações 1.22, 1.23 e 1.24 definem o algoritmo "Radix-4" para N = 16.

Baseado nessas equações, é apresentado o diagrama de fluxo na base 4 mostrado na Figura 1.5 .

SERVIÇO DE BIBLIOTECA E INFORMAÇÃO - IFQSC
FÍSICA

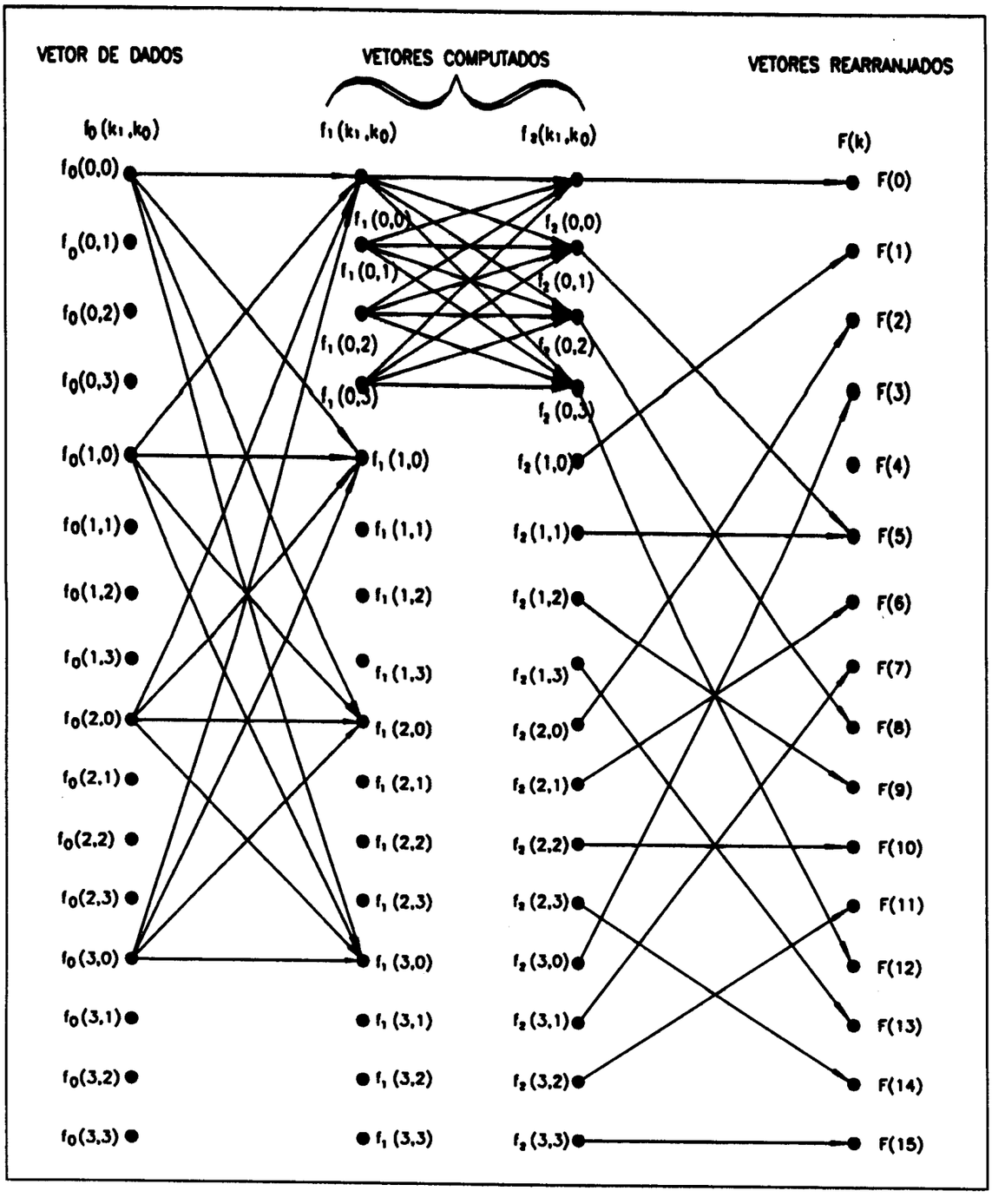

Figura 1.5 - Diagrama de Fluxo simplificado para N = 16

Assim o algoritmo "Radix-4" pode ser descrito como:

Rotina "Radix-4"

lê vetores de dados complexos

Rotina "bit-reverse"

Rotina de transformação - chamada de cool2

Fim da ROTINA "Radix-4".

1.3.2 - Implementação do algoritmo FFT-2D.

Para o algoritmo da FFT-2D, foi utilizado o método "Radix-4", explicado anteriomente, para as linhas da matriz e depois para as colunas da matriz, simplesmente quebrando a FFT-2D em várias FFT-1D (unidimensionais). Isto é possível, fazendo a FFT-1D primeiramente para as linhas e depois para as colunas de dados da matriz.

Exemplificando, se tivermos uma imagem bidimensional, então podemos separar os dados da imagem em linhas e colunas, como mostrado na Figura 1.6.

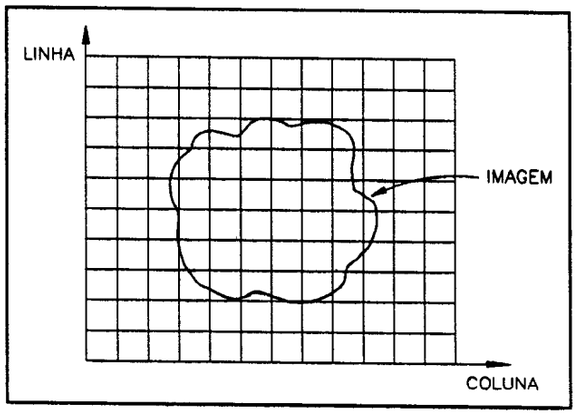

Figura 1.6 - Imagem bidimensional (linhas X colunas).

Onde se obtem a transformada total da imagem, após realizar as transformadas das linhas e depois das colunas.

Assim o algoritmo da FFT-2D fica:

Rotina FFT-2D

Para linha = 0 até linha = N-1

fftunidimensional

transpõe a matriz de dados

Para coluna = 0 até coluna = N-1

fftunidimensional

Fim da Rotina FFT-2D.

O custo operacional é simplesmente o custo de todas as linhas da transformada vezes o custo de todas as colunas, portanto é de aproximadamente $N^2 \log_2 N$.

A implementação do algoritmo FFT-2D em linguagem OCCAM2, realizada para transputers é apresentado no Apêndice-A.

Capítulo 2
Transputer e Linguagem OCCAM

2.1 - Introdução.

Nesse capítulo apresentaremos as características básicas do transputer e da linguagem OCCAM, onde o transputer foi desenvolvido pela INMOS Ltd, para executar eficientemente a linguagem OCCAM.

2.2 - Transputer T800.

As especificações básicas do transputer T800 são dadas pelos seguintes elementos apresentados abaixo e ilustrado na Figura 2.1:

- um processador inteiro de 32 bits,
- um processador de ponto flutuante para cálculos de 32 bits ou de 64 bits,
- uma unidade de memória com capacidade de 4Kbytes,
- quatro ligações("links") com dois canais cada,
- interface para memória externa de até 4 Gbytes.

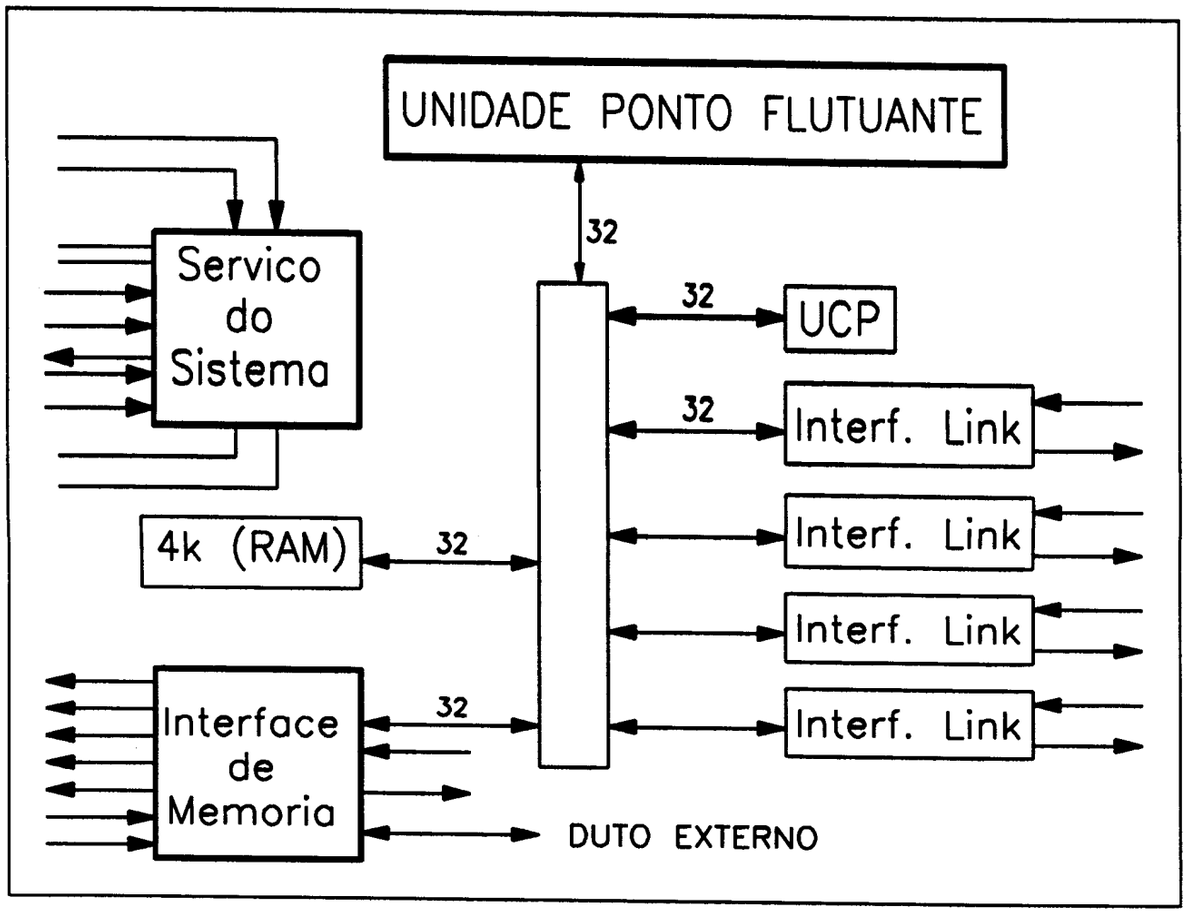

Figura 2.1 - Arquitetura do T800

O que difere a arquitetura dos transputers das demais arquiteturas são os "links", que realizam as comunicações com outros transputers ou com outros processos,sendo que os processos são sincronizados e são bem suportados em sistemas multiprocessados.

O transputer pode ser utilizado em um sistema de processamento único ou em uma rede, com o intuito de trazer alto desempenho em sistemas concorrentes. Uma rede de transputers é construída utilizando a comunicação ponto a ponto disponível nos canais dos transputers.

Para maiores referências é aconselhavél uma leitura dos manuais da INMOS [THE-87].

2.2.1 - Unidade de processamento inteiro.

A tabela 2.1 representa os principais registradores de 32 bits da unidade de processamento inteiro.

Tabela 2.1 - Registradores inteiros de 32 bits

Registrador	Função
Iptr	Aponta a próxima instrução a ser executada
Wptr	Aponta a região de trabalho do processo corrente
Areg	Pilha de avaliação
Breg	
Creg	
Oreg	Contém o registrador de operando

Para entender o funcionamento da pilha de avaliação é dado um exemplo na Figura 3.2 que representa o cálculo da expressão aritmética " c = a*a + 2*b ".

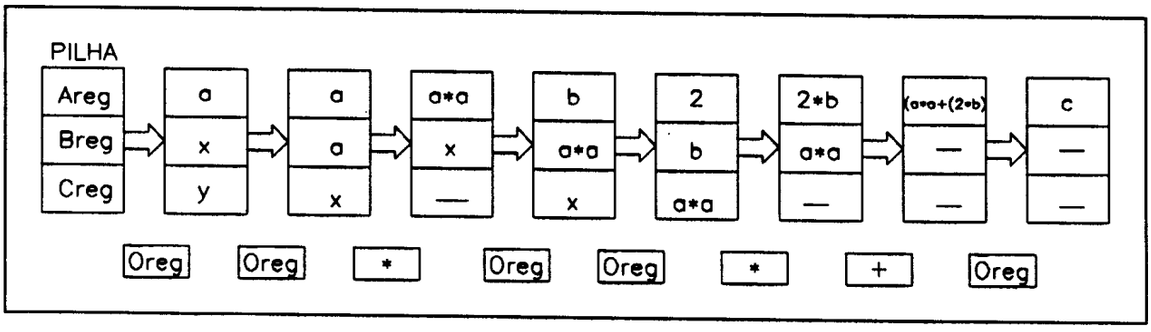

Figura 2.2 - Funcionamento da pilha de avaliação

Ou seja, carrega-se o dado "a" em Areg, o próximo dado a ser carregado "a" entrará em Areg e consequentemente o primeiro dado "a" vai para Breg.

Se já houver um operando em Oreg, este é executado entre os dados contidos em Areg e Breg, e o resultado é inserido em Areg. Desse modo Breg fica vazio e Creg fica com valor indefinido.

As operações no transputer são executadas nesses registradores e utilizam a notação pós-fixa, onde os operandos são digitados antes da operação. Um exemplo é realizar a multiplicação entre dois números a e b, assim deve-se proceder da seguinte maneira:

• digite a depois digite b depois digite * ===> a b *

Assim obtemos a multiplicação de a por b.

2.2.2 - As instruções de máquina.

Uma instrução de máquina contém 1byte, sendo que os 4 bits mais significativos contém a operação, e os 4 bits menos significativos o operando.

No transputer há 16 instruções básicas, sendo uma instrução de código de operação (opr), mais duas instruções para prefixar os operandos (pfix, nfix), os quais permitem a extensão dos códigos de operação e as 13 instruções restantes para valores literais ou endereço de offset.

Os 16 operadores básicos são mostrados na Tabela 2.2.

Tabela 2.2 - Operadores básicos do T800

Operador	Função
pfix	Prefixo
nfix	Prefixo negativo
opr	Interpreta Oreg como código de operação
ldc	Carrega constante
ldl	Carrega variável local
stl	Armazena em variável local
ldlp	Carrega ponteiro local
adc	Adiciona constante
eqc	Teste de igualdade com constante
j	Deslocamento "jump" incondicional
cj	Deslocamento "jump" condicional
ldnl	Carrega variável não local
stnl	Armazena em variável não local
ldnlp	Carrega ponteiro não local
call	Chamada de subrotina
ajw	Ajusta região de trabalho para subrotina

2.2.3 - Estrutura da Unidade de processamento de ponto flutuante.

Nessa unidade, estão presentes três novos registradores, que juntamente com a unidade de processamento inteiro, fazem as operações em ponto flutuante.

Na Tabela 2.3 é apresentado esses novos registradores.

Tabela 2.3 - Registradores adicionais para Unidade de Ponto flutuante.

Registrador (32 ou 64 bits)	Função
FAreg	Pilha de avaliação em ponto-flutuante
FBreg	
FCreg	

Os registradores podem ser de 32 ou 64 bits, dependendo da especificação desejada pelo programador. Para entender a atuação desses registradores adicionais é dado um exemplo simples:

Exemplo : ldlp 5

fpldnlsn

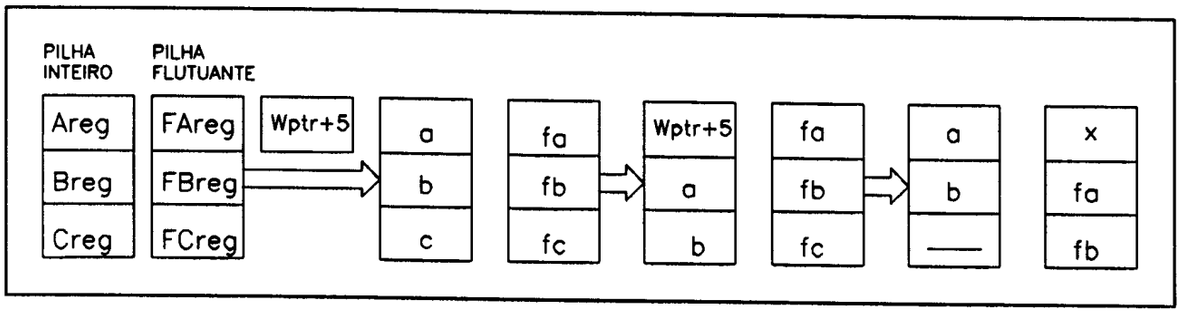

Figura 2.3 - Funcionamento da pilha de avaliação em ponto-flutuante.

Na Figura 2.3 "x" representa o conteúdo da posição endereçada por Wptr+5.

Inicialmente carrega-se o endereço (ponteiro local) (Wptr + 5) em Areg, sendo o carregamento dos dados realizado como descrito anteriormente na unidade de processamento inteiro. Em seguida o conteúdo do endereço em Areg dado pela variável x, é carregado em FAreg.

2.3 - Linguagem OCCAM.

A linguagem OCCAM foi desenvolvida como uma implementação de um subconjunto de CSP de Hoare, para explorar o máximo paralelismo de sistemas concorrentes. Isto ocorre devido a sua implementação de paralelismo que está baseado no conceito de comunicação ponto-a-ponto entre processos, isto é, os processos só se comunicam através de canais unidirecionais [HOA-78].

Um processo em OCCAM fornece uma construção em blocos para a linguagem e o transputer fornece uma implementação direta do processo. Assim o programa OCCAM pode ser descrito como um processo que inicia, realiza ações e termina.

A seguir apresentaremos resumidamente a linguagem OCCAM. Para maiores detalhes pode-se consultar [POU-86].

2.3.1 - Processos básicos.

Há cinco processos básicos na linguagem OCCAM:

- Atribuição
- Entrada de dados por canal
- Saída de dados por canal
- SKIP
- STOP

Na Tabela 2.4 são representadas as notações e as suas funções.

Tabela 2.4 - Processos primitivos

Processo	Notação	Função
Atribuição	:=	Altera o valor da variável da esquerda pelo valor da direita. Ex : a := (expressão)
Entrada de dados por canal	?	Altera o valor da variável pelo valor enviado pelo canal. Ex: c ? x
Saída de dados por canal	!	A variável será enviada pelo canal. Ex: c ! x
SKIP	SKIP	Inicia um processo, mas não realiza nada, e depois termina.
STOP	STOP	Inicia um processo, nunca faz nada e também nunca termina.

Os processos primitivos de entrada de dados por canal e saída de dados por canal são sempre utilizadas em conjunto em processos paralelos, um exemplo para demonstração é mostrado a seguir:

```
PAR
  SEQ
    ...
    c ! x        -- o valor x é enviado para o canal c.
    ...
  SEQ
    ...
    c ? y        -- o conteúdo do canal c é repassado para a variável y.
    ...
```

2.3.2 - Tipos de Dados Primitivos.

São seis os tipos de dados primitivos que estão descritos na Tabela 2.5 quanto ao tipo e suas funções:

Tabela 2.5 - Tipos de dados primitivos

Tipo de dados primitivos	Função
CHAN OF	Canal de comunicação entre processos.
TIMER	Temporizador que pode ser usado para cada processo concorrente.
BOOL	Variável verdadeira ou falsa
BYTE	Corresponde a 8 bits
INT	Inteiro, que pode ser extendido para INT16, INT32, INT64 em número de bits.
REAL	Ponto flutuante, com duas variações REAL32 e REAL64.

Há ainda o tipo vetor, que é a estrutura de dados básica do OCCAM, e que contém uma cadeia de valores de um determinado tipo de dado primitivo. Sua representação é semelhante ao da linguagem C.

Exemplo : float variável [] (linguagem C).

[]REAL32 variável (linguagem OCCAM).

Para declarar uma variável sendo de um tipo, a declaração é da seguinte forma:

tipo nome:

Onde:

- **tipo** representa um dos tipos de dados primitivos
- **nome** o nome da variável
- **:** indica que a declaração é válida para o processo a seguir.

Assim a declaração para cada tipo de dados fica:

- **CHAN OF INT c:** (onde c representa o nome da variável)

Isto significa que está sendo declarado um canal chamado c do tipo INT. Sendo que o tipo

pode conter também um protocolo criado pelo programador, que será explicado mais adiante nessa seção 2.3.3.3.

• **TIMER tempo:**

Isto significa que "tempo" é um contador de tempo, sendo que o valor retomado é do tipo INT, para manipular esse dado são apresentados as seguintes palavras chaves:

- AFTER (que espera o tempo declarado após a palavra);
- PLUS (é o operador "+", sem teste de estouro (overflow));
- MINUS (é o operador "-", sem teste de estouro (overflow)).

• **BOOL verdade:**

Isto significa que verdade pode ser TRUE ou FALSE aplicado para comparação ou teste entre valores de mesmo tipo de dados.

• **BYTE a:**

Como em OCCAM não há uma representação para caracteres alfabéticos ou de palavras específicas, esses serão representados por número do tipo BYTE que associa a com seu valor em código ASCII.

• **INT a:**

Isto significa que a variável "a" é do tipo inteiro.

• **REAL32 b:**

Isto significa que a variável "b" é um número de ponto flutuante de 32 bits.

Para representarmos valores constantes de um determinado tipo de dado primitivo é utilizado a seguinte palavra chave:

• **VAL INT constante IS 0:**

Isto significa que a variável "constante" tem sempre o valor zero.

Para declararmos vetores que são representados através de colchetes, deve-se proceder da seguinte forma:

• **[50]INT vet:**

Isto significa que vet tem 50 elementos do tipo inteiro, e podemos multidimensioná-lo indicando o alcance para cada dimensão, por exemplo:

Ex.1 - **[ind1][ind2]...[indn]INT vet1:** onde as dimensões são dadas por ind1, ind2 até indn sucessivamente.

Se desejarmos apenas uma parte do vetor vet1 podemos utilizar a seguinte notação:

Ex.2 - **[vet FROM inicio FOR nelem]** Essa notação significa que só será acessado a parte do vetor que começa em início e compreende **nelem** elementos do vetor. O mesmo procedimento é realizado para os vetores multidimensionais, mas considerando que variamos apenas uma dimensão e as outras são constantes.

E para representarmos uma cadeia de caracteres, usamos um vetor de **BYTE**.

Ex.3 - **[cadeia]BYTE palavra:** onde **cadeia** representa o comprimento máximo da palavra.

Para os outros tipos de dados primitivos a forma de declaração é semelhante.

2.3.3 - Processos em OCCAM.

Como foi dito anteriomente os processos em OCCAM são de cinco tipos primitivos: Atribuição, entrada e saída de dados por canal, SKIP e STOP.

Esses processos elementares podem ser combinados em processos mais complexos através de construções.

As classes de construção são dadas na Tabela 2.6.

Tabela 2.6 - Classes de construção de processos.

Construção	Função
SEQ	Composição sequencial
PAR	Composição paralela
IF	Condicional
WHILE	Repetição
ALT	Composição alternativa

• SEQ - A construção SEQ é seguida de um ou mais processos:

```
SEQ
  processo1
  processo2
  ...
  processon
```

Os processos serão executados um após o outro. Uma observação importante é que para um processo ser considerado dentro de uma estrutura, esse deve estar identado dois espaços em relação ao identificador da estrutura, como mostrado acima. O que vale para todas as construções abordadas a seguir.

• PAR - A construção PAR é seguida de um ou mais processos executados em paralelo.

```
PAR
  processo1
  processo2
  ...
  processon
```

Quando cada um dos processos que aparece dentro de um PAR é alocado cada qual em um processador diferente, utilizando a notação PLACED PAR, o qual será comentado mais adiante, aí então teremos o verdadeiro paralelismo, mas quando os processos são alocados dentro de um único processador, então esses serão executados concorrentemente através da técnica de "time-sharing".

• IF - A construção IF determina condição de execução, dependendo de qual instrução seja verdadeira primeiro, um ou mais processos serão executados.

```
IF
  condição1
    processo1
  condição2
    processo2
  ...
  condiçãon
    processon
```

• WHILE - A construção WHILE faz com que o processo identado a ele seja executado repetidas vezes até que a condicão não mais seja válida.

```
WHILE condição
  processo
```

• ALT - A construção ALT permite uma escolha entre vários processos de acordo com qual canal, dentre vários, irá receber primeiro um dado.

```
ALT
  guarda1
    processo1
  guarda2
    processo2
  ...
```

Para entendermos melhor essa construção, que é exclusiva da linguagem OCCAM, daremos um exemplo:

```
ALT
  c1 ? b
    a:= a - b
  c2 ? d
    SEQ
        a:= a + d
        g:= 1
  (a>0) & xy ? e
    a:= 0
```

Se o canal "c1" for o que receber primeiro o dado, então o dado enviado por esse canal será armazenado na variável "b", sendo em seguida o valor de "b" subtraído do valor atual de "a".

Caso o primeiro dado chegue pelo canal "c2", então esse deve ser armazenado em "d" e em seguida executar o processo sequencial.

Se o dado chegar antes pelo canal "xy" temos uma condição adicional a realizar, a de que "a>0", então no instante que ALT for executado, o valor de "a" será testado, caso o valor for menor ou igual a "a" então o canal "xy" não será testado, isto é, a chegada do dado será ignorada, caso contrário o processo será executado.

2.3.3.1 - Replicadores.

Há ainda a ressaltar que existem mais quatro tipos de forma de repetição que são os replicadores sequenciais, paralelos, alternados e condicionais. Esses são definidos a seguir, onde inicio e contador são do tipo INT:

- **SEQ passo = inicio FOR contador**
  processo[passo]

Isto significa que o processo será executado um número de "**contador**" vezes sequencialmete, o qual tem a mesma função do FOR na linguagem C.

- **PAR passo = inicio FOR contador**
  processo[passo]

Isto significa que o processo será executado concorrentemente um número de "**contador**" vezes.

- **ALT passo = inicio FOR contador**
  in[passo] ? valor1[passo]

Isto significa que há um número "contador" de canais "in" e para cada canal está associado uma variável que será enviada pelos canais, sendo que apenas um desses canais será habilitado por vez, já que na instrução ALT o primeiro canal que chegar é que vai ser executado.

- IF passo:= início FOR contador
  processo[passo]:= tecla

Para o replicador IF, esse tem o mesmo efeito do replicador WHILE.

2.3.3.2 - Prioridades.

Pode ser introduzido prioridade entre os processos concorrentes na linguagem OCCAM, através do recurso PRI, o que só vale para as construções PAR e ALT.

A sua construção tem a seguinte estrutura:

```
PRI PAR
  processo1  -- processo com alta prioridade
  PAR
    processo2  -- dois processos com prioridades iguais
    processo3
  processo4  -- processo com baixa prioridade
```

Essa construção é muito utilizada para aplicações de tempo real, onde podemos alocar a alta prioridade para o processo que estiver tratando das respostas, e o programador pode assegurar então que outro processo não será executado até que o evento em tempo-real seja tratado.

Como em um sistema de "run-time" deve suportar filas distintas para cada nível de prioridade, então uma implementação pode ser limitada pelo número de componentes contidos em um PRI PAR. Por exemplo, no transputer somente dois níveis de prioridade são suportados alta e baixa.

E para PRI ALT a construcão segue o mesmo padrão, sendo que a sua utilização se deve

a sua resposta rápida para a ocorrência de eventos.

```
PRI ALT
  a > 0 & c ? x
    processo1
  b > 0 & c ? y
    processo2
```

2.3.3.3 - Protocolos.

Para processos concorrentes a utilização de protocolos é frequente, pois desse modo pode-se alocar a um canal um ou mais tipos de dados primitivos. Há três formas de protocolo que são:

• **Protocolos sequenciais**: onde o canal transmite diversos dados sequencialmente.

A sua declaração fica: **PROTOCOL c IS tipo1; tipo2:**

Assim a sua chamada é:

```
CHAN (INT; REAL32) c:
INT a:
REAL32 b:
  PAR
    c ? 1; 5.5
    ...
    c ! a; b
```

• **Protocolos de vetor**: onde o canal transmite uma cadeia de dados, onde primeiro é enviado o tamanho da cadeia e depois os dados especificados.

A sua declaração fica: **PROTOCOL c IS INT::[]INT:**

Assim a sua chamada é:

```
CHAN OF ([INT]INT) c:
[4]INT vetor:
[6]INT v:
INT a:
  PAR
    c ! 3 :: vetor
    ...
    c ? a :: [v FROM 0 FOR a]
```

• **Protocolos variantes:** onde o canal transmite diferentes tipos de dados primitivos, isto se dá devido a transmissão de um identificador do tipo seguido pelo dado.

A sua declaração fica:
```
PROTOCOL canal1
  CASE
    fixo; INT
    flutuante; REAL32
```

Assim a sua chamada fica:
```
CHAN OF canal1 valor:
  INT a:
  REAL32 b:
  PAR
    SEQ
      valor ! fixo; 3
      valor ! flutuante; 2.1
      ...
      valor ? CASE
        fixo ; a
          processo1
        flutuante ; b
          processo2
```

2.3.4 - Paralelização em transputers.

A alocação de processos em transputers diferentes, para a realização do paralelismo real são:

• **Alocação de processadores:** É implementada através da estrutura **PLACED PAR**, que indica qual processo irá ser executado em algum determinado processador.

A sua construção fica:
```
PLACED PAR
  PROCESSOR 0   T8
    processo1
  PROCESSOR 1   T4
    processo2
```

• **Alocação de "links"**: Indica qual "link" físico do transputer está associado ao canal lógico (linknum).

A sua construção fica: PLACE que.canal AT linknum

• **Alocação de memória de dados**: Indica em que posição da memória o dado deve ser inserido.

A sua construção fica: PLACE variável AT local.memória

2.3.5 - Rotinas e Funções.

• **Rotinas** : A palavra chave é **PROC**, e a frente coloca-se um nome de chamada e entre parênteses os parâmetros.

A sua construção é da seguinte forma:

```
PROC nome.chamada ( tipo1 variável; tipo2 variável...)
  rotina.a.ser.executada
```

• **Funções** : A palavra chave é **VALOF**, e esta executa uma função e seu valor é retornado pela palavra chave **RETURN**. E a chamada é realizada pela notação **FUNCTION**.

A sua construção é da seguinte forma:

```
REAL32 FUNCTION cálculo ( VAL []REAL32 valores)
  REAL32 soma:
  VALOF
    SEQ
      soma:= 0.0(REAL32)
      SEQ i:=0 FOR SIZE valores
        soma:= soma + valores[i]
      RESULT soma/(REAL32 ROUND SIZE valores)
```

Capítulo 3

FFT-2D em Transputer: Processamento Sequencial

3.1 - Introdução.

Nesse capítulo é abordado o desempenho obtido pela FFT-2D para o processamento sequencial, apenas como base de comparação com os resultados obtidos do processamento paralelo realizado para a FFT-2D para rede de transputers.

Há ainda a ressaltar que para melhorar o desempenho, foi realizado a conversão de algumas rotinas da transformada dada em linguagem OCCAM para a linguagem montadora do transputer, o qual obteve bons resultados, como será comentado no decorrer desse capítulo.

3.2 - Implementação da FFT-2D sequencial.

A implementação da FFT-2D sequencial foi realizada para verificar quanto tempo é gasto no cálculo da FFT-2D por um micro tipo PC-XT.

Como foi dito anteriomente o algoritmo adotado para esse trabalho consiste do método "Radix-4".

O esquema seguido para a implementação é mostrado a seguir na Figura 3.1.

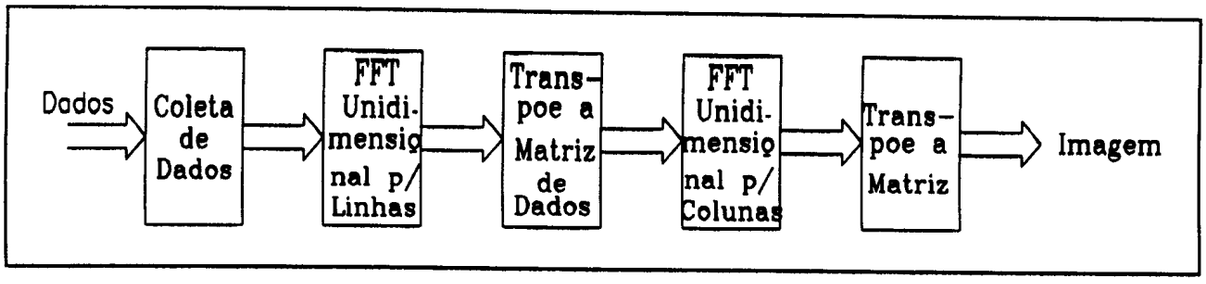

Figura 3.1 - Esquema da implementação do algoritmo da FFT.

O exemplo adotado para a implementação, é a plotagem de uma imagem de corte de uma laranja obtida por ressonância magnética nuclear (RMN), onde a imagem tem a dimensão de 256 X 256 (256 linhas e 256 colunas), como mostrada na Figura 3.2.

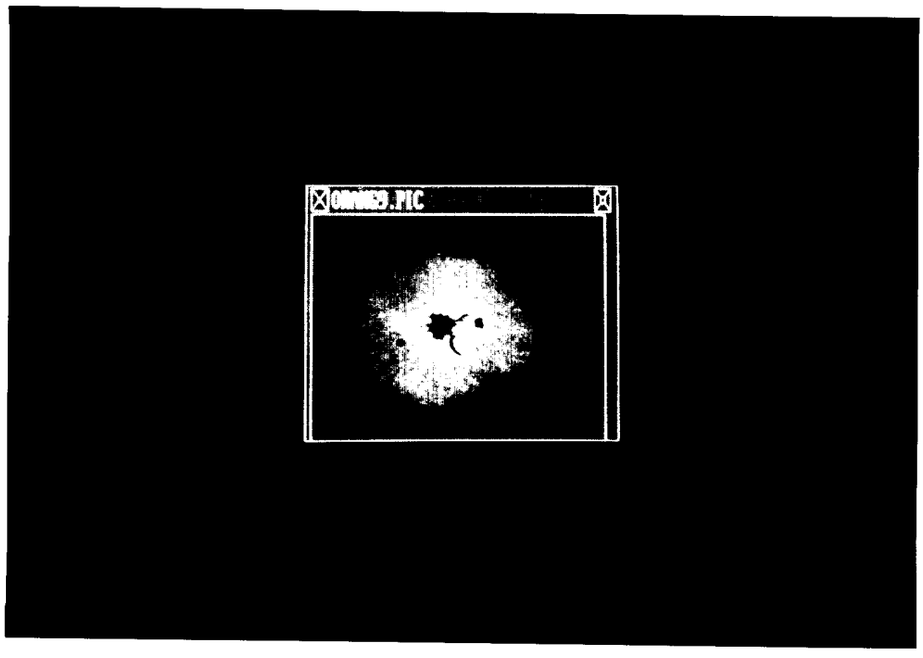

Figura 3.2 - Imagem de uma meia laranja obtida por RMN.

A seguir é apresentado o fluxograma do método "Radix-4" , utilizado para o cálculo da FFT, dado pela Figura 3.3.

SERVIÇO DE ... IFQSC

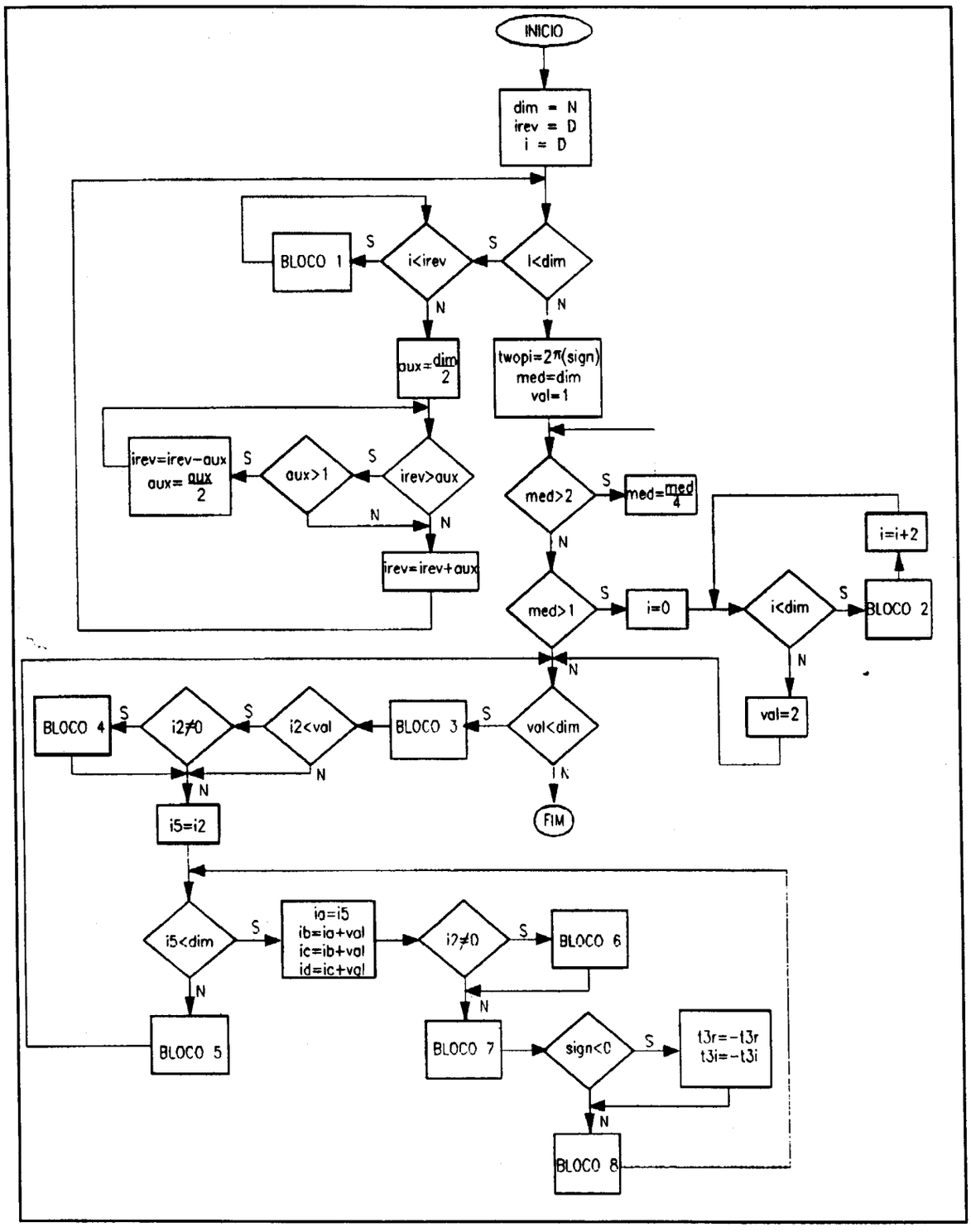

Figura 3.3 - Diagrama de Blocos do método Radix-4 unidimensional.

Os conteúdos dos blocos 1 ao 8, citados na Figura 3.3 estão expostos a seguir:

BLOCO 1 : muda os dados do vetor de i para irev e vice-versa, consiste a operação "bit-reverse".

```
tmp := FR[i]
FR[i]:= FR[irev]
FR[irev]:= tmp
tmp1:= FI[i]
FI[i]:= FI[irev]
FI[irev]:= tmp1
```

BLOCO 2 : muda os dados do vetor a cada duas posições a contar de i.

```
next:= i+2
tmpr:= FR[i]
tmpi:= FI[i]
FR[next]:= FR[i] - tmpr
FI[next]:= FI[i] - tmpi
FR[i]:= FR[i] + tmpr
FI[i]:= FI[i] + tmpi
```

BLOCO 3 : inicializa valores para calculo de ω

```
ip3:= val * 4
theta:= twopi / ip3
sinth:= sin( theta/2 )
wstpr:= sinth * sinth * (-2)
wstpi:= sin(theta)
wr:= 1.0
wi:= 0.0
```

BLOCO 4 : calcula os valores de ω

```
w2r:= (wr * wr) - (wi * wi)
w2i:= (wr * wi) * 2
w3r:= (w2r * wr) - (w2i * wi)
w3i:= (w2r * wi) + (w2i * wr)
```

BLOCO 5 : calcula as "butterflys" da fase intermediária

```
tmpr:= wr
wr:= (wstpr * tmpr) - (wstpi * wi) + tmpr
wi:= (wstpr * wi) + (wstpi * tmpr) + wi
```

BLOCO 6 : calcula as "butterflys" da fase intermediária

```
tmpr:= FR[ib]
FR[ib]:= (w2r * FR[ib]) - (w2i * FI[ib])
```

```
FI[ib]:= (w2r * FI[ib]) + (w2i * tmpr)
tmpr:= FR[ic]
FR[ic]:= (w2r * FR[ic]) - (w2i * FI[ic])
FI[ic]:= (w2r * FI[ic]) + (w2i * tmpr)
tmpr:= FR[id]
FR[id]:= (w2r * FR[id]) - (w2i * FI[id])
FI[id]:= (w2r * FI[id]) + (w2i * tmpr)
```

BLOCO 7 : cálculo da "butterflys" para "Radix-4"

```
t0r:= FR[ia] + FR[ib]
t0i:= FI[ia] + FI[ib]
t1r:= FR[ia] - FR[ib]
t1i:= FI[ia] - FI[ib]
t2r:= FR[ic] + FR[id]
t2i:= FI[ic] + FI[id]
t3r:= FR[ic] - FR[id]
t3i:= FI[ic] - FI[id]
FR[ia]:= t0r + t2r
FI[ia]:= t0i + t2i
FR[ic]:= t0r - t2r
FI[ic]:= t0i - t2i
```

BLOCO 8 : parte do cálculo da "butterfly" para "Radix-4"

```
FR[ib]:= t1r - t3i
FI[ib]:= t1i + t3r
FR[id]:= t1r + t3i
FI[id]:= t1i - t3r
```

3.3 - Implementação da FFT-2D em linguagem montadora.

Como comentado anteriormente, foram convertidas algumas rotinas implementadas em linguagem OCCAM pura para a linguagem montadora do transputer. Essas rotinas são os blocos 2, 5, 6, 7 e 8 apresentados na Figura 3.3 da seção anterior, pois elas representam a operação de "bit-reverse" das fases intermediárias do cálculo da FFT e da operação de "butterfly".

Foi introduzida também uma rotina "pos" que simula o ponteiro da linguagem C para a linguagem montadora do transputer, a fim de agilizar o endereçamento de dados e aumentar o

desempenho do transputer, além de agilizar as operações aritméticas mais requisitadas, o qual resultou num decaimento do tempo de processamento dos resultados da FFT-2D considerávelmente. Essa rotina "pos" está inserida no Apêndice B.

Para simples ilustração de como as rotinas em linguagem OCCAM são convertidas para a linguagem montadora, é dado a seguir a rotina oper, que representa o bloco 2 da Figura 3.3.

• rotina oper em linguagem OCCAM, que representa a operação de "bit-reverse" de uma das fases intermediárias do cálculo da transformada.

```
SEQ
  tmpr:= FR[i]
  tmpi:= FI[i]
  FR[i]:= FR[k] - tmpr
  FI[i]:= FI[k] - tmpi
  FR[k]:= FR[k] + tmpr
  FI[k]:= FI[k] + tmpi
```

• rotina oper em linguagem montadora do transputer.

```
SEQ
GUY
   ldl        pont   -- carrega variável local pont
   fpldnlsn          -- lê variável da memória apontado por pont
   ldlp       tmpr   -- carrega ponteiro local tmpr
   fpstnlsn          -- tmpr recebe conteúdo de pont
   fpchkerr          -- checa erro
                        tmpr:= FR[i]

   ldl        pont   -- carrega variável local pont
   adc        4      -- vai para o próximo byte
   dup               -- duplica a variável de pont+meio byte
   stl        fp1I   -- fp1I recebe um dos conteúdos duplicados
   fpldnlsn          -- lê variável da memória apontado por pont
   ldlp       tmpi   -- carrega ponteiro local tmpi
   fpstnlsn          -- tmpi recebe o conteúdo duplicado
   fpchkerr          -- checa erro
                        tmpi:= FI[i]

   ldl        pont1  -- carrega variável local pont1
   adc        4      -- vai para o próximo byte
   stl        fpI    -- fpI recebe conteúdo de pont1+meio byte
   ldl        pont1  -- carrega variável local pont1
```

```
fpldnlsn          -- lê variável da memória apontado por pont1
ldlp      tmpr    -- carrega ponteiro local tmpr
fpldnlsn          -- lê variável da memória apontado por tmpr
fpsub             -- subtrai tmpr de pont1+meio byte
ldl       pont    -- carrega variável local pont
fpstnlsn          -- pont recebe resultado da subtração
fpchkerr          -- checa erro
                     FR[i]:= FR[k] - tmpr

ldl       fpI     -- carrega variável local fpI
fpldnlsn          -- lê variável da memória apontado por fpI
ldlp      tmpi    -- carrega ponteiro local tmpi
fpldnlsn          -- lê variável da memória apontado por tmpi
fpsub             -- subtrai fpI de tmpi
ldl       fp1I    -- carrega variável local fp1I
fpstnlsn          -- tmpi recebe resultado da subtração
fpchkerr          -- checa erro
                     FI[i]:= FI[k] - tmpi

ldl       pont1   -- carrega variável local pont1
fpldnlsn          -- lê variável da memória apontado por pont1
ldlp      tmpr    -- carrega ponteiro local tmpr
fpldnladdsn       -- soma tmpr e pont1
ldl       pont1   -- carrega variável local pont1
fpstnlsn          -- pont1 recebe resultado da soma
fpchkerr          -- checa erro
                     FR[k]:= FR[k] + tmpr

ldl       fpI     -- carrega variável local fpI
fpldnlsn          -- lê variável da memória apontado por fpI
ldlp      tmpi    -- carrega ponteiro local tmpi
fpldnladdsn       -- soma tmpi com fpI
ldl       fpI     -- carrega variável fpI
fpstnlsn          -- fpI recebe resultado da soma
fpchkerr          -- checa erro
                     FI[k]:= FI[k] + tmpi
```

Na rotina em linguagem montadora a instrução "GUY" indica que o que segue abaixo está em linguagem montadora do transputer. Aparecem também duas novas variáveis não declaradas anteriormente que são "**pont**" e "**pont1**" , que contém os valores do vetor F, que foram calculados na rotina "**pos**".

No lado direito da rotina, como comentário, estão as expressões correspondentes na linguagem OCCAM.

O programa implementado em linguagem montadora está contido no Apêndice B.

3.4 - Performance da FFT-2D sequencial.

Nessa seção será dada a performance obtida pela implementação da FFT-2D sequencial tanto para a implementação puramente na linguagem OCCAM, quanto com a inserção de trechos em linguagem montadora indicados.

Para facilitar a visualização da implementação, o programa foi separado em duas partes, Controlador e FAZFFT.

• Controlador : dentro dele estão contidas as rotinas de leitura de dados e a distribuição dos dados em linhas e colunas para a realização da FFT.

• FAZFFT : contém as rotinas que realizarão a FFT, dos dados enviados pelo controlador.

Abaixo temos um diagrama desse rearranjo da implementação:

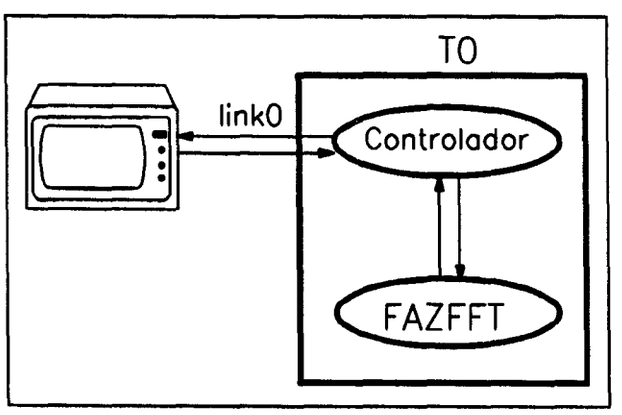

Figura 3.4 - Controlador e FAZFFT.

3.4.1 - Performance para implementação na linguagem OCCAM.

Aqui será dado o tempo gasto para que um transputer realize o processamento da FFT-2D, sem a utilização dos "links" de comunicação existente no transputer.

Assim na Tabela 3.1 são mostrados os tempos de processamento coletados para as seguintes dimensões da imagem:

Tabela 3.1 - Tempo de processamento para o algoritmo puramente em linguagem OCCAM.

dimensão da imagem (linhas X colunas)	tempo de leitura do arquivo de dados no PC (s)	tempo de processamento da FFT (s)
8 X 256	5.426	0.326
16 X 256	10.803	0.679
32 X 256	21.486	1.443
64 X 256	42.599	2.981
128 X 256	71.828	6.367

3.4.2 - Performance para implementação com rotinas em linguagem montadora.

Os tempos de processamento para as seguintes dimensões da matriz imagem para a implementação com rotinas em linguagem montadora é dada na Tabela 3.2.

Tabela 3.2 - Tempo de processamento para o algoritmo com rotinas implementadas em linguagem montadora.

dimensão da imagem (linhas X colunas)	tempo de leitura do arquivo de dados no PC (s).	tempo de processamento da FFT (s).
8 X 256	5.494	0.204
16 X 256	10.740	0.431
32 X 256	21.380	0.961
64 X 256	42.645	2.016
128 X 256	71.752	4.438

3.5 - Resultados.

Pelas Tabelas 3.1 e 3.2 montamos o gráfico em barras para termos uma melhor visão dos tempos de processamento da FFT-2D nas duas implementações realizadas nesse capítulo. Podemos observar que temos um bom resultado no tempo de processamento da FFT-2D para a implementação com rotinas em linguagem montadora, mas a sua maior desvantagem é na depuração, que é muito complexa e complicada. É recomendado o uso da linguagem montadora na implementação quando houver uma necessidade de reduzir o tempo de processamento maior do que foi encontrado utilizando uma rede de transputers.

As conclusões desses resultados serão apresentados posteriormente no Capítulo 5.

SERVIÇO DE BIBLIOTECA E INFORMAÇÃO - IFQSC
FÍSICA

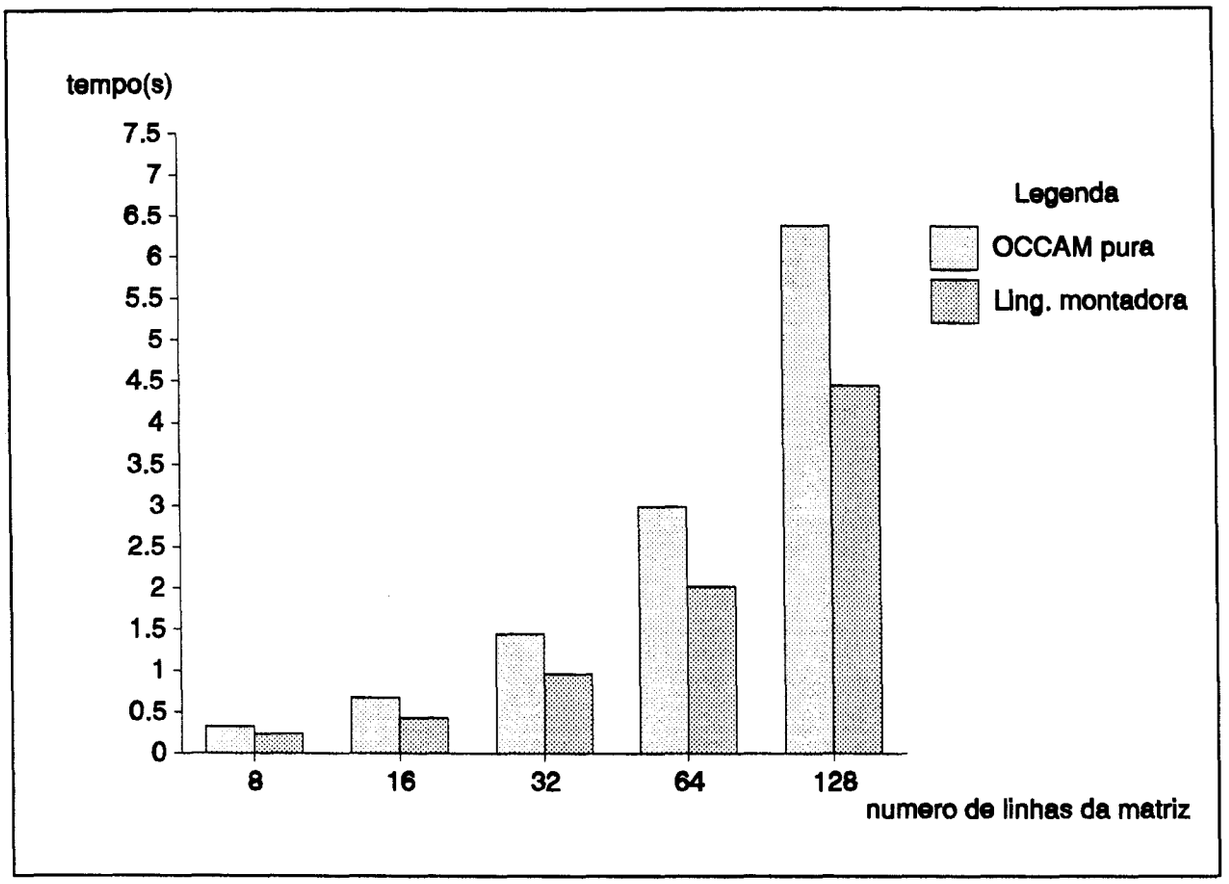

Figura 3.5 - Tempo de processamento da FFT-2D sequencial

Capítulo 4
FFT-2D paralelo em rede de transputer

4.1 - Introdução.

Nesse capítulo será mostrado a implementação da FFT-2D paralela em uma rede de transputer, quando exploramos os seus "links" de comunicação entre os transputers e assim poder definir a sua performance.

Para explorar o paralelismo há três técnicas de exploração, que são descritas a seguir.

• Divisão geométrica: Utiliza-se quando um problema, que apresenta processamentos semelhantes para grande quantidade de dados diferentes, podendo ter seus dados divididos entre várias unidades de processamentos, sendo todas elas encarregadas de realizar o mesmo tipo de processamento.

• Divisão algorítmica: Quando o algoritmo para o processamento de certos dados apresentam diversas fases, sendo que cada fase possa ser realizada em paralelo, pode-se implementar a divisão algorítmica alocando processadores diferentes para as diversas fases de processamento.

• Divisão em forma de "farm" (fazenda): Que são aplicados em algoritmos que podem ser divididos formando pacotes discretos do processamento. Os processos podem ser idênticos ou não, mas devem apresentar independência entre si, sendo que uma unidade de processamento é encarregado da distritribuição dos pacotes entre as outras unidades de processamento e fazer o gerenciamento da entrada e saída de dados pelos canais de comunicação, determinando a granularidade do problema.

Para o esse trabalho foi adotado basicamente a divisão em forma de "farm", por apresentar maior facilidade na implementação, já que o transputer trabalha com divisões em blocos para alocação de processos, apesar de apresentar também características da divisão geométrica, pois

cada transputer estará realizando o processamento da FFT de um conjunto coluna ou de um conjunto de linhas concorrentemente.

4.2 - FFT-2D paralela.

Após definido o algoritmo da FFT-2D, e os recursos dados pela linguagem OCCAM e do transputer, deverá ser escolhida a melhor forma de paralelização apropriado ao problema.

Um bom algoritmo paralelo utilizando a divisão em forma de "farm", deve apresentar uma boa independência de dados, de tal forma que o algoritmo possa ser seccionado, distribuído e realizado concorrentemente.

Para que isso seja garantido é preciso que o algoritmo apresente algumas características, dentre elas podemos destacar:

• distribuição uniforme, onde o algoritmo pode ser particionado entre as várias unidades de processamento (conjunto de transputers).

• a granularidade seja uniforme, ou seja, cada unidade de processamento (transputer) é encarregada de realizar a mesma quantidade de processamento.

• expansibilidade do algoritmo, consiste em quando houver acréscimo de mais unidades de processamento (transputers), isso não afete a eficiência do programa, isto é, o tempo total de processamento decai proporcionalmente.

• os canais de comunicação sejam eficientes, tal que não ocorra sobrecarga dos canais de comunicação ou de que um dado fique preso em algum canal, e são também esses canais que irá determinar a performance da máquina paralela.

No transputer há quatro "links" de comunicação, como mostra a Figura 4.1.

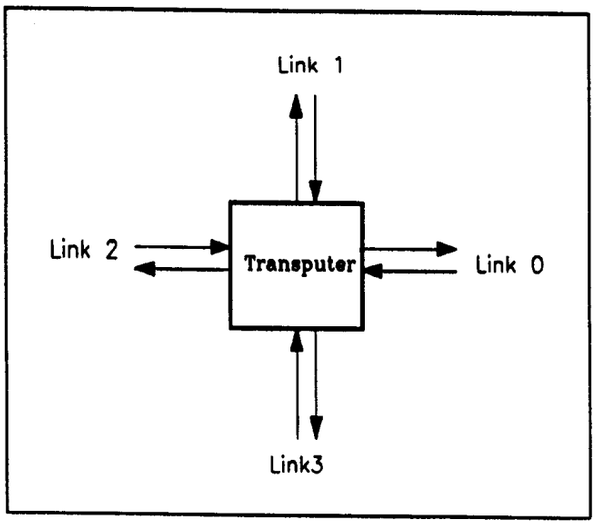

Figura 4.1 - Os quatro "links" de um transputer.

Assim para montar uma rede com transputer, há várias formas de conexão entre elas utilizando os seus "links", a Figura 4.2 mostra algumas das topologias possíveis com transputers.

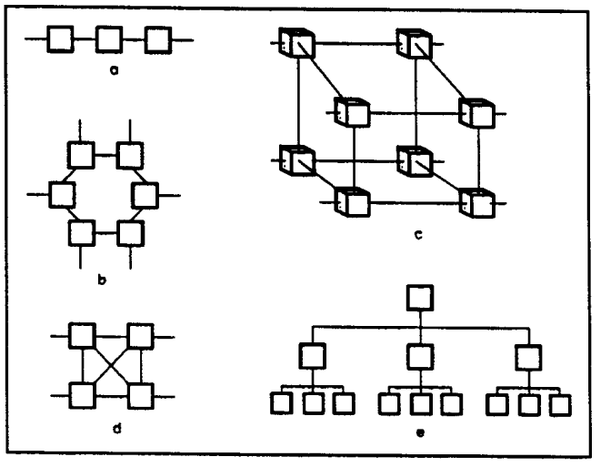

Figura 4.2 - Topologias possíveis com transputers.

Na Figura 4.2 as figuras numeradas são: (a) topologia linear; (b) topologia em anel; (c) topologia em cubo; (d) topologia com quatro transputers inteiramente interligadas; (e) topologia em árvore.

4.3 - Implementação da FFT-2D em uma rede de transputers.

Para a distribuição do algoritmo da FFT-2D em uma rede de transputers, observou-se a independência da matriz de dados da imagem por linhas e por colunas, como explicado no Capítulo 1, desta forma podemos subdividir o processamento da transformada bidimensional em varias transformadas unidimensionais distribuídas para as linhas e colunas da matriz de dados, assim há um número de FFT-1D para linhas e um número FFT-1D para as colunas.

Implementando a FFT-2D para uma rede de transputers, foi primeiramente verificado como será realizada a interconexão dos seus "links", levando em consideração que para cada transputer há apenas quatro "links", e também a melhor topologia a ser adotada para obter a melhor performance quando for necessário expandir o número de transputers na rede.

Para a escolha da topologia foram considerados dois objetivos.

- implementação das intercomunicações entre os transputers a mais simples possível para a rede.
- facilidade de aumentar o número de transputers visando o aumento do desempenho.

Assim a topologia escolhida nesse trabalho é a de árvore, pois essa topologia garante uma melhor distribuição do algoritmo quando houver acréscimo de mais transputers na rede.

Uma observação a ser considerada é que para essa topologia de árvore sua configuração é desbalanceada para número de transputers múltiplo de dois e acima de quatro, sendo que haverá um aumento de comunicação em apenas um dos "link" de comunicação, mas a distribuição de processamento a cada transputer permanece o mesmo.

Já com a topologia em cubo o número máximo de transputer interligados é de 16, expansível para 32 se utilizarmos em cada cubo a topologia de 4 transputer inteiramente interligadas, mesmo deste modo a sua expansibilidade fica muito comprometida, a comunicação entre os processadores apenas entre quatro cubos próximos, deste modo a comunicação fica bem restrita. E para a topologia linear ou em anel a expansibilidade é possível e mais fácil, mas com sobrecarga nos "links" de comunicação, pois todos os dados percorrem pelos canais para a distribuição entre os transputers.

A sub-divisão nos dados da matriz imagem foi obtida de tal forma, que cada transputer da rede realiza uma quantidade fixa de processamento de linhas e de colunas da matriz imagem.

Então a implementação pode ser esquematizada como mostra a Figura 4.3.

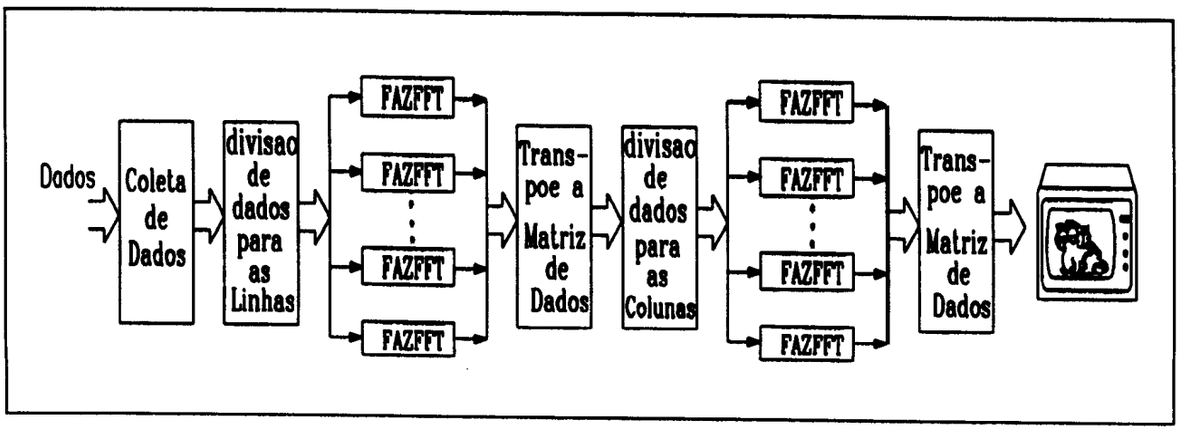

Figura 4.3 - Implementação da FFT-2D numa rede de transputers.

Mas para essa implementação é requisitado um número de transputer injustificável pelos nossos recursos. Assim nesse trabalho foi utilizado uma placa da MICROWAY QUADPUTER, que contém apenas quatro transputer e as suas ligações são:

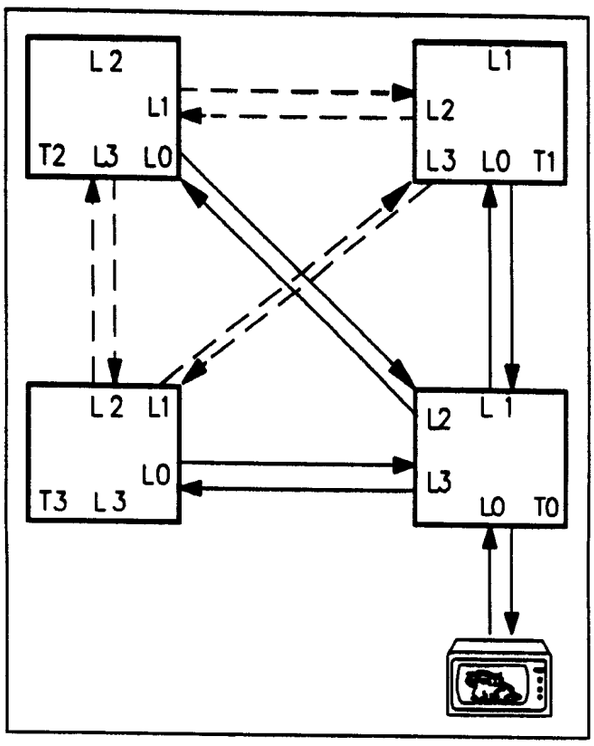

Figura 4.4 - Topologia utilizada na implementação.

Na Figura 4.4 as linhas contínuas representam as ligações utilizadas para a implementação desse trabalho e as linhas tracejadas as ligações já existente na placa da MICROWAY QUADPUTER.

A implementação paralela nesse trabalho utiliza o método de divisão em forma de "farm", onde há um transputer que faz o controle da comunicação de dados entre os transputers e os outros transputers que realizam a transformada para cada linha e depois para cada coluna, ilustrada na Figura 4.5.

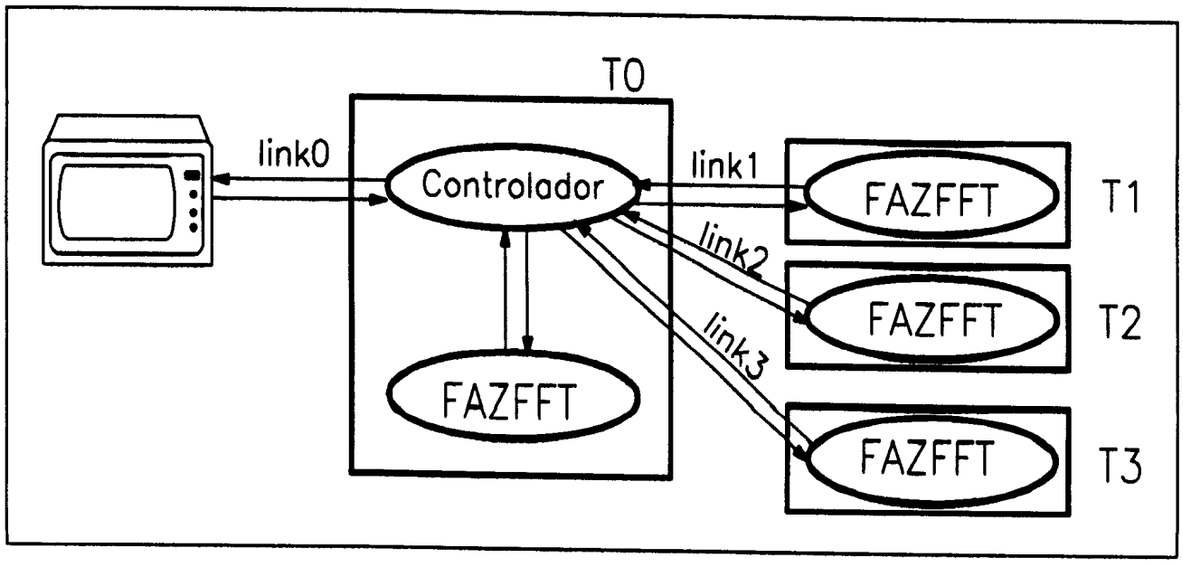

Figura 4.5 - Atribuições de tarefas para cada transputer.

Onde o transputer T0 é encarregado do gerenciamento dos dados da matriz imagem, ou seja é ele que controlará a distribuição dos dados para os outros transputers, além de realizar também uma parte da transformada.

Assim o Controlador contém as rotinas de coleta de dados e distribuição dos dados para a rede de transputers e uma rotina que realiza a FFT chamada de FAZFFT.

Cada outro transputer realizará apenas a rotina FAZFFT.

No apêndice C estão as rotinas Controlador e FAZFFT utilizadas para as implementações realizadas nesse trabalho.

4.4 - Desempenho da implementação.

Para verificar como a paralelização provocado pelo hardware afetará o tempo de processamento da FFT-2D, faremos a seguinte distribuição:

- utilizando apenas um transputer.
- utilizando dois transputers.
- utilizando quatro transputers.

4.4.1 - Utilizando apenas um transputer.

O esquema para essa implementação é dado na Figura 4.6.

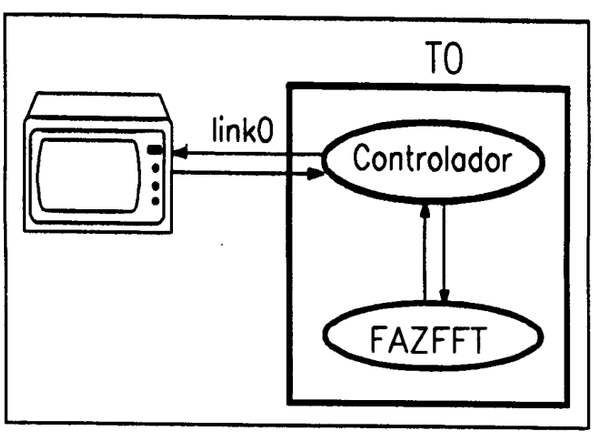

Figura 4.6 - Configuração para um transputer

A configuração do programa desenvolvido para a habilitação do "link" , para o hospedeiro, é dado a seguir:

```
CHAN OF SP from.host, to.host:
PROCESSOR 0 T8
  PLACE from.host AT link0.in:
  PLACE to.host   AT link0.out:
  main(from.host, to.host)
```

Onde **main** contém o Controlodor e FAZFFT mais as rotinas para criar arquivo para a geração de imagem.

O tempo de processamento nessa configuração é dada na Tabela 4.1.

Tabela 4.1 - Tempo de processamento da FFT-2D para 1 transputer.

dimensão da matriz de dados	tempo (s) da FFT-2D
8 X 256	0.330
16 X 256	0.690
32 X 256	1.469
64 X 256	3.038

4.4.2 - Utilizando dois transputers.

Para essa implementação, a divisão do algoritmo é mostrada na Figura 4.7.

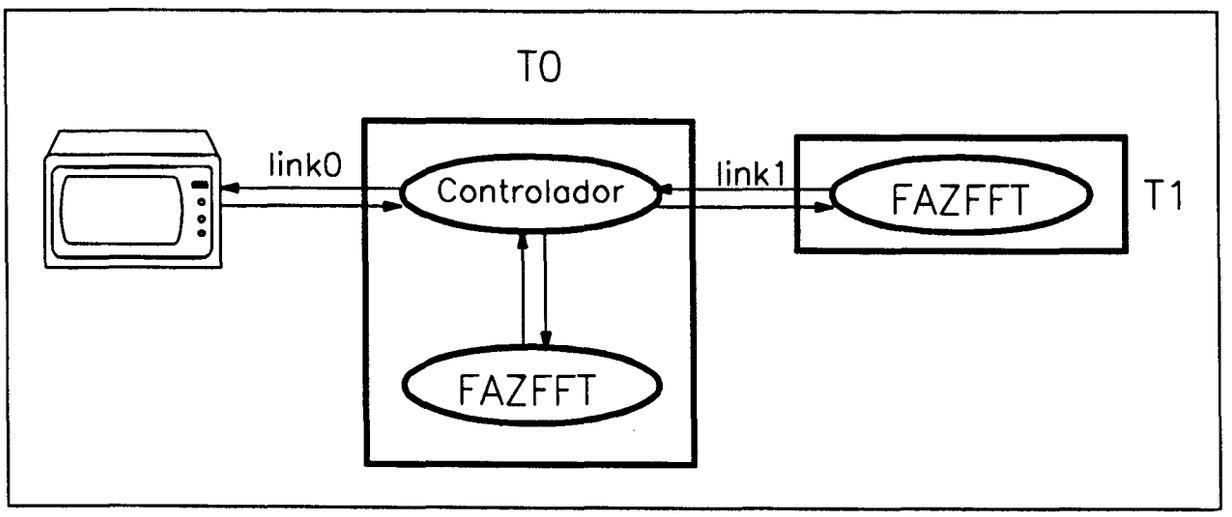

Figura 4.7 - Configuração para dois transputer

E o programa para a interconexão dos "links" é dada pelos seguintes comandos e atribuições:

```
--{{{ canais
        CHAN OF SP from.host, to.host:
        CHAN OF cteste entrada, saida:
        VAL INT np IS 1:
--}}}
```

```
PLACED PAR
PROCESSOR 0 T8
PLACE from.host  AT link0.in:
PLACE to.host    AT link0.out:
PLACE entrada  AT link1.out:
PLACE saida AT link1.in:
  main(from.host, to.host, entrada, saida)

PROCESSOR 1 T8
PLACE entrada AT  link0.in:
PLACE saida AT link0.out:
  FAZFFT(1, entrada, saida)
```

Onde "**main**" contém o Controlador e as rotinas que criam os arquivos para gerar imagem.

O tempo de processamento da FFT nessa situação para diferentes dimensões dos dados da matriz imagem é mostrado na Tabela 4.2.

Tabela 4.2 - Tempo de processamento da FFT-2D para 2 transputers.

dimensão da matriz de dados	tempo (s) da FFT
8 X 256	0.209
16 X 256	0.426
32 X 256	0.884
64 X 256	1.818

4.4.3 - Utilizando quatro transputers.

A implementação para a FFT tem a seguinte divisão do programa:

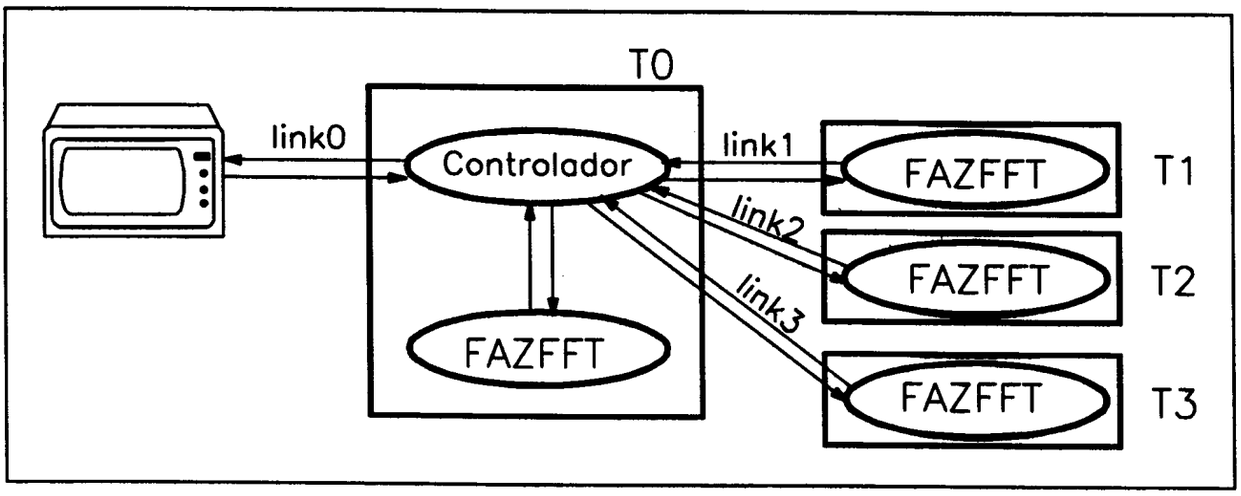

Figura 4.8 - Configuração para quatro transputers

E para essa configuração a implementacão do programa tem a seguinte forma:

```
--{{{ habilitação dos "links" dos transputers
        CHAN OF SP from.host, to.host:
        [3]CHAN OF cteste entrada, saida:
        VAL INT np IS 4:  -- número de processadores
--}}}
      PLACED PAR
      PROCESSOR 0 T8
        PLACE from.host  AT link0.in:    -- alocação dos canais
        PLACE to.host    AT link0.out:
        PLACE entrada[0]  AT link1.out:
        PLACE entrada[1]  AT link2.out:
        PLACE entrada[2]  AT link3.out:
        PLACE saida[0] AT link1.in:
        PLACE saida[1] AT link2.in:
        PLACE saida[2] AT link3.in:
        main(from.host,  to.host,  entrada[0],  entrada[1],  entrada[2],
             saida[0], saida[1], saida[2])
      PLACED PAR j=1 FOR np-1
        PROCESSOR j T8
          PLACE entrada[j-1] AT  link0.in:
          PLACE saida[j-1] AT link0.out:
          FAZFFT(j-1, entrada[j-1], saida[j-1])
```

A rotina **main** contém o Controlador e as rotinas que geram a matriz de dados para a imagem.

O tempo de processamento nessa situação com dimensões dos dados da matriz imagem é dado na Tabela 4.3.

Tabela 4.3 - Tempo de processamento da FFT-2D para 4 transputers.

dimensão da matriz de dados	tempo (s) da FFT
8 X 256	0.130
16 X 256	0.261
32 X 256	0.534
64 X 256	1.091

4.5 - Resultados.

Com os tempos de processamento obtidos com as três implementações, comentadas anteriormente, para o cálculo da FFT-2D foi montado um gráfico em barras para uma visualização separada de cada configuração adotada para a rede de transputer. Como podemos observar pela figura 4.9 com a rede de quatro transputer têm-se um bom resultado na velocidade de processamento da FFT-2D. As conclusões referentes as implementações para as diferentes configurações da rede serão abordadas posteriormente no Capítulo 5.

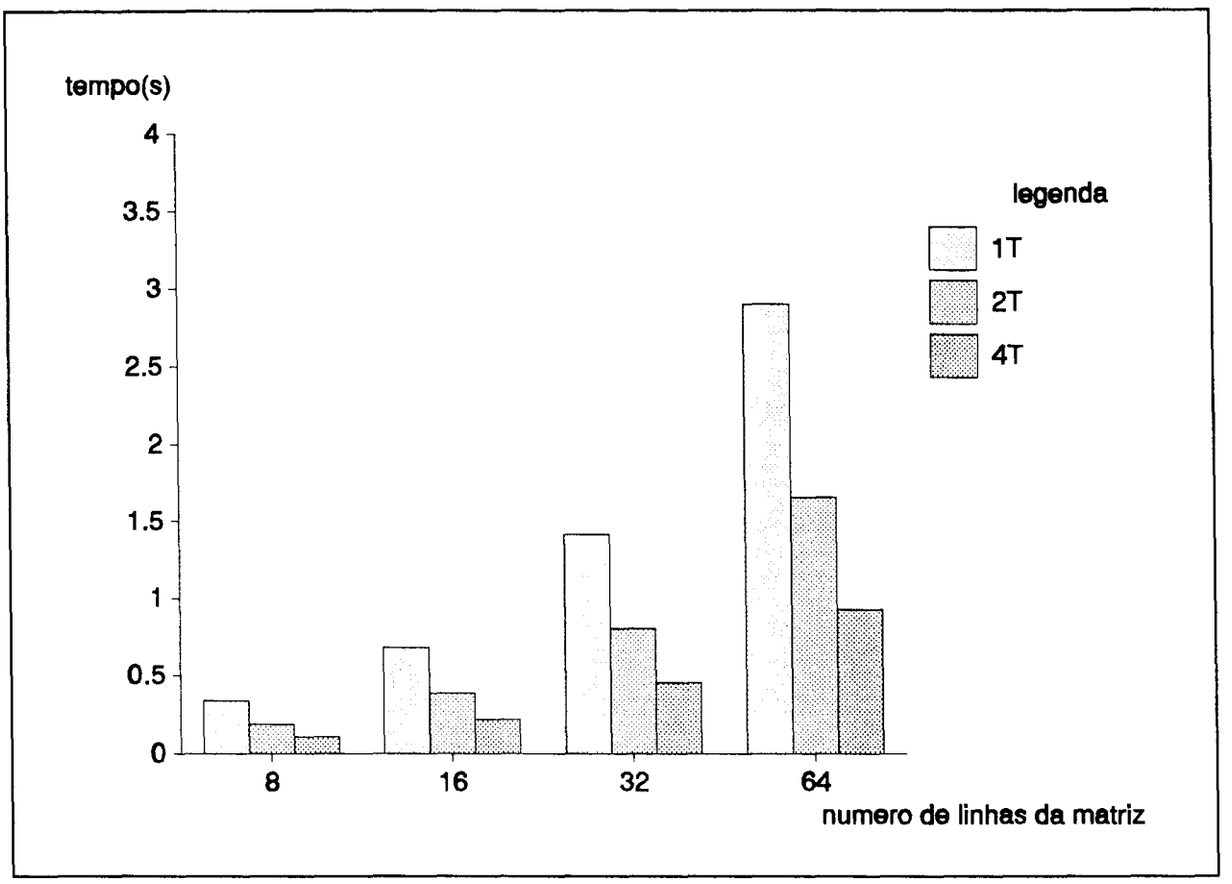

Figura 4.9 - Tempo de processamento para um, dois e quatro transputers.

Capítulo 5
Resultados e Conclusões

Nesse capítulo estão os resultados obtidos na implementação sequencial, apresentada no Capítulo 3, bem como as realizadas para as redes com dois e quatro transputers, tal como apresentado no Capítulo 4.

Para poder concluir os diferentes desempenhos obtidos para os vários conjuntos de dados, é dado nesse capítulo uma avaliação global da eficiência dos algoritmos desenvolvidos.

5.1 - Resultados.

5.1.1 - Análise do desempenho da FFT-2D sequencial em um transputer.

No Capítulo 3, para a execução do programa da FFT-2D em uma implementação puramente sequencial, não foram utilizados os "links" de comunicação entre os transputers.

A partir dessa implementação foram construídas duas tabelas. Na Tabela 3.1 o programa foi desenvolvido utilizando somente a linguagem OCCAM, enquanto a Tabela 3.2 as rotinas de cálculo das "butterflys" foram escritas em linguagem montadora.

Com os tempos desses dois programas, mencionados acima, foi construido a Tabela 5.1. E para obter o desempenho da adoção de rotinas em linguagem montadora foi calculado em porcentagem a redução do tempo de um programa a outro, para tamanhos diferentes da matriz.

Tabela 5.1 - Redução no tempo de processamento em % devido a adoção de rotinas em linguagem montadora numa implementação sequencial em linguagem OCCAM.

dimensão da imagem	redução do tempo em %
8 X 256	37,61
16 X 256	36,62
32 X 256	33,50
64 X 256	32,44
128 X 256	30,30
média da redução do tempo em %	34,10

O tempo de processamento da FFT-2D com rotinas em linguagem montadora reduziu, dependendo do tamanho da matriz, de 37,61% a 30,30% em relação ao tempo de processamento da FFT-2D na linguagem puramente em OCCAM.

5.1.2 - Análise do desempenho entre implementação sequencial e paralelo em um transputer.

Comparando os tempos obtidos para a implementação paralela no Capítulo 4, utilizando os links de comunicação entre Controlador e FAZFFT, com as da implementação sequencial no Capítulo 3 realizada em apenas um transputer, é dado a seguir na Tabela 5.2.

Tabela 5.2 - Tempo de processamento da FFT-2D sequencial e paralelo

dimensão da matriz	t(s) sequencial	t(s) paralelo
8 X 256	0.327	0.330
16 X 256	0.680	0.690
32 X 256	1.445	1.469
64 X 256	2.984	3.088

Dessa tabela 5.3 montamos o gráfico da Figura 5.1:

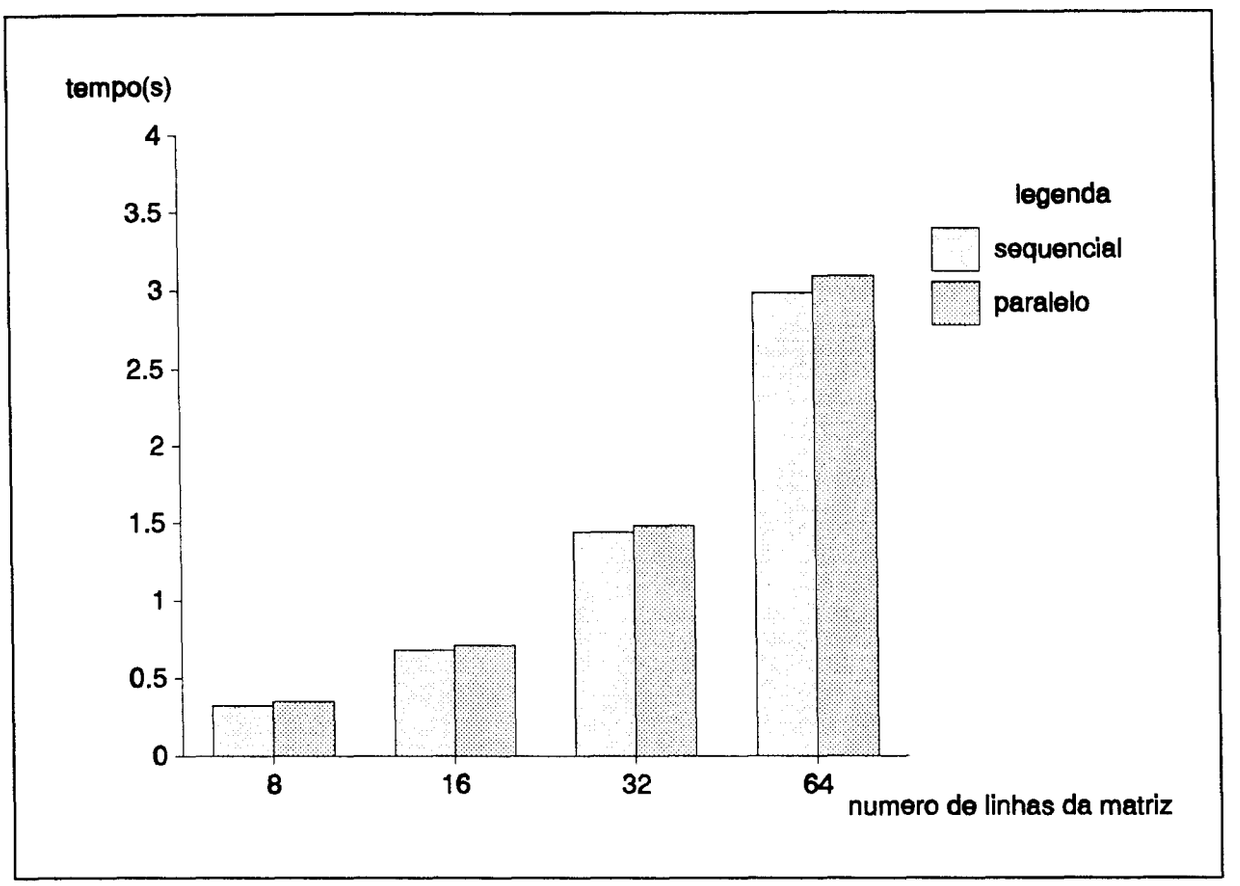

Figura 5.1 - Tempo de processamento da FFT-2D para um transputer.

Nota-se da Figura 5.1 que houve,um aumento não muito grande no tempo de processamento para a implementação paralela. Este aumento ocorre devido as comunicações interna do transputer.

5.1.3 - Análise do desempenho da FFT-2D em rede de transputer.

Na implementação paralela do algoritmo da FFT-2D apresentada no Capítulo 4, que é o

objetivo principal deste trabalho, foi apresentada a paralelização do hardware pelos "links" de comunicação dos transputers utilizados na formação das redes desejadas.

Assim, é mostrado nessa seção a performance obtida na rede de transputers, em tempo de processamento do algoritmo da FFT-2D.

Os tempos de processamento obtidos estão representados na Tabela 5.3 e na Figura 5.2.

Tabela 5.3 - Tempo de processamento paralelo da FFT-2D.

dimensão	tempo 1T (s)	tempo 2T (s)	tempo 4T (s)
4 X 256	0.158	0.103	0.067
8 X 256	0.330	0.209	0.130
16 X 256	0.690	0.426	0.261
32 X 256	1.469	0.884	0.534
64 X 256	3.038	1.818	1.091

Da Tabela 5.3 montamos o gráfico da Figura 5.2:

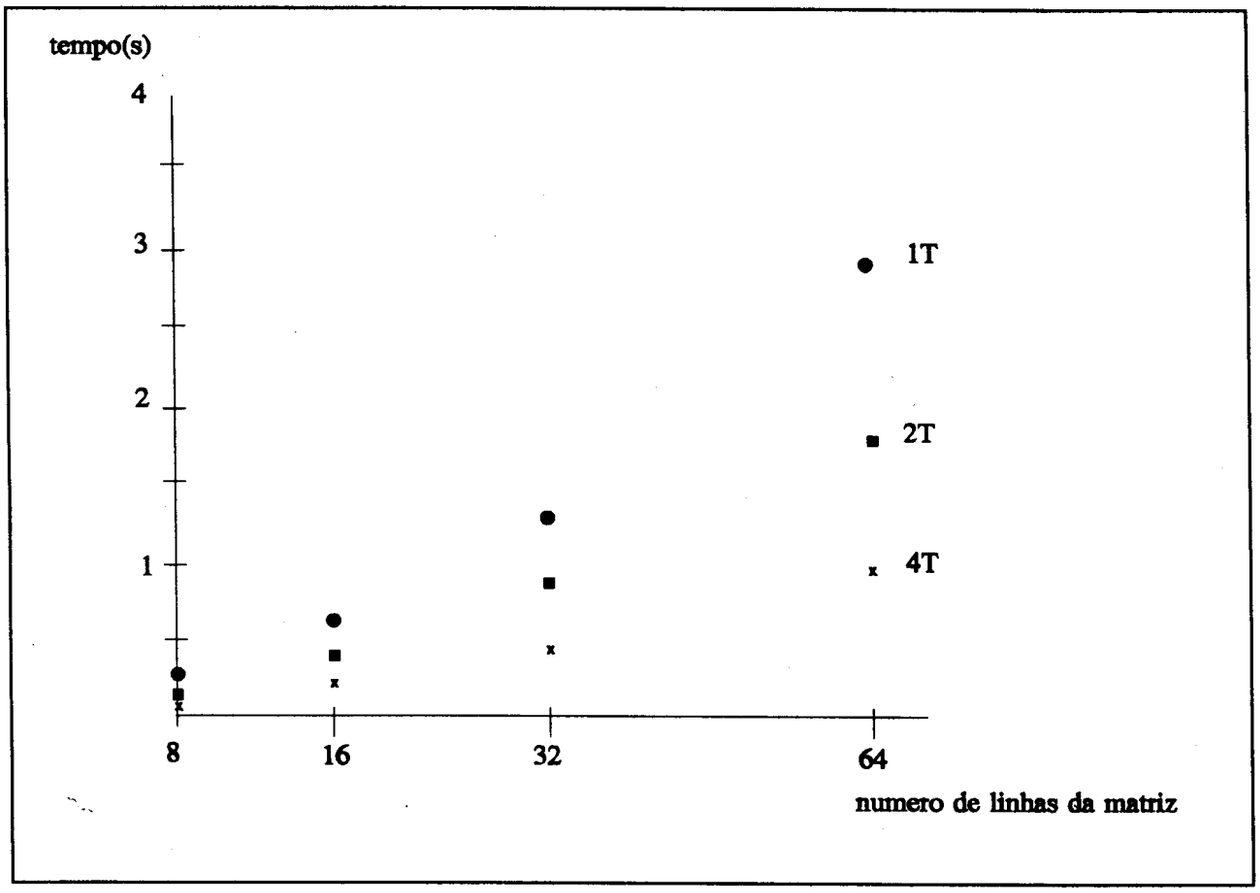

Figura 5.2 - Tempo de processamento paralelo da FFT-2D em um, dois e quatro transputers.

A partir da Figura 5.2 conclui-se que para o aumento da matriz de dados a parelelização faz com que se tenha os resultados da transformação mais rapidamente, devido ao fato que num mesmo instante há dois processamentos ocorrendo em uma rede de dois transputers, o mesmo conceito para uma rede de quatro transputers.

Também pode-se calcular o "Speedup" e a eficiência do algoritmo para cada configuração da rede de transputers utilizado, para determinar a performance do algoritmo.

O "Speedup" (S_p) é o aumento da velocidade observada de um computador paralelo com "p" processadores (Tp) com um processador (T1).

A Tabela 5.4 mostra os "Speedup" alcançado para cada configuração da rede utilizada.

$$S_p = \frac{T1}{Tp}$$

Tabela 5.4 - "Speedup" para 2 e 4 transputers.

dimensão	"Speedup" 2T	"Speedup" 4T
4 X 256	1.53	2.35
8 X 256	1.57	2.53
16 X 256	1.61	2.64
32 X 256	1.66	2.75
64 X 256	1.67	2.78

Para situação ideal o "Speedup" tem valor igual ao número de processadores na rede, mas como na prática há tanto a sobrecarga de comunicação quanto a impossibilidade de aproveitamento total da capacidade de processamento entre os processadores, isto provoca um aumento no tempo de execução no computador paralelo, fazendo com que o "Speedup" seja menor.

Pelos dados coletados na Tabela 5.4, observa-se que os "Speedups" encontrados são compatíveis com a teoria, sendo menores do que no caso ideal.

Para ilustrar os "Speedup" entre as configurações para cada implementação utilizada é construído a Figura 5.3 dada a seguir.

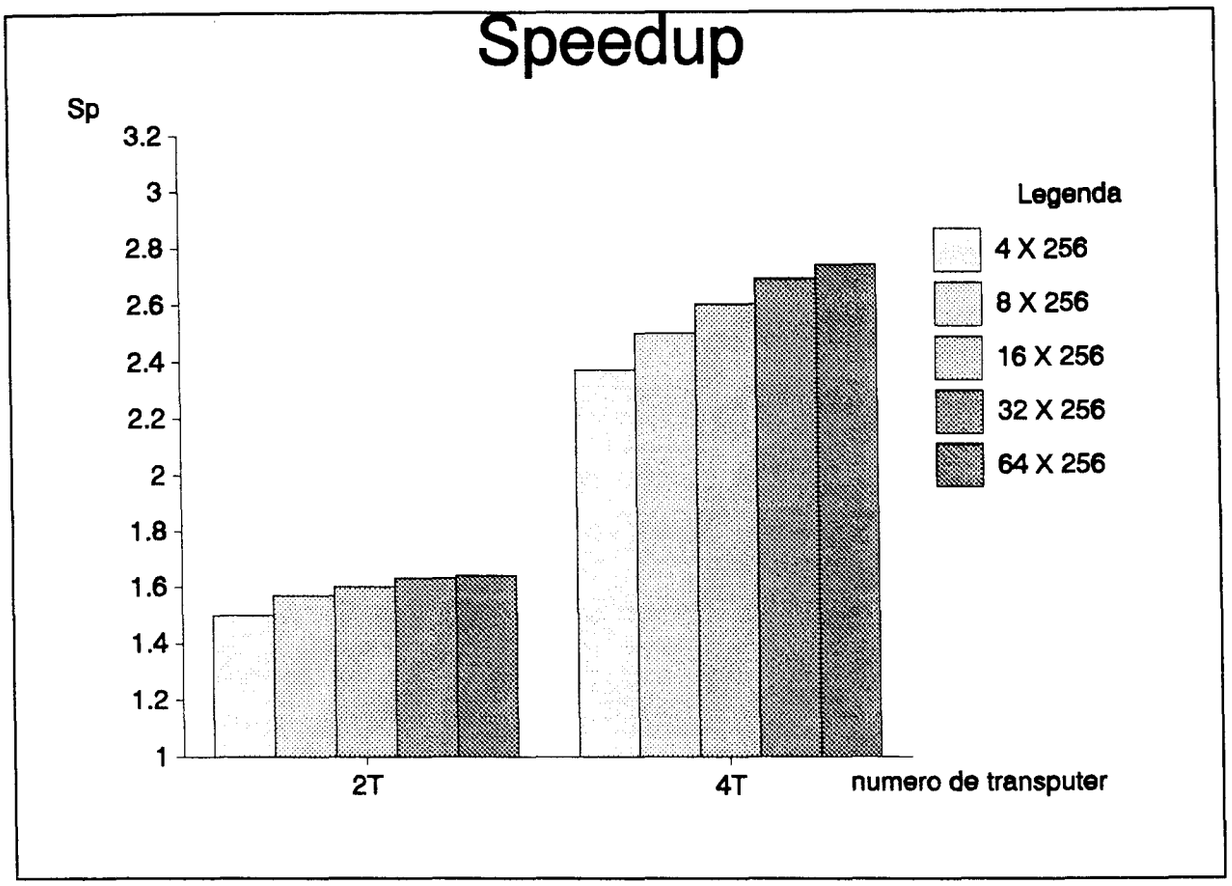

Figura 5.3 - "Speedup" para dois e quatro transputers.

O cálculo da eficiência (E_p) de um algoritmo paralelo numa máquina pararela é definido como:

$$E_p = \frac{S_p}{p}$$

onde S_p é o "Speedup" e "p" o número de processadores na máquina paralela.

Na situação ideal, onde "Speedup" é igual a p, a eficiência é máxima e vale 1 (100%). Na prática, a eficiência será menor que um devido as perdas de paralelismo no algoritmo e de algumas sobrecargas de comunicação.

A eficiência encontrada para cada configuração da implementação realizada no cálculo

da FFT-2D é dado a Tabela 5.5.

Tabela 5.5 - Eficiência para 2 e 4 transputers.

dimensão	E_p 2T	E_p 4T
4 X 256	0.767	0.590
8 X 256	0.789	0.635
16 X 256	0.810	0.661
32 X 256	0.831	0.688
64 X 256	0.835	0.696

Para ilustrar a eficiência é dado a Figura 5.4.

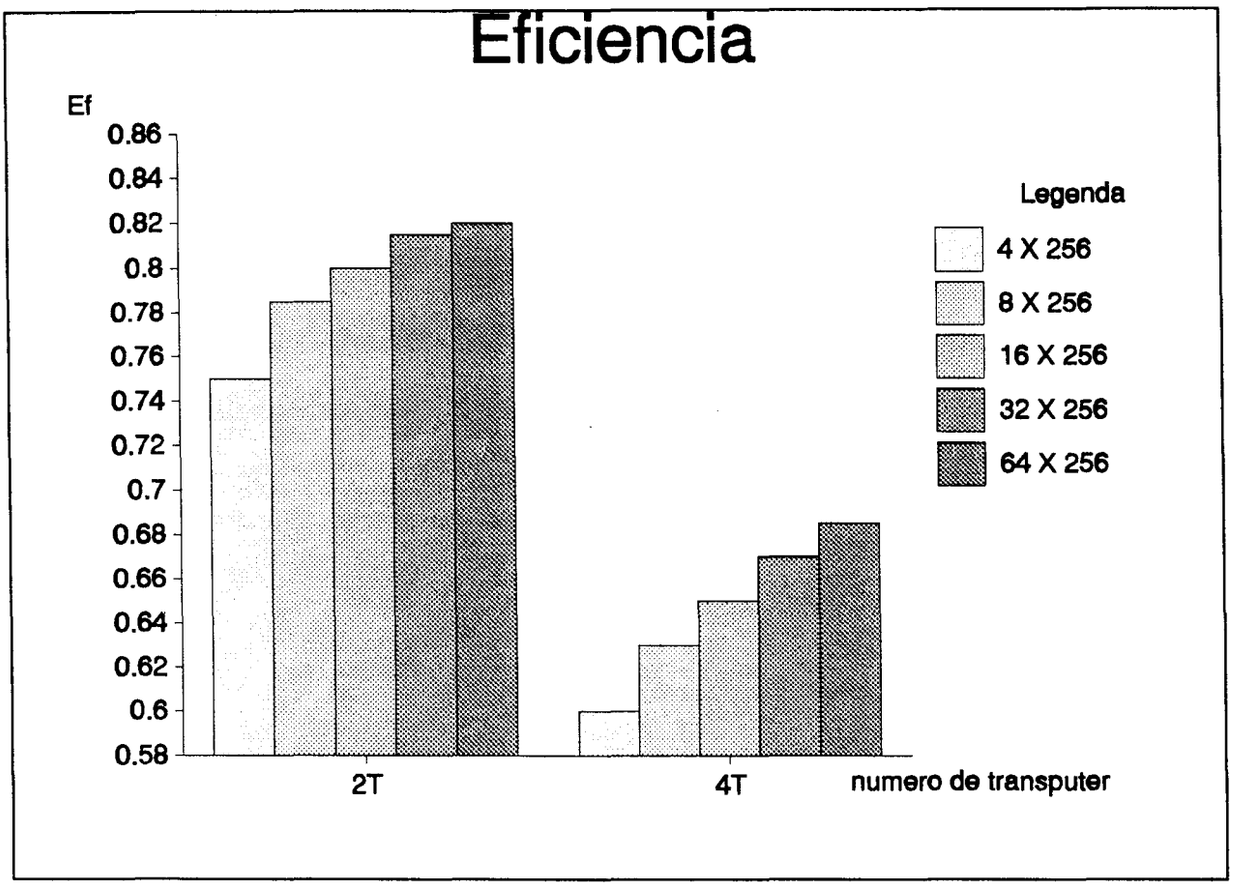

Figura 5.4 - Eficiência para a rede com dois e quatro transputers.

5.2 - Conclusão.

Pelos resultados obtidos na seção anterior, pode-se concluir que foi possível utilizando a técnica de divisão algorítmica em forma de "farm", realizar a paralelização do algoritmo da FFT-2D e diminuir o seu tempo de processamento.

Para uma rede de dois transputer conseguimos uma eficiência de 84,34% para o algoritmo da FFT-2D, o que implica que na rede de transputer o algoritmo está sendo executado 1.69 vezes mais rápido que num único processador.

E para uma rede de quatro transputer temos uma eficiência de 69,34%, sendo que temos a execução do algoritmo 2.77 vezes mais rápido que num único processador.

Os resultados acima, indicam que para um número maior de transputers na rede e para grande dados da matriz a paralelização do algoritmo da FFT-2D tem eficiência melhor, mesmo com o "overhead" advindo da comunicação de dados devido a distribuição dos mesmos na rede de transputers.

REFERÊNCIAS BIBLIOGRÁFICAS

[BRI-74] - BRIGHAM, E. O. **The Fast Fourier Transform.** New Jersey, Prentice-Hall. 1974.

[COO-65] - COOLEY, J. W.; TUCKEY, J. W. An algorithm for the machine calculation of complex Fourier series. **Mathematics of Computation**, v.19, n.90, p. 297-301, 1965.

[COO-69] - COOLEY, J. W.; GARWIN, R. L.; RADER, C. M.; BOGERT, B. P.; STOCKHAM, T.C. The 1968 Arden House Workshop on fast Fourier transform processing. **IEEE Trans. on Audio and Eletroacoustics**, v.AU-15, n.02, p. 76-79, 1969.

[COS-86] - COSTA, L.F. Arquitetura dedicada para o cálculo da Transformada de Fourier. **Tese de mestrado**, Instituto de Física e Química de São Carlos, 1986.

[DAN-42] - DANIELSON, G. C.; LANCZOS, C. Some Improvements in Pratical Fourier Analysis and Their Application to X-Ray Scattering from Liquids. **J. Franklin Institute**, v.233, p. 365-380, 1942.

[DUD-84] - DUDGEON, D. E.; MERSEREAU, R. M. **Multidimensional Digital Signal Processing**. New Jersey, Prentice-Hall, 1984.

[GUR-85] - GURD, J. R.; KIRKHAM, C. C.; WATSON, I. The Manchester prototype dataflow computer. **Communications of the ACM**, v.28, n.01, p.34-52, 1985.

[HOA-78] - HOARE, C. A. R. Communicating sequencial processes. **Communications of the ACM**, v.21, n.08, p. 666-677, 1978.

[HOC-88] - HOCKNEY, R. W.; JESSHOPE, C. R. **Parallel Computer 2.** Philadelphia, IOP Publishing, 1988.

SERVIÇO DE BIBLIOTECA ... FÍSICA

[OPP-75] - OPPENHEIM, A. V.; SCHAFER, R. W. **Digital Signal Processing. New Jersey, Prentice-Hall**, 1975.

[PAI-90] - PAIVA, M. S. Projeto de uma Arquitetura de Hardware para Visualização de Imagens Digitais. **Tese de Doutorado.** Instituto de Física e Química de São Carlos, 1990.

[POU-86] - POUTAIN, D. **A Tutorial Instruction to OCCAM Programming.** INMOS Ltd., Bristol, 1986.

[RUD-66] - RUDNICK, P. Notes on the Calculation of Fourier Series. **Mathematics of Computation**, v.20, p. 429-430, 1966.

[RUN-64] - RUNGE C.; KÖNING, H. Die Grundlehren der Mathematischen Wissenschaften. **Vorlesungen über Numerischen Rechnen**, v.11, Julius Springer, Berlim, 1964.

[SEI-85] - SEITZ, C. L. The cosmic cube. **Communications of the ACM**, v.28, n.01, p. 22-33, 1985.

[STU-39] - STUMPFF, K. **Tafeln und Aufgaben für Harmonischen Analyse und Periodogrammrechnung.**, Julius Springer, Berlin, 1939.

[TDS-88] - TDS - **Transputer Development System.** New York, Prentice-Hall, 1988.

[THE-87] - **The Transputer Instruction Set - A Compiler Write 's Guide.** INMOS Ltd., Bristol, 1987.

[THO-63] - THOMAS, L. H. **Using a Computer to Solve Problems in Phisics Applications of Digital Computer.** Boston, Guim, 1963.

[TOR-88] - TORRE NETO, A. Projeto e construção de um digitalizador e premediador

de dois canais para TRMN. **Tese de doutorado.** Imstituto de Física de São Carlos, 1988.

[TRA-91] - TRAINA, A. J. M. Estudo e Implementação de "Soft" Dedicado para um sistema de visualização de imagem. **Tese de Doutorado**, Instituto de Física e Química de São Carlos, 1991.

[YAT-37] - YATES, F. The Design and Analysis of Fatorial Experiments. Commonwealth Agriculture Bureaux. Burks, England Farnam Royal, 1937.

APÊNDICE A

Nesse apêndice estão as bibliotecas criadas, bem como a rotina principal para a execução do programa da FFT-2D sequencial, utilizando somente a linguagem OCCAM.

```
*****************************************************************************************
Biblioteca window.tsr - cria janela para saida de dados na tela.
*****************************************************************************************

#USE strmhdr
#USE streamio

PROC window(CHAN OF SS screen)  -- so cria o contorno da janela

  INT i, x, y:
  PROC clean.screen( )
    ss.write.string(screen,"*#1B[2J")
  :
  SEQ
    clean.screen( )
    SEQ
      {{{  criando a janela
      SEQ i=0 FOR 14
        ss.write.string(screen,"%%%%%")
      x:= 0
      y:= 0
      SEQ i=0 FOR 22
        SEQ
          ss.goto.xy(screen, y, x)
          ss.write.string(screen,"%")
          x:= x+1
      x:= 0
      y:= 70
      SEQ i=0 FOR 22
        SEQ
          ss.goto.xy(screen, y, x)
          ss.write.string(screen,"%")
          x:= x+1
      ss.goto.xy(screen,0,x)
      SEQ i=0 FOR 14
        ss.write.string(screen,"%%%%%")
      ss.write.string(screen,"%")
      }}}
    ss.write.nl(screen)
:
```

****

Biblioteca subrot.tsr -- Guarda os valores das constantes e os protocolos dos canais

****

```
{{{ variaveis constantes
VAL INT ROW IS 1:
VAL INT COLUMN IS 0:
VAL INT R IS 0:
VAL INT I IS 1:
VAL INT MAXROW IS 64:
VAL INT MAXCOL IS 256:
}}}
{{{ protocolos dos canais para processamento sequencial e paralelo
PROTOCOL canais IS INT;INT::[][MAXCOL]REAL32; INT::[][MAXCOL]REAL32:
PROTOCOL can IS INT;INT::[][MAXROW]REAL32; INT::[][MAXROW]REAL32:
PROTOCOL cteste IS INT; INT::[]REAL32; INT::[]REAL32; INT:
PROTOCOL cteste1 IS INT; INT::[]REAL32; INT:
}}}
```

****

Biblioteca ctsubrot.tsr -- rotina que contém as operações aritméticas para o cálculo das butterflys.

****

```
{{{
#USE sphdr
#USE userio
#USE ioconv
#USE strings
#USE snglmath
#USE "subrot.tsr"
}}}
PROC oper( []REAL32 FR, FI, VAL INT j, l)
  {{{ oper
  INT n:
  REAL32 tmpr, tmpi:
  SEQ
    {{{
    tmpr:= FR[j]
    FR[j]:= FR[l] - tmpr
    FR[l]:= FR[l] + tmpr
    }}}
    {{{
    tmpi:= FI[j]
    FI[j]:= FI[l] - tmpi
    FI[l]:= FI[l] + tmpi
    }}}
  }}}
:
```

```
PROC oper1( VAL INT ja, jb, jc, jd, sign, []REAL32 FR, FI)
 {{{ oper1
 REAL32 t0r, t0i, t1r, t1i, t2r, t2i, t3r, t3i:
 SEQ
  {{{ atribui valores para t..r e t..i
   {{{
   t0r:= FR[ja] + FR[jb]
   t1r:= FR[ja] - FR[jb]
   t2r:= FR[jc] + FR[jd]
   t3r:= FR[jc] - FR[jd]
   }}}
   {{{
   t0i:= FI[ja] + FI[jb]
   t1i:= FI[ja] - FI[jb]
   t2i:= FI[jc] + FI[jd]
   t3i:= FI[jc] - FI[jd]
   }}}
   {{{
   FR[ja]:= t0r + t2r
   FR[jc]:= t0r - t2r
   }}}
   {{{
   FI[ja]:= t0i + t2i
   FI[jc]:= t0i - t2i
   }}}
  }}}
  IF
   sign < 0
    {{{
    SEQ
     t3r:= t3r * (REAL32 ROUND (-1))
     t3i:= t3i * (REAL32 ROUND (-1))
    }}}
   TRUE
    SKIP
  {{{ calcula para FR[i1R][j1R] e FI[i1I][j1I]
   {{{
   FR[jb]:= t1r - t3i
   FR[jd]:= t1r + t3i
   }}}
   {{{
   FI[jb]:= t1i + t3r
   FI[jd]:= t1i - t3r
   }}}
  }}}
 }}}
:
```

****

Biblioteca busca.tsr -- Rotina que procura um determinado caracter em um arquivo

```
*****************************************************************************************

{{{
#USE sphdr
#USE userio
#USE ioconv
#USE strings
#USE snglmath
#USE "subrot.tsr"
}}}
PROC busca(VAL BYTE key, VAL []BYTE dado, INT i, VAL INT bytes.read )
  {{{  procedimento busca
  SEQ
    i:=0
    WHILE ((dado[i] <> key) AND (i<bytes.read))
      i:= i+1
  }}}
:
}}}
*****************************************************************************************
Biblioteca tipoleitura.tsr -- Rotina que contém as rotinas com tipo de leitura do arquivo
*****************************************************************************************


SC leitura.tsr  -- leitura da forma leitura centralizada.

{{{  bibliotecas
#USE strmhdr
#USE sphdr
#USE "subrot.tsr"
#USE "busca00.tsr"
#USE strings
#USE splib
#USE solib
#USE sklib
#USE streamio
#USE ioconv
}}}
PROC le.arq.center( CHAN OF SS screen, CHAN OF SP fs, ts, VAL INT32 streamid,
                VAL INT nlin, ncol, [][]REAL32 FR, FI)
  {{{  learqcenter
  {{{  declaracao das variaveis locais
  [100]BYTE vet:
  INT x, k, l, i, j, ii, iaux, sentido, cont, pos:
  INT canal, auxcol:
  BYTE key1, result, result1:
  INT bytes.read, ptr, number:
  BOOL error, ok:
  INT retorno:
  }}}
```

```
SEQ
  {{{ inicializacao das variaveis
  ii:= 0
  sentido:= 1
  cont:= 2
  x:= nlin/2
  }}}
  so.eof (fs, ts, streamid, result1)
  k:=0
  WHILE ((k < nlin) AND (result1 <> spr.ok))
    {{{
    SEQ
      {{{ ajusta o sentido
      i:= x + (ii * sentido)
      sentido:= (-1) * sentido
      }}}
      {{{ ajusta o contador
      IF
        cont = 2
          SEQ
            ii:= ii + 1
            cont:= 1
        cont <> 2
          cont:= cont + 1
      }}}
      {{{ le o numero da aquisicao
      {{{ pula duas linhas
      so.gets(fs, ts, streamid, bytes.read, vet, result)
      so.gets(fs, ts, streamid, bytes.read, vet, result)
      }}}
      IF
        k = 0
          {{{
          SEQ
            so.gets(fs, ts, streamid, bytes.read, vet, result)
            {{{ rotina busca
            IF
              result = spr.ok
                SEQ
                  busca('#', vet, retorno, bytes.read)
                  STRINGTOINT(error, iaux, [vet FROM retorno+1 FOR 3])
              TRUE
                SKIP
            }}}
          }}}
        k <> 0
          {{{
          SEQ
            so.gets(fs, ts, streamid, bytes.read, vet, result)
            so.gets(fs, ts, streamid, bytes.read, vet, result)
            {{{ rotina busca
```

```
      IF
        result = spr.ok
          SEQ
            busca('#', vet, retorno, bytes.read)
            STRINGTOINT(error, iaux, [vet FROM retorno+1 FOR 3])
        TRUE
          SKIP
      }}}
    }}}
  TRUE
    SKIP
}}}
{{{ pula uma linha do arquivo
so.gets(fs, ts, streamid, bytes.read, vet, result)
}}}
{{{ le o numero do canal e de words da parte real
so.gets(fs, ts, streamid, bytes.read, vet, result)
key1:='('
{{{ rotina busca
IF
  result = spr.ok
    SEQ
      busca(key1, vet, retorno, bytes.read)
      STRINGTOINT(error, canal, [vet FROM retorno+1 FOR 3])
  TRUE
    SKIP
}}}
STRINGTOINT(error, auxcol, [vet FROM retorno+4 FOR 3])
}}}
{{{ le o dado real
j:=0
SEQ l=0 FOR 32
  SEQ
    so.gets(fs, ts, streamid, bytes.read, vet, result)
    vet[bytes.read]:= ' '
    IF
      result = spr.ok
        SEQ
          ok:= TRUE
          ptr:= 0
          SEQ pos=0 FOR 8
            SEQ
              next.int.from.line(vet, ptr, number, ok)
              FR[i][j]:= REAL32 ROUND number
              j:=j+1
      TRUE
        SKIP
}}}
{{{ pula uma linha do arquivo
so.gets(fs, ts, streamid, bytes.read, vet, result)
}}}
```

```
      {{{ le o numero do canal e de words da parte imaginaria
      so.gets(fs, ts, streamid, bytes.read, vet, result)
      key1:='('
      {{{ rotina busca
      IF
        result=spr.ok
          SEQ
            busca( key1, vet, retorno, bytes.read)
            STRINGTOINT(error, canal, [vet FROM retorno+1 FOR 3])
        TRUE
          SKIP
      }}}
      STRINGTOINT(error, auxcol, [vet FROM retorno+4 FOR 3])
      }}}
      {{{ le o dado imaginario
      j:=0
      SEQ l=0 FOR 32
        SEQ
          so.gets(fs, ts, streamid, bytes.read, vet, result)
          vet[bytes.read]:= ' '
          IF
            result = spr.ok
              SEQ
                ok:= TRUE
                ptr:= 0
                SEQ pos=0 FOR 8
                  SEQ
                    next.int.from.line(vet, ptr, number, ok)
                    FI[i][j]:= REAL32 ROUND number
                    j:=j+1
            TRUE
              SKIP
      }}}
      k:= k+1
      so.eof (fs, ts, streamid, result1)
    }}}
  }}}
:


{{{ SC leitura1.tsr -- forma de leitura dos dados do tipo leitura por linha
{{{ bibliotecas
#USE strmhdr
#USE streamio
#USE solib
#USE sklib
#USE splib
#USE sphdr
#USE ioconv
#USE strings
#USE "subrot.tsr"
```

```
#USE "ctsubrot.tsr"
#USE "busca00.tsr"
}}}
PROC leporlinha( CHAN OF SS screen, CHAN OF SP fs, ts, VAL INT32 streamid,
                 VAL INT nlin, ncol, [][]REAL32 MR, MI)
  {{{ leporlinha
  {{{ declaracao
  {{{ declaracao das variaveis locais
  [100]BYTE dado:
  INT x, k, l, i, j, ii, iaux, sentido, cont, retorno1, pos:
  INT canal, auxcol:
  BYTE key, result,result1:
  INT bytes.read, ptr, number:
  BOOL error, ok:
  }}}
  {{{ declaracao de variaveis locais
  INT size:
  }}}
  }}}
  SEQ
    so.eof (fs, ts, streamid, result1)
    k:=0
    WHILE ((k < nlin) AND (result1 <> spr.ok))
      SEQ
        {{{ lendo por linha
        {{{ le o numero da aquisicao
        {{{ pula duas linhas
        so.gets(fs, ts, streamid, bytes.read, dado, result)
        so.gets(fs, ts, streamid, bytes.read, dado, result)
        }}}
        IF
          k = 0
            {{{
            SEQ
              so.gets(fs, ts, streamid, bytes.read, dado, result)
              {{{ rotina busca
              IF
                result=spr.ok
                  SEQ
                    busca( '#', dado, i, bytes.read)
                    STRINGTOINT(error, iaux, [dado FROM i+2 FOR 3])
                TRUE
                  SKIP
              }}}
            }}}
          k <> 0
            {{{
            SEQ
              so.gets(fs, ts, streamid, bytes.read, dado, result)
              so.gets(fs, ts, streamid, bytes.read, dado, result)
              {{{ rotina busca
```

```
    IF
      result=spr.ok
        SEQ
          busca('#', dado, i, bytes.read)
          STRINGTOINT(error, iaux, [dado FROM i+2 FOR 3])
      TRUE
        SKIP
    }}}
  }}}
 TRUE
   SKIP
}}}
{{{ pula uma linha do arquivo
so.gets(fs, ts, streamid, bytes.read, dado, result) -- pula 1 linha
}}}
{{{ le o numero do canal e de words da parte real
so.gets(fs, ts, streamid, bytes.read, dado, result)
key:='('
{{{ rotina busca
IF
 result=spr.ok
   SEQ
     busca( key, dado, i, bytes.read)
     STRINGTOINT(error, canal, [dado FROM i+2 FOR 3])
 TRUE
   SKIP
}}}
STRINGTOINT(error, auxcol, [dado FROM i+4 FOR 3])
}}}
{{{ le o dado real
j:=0
SEQ l=0 FOR 32
 SEQ
   so.gets(fs, ts, streamid, bytes.read, dado, result)
   dado[bytes.read]:= ' '
   IF
     result = spr.ok
       SEQ
         ok:= TRUE
         ptr:= 0
         SEQ pos=0 FOR 8
           SEQ
             next.int.from.line(dado, ptr, number, ok)
             MR[i][j]:= REAL32 ROUND number
             j:=j+1
     TRUE
       SKIP
}}}
{{{ pula uma linha do arquivo
so.gets(fs, ts, streamid, bytes.read, dado, result) -- pula 1 linha
}}}
```

```
    {{{ le o numero do canal e de words da parte imaginaria
    so.gets(fs, ts, streamid, bytes.read, dado, result)
    key:='('
    {{{ rotina busca
    IF
      result=spr.ok
        SEQ
          busca( key, dado, i, bytes.read)
          STRINGTOINT(error, canal, [dado FROM i+2 FOR 3])
      TRUE
        SKIP
    }}}
    STRINGTOINT(error, auxcol, [dado FROM i+4 FOR 3])
    }}}
    {{{ le o dado imaginario
    j:=0
    SEQ l=0 FOR 32
      SEQ
        so.gets(fs, ts, streamid, bytes.read, dado, result)
        dado[bytes.read]:= ' '
        IF
          result = spr.ok
            SEQ
              ok:= TRUE
              ptr:= 0
              SEQ pos=0 FOR 8
                SEQ
                  next.int.from.line(dado, ptr, number, ok)
                  MI[i][j]:= REAL32 ROUND number
                  j:=j+1
          TRUE
            SKIP
    }}}
    }}}
    k:= k+1
    so.eof (fs, ts, streamid, result1)
  }}}
:
```

****

Biblioteca transfor.tsr -- Rotina que contém as rotinas do algorítmo de CT para o método "Radix-4" e a rotina de filtragem dos dados da matriz.

****

SC transfft.tsr -- Rotina que contém o método "Radix-4" que engloba as subrotinas bitreverso, que realiza a reversão dos dados da matriz e "cool2" que é o algorítmo de CT.

```
{{{ bibliotecas utilizadas pelas subrotinas
#USE sphdr
#USE userio
#USE ioconv
#USE strings
#USE snglmath
#USE "subrot.tsr"
#USE "ctsubrot.tsr"
}}}

-- Faz o bitreverso dos dados de entrada para calcular a FFT
PROC bitreverso( VAL INT row, []REAL32 FR, FI, []INT dim, VAL INT i)
  {{{ bitreverso
  {{{ declaracao de variaveis
  INT j, n, d:
  INT i, k, irev, aux, dimaux:
  REAL32 tmp, tmp1:
  }}}
  SEQ
    dimaux:= dim[row]
    irev:=0
    SEQ i=0 FOR dimaux
      SEQ
        IF
          i < irev
            {{{ faz unscrambling
            SEQ
              {{{ muda parte real
              tmp:= FR[irev]
              FR[irev]:= FR[i]
              FR[i]:= tmp
              }}}
              {{{ muda parte imaginaria
              tmp1:= FI[irev]
              FI[irev]:= FI[i]
              FI[i]:= tmp1
              }}}
            }}}
          TRUE
            SKIP
        {{{ vai para o proximo endereco
        aux:= dimaux/2
        WHILE( (irev >= aux) AND (aux > 1) )
          SEQ
            irev:= irev - aux
            aux:= aux/2
        }}}
        irev:= irev + aux
  }}}
:
```

```
-- algorítmo de CT pelo método "Radix-4"
PROC cool2( VAL INT sign, row, []REAL32 FR, FI, VAL []INT dim, VAL INT lc)
  {{{ cool2
  {{{ variaveis
  REAL32 twopi, tmpr, tmpr1, tmpr2, wr, wi, wstpr, wstpi:
  REAL32 theta, sinth, w2r, w2i, w3r, w3i:
  INT j:
  INT cont, i , next, med, valor, ia, ib, ic, id:
  INT i5, i2, ip3, dimaux:
  REAL32 TWOPI:
  REAL32 tmpi, tmpi1, tmpi2:
  REAL32 a, b, a1, b1, a2, b2:
  }}}
  SEQ
    {{{ atribuindo valores para twopi
    TWOPI:= 6.2831853072(REAL32)
    twopi:= TWOPI * (REAL32 ROUND sign)
    }}}
    {{{ iniciando valores para med e valor e dimaux
    dimaux:= dim[COLUMN]
    med:= dimaux
    valor:= 1
    }}}
    {{{ colocar em potencia de quatro se maior que dois
    WHILE ( med >2 )
      med := med/4
    }}}
    {{{ se potencia de dois ajusta para potencia de quatro
    IF
      med > 1
        {{{
        SEQ
          i:=0
          WHILE ( i < dimaux )
            SEQ
              next:= i+1
              {{{ faz unscrambling
              oper(FR, FI, next, i)
              }}}
              i:=i+2
          valor:= 2
        }}}
      TRUE
        SKIP
    }}}
    {{{ FFT em potencia de quatro
    WHILE ( valor < dimaux )
      SEQ
        {{{ iniciacao das variaveis
        ip3:= valor * 4
        theta:= twopi / (REAL32 ROUND ip3)
```

```
sinth:= SIN(theta/(REAL32 ROUND 2))
wstpr:= sinth * (sinth * (REAL32 ROUND (-2)))
wstpi:= SIN(theta)
wr:= 1.0(REAL32)
wi:= 0.0(REAL32)
}}}
SEQ i2=0 FOR valor
  SEQ
    {{{  na primeira vez pula
    IF
      i2 <> 0
        SEQ
          w2r:= (wr * wr) - (wi * wi)
          w2i:= (wr * wi) * (REAL32 ROUND 2)
          w3r:= (w2r * wr) - (w2i * wi)
          w3i:= (w2r * wi) + (w2i * wr)
      TRUE
        SKIP
    }}}
    i5:=i2
    WHILE (i5 < dimaux)
      SEQ
        {{{  iniciacao das variaveis locais
        ia:= i5
        ib:= ia + valor
        ic:= ib + valor
        id:= ic + valor
        }}}
        {{{  efetuando operacao nas linhas ou colunas
        IF
          i2 <> 0
            SEQ
              {{{  calculo
              {{{
              {{{  temp1
              tmpr:= FR[ib]
              FR[ib]:= (w2r * FR[ib]) - (w2i * FI[ib])
              FI[ib]:= (w2r * FI[ib]) + (w2i * tmpr)
              }}}
              {{{  temp2
              tmpr:= FR[ic]
              FR[ic]:= (wr * FR[ic]) - (wi * FI[ic])
              FI[ic]:= (wr * FI[ic]) + (wi * tmpr)
              }}}
              {{{  temp3
              tmpr:= FR[id]
              FR[id]:= (w3r * FR[id]) - (w3i * FI[id])
              FI[id]:= (w3r * FI[id]) + (w3i * tmpr)
              }}}
              }}}
              }}}
```

```
            TRUE
              SKIP
          }}}
          {{{  chamada para rotina oper1
          oper1 (ia, ib, ic, id, sign, FR, FI)
          }}}
          i5:= i5 + ip3
        {{{  ajuste das variaveis
        tmpr:= wr
        wr:=((wstpr * tmpr) - (wstpi * wi)) + tmpr
        wi:=((wstpr * wi) + (wstpi * tmpr)) + wi
        }}}
      valor:= ip3
    }}}
  }}}
:


SC filtrage.tsr -- faz a filtragem dos dados
{{{
#USE "subrot.tsr"
#USE strmhdr
#USE sphdr
#USE streamio
#USE userio
}}}
PROC filtragem(CHAN OF SS screen,INT ncol, nlin, [][]REAL32 MR, MI)
  {{{  filtragem
  {{{  declaracao das variaveis
  REAL32 mediaR, mediaI:
  [256]REAL32 medR, medI:
  INT lim, j:
  }}}
  SEQ
    {{{  filtragem dos dados de entrada
    IF
      ncol > 10
        lim:= ncol - 10
      ncol < 10
        lim:= ncol
      TRUE
        SKIP
    -- Calcula a media dos ultimos 10 pontos de
    -- cada linha da matriz real e imaginaria
    mediaR:= 0.0(REAL32)
    mediaI:= 0.0(REAL32)
    SEQ i=0 FOR nlin-1
      SEQ
        medR[i]:= 0.0(REAL32)
        medI[i]:= 0.0(REAL32)
        j:=lim
```

```
      WHILE (j < ncol)
        SEQ
          medR[i]:= medR[i] + MR[i][j]
          medI[i]:= medI[i] + MI[i][j]
          j:=j+1
      medR[i]:= medR[i]/(10.0(REAL32))
      medI[i]:= medI[i]/(10.0(REAL32))
      mediaR:= mediaR + medR[i]
      mediaI:= mediaI + medI[i]
    mediaR:= mediaR/( REAL32 ROUND nlin)
    mediaI:= mediaI/( REAL32 ROUND nlin)
    ss.write.nl(screen)
    ss.goto.xy(screen, 5, 7)
    ss.write.string(screen,"mediaR:= ")
    ss.goto.xy(screen, 15, 7)
    ss.write.real32(screen, mediaR, 5, 5)
    ss.goto.xy(screen, 5, 9)
    ss.write.string(screen,"mediaI:= ")
    ss.goto.xy(screen, 15, 9)
    ss.write.real32(screen, mediaI, 5, 5)
    -- RETIRAR A MEDIA( BASE ) DE TODOS OS PONTOS R E I
    SEQ i=0 FOR nlin
      SEQ j=0 FOR ncol
        SEQ
          MR[i][j]:= MR[i][j] - mediaR
          MI[i][j]:= MI[i][j] - mediaI
    }}}
  }}}
:
```

****

Biblioteca met.tsr -- Contém as rotinas que serão utilizados na criação dos arquivos de saída para a geração de imagens.

****

```
SC reestrutura.tsr  -- reeestrutura a matriz para saida da imagem
{{{
#USE "subrot.tsr"
#USE strmhdr
#USE streamio
#USE sphdr
#USE userio
#USE ioconv
```

```
#USE strings
#USE snglmath
}}}
PROC reestrutura( []INT dim, [][]REAL32 F)
  {{{  reestrutura
  INT i, j, dlin, dcol, d, n:
  REAL32 salva:
  SEQ
    dlin:= dim[1]/2
    dcol:= dim[0]/2
    {{{  troca linha
    SEQ i=0 FOR dlin
      SEQ j=0 FOR dcol
        SEQ
          salva:=F[i][j]
          F[i][j]:= F[i+dlin][j+dcol]
          F[i+dlin][j+dcol]:= salva
    }}}
    {{{  troca coluna
    i:=dlin
    WHILE ( i < dim[1])
      SEQ
        SEQ j=0 FOR dcol
          SEQ
            salva:=F[i][j]
            F[i][j]:= F[i-dlin][j+dcol]
            F[i-dlin][j+dcol]:= salva
        i:= i+1
    }}}
  }}}
:


SC preparasaida.tsr  -- coloca os dados para a saida da imagem
{{{
#USE "subrot.tsr"
#USE strmhdr
#USE streamio
#USE sphdr
#USE userio
#USE splib
#USE solib
#USE sklib
#USE ioconv
#USE strings
#USE snglmath
}}}
PROC preparasaida( CHAN OF SS screen, CHAN OF SP fs, ts, INT32 streamid, [][]INT ponto, VAL []INT dim)
  {{{  preparasaida
  BYTE result:
  INT key:
```

SERVIÇO DE BIBLIOTECA E INFORMAÇÃO - IFOSC
FÍSICA

```
[50]BYTE file:
INT len:
BOOL error:
VAL INT ROW IS 1:
VAL INT COLUMN IS 0:
SEQ
  {{{ comentario de entrada
  ss.goto.xy(screen, 5, 11)
  ss.write.text.line(screen,"Entre com o nome do arquivo:")
  ss.goto.xy(screen, 35, 11)
  so.read.echo.line(fs, ts, len, file, result)
  }}}
  VAL imagem IS [file FROM 0 FOR len]:
  so.open(fs, ts, imagem, spt.text, spm.output, streamid, result)
  IF
    result = spr.ok
      SEQ
        ss.goto.xy(screen, 5, 13)
        ss.write.text.line(screen,"Gerando o arquivo de saida ...")
        {{{ le numero de linha e escreve no arquivo de saida
        ss.goto.xy(screen, 5, 15)
        ss.write.string(screen,"MAXLIN:=")
        so.fwrite.string(fs, ts, streamid, "(", result)
        ss.goto.xy(screen, 17, 15)
        so.read.echo.int(fs, ts, key, error)
        so.fwrite.int(fs, ts, streamid, key, 0, result)
        }}}
        {{{ le numero de coluna e escreve no arquivo de saida
        ss.goto.xy(screen, 5, 17)
        ss.write.string(screen,"MAXCOLUMN:=")
        ss.goto.xy(screen, 17, 17)
        so.fwrite.string(fs, ts, streamid, ",", result)
        so.read.echo.int(fs, ts, key, error)
        so.fwrite.int(fs, ts, streamid, key, 0, result)
        so.fwrite.string(fs, ts, streamid, ")", result)
        so.fwrite.nl(fs, ts, streamid, result)
        }}}
        SEQ i=0 FOR dim[ROW]
          SEQ
            SEQ j=0 FOR dim[COLUMN]
              SEQ
                so.fwrite.int(fs, ts, streamid, ponto[i][j], 0, result)
                so.fwrite.string(fs, ts, streamid, " ", result)
            so.fwrite.nl(fs, ts, streamid, result)
        so.fwrite.nl(fs, ts, streamid, result)
    TRUE
      SEQ
        ss.goto.xy(screen, 5, 18)
        ss.write.text.line(screen, "ARQUIVO NAO ENCONTRADO")
        ss.goto.xy(screen, 5, 20)
        ss.write.int(screen,(INT result), 0)
```

```
  so.close(fs, ts, streamid, result)
 }}}
:

SC normaliza.tsr  -- faz a normalizacao dos dados obtidos, para melhorar a imagem
{{{
#USE "subrot.tsr"
#USE strmhdr
#USE sphdr
#USE streamio
#USE userio
#USE splib
#USE solib
#USE sklib
#USE ioconv
#USE strings
#USE snglmath
}}}
PROC normaliza( CHAN OF SS screen, VAL []REAL32 fponto, []INT ponto, VAL []INT dim )
 {{{  normaliza
 INT i, j:
 REAL32 a, b, max, min:
 VAL INT HUGE IS 256:
 {{{
 VAL INT ROW IS 1:
 VAL INT COLUMN IS 0:
 }}}
 SEQ
  min:= (REAL32 ROUND HUGE)
  max:= -(REAL32 ROUND HUGE)
  SEQ i=0 FOR dim[ROW]
   SEQ j=0 FOR dim[COLUMN]
    SEQ
     {{{
     IF
      min > fponto[i][j]
       min:= fponto[i][j]
      TRUE
       SKIP
     IF
      max < fponto[i][j]
       max:= fponto[i][j]
      TRUE
       SKIP
     }}}
  {{{  escreve o valor maximo
  ss.goto.xy(screen, 5, 3)
  ss.write.string(screen,"Os dados normalizados tem:")
  ss.goto.xy(screen, 5, 5)
  ss.write.string(screen,"MAX:=")
  ss.goto.xy(screen, 12, 5)
```

```
  ss.write.real32(screen, max, 3, 3)
  }}}
  {{{  escreve o valor minimo
  ss.goto.xy(screen, 5, 7)
  ss.write.string(screen,"MIN:=")
  ss.goto.xy(screen, 12, 7)
  ss.write.real32(screen, min, 3, 3)
  }}}
  a:= ((REAL32 ROUND 127)/(max - min))
  b:= (REAL32 ROUND 127) * (min/(max - min))
  SEQ i=0 FOR dim[ROW]
    SEQ j=0 FOR dim[COLUMN]
      ponto[i][j] := (INT ROUND (((a * fponto[i][j]) - b) + 0.5(REAL32)))
 }}}
:


SC imprimi.tsr  -- coloca os dados no arquivo de saida
{{{
#USE "subrot.tsr"
#USE strmhdr
#USE sphdr
#USE streamio
#USE userio
#USE splib
#USE solib
#USE sklib
#USE ioconv
#USE strings
#USE snglmath
}}}
PROC imprimi(CHAN OF SP fs, ts, []INT dim, [][]INT ponto)
 {{{
 VAL INT ROW IS 1:
 VAL INT COLUMN IS 0:
 }}}
 {{{  imprime
 SEQ
   SEQ i=0 FOR dim[ROW]
     SEQ
       SEQ j=0 FOR dim[COLUMN]
         so.write.int(fs, ts, ponto[i][j], 5)
       so.write.nl(fs, ts)
   so.write.nl(fs, ts)
 }}}
:
```

****

Biblioteca auxiliar.tsr -- Contém as rotinas de chamada para a transformada e da leitura de dados.

****

```
SC ledados  -- escolhe como os dados serão lidos
{{{  bibliotecas
#USE "subrot.tsr"
#USE strmhdr
#USE streamio
#USE "janela.tsr"
#USE "tipoleit.tsr"
#USE "busca00.tsr"
#USE solib
#USE sklib
#USE splib
#USE sphdr
#USE ioconv
#USE strings
#USE snglmath
}}}
PROC ledados( CHAN OF SS screen, CHAN OF KS keyboard, CHAN OF SP fs, ts,
            INT32 streamid, INT nlin, ncol, maxlin, maxcol, [][]REAL32 MR, MI)
  {{{  ledados
  {{{  declaracao das variaveis e canais
  BYTE result, caract, key:
  [3]BYTE resposta, char1:
  [100]BYTE dado:
  INT i, j, k, dlin, dcol, lim, teste, char:
  INT len, bytes.read:
  INT32 position:
  BOOL flag, error:
  INT flag.center:
  INT sign:
  {{{  variaveis de temporizacao
  TIMER tempo:
  INT t1, t2, t3, t4:
  REAL32 segundo:
  INT seg:
  }}}
  }}}
  SEQ
    {{{  ledados
    window(screen)
    {{{  pula 8 linhas do arquivo de dados e comentario
    i:=0
    WHILE ( i < 9 )
      SEQ
        so.gets(fs, ts, streamid, bytes.read, dado, result)
        IF
          result = spr.ok
            {{{  imprime dado
            SEQ
              ss.goto.xy(screen, 5, 3+i)
              ss.write.text.line(screen, [dado FROM 0 FOR bytes.read])
            }}}
```

```
    TRUE
      SKIP
  i:= i + 1
}}}
{{{ lendo numero de linhas de dados
so.gets(fs, ts, streamid, bytes.read, dado, result)
ss.goto.xy(screen, 5, 13)
ss.write.text.line(screen,[dado FROM 0 FOR bytes.read])
key:=':'
{{{ rotina busca
IF
  result=spr.ok
    SEQ
      busca( key, dado, i, bytes.read)
      STRINGTOINT(error, nlin, [dado FROM i+2 FOR 3])
  TRUE
    SKIP
}}}
}}}
{{{ lendo numero de colunas de dados
so.gets(fs, ts, streamid, bytes.read, dado, result)
ss.goto.xy(screen, 5, 15)
ss.write.text.line(screen,[dado FROM 0 FOR bytes.read])
{{{ rotina busca
IF
  result=spr.ok
    SEQ
      busca( key, dado, i, bytes.read)
      STRINGTOINT(error, ncol, [dado FROM i+2 FOR 3])
  TRUE
    SKIP
}}}
}}}
{{{ ignora 2 linhas do arquivo de dados
j:=0
WHILE j<2
  SEQ
    so.gets(fs, ts, streamid, bytes.read, dado, result)
    ss.goto.xy(screen, 5, 17+j)
    ss.write.text.line(screen, [dado FROM 0 FOR bytes.read])
    j:= j+1
}}}
{{{ verificando se e center colect
so.gets(fs, ts, streamid, bytes.read, dado, result)
IF
  dado[0] = ';'
    SEQ
      ss.goto.xy(screen, 8, 19)
      ss.write.text.line(screen, " Center Colect!!")
      flag.center:= 1
  TRUE
```

```
    SKIP
}}}
{{{ inicializa as matrizes - real e imaginario
SEQ i=0 FOR nlin
  SEQ j=0 FOR ncol
    SEQ
      MR[i][j]:=0.0(REAL32)
      MI[i][j]:=0.0(REAL32)
}}}
ss.goto.xy(screen, 5, 21)
ss.write.string(screen," tecle barra de espaco para prosseguir!")
ks.read.char(keyboard, char)
window(screen)
{{{ lendo o tamanho do arquivo de saida- linha
flag:=TRUE
WHILE (flag)
  SEQ
    ss.goto.xy(screen, 5, 2)
    ss.write.text.line(screen,"Entre com o tamanho da matriz resultado")
    ss.goto.xy(screen, 5, 4)
    ss.write.string(screen,"Entre com o numero de LINHAS: ")
    ss.goto.xy(screen, 35, 4)
    so.read.echo.int(fs, ts, maxlin, error)
    {{{ mensagem de erro
    IF
      error = FALSE
        flag:= FALSE
      TRUE
        SEQ
          ss.goto.xy(screen, 5, 4)
          ss.write.text.line(screen,"numero de linhas MAXIMO = 512 !")
    }}}
}}}
{{{ lendo o tamanho do arquivo de saida - coluna
flag:=TRUE
WHILE (flag)
  SEQ
    ss.goto.xy(screen, 5, 6)
    ss.write.string(screen,"Entre com o numero de COLUNAS: ")
    ss.goto.xy(screen, 36, 6)
    so.read.echo.int(fs, ts, maxcol, error)
    {{{ mensagem de erro
    IF
      error = FALSE
        flag:=FALSE
      TRUE
        SEQ
          ss.goto.xy(screen, 5, 6)
          ss.write.text.line(screen,"numero de colunas MAXIMO = 512 !")
    }}}
}}}
```

```
  ss.goto.xy(screen, 5, 9)
  ss.write.text.line(screen,"Leitura dos dados...")
  {{{ se for center colect chama le.arq.center
  IF
    flag.center = 1
      SEQ
        ss.goto.xy(screen, 5, 11)
        ss.write.text.line(screen,"Lendo da forma Center Collect - aguarde...")
        tempo ? t1
        le.arq.center(screen, fs, ts, streamid, nlin, ncol, MR, MI)
        tempo ? t2
        ss.goto.xy(screen, 5, 14)
        ss.write.string(screen,"O tempo de leitura foi de: ")
        segundo:=(REAL32 ROUND(t2 MINUS t1))*(64.0(REAL32)*POWER(10.0(REAL32),-6.0(REAL32)))
        ss.goto.xy(screen, 33, 14)
        ss.write.real32(screen, segundo, 5,3)
    TRUE
      SEQ
        ss.goto.xy(screen, 5, 11)
        ss.write.text.line(screen,"Nao e Center Collect")
        ss.goto.xy(screen, 5, 14)
        ss.write.text.line(screen,"lendo por linha")
        tempo ? t1
        leporlinha(screen, fs, ts, streamid, nlin, ncol, MR, MI)
        tempo ? t2
        ss.goto.xy(screen, 5, 14)
        ss.write.string(screen,"O tempo de leitura foi de: ")
        segundo:=(REAL32 ROUND(t2 MINUS t1))*(64.0(REAL32)*POWER(10.0(REAL32),-6.0(REAL32)))
        ss.goto.xy(screen, 33, 14)
        ss.write.real32(screen, segundo, 5,3)
  }}}
  }}}
}}}
:


SC fftuni.tsr  -- chamada das rotinas de transformacao por linhas ou por colunas
{{{ bibliotecas utilizadas
#USE strmhdr
#USE streamio
#USE "subrot.tsr"
#USE "transfor.tsr"
#USE sphdr
#USE userio
#USE ioconv
#USE strings
#USE snglmath
}}}
PROC fft( VAL INT sign, []REAL32 FR, FI, []INT dim, VAL INT i)
  {{{ transformada rapida para linhas
  SEQ
```

```
    {{{
    bitreverso( COLUMN, FR, FI, dim, i)
    cool2( sign, COLUMN, FR, FI, dim, i)
    }}}
  }}}
:
```

****

Programa Principal "sequencial.tsr" -- Programa implementado utilizando somente a linguagem OCCAM.

****

```
{{{ bibliotecas utilizadas
#USE sphdr
#USE streamio
#USE userio
#USE splib
#USE solib
#USE sklib
#USE ioconv
#USE strings
#USE snglmath
{{{ bibliotecas criadas para o processamento
#USE "janela.tsr"
#USE "subrot.tsr"
#USE "met.tsr"
#USE "transfor.tsr"
#USE "auxiliar.tsr"
}}}
}}}
SEQ
  {{{ declaracao das variaveis globais
  {{{ dimensao da matriz imagem
  [2]INT dim:                    -- dimensao da matriz imagem para FFT linha
  [2]INT dim1:                   -- dimensao da matriz imagem para FFT coluna
  INT maxlin, maxcol:              -- dimensao da matriz imagem expandida
  }}}
  {{{ variaveis temporais
  TIMER tempo:
  INT t1, t2, t3, t4:
  REAL32 segundo:
  INT seg:
  }}}
  {{{ indicador da transformada direta ou reversa
  INT sign:
  }}}
  {{{ declaracao da matriz para transformada
  [MAXROW][MAXCOL]REAL32 MR, MI:        -- para transformada das linhas
  [MAXCOL][MAXROW]REAL32 DR, DI:       -- para transformada das colunas
  }}}
  }}}
```

```
SEQ
  {{{ main
  {{{ declaracao das variaveis locais
  {{{ canal para comunicaco com o hospedeiro
  CHAN OF SP fs IS from.isv:          -- canal para o hospedeiro
  CHAN OF SP ts IS to.isv:
  INT32 streamid:                     -- identidficador do arquivo
  {{{ variaveis para habilitacao do arquivo
  INT len, bytes.read:
  INT char:
  BYTE result:
  }}}
  }}}
  {{{ guarda o nome do arquivo que contem a matriz imagem
  VAL max.string IS 50:
  [max.string]BYTE name:
  INT nlin, ncol:
  }}}
  {{{ variaveis auxiliares
  [3]BYTE resposta:
  INT i, j, k, dlin, dcol, lim, teste:
  }}}
  {{{ matrizes auxiliares dos dados para geracao da imagem
  [64][256]INT ponto:
  [64][256]REAL32 fponto:
  }}}
  }}}
  SEQ
    {{{ leitura do arquivo que contem a matriz imagem obtida por RMN
    window(screen)
    ss.goto.xy(screen, 3,3)
    ss.write.text.line(screen, "Digite o nome do arquivo: ")
    ss.goto.xy(screen,29, 3)
    so.read.echo.line(fs, ts, len, name, result)
    VAL muda.nome IS [name FROM 0 FOR len]:
    so.open( fs, ts, muda.nome, spt.text, spm.input, streamid, result)
    }}}
    IF
      result = spr.ok
        SEQ
          {{{ leitura dos dados da matriz imagem
          ledados( screen, keyboard, fs, ts, streamid, nlin, ncol, maxlin, maxcol, MR, MI)
          so.close(fs, ts, streamid, result)
          ss.goto.xy(screen, 5, 21)
          ss.write.string(screen," tecle barra de espaco para prosseguir!")
          ks.read.char(keyboard, char)
          window(screen)
          {{{ retira o plano de base?
          ss.goto.xy(screen, 5, 5)
          ss.write.text.line(screen,"Deve ser extraido o plano de base?")
          ss.goto.xy(screen, 40, 5)
```

```
ks.read.echo.text.line(keyboard, screen, len, resposta, char)
IF
  ((resposta[0] = 's' ) OR (resposta[0] = 'S'))
    SEQ
      filtragem(screen, ncol, nlin, MR, MI)
  TRUE
    SKIP
}}}
{{{  expansao da imagem?
ss.goto.xy(screen, 5, 13)
ss.write.text.line(screen,"Deseja que a imagem seja expandida?")
ss.goto.xy(screen, 42, 13)
ks.read.echo.text.line(keyboard, screen, len, resposta, char)
IF
  ((resposta[0] = 's' ) OR (resposta[0] = 'S'))
    SEQ
      dim[ROW]:= maxlin
      dim[COLUMN]:= maxcol
  ((resposta[0] = 'n') OR (resposta[0] = 'N'))
    SEQ
      dim[ROW]:= nlin
      dim[COLUMN]:= ncol
  TRUE
    SKIP
}}}
}}}
{{{  processando a transformada de fourier rapida
ss.goto.xy(screen, 5, 15)
ss.write.text.line(screen,"Processando a Transformada de Fourier...")
tempo ? t3                -- tempo inicial para o processamento
SEQ i=0 FOR nlin          -- transformada para as linhas
  fft(-1, MR[i], MI[i], dim, i)
{{{  transposicao da matriz imagem
SEQ i=0 FOR nlin
  SEQ j=0 FOR ncol
    SEQ
      DR[j][i]:= MR[i][j]
      DI[j][i]:= MI[i][j]
}}}
dim1[0]:=dim[1]
dim1[1]:=dim[0]
SEQ i=0 FOR ncol          -- transformada para as colunas
  fft(-1, DR[i], DI[i], dim1, i)
{{{  transposicao da matriz imagem
SEQ i=0 FOR nlin
  SEQ j=0 FOR ncol
    SEQ
      MR[i][j]:= DR[j][i]
      MI[i][j]:= DI[j][i]
}}}
MR[0][0]:=MR[0][1]
```

```
MR[0][0]:=MR[0][1]
tempo ? t4                -- tempo final para o processamento
ss.goto.xy(screen, 5, 17)
ss.write.string(screen,"O tempo gasto pela FFT foi de: ")
-- tempo de processamento em segundos
segundo:=(REAL32 ROUND(t4 MINUS t3))*(64.0(REAL32)*POWER(10.0(REAL32),-6.0(REAL32)))
ss.goto.xy(screen, 35, 17)
ss.write.real32(screen, segundo, 5,3)
ss.goto.xy(screen, 5, 21)
ss.write.string(screen," tecle barra de espaco para prosseguir!")
ks.read.char(keyboard, char)
}}}
{{{  gerando imagem
SEQ
  INT pd.command, char:
  SEQ
    pd.command := 0
    WHILE ( pd.command <> 5)
      SEQ
        {{{  menu
        window(screen)
        ss.goto.xy(screen, 5, 3)
        ss.write.text.line(screen,"As opcoes de imagens sao:")
        ss.goto.xy(screen, 5, 5)
        ss.write.text.line(screen,"Desenha pela parte real                - opcao 1")
        ss.goto.xy(screen, 5, 7)
        ss.write.text.line(screen,"Desenha pela parte imaginaria          - opcao 2")
        ss.goto.xy(screen, 5, 9)
        ss.write.text.line(screen,"Desenha pelo Modulo (sqrt a**2 + b**2)  - opcao 3")
        ss.goto.xy(screen, 5, 11)
        ss.write.text.line(screen,"Desenha pela Fase (arctag Imag/Real)  - opcao 4")
        ss.goto.xy(screen, 5, 13)
        ss.write.text.line(screen,"Nenhuma das anteriores                 - opcao 5")
        }}}
        ss.goto.xy(screen, 5, 18)
        ss.write.text.line(screen,"Entre com a opcao desejada: ")
        ss.goto.xy(screen, 35, 18)
        ks.read.echo.int(keyboard, screen, pd.command, char)
        IF
          pd.command = 1
            SEQ
              reestrutura( dim, MR )
              window(screen)
              normaliza ( screen, MR, ponto, dim)
              {{{
              ss.goto.xy(screen, 15, 9)
              ss.write.text.line(screen,"FFT - da parte real")
              }}}
              preparasaida (screen, fs, ts, streamid, ponto, dim)
          pd.command = 2
            SEQ
```

```
            reestrutura( dim, MI )
            window(screen)
            normaliza( screen, MI, ponto, dim )
            {{{
            ss.goto.xy(screen, 15, 9)
            ss.write.text.line(screen,"FFT - da parte imaginaria")
            }}}
            preparasaida (screen, fs, ts, streamid, ponto, dim )
          pd.command = 3
           SEQ
            {{{  tirando a raiz quadrada dos pontos
            SEQ i=0 FOR dim[ROW]
              SEQ j=0 FOR dim[COLUMN]
                fponto[i][j]:= SQRT((MR[i][j] * MR[i][j])+(MI[i][j] * MI[i][j]))
            }}}
            reestrutura( dim, fponto )
            window(screen)
            normaliza( screen, fponto, ponto, dim)
            {{{
            ss.goto.xy(screen, 15, 9)
            ss.write.text.line(screen,"FFT - raiz quadrada")
            }}}
            preparasaida (screen, fs, ts, streamid, ponto, dim)
          pd.command = 4
           SEQ
            {{{  tirando o arco-tangente dos pontos
            SEQ i=0 FOR dim[ROW]
              SEQ j=0 FOR dim[COLUMN]
                fponto[i][j]:= ATAN(MR[i][j] /MI[i][j])
            }}}
            reestrutura( dim, fponto)
            window(screen)
            normaliza(screen, fponto, ponto, dim)
            {{{
            ss.goto.xy(screen, 15, 9)
            ss.write.text.line(screen,"FFT - arco tangente")
            }}}
            preparasaida (screen, fs, ts, streamid, ponto, dim)
          pd.command = 5
           SKIP
      }}}
  TRUE
    {{{  se der erro da mensagem
    SEQ
      ss.write.text.line(screen, "   Nao le arquivo")
      ss.write.int(screen, (INT result), 0)
      ss.write.nl(screen)
    }}}
{{{  comentario de saida
ss.goto.xy(screen, 10, 21)
ss.write.text.line(screen, "Aperte qualquer tecla para voltar ao TDS")
```

```
  INT any:
  keyboard ? any
  ss.write.nl(screen)
  }}}
}}}
```

APÊNDICE B

SERVIÇO DE BIBLIOTECA E INFORMAÇÃO - IFQSC
FÍSICA

Nesse apêndice estão as rotinas que foram programadas em linguagem montadora do transputer.

Essas rotinas foram colocadas todas em bibliotecas auxiliares.

O programa principal é idêntico ao apresentado no Apêndice A, com uma única ressalva, que é a mudança da matriz analisada de dois parâmetros para três. Foi realizada essa mudança para melhor acesso dos dados utilizando a linguagem montadora.

****

Biblioteca fftassem.tsr -- Contém as rotinas para o cálculo das "butterflys", além da rotina que calcula a posição de um dado da matriz na memória.

****

```
SC oper  -- rotina em linguagem montadora par a operacao de "butterfly"
{{{
#USE strmhdr
#USE sphdr
#USE userio
#USE ioconv
#USE strings
#USE snglmath
}}}
PROC oper( INT pont, pont1, fpI, fp1I)
  {{{  oper
  REAL32 tmpr, tmpi:
  SEQ
    GUY
      {{{
      LDL       pont
      FPLDNLSN
      LDLP      tmpr      -- le *pont
      FPSTNLSN
      FPCHKERR

      LDL       pont
      ADC       4         -- pont + 1
      DUP
      STL       fp1I      -- fp1I:= pont + 1
      FPLDNLSN
      LDLP      tmpi
      FPSTNLSN
      FPCHKERR

      LDL       pont1
      ADC       4         -- pont1 + 1
      STL       fpI
```

```
    LDL       pont1
    FPLDNLSN            -- *pont1
    LDLP      tmpr
    FPLDNLSN            -- *tmpr
    FPSUB               -- *pont1 - tmpr
    LDL       pont
    FPSTNLSN
    FPCHKERR

    LDL       fpI
    FPLDNLSN            -- *fpI
    LDLP      tmpi
    FPLDNLSN            -- *tmpi
    FPSUB               -- *fpI - tmpi
    LDL       fp1I
    FPSTNLSN
    FPCHKERR

    LDL       pont1
    FPLDNLSN            -- *pont1
    LDLP      tmpr
    FPLDNLADDSN         -- *tmpr
    LDL       pont1
    FPSTNLSN
    FPCHKERR

    LDL       fpI
    FPLDNLSN            -- *fpI
    LDLP      tmpi
    FPLDNLADDSN         -- *tmpi
    LDL       fpI
    FPSTNLSN
    FPCHKERR
    }}}
  }}}
:


SC oper1  -- rotina em linguagem montadora para calculo da "butterfly"
{{{
#USE strmhdr
#USE sphdr
#USE userio
#USE ioconv
#USE strings
#USE snglmath
}}}
PROC oper1( INT fpR, fp1R, fpI, fp1I, VAL INT sign)
```

```
{{{ oper1
REAL32 t0r, t0i, t1r, t1i, t2r, t2i, t3r, t3i:
SEQ
  GUY
    {{{
    LDL       fpR
    FPLDNLSN
    LDL       fp1R
    FPLDNLADDSN
    LDLP      t0r
    FPSTNLSN
    FPCHKERR

    LDL       fpR
    ADC       4          -- fpR + 1
    FPLDNLSN
    LDL       fp1R
    ADC       4          -- fp1R + 1
    FPLDNLADDSN
    LDLP      t0i
    FPSTNLSN
    FPCHKERR

    LDL       fpR
    FPLDNLSN
    LDL       fp1R
    FPLDNLSN
    FPSUB
    LDLP      t1r
    FPSTNLSN
    FPCHKERR

    LDL       fpR
    ADC       4          -- fpR + 1
    FPLDNLSN
    LDL       fp1R
    ADC       4          -- fp1R + 1
    FPLDNLSN
    FPSUB
    LDLP      t1i
    FPSTNLSN
    FPCHKERR

    LDL       fpI
    FPLDNLSN
    LDL       fp1I
    FPLDNLADDSN
    LDLP      t2r
    FPSTNLSN
    FPCHKERR
```

```
LDL       fpI
ADC       4          -- fpR + 1
FPLDNLSN
LDL       fp1I
ADC       4          -- fp1R + 1
FPLDNLADDSN
LDLP      t2i
FPSTNLSN
FPCHKERR

LDL       fpI
FPLDNLSN
LDL       fp1I
FPLDNLSN
FPSUB
LDLP      t3r
FPSTNLSN
FPCHKERR

LDL       fpI
ADC       4          -- fpR + 1
FPLDNLSN
LDL       fp1I
ADC       4          -- fp1R + 1
FPLDNLSN
FPSUB
LDLP      t3i
FPSTNLSN
FPCHKERR

LDLP      t0r
FPLDNLSN
LDLP      t2r
FPLDNLADDSN
LDL       fpR
FPSTNLSN
FPCHKERR

LDLP      t0i
FPLDNLSN
LDLP      t2i
FPLDNLADDSN
LDL       fpR
ADC       4
FPSTNLSN
FPCHKERR

LDLP      t0r
FPLDNLSN
LDLP      t2r
FPLDNLSN
```

```
FPSUB
LDL     fpI
FPSTNLSN
FPCHKERR

LDLP    t0i
FPLDNLSN
LDLP    t2i
FPLDNLSN
FPSUB
LDL     fpI
ADC     4
FPSTNLSN
FPCHKERR

LDC     0
LDL     sign
GT
CJ      .L5
LDLP    t3r
FPLDNLSN
FPLDZEROSN
FPREV
FPSUB
LDLP    t3r
FPSTNLSN
FPCHKERR

LDLP    t3i
FPLDNLSN
FPLDZEROSN
FPREV
FPSUB
LDLP    t3i
FPSTNLSN
FPCHKERR

:L5

LDLP    t1r
FPLDNLSN
LDLP    t3i
FPLDNLSN
FPSUB
LDL     fp1R
FPSTNLSN
FPCHKERR

LDLP    t1i
FPLDNLSN
LDLP    t3r
```

```
    FPLDNLADDSN
    LDL      fp1R
    ADC      4
    FPSTNLSN
    FPCHKERR

    LDLP     t1r
    FPLDNLSN
    LDLP     t3i
    FPLDNLADDSN
    LDL      fp1I
    FPSTNLSN
    FPCHKERR

    LDLP     t1i
    FPLDNLSN
    LDLP     t3r
    FPLDNLSN
    FPSUB
    LDL      fp1I
    ADC      4
    FPSTNLSN
    FPCHKERR
    }}}
  }}}
:
```

SC pos -- rotina que calcula a posição do dado na memória.

```
PROC pos( INT next, j, R)
  {{{ parte em assembly
  GUY
    LDL      next
    LDL      MAXSIZE
    PROD
    LDL      j
    ADD
    LDC      2
    PROD
    LDC      R
    ADD
    LDLP     F
    WSUB
    STL      fpR
    FPCHKERR
  }}}
:
```

APÊNDICE C

Nesse apêndice estão as rotinas Controlador e FAZFFT utilizadas para a implementação paralela do algorítmo da FFT-2D.

```
{{{ bibliotecas utilizadas
#USE strmhdr
#USE streamio
#USE sphdr
#USE userio
#USE splib
#USE solib
#USE sklib
#USE ioconv
#USE spinterf
#USE strings
#USE snglmath
}}}
#USE "subrot.tsr"

SC controle
#USE strmhdr
#USE streamio
#USE sphdr
#USE userio
#USE splib
#USE solib
#USE sklib
#USE ioconv
#USE spinterf
#USE strings
#USE snglmath

{{{
#USE "subrot.tsr"
#USE "janela.tsr"
#USE "tipoleit.tsr"
#USE "ctsubrot.tsr"
#USE "transfor.tsr"
#USE "met.tsr"
#USE "auxiliar.tsr"
#USE linkaddr
#USE t4board
}}}
PROC controlador(CHAN OF SS screen, CHAN OF KS keyboard, CHAN OF SP from.isv,
            to.isv, CHAN OF cteste entrada0, entrada1, entrada2, entrada3,
            saida0, saida1, saida2, saida3)
  {{{ declaracao de variaveis
  VAL MAXROW IS 8:
  {{{ dimensao da matriz
  [2]INT dim:
  }}}
```

```
{{{ variaveis de temporizacao
INT maxlin, maxcol:
REAL32 te:
TIMER tempo:
INT t3, t4:
}}}
{{{ declaracao das variaveis e canais
{{{ variaveis de i/o
INT32 streamid:
VAL max.string IS 50:
[max.string]BYTE name:
}}}
BYTE result:
[10]BYTE resposta:
INT len, bytes.read, nlin, ncol:
INT p, nl, nc, np:
[MAXROW][MAXCOL]INT ponto:
[MAXROW][MAXCOL]REAL32 fponto:
INT pd.command, char:
[MAXROW][MAXCOL]REAL32 MR, MI:
[MAXCOL][MAXROW]REAL32 DR, DI:
VAL INT np IS 4:
}}}
INT particao:
}}}
SEQ
  {{{ controlador
  {{{ entrada
  window(screen)
  {{{ comentario de entrada
  ss.goto.xy(screen, 3,3)
  ss.write.text.line(screen, "Digite o nome do arquivo: ")
  ss.goto.xy(screen,29, 3)
  so.read.echo.line(from.isv, to.isv, len, name, result)
  }}}
  VAL muda.nome IS [name FROM 0 FOR len]:
  so.open( from.isv, to.isv, muda.nome, spt.text, spm.input, streamid, result)
  }}}
  IF
    result = spr.ok
      SEQ
        {{{ coleta de dados
        ledados( screen, keyboard, from.isv, to.isv, streamid, nlin, ncol, maxlin, maxcol, MR, MI)
        so.close(from.isv, to.isv, streamid, result)
        ss.goto.xy(screen, 5, 21)
        ss.write.string(screen," tecle barra de espaco para prosseguir!")
        ks.read.char(keyboard, char)
        window(screen)
        {{{ deve ser retirado o plano de base?
        ss.goto.xy(screen, 5, 5)
        ss.write.text.line(screen,"Deve ser extraido o plano de base?")
```

```
ss.goto.xy(screen, 40, 5)
ks.read.echo.text.line(keyboard, screen, len, resposta, char)
IF
  ((resposta[0] = 's' ) OR (resposta[0] = 'S'))
    SEQ
      filtragem(screen, ncol, nlin, MR, MI)
  TRUE
    SKIP
}}}
{{{ deve ser expandida a imagem?
ss.goto.xy(screen, 5, 13)
ss.write.text.line(screen,"Deseja que a imagem seja expandida?")
ss.goto.xy(screen, 42, 13)
ks.read.echo.text.line(keyboard, screen, len, resposta, char)
IF
  ((resposta[0] = 's' ) OR (resposta[0] = 'S'))
    SEQ
      dim[ROW]:= maxlin
      dim[COLUMN]:= maxcol
  ((resposta[0] = 'n') OR (resposta[0] = 'N'))
    SEQ
      dim[ROW]:= nlin
      dim[COLUMN]:= ncol
  TRUE
    SKIP
}}}
}}}
ss.goto.xy(screen, 5, 15)
ss.write.text.line(screen,"Processando a Transformada de Fourier...")
{{{ enviando e recebendo pelos canais 1 e 2
tempo ? t3
particao:= (nlin*ncol)/np
nl:= nlin/np
{{{ retype para os dados passados pelo canal
[2048]REAL32 vet.temp.real RETYPES MR:
[2048]REAL32 vet.temp.imag RETYPES MI:
}}}
PAR
  SEQ
    {{{ para transputer1
    in IS entrada0:
    out IS saida0:
    INT a, a1, a2:
    {{{ enviando e recebendo dados da matriz para o canal 1
    SEQ
      in ! nl;512::[vet.temp.real FROM 0 FOR particao];512::[vet.temp.imag FROM 0
FOR particao]; ncol
      out ? a1; a::[vet.temp.real FROM 0 FOR particao];a::[vet.temp.imag FROM 0 FOR
particao];a2
    }}}
    }}}
```

```
SEQ
  {{{ para transputer2
  in IS entrada1:
  out IS saida1:
  INT a, a1, a2:
  {{{ enviando e recebendo dados da matriz para o canal 1
  SEQ
    in ! nl;512::[vet.temp.real FROM particao FOR particao];512::[vet.temp.imag FROM
particao FOR particao]; ncol
    out ? a1; a::[vet.temp.real FROM particao FOR particao];a::[vet.temp.imag FROM
particao FOR particao];a2
  }}}
  }}}
SEQ
  {{{ para transputer3
  in IS entrada2:
  out IS saida2:
  INT a, a1, a2:
  {{{ enviando e recebendo dados da matriz para o canal 1
  SEQ
    in ! nl;512::[vet.temp.real FROM (2*particao) FOR particao];
          512::[vet.temp.imag FROM (2*particao) FOR particao]; ncol
    out ? a1; a::[vet.temp.real FROM (2*particao) FOR particao];
            a::[vet.temp.imag FROM (2*particao) FOR particao];a2
  }}}
  }}}
SEQ
  {{{ para transputer0
  in IS entrada3:
  out IS saida3:
  INT a, a1, a2:
  {{{ enviando e recebendo dados da matriz para o canal 1
  SEQ
    in ! nl;512::[vet.temp.real FROM (3*particao) FOR particao];
          512::[vet.temp.imag FROM (3*particao) FOR particao]; ncol
    out ? a1; a::[vet.temp.real FROM (3*particao) FOR particao];
            a::[vet.temp.imag FROM (3*particao) FOR particao];a2
  }}}
  }}}
{{{ fazendo transposicao
SEQ i=0 FOR dim[ROW]
  SEQ
    SEQ j=0 FOR dim[COLUMN]
    {{{
      SEQ
        DR[j][i]:= MR[i][j]
        DI[j][i]:= MI[i][j]
    }}}
}}}
nc:= ncol/np
{{{ retype para os dados passados pelo canal
```

```
[2048]REAL32 vet.temp.real RETYPES DR:
[2048]REAL32 vet.temp.imag RETYPES DI:
}}}
PAR
  SEQ
    {{{ para transputer1
    in IS entrada0:
    out IS saida0:
    INT b, b1, b2:
    {{{ enviando e recebendo dados da matriz para o canal 1
    SEQ
      in ! nc;512::[vet.temp.real FROM 0 FOR particao];512::[vet.temp.imag FROM 0
FOR particao]; nlin
      out ? b1; b::[vet.temp.real FROM 0 FOR particao];b::[vet.temp.imag FROM 0 FOR
 particao];b2
    }}}
    }}}
  SEQ
    {{{ para transputer2
    in IS entrada1:
    out IS saida1:
    INT b, b1, b2:
    {{{ enviando e recebendo dados da matriz para o canal 1
    SEQ
      in ! nc;512::[vet.temp.real FROM particao FOR particao];512::[vet.temp.imag FROM
  particao FOR particao]; nlin
      out ? b1; b::[vet.temp.real FROM particao FOR particao];b::[vet.temp.imag FROM
 particao FOR particao];b2
    }}}
    }}}
  SEQ
    {{{ para transputer3
    in IS entrada2:
    out IS saida2:
    INT b, b1, b2:
    {{{ enviando e recebendo dados da matriz para o canal 1
    SEQ
      in ! nc;512::[vet.temp.real FROM (2*particao) FOR particao];
            512::[vet.temp.imag FROM (2*particao) FOR particao]; nlin
      out ? b1; b::[vet.temp.real FROM (2*particao) FOR particao];
              b::[vet.temp.imag FROM (2*particao) FOR particao];b2
    }}}
    }}}
  SEQ
    {{{ para root
    in IS entrada3:
    out IS saida3:
    INT b, b1, b2:
    {{{ enviando e recebendo dados da matriz para o canal 1
    SEQ
      in ! nc;512::[vet.temp.real FROM (3*particao) FOR particao];
```

SERVIÇO DE BIBLIOTECA E INFORMAÇÃO - IFQSC
FÍSICA

```
            512::[vet.temp.imag FROM (3*particao) FOR particao]; nlin
        out ? b1; b::[vet.temp.real FROM (3*particao) FOR particao];
              b::[vet.temp.imag FROM (3*particao) FOR particao];b2
      }}}
      }}}
    {{{ voltando a matriz original
    SEQ i=0 FOR dim[ROW]
     SEQ
      SEQ j=0 FOR dim[COLUMN]
      {{{
       SEQ
        MR[i][j]:= DR[j][i]
        MI[i][j]:= DI[j][i]
      }}}
    }}}
    {{{ reordenando o topo da matriz r e i
    MR[0][0]:=MR[0][1]
    MI[0][0]:=MI[0][1]
    }}}
    tempo ? t4
    te:= (REAL32 ROUND (t 4MINUSt3)) * (64.0(REAL32) * POWER(10.0(REAL32),
-6.0(REAL32)))
    {{{ tempo de processamento da fft
    ss.goto.xy(screen, 5, 17)
    ss.write.string(screen,"O tempo gasto pela FFT foi de: ")
    ss.goto.xy(screen, 35, 17)
    ss.write.real32(screen, te, 5,3)
    }}}
    }}}
    {{{ gerando imagem
    ss.goto.xy(screen, 5, 21)
    ss.write.string(screen," tecle barra de espaco para prosseguir!")
    ks.read.char(keyboard, char)
    SEQ
     {{{ menu
     window(screen)
     ss.goto.xy(screen, 5, 3)
     ss.write.text.line(screen,"As opcoes de imagens sao:")
     ss.goto.xy(screen, 5, 5)
     ss.write.text.line(screen,"Desenha pelo Modulo (sqrt a**2 + b**2)  - opcao 1")
     ss.goto.xy(screen, 5, 13)
     ss.write.text.line(screen,"Nenhuma das anteriores                - opcao 5")
     }}}
     {{{
     ss.goto.xy(screen, 5, 18)
     ss.write.text.line(screen,"Entre com a opcao desejada: ")
     ss.goto.xy(screen, 35, 18)
     ks.read.echo.int(keyboard, screen, pd.command, char)
     }}}
     WHILE ( pd.command <> 2)
      SEQ
```

```
        {{{ geraimagem
        {{{ gerando imagem
        SEQ
          {{{
          CASE pd.command
            1
              SEQ

            2
              SKIP
          }}}
        }}}
        }}}
        {{{
        {{{ menu
        window(screen)
        ss.goto.xy(screen, 5, 3)
        ss.write.text.line(screen,"As opcoes de imagens sao:")
        ss.goto.xy(screen, 5, 5)
        ss.write.text.line(screen,"Desenha pelo Modulo (sqrt a**2 + b**2)  - opcao 3")
        ss.goto.xy(screen, 5, 11)
        ss.write.text.line(screen,"Desenha pela Fase (arctag Imag/Real)  - opcao 4")
        ss.goto.xy(screen, 5, 13)
        ss.write.text.line(screen,"Nenhuma das anteriores                - opcao 5")
        }}}
        {{{
        ss.goto.xy(screen, 5, 18)
        ss.write.text.line(screen,"Entre com a opcao desejada: ")
        ss.goto.xy(screen, 35, 18)
        ks.read.echo.int(keyboard, screen, pd.command, char)
        }}}
        }}}
      }}}
    {{{
    TRUE
      {{{ se der erro da mensagem
      SEQ
        ss.write.text.line(screen, "  Nao le arquivo")
        ss.write.int(screen, (INT result), 0)
        ss.write.nl(screen)
      }}}
    }}}
  {{{ comentario de saida
  ss.goto.xy(screen, 10, 21)
  ss.write.text.line(screen, "digite qualquer tecla para retornar so TDS")
  INT any:
  keyboard ? any
  ss.write.nl(screen)
  }}}
  }}}
:
```

```
SC fazfft
#USE strmhdr
#USE userio
#USE ioconv
#USE streamio
{{{
#USE "subrot.tsr"
#USE "janela.tsr"
#USE "tipoleit.tsr"
#USE "ctsubrot.tsr"
#USE "transfor.tsr"
#USE "met.tsr"
#USE "auxiliar.tsr"
}}}
}}}
PROC fazfft(VAL INT i, CHAN OF cteste ent, sai)
  SEQ
    {{{  transformada para linhas
    {{{  declaracao da matriz
    [2]INT d1:
    INT nlin, ncol, aux:
    [512]REAL32 vetor.temp.real, vetor.temp.imag:
    }}}
    SEQ
      {{{  recebendo a matriz real e imaginaria do canal 1
      ent ? nlin; aux::vetor.temp.real; aux::vetor.temp.imag;ncol
      d1[COLUMN]:= ncol
      d1[ROW]:= nlin
      }}}
      {{{
      [2][256]REAL32 BR RETYPES vetor.temp.real:
      [2][256]REAL32 BI RETYPES vetor.temp.imag:
      }}}
      {{{  processando a transformada de fourier rapida
      SEQ k=0 FOR nlin
        fftbilin(-1, BR[k], BI[k], d1, k)
      }}}
      {{{  enviando dados para o canal 1
      sai ! nlin; aux::vetor.temp.real; aux::vetor.temp.imag;ncol
      }}}
    }}}
    {{{  transformada para colunas
    {{{  declaracao da matriz
    [2]INT dim:
    INT nlin, ncol, aux:
    [512]REAL32 vetor.temp.real, vetor.temp.imag:
    }}}
    SEQ
      {{{  recebendo a matriz real e imaginaria do canal 1
      ent ? ncol; aux::vetor.temp.real; aux::vetor.temp.imag;nlin
      dim[COLUMN]:= nlin
```

```
    dim[ROW]:= ncol
    }}}
    {{{
    [64][8]REAL32 CR RETYPES vetor.temp.real:
    [64][8]REAL32 CI RETYPES vetor.temp.imag:
    }}}
    {{{  processando a transformada de fourier rapida
    {{{  fazendo transformada
    SEQ j=0 FOR ncol
      fftbilin(-1,CR[j], CI[j], dim, j)
    }}}
    }}}
    {{{  enviando dados para o canal 1
    sai ! ncol; aux::vetor.temp.real; aux::vetor.temp.imag;nlin
    }}}
  }}}
:

PROC main(CHAN OF SP from.host, to.host, CHAN OF cteste entrada0,
        entrada1, entrada2, saida0, saida1, saida2)
 {{{  main
 #USE t4board
 CHAN OF SS screen:
 CHAN OF KS keyboard:
 [3]CHAN OF SP f.isv, t.isv:
 CHAN OF BOOL stopper, mstopper:
 CHAN OF cteste entrada3, saida3:
 PAR
  -- ==========================================
  SEQ
   --==========================================
   PAR
    {{{  corpo principal
    controlador(screen, keyboard, f.isv[2], t.isv[2], entrada0,
              entrada1, entrada2, entrada3, saida0, saida1, saida2, saida3)
    fazfft(0, entrada3, saida3)
    }}}

:
```