ANÁLISE E IMPLEMENTAÇÁO DA FFT EM ARQUITETURAS HIPERCÚBICAS

AUTORES:

Celso Starec
Gabriel P. Silva
Ricardo G. Donadio

CURSO:

PROGRAMAÇÁO PARALELA

PROFs:

VALMIR C. BARBOSA E CLÁUDIO L. AMORIM

COPPE SISTEMAS - UFRJ
Julho de 1989

RESUMO

Este relatório apresenta um estudo sobre uma implementação paralela da transformada rápida de Fourier (FFT). Trabalhos anteriores [Cve87b] trataram da análise da FFT em arquiteturas paralelas com memória compartilhada. Este estudo se aterá ao comportamento da FFT em arquiteturas hipercúbicas.

Primeiro apresenta-se uma pequena revisão da transformada de Fourier original, contínua e discreta, e então, a sua implementação computacional mais conhecida: a FFT. A seguir é mostrado um mapeamento que resulta em um menor custo de comunicação e, conseqüentemente, em um melhor desempenho do algoritmo. A partir daí desenvolve-se um modelo simples de predição de desempenho. Finalmente, é apresentado um programa paralelo, escrito em "Parallel C", seguido da análise de resultados. Este programa é executado em um sistema de desenvolvimento para PC, baseado em Transputers T800 da Inmos.

ABSTRACT

This technical report presents a study about parallel implementation of the Fast Fourier Transform (FFT). Previous works [Cve87b] treated this problem on a shared memory parallel architecture. This study is oriented towards the implementation of the FFT on an hypercube architecture.

First, a brief review of the original Fourier transform, continuous and discrete, is presented, and then its most used computational implementation: the FFT. A possible mapping which results in a low communication overhead for the FFT algorithm and a simple performance model are presented. At last, a parallel program written in "Parallel C" is presented, followed by the analysis of the results. The program runs on a PC "Transputer Development System" board, a Transputer T800 based system of Inmos Ltd.

1. INTRODUCAO

A primeira tentativa de apresentação da transformada de Fourier foi feita em 1807, mas o trabalho na época foi recusado por Lagrange. Mais tarde, em 1822, Fourier conseguiu apresentá-la em seu livro "Théorie analytique de la chaleur". No entanto, aquela versão original era de difícil implementação computacional até bem pouco tempo atrás, devido a complexidade do algoritmo ser $O(n^2)$. Somente em 1965, Cooley e Tukey vieram a propor o algoritmo que hoje é conhecido como transformada rápida de Fourier (Fast Fourier Transform - FFT) e possui complexidade O(nlogn). Na realidade, FFT é a denominação dada a um conjunto de algoritmos cuja complexidade é O(nlogn) e são baseados no algoritmo de Cooley e Tukey. A FFT passou então a ter ampla utilização nas mais diversas áreas, entre as quais podemos destacar :

- . processamento de sinais em geral
- . processamento de voz / imagem
- . processamento de sinais geofísicos
- . equações diferenciais
- . convoluções
- . análise espectral
- . reconhecimento de padrões

Apresenta-se a seguir a transformada contínua de Fourier de um sinal x(t), definida por :

$$X(w) = \int_{-\infty}^{+\infty} x(t)\, e^{-jwt}\, dt$$

A transformada discreta de Fourier (DFT) é obtida através de uma operação equivalente aplicada a uma seqüência de N amostras do sinal x(t) :

$$X(k) = \sum_{n=0}^{N-1} x(m) \; W_N^{kn} \quad ; \; k = 0, \ldots, N-1$$

onde $W_N = e^{-j2\pi/N}$

A seguir mostra-se como obter um algoritmo que implementa o cálculo da FFT a partir da transformada discreta de Fourier. Dados N pontos amostrados do sinal de entrada, onde N é uma potência de dois, dividiremos o somatório em duas parcelas, uma com as seqüências pares e outra com as ímpares :

$$X(k) = \sum_{r=0}^{(N/2)-1} x(2r) \; W_N^{2rk} + \sum_{r=0}^{(N/2)-1} x(2r+1) \; W_N^{(2r+1)k}$$

desenvolvendo :

$$X(k) = \sum_{r=0}^{(N/2)-1} x(2r) \; (W_N^2)^{rk} + W_N^k \sum_{r=0}^{(N/2)-1} x(2r+1) \; (W_N^2)^{rk}$$

Observe que $W_N^2 = e^{-j2\pi/(N/2)} = W_{N/2}$

Então o somatório pode ser reescrito como :

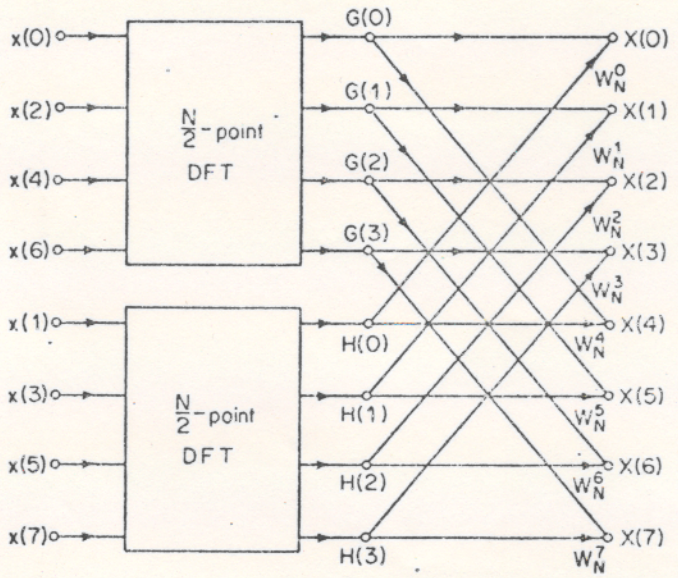
$$X(k) = \sum_{r=0}^{(N/2)-1} x(2r) \; W_{N/2}^{rk} + W_N^k \sum_{r=0}^{(N/2)-1} x(2r+1) \; W_{N/2}^{rk}$$

Ou seja :

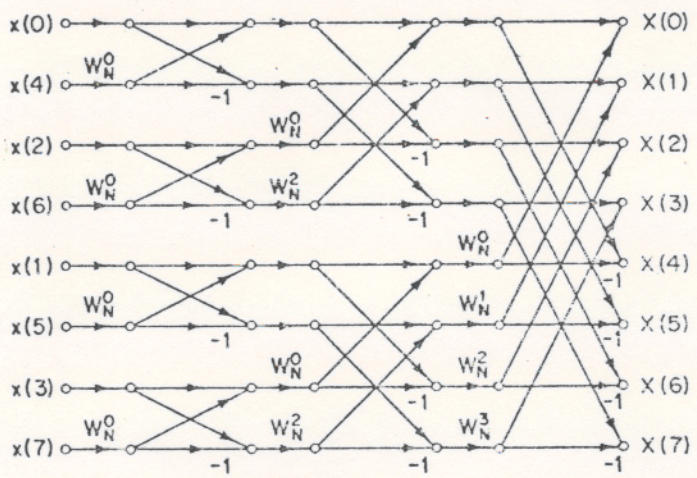
$$X(k) = G(k) + W_N^k \; H(k)$$

Onde G(k) e H(k) são duas novas transformadas discretas aplicadas sobre N/2 pontos. AO método de se dividir os somatórios em outros dois sucessivamente dá-se o nome de radix-2.

Como pode-se ver, a transformada discreta de Fourier assume uma forma recursiva, onde os resultados de um passo intermediário são utilizados no passo seguinte, como no no grafo a seguir :



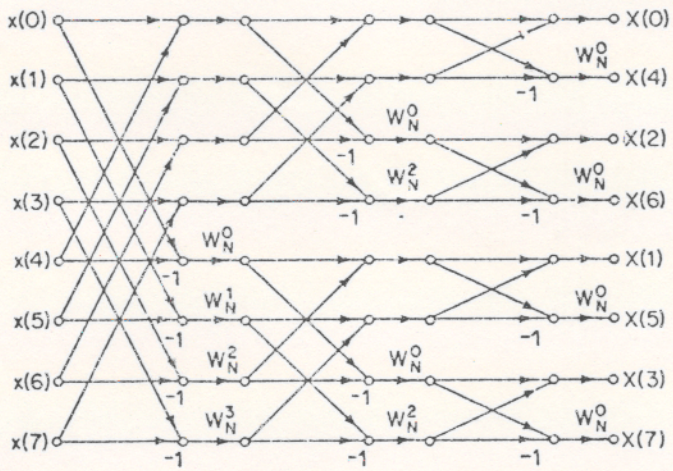
A esta decomposição dá-se o nome de decimação no tempo. Se aplicarmos esta operação recursivamente obteremos o seguinte grafo final :



Como pode-se verificar os dados no final do algoritmo estão arrumados na ordem binária reversa ("bit reversal") em relação a entrada :

X(000) = X(000)		X(100) = X(001)	
X(001) = X(100)		X(101) = X(101)	
X(010) = X(010)		X(110) = X(011)	
X(011) = X(110)		X(111) = X(111)	

Um processo análogo usado para se avaliar a FFT de uma seqüência de pontos é chamado decimação na freqüência. O grafo obtido é o seguinte :



Observe que o grafo é justamente o inverso do anterior.

Resta considerar a transformada inversa, que nos restitui a seqüência de entrada x(n) :

$$x(n) = 1/N \sum_{k=0}^{N-1} X(k) \; W_N^{-kn} \quad ; \; n = 0, \ldots, N-1$$

Observe que a implementação da inversa pode seguir o mesmo algoritmo descrito pelo grafo anterior. Basta usar os complexos conjugados de W_N como novos fatores multiplicativos e normalizar a seqüência resultante do final do algoritmo. Uma técnica de implementação, mais prática, consiste em utilizar a FFT direta, porém, substituíndo-se cada elemento da seqüência de entrada por seu conjugado normalizado.

2. O MODELO

O grafo computacional da FFT pode ser dividido em passos bem definidos. Para um problema com seqüência de entrada de N pontos, onde N é uma potência de dois, têm-se $\log_2 N$ passos. Os N valores complexos calculados num determinado passo dependem apenas dos N resultados do passo anterior. Em cada passo são executadas operações do mesmo tipo chamadas "butterflies". Cada "butterfly" consiste em uma multiplicação e duas somas complexas realizadas entre dois pontos distantes 2^{i-1} entre si, onde i é o indexador do passo.

Seja $Nproc=2^d$ o número de processadores, onde d é a dimensão do hipercubo. Distribuindo-se os pontos de entrada igualmente entre os processadores, têm-se N/Nproc pontos em cada processador. Considerando-se a FFT por decimação no tempo, pode-se adotar uma estratégia para resolver o problema em duas etapas.

A primeira etapa seria calcular uma FFT de N/Nproc pontos seqüencialmente em cada processador, o que é possível dada a natureza recursiva do algoritmo. Esta tarefa levaria $\log_2(N/Nproc)$ passos. A segunda etapa envolve os passos que exigem comunicação entre os nós, necessários para o cálculo da FFT completa, e seria feita em $\log_2 N - \log_2(N/Nproc)$ passos, ou seja, d passos.

Para que o custo decorrente da comunicação seja mínimo, nos passos em que houver comunicação, a troca de mensagens deverá se efetuar somente entre processadores que sejam vizinhos no cubo. Assim sendo, não haverá necessidade de roteamento.

Verifica-se que no hipercubo cada processador está ligado a d vizinhos cuja identificação binária difere de apenas um bit, ou seja : 1, 2, 4, ... ,2^d. Pode-se observar que existe um isomorfismo entre o grafo computacional da FFT e o padrão de interconexão do hipercubo. Partindo de uma distribuição dos pontos pelos processadores adequada, pode-se mapear cada "butterfly" de um passo externo a um único canal de comunicação de maneira que esta se dê sempre entre vizinhos.

Existem mapeamentos dos dados nos processadores para os quais esta propriedade não se verifica, requerendo as "butterflies" mais de um canal. A determinação e quantificação de quais configurações resultam em um melhor mapeamento, considerando-se um critério mais abrangente, não é objetivo deste trabalho e poderá ser realizada em um estudo posterior.

Apresenta-se agora um pequeno modelo de desempenho para esta configuração.

Sejam :

N - # de pontos amostra

M - # de passos do algoritmo = $\log_2 N$

d - dimensão do cubo

Nptno - # de pontos por nó = N/Nproc

t_{pi} - tempo de execução por passo interno por ponto

t_{pe} - tempo de execução por passo externo por ponto

T_t - tempo total do algoritmo

Então :

$$T_t = Nptno\ [(M-d)t_{pi} + dt_{pe}]$$

$$\Longrightarrow T_t(M,d) = 2^{(M-d)}\ [(M-d)t_{pi} + dt_{pe}] \quad ; d \leq M \qquad (1)$$

Aumentando-se a dimensão do cubo de 1 unidade, dobra-se a capacidade de processamento da máquina, e provavelmente acelera-se a execução do algoritmo. Substituindo d por d+1 na expressão (1), vem :

$$T_t'(M,d+1) = 2^{(M-(d+1))}\ [(M-(d+1))t_{pi} + (d+1)t_{pe}] \qquad (2)$$

$T_t' < T_t$, desenvolvendo têm-se :

$$t_{pi} / t_{pe} > (1-d)/(1+M-d) \qquad (3)$$

Sabendo-se que tanto t_{pi} quanto t_{pe} são valores positivos não nulos, a relação acima se verifica a partir de d=1. Isto significa que o tempo total de execução diminui monotonicamente com o aumento da dimensão do cubo, a partir de dois de processadores.

Dependendo da relação entre t_{pi} e t_{pe}, pode-se ter um aumento do tempo total de execução quando se executa em dois processadores ao invés de apenas um.

Um valor notável para a relação t_{pe} / t_{pi} ocorre quando o tempo de execução não se altera quando se passa de um para dois processadores, ou seja, $T_t(M,0) = T_t(M,1)$. Desenvolvendo-se esta expressão obtém-se :

$$t_{pe} / t_{pi} = M + 1 \qquad (4)$$

Outro valor notável desta relação ocorre quando o tempo de execução é o mesmo tanto para o número mínimo quanto máximo de processadores, ou seja, $T_t(M,0) = T_t(M,M)$. Desenvolvendo-se esta expressão obtem-se :

$$t_{pe} / t_{pi} = 2^M \qquad (5)$$

Baseado no modelo, em que por hipótese os canais têm a mesma velocidade, apresenta-se gráficos para diferentes valores de t_{pe} / t_{pi} num problema em que M=12. A saber :

i) Gráfico das curvas do tempo total estimado para a execução em função da dimensão do cubo (Fig. 3.1);

ii) Gráfico das curvas de aceleração incremental (marginal), a que se obteria aumentando em 1 a dimensão do cubo, ou seja dobrando o número de processadores, em função da dimensão do cubo (Fig. 3.2);

iii) Gráfico das curvas de eficiência (speedup X número de processadores) em função da dimensão do cubo (Fig. 3.3).

Os valores de t_{pe} / t_{pi} usados nos gráficos são :

i) $t_{pe} / t_{pi} = 54 / 17$, valores extraídos das medições realizadas na implementação da FFT paralela por decimação na freqüência sem cálculo dos coeficientes W_N ;

ii) t_{pe} / t_{pi} de acordo com a equação (4), e $t_{pi} = 17$;

iii) t_{pe} / t_{pi} de acordo com a equação (5), e $t_{pi} = 17$.

3. IMPLEMENTACAO

O programa paralelo desenvolvido é organizado em duas etapas. A primeira calcula por decimação na freqüência a FFT de uma seqüência inicial de pontos. A seguir, retorna-se a esta seqüência pela aplicação da transformada inversa. A transformada inversa é realizada pela aplicação da FFT por decimação no tempo a uma seqüência cujos pontos são os valores complexos conjugados e normalizados dos pontos gerados na etapa anterior.

Esta abordagem elimina a possível necessidade de comunicação extra para reorganizar dados causada pelo ordenamento "bit reversal", cuja necessidade depende da aplicação a que a(s) transformada(s) se destina(m). Uma solução proposta em [Fox88] realizaria esta comunicação sobre um anel lógico no hipercubo, não apresentando interesse maior para este trabalho.

A implementação da FFT paralela é feita através de rotinas idênticas rodando em cada transputer. Cada rotina é composta de uma FFT sequêncial de Nptno pontos, tal como sugerida em [Opp75], mais uma parte restante que envolve a comunicação.

A cada passo externo do algoritmo corresponde um "link" de comunicação. Existe um mapeamento do grafo computacional que faz com que num determinado passo cada processador execute sempre a mesma parte da "butterfly" e é possível decidir qual das partes em função da identificação do processador (Pid). A identificação de cada transputer é recebida no início das "tasks" através do comando "BIND" no arquivo de configuração.

Optou-se por calcular os coeficientes W_N ("twiddle factors") a cada passo ao invés de tê-los pré-calculados numa tabela, como usualmente é feito em processamento de sinais. Pode-se também medir os tempos equivalentes à implementação utilizando tabelas, simplesmente não executando os trechos relativos aos cálculos destes coeficientes.

Esta opção envolve um compromisso entre tempo de execução e memória para armazenamento dos coeficientes. Em hipercubos de grande dimensão pode não ser eficiente armazenar a tabela em memória pois o número de coeficientes por nó seria N/2, enquanto os pontos da seqüência N/Nproc.

O cálculo dos coeficientes é feito, na terminologia de processamento de sinais, recursivamente. Na prática, isto significa que ao invés de se calcular uma série trigonométrica para cada coeficiente calcula-se desta maneira apenas o coeficiente inicial (W^{ini}) e um fator (W^{dr}). Cada novo coeficiente é gerado pela multiplicação complexa do coeficiente atual por W^{dr}.

Um problema inerente ao desenvolvimento do algoritmo paralelo é a determinação do coeficiente correto a ser usado em por cada processador para as "butterflies" que executará, nos passos do algoritmo que exigem comunicação externa.

A cada processador está associada uma seqüência de coeficientes. Uma vez que W^{dr} é função apenas do passo do algoritmo, e facilmente determinável, o problema se resume na determinação de W^{ini} para cada processador.

A cada passo externo a cardinalidade da seqüência de todos os coeficientes neste passo do algoritmo (LE1) é uma potência de dois do número de pontos por processador (Nptno).

Os indices dr e ini podem ser calculados da seguinte maneira :

```
dr  = (N/2) / LE1;
ini = [Pid  mod  (LE1/Nptno)] * Nptno * dr
```

Uma listagem das rotinas da FFT paralela por decimação no tempo e por decimação na freqüência pode ser encontrada em anexo, incluindo também a apresentação dos comandos de configuração específicos do sistema de desenvolvimento utilizado.

4. RESULTADOS

Apresentam-se aqui os resultados obtidos na execução das rotinas que realizam a FFT paralela por decimação na freqüência e no tempo. As medidas foram realizadas com resolução de um microsegundo, através de uma "thread" de alta prioridade, que faz a leitura do "clock" de mesma prioridade do T800. Nos tempos medidos não foram computados o "overhead" devido à carga inicial dos dados e do código.

Os gráficos 4.1 e 4.2 apresentam o "Overhead" da FFT paralela por decimação na freqüência e no tempo respectivamente, em função da tamanho do problema (M). Em cada gráfico são encontradas quatro curvas. Duas se referem a um cubo com dimensão um (2-x) e as outras duas a um cubo com dimensão dois (4-x). Cada par, por sua vez, representa a computação com (X-b) e sem (X-a) o cálculo recursivo dos coeficientes.

Os gráficos 4.3 e 4.4 apresentam o "Speedup" da FFT paralela por decimação na freqüência e no tempo respectivamete, em função do tamanho do problema. As quatro curvas levantadas foram submetidas às mesmas condições anteriores.

$$\text{Speedup} = T_t(M,0)/T_t(M,d)$$

$$\text{Overhead} = [(Nproc * T_t(M,d)) - T_t(M,0)]/T_t(M,0)$$

para d=1 e 2

Observa-se que o "Overhead" cai assintoticamente com M, ou seja quanto maior a granularidade do problema menor o "Overhead" associado, como era de se esperar.

5. CONCLUSAO

Este trabalho apresentou uma implementação paralela da FFT e mostrou um modelo simples capaz de estimar o desempenho e a eficiência da mesma em arquiteturas hipercúbicas.

Os parâmetros do modelo são extraídos diretamente da implementação num cubo de pequena dimensão, e permitem a extrapolação para tamanhos arbitrários do problema e da máquina alvo.

Aspectos relevantes da implementação são o tempo de comunicação e o desbalanceamento de carga imposto pela característica da "butterfly" expressos pelo parâmetro t_{pe} do modelo. São possíveis ainda melhoramentos como o sobreposição de processamento e comunicação de maneira a diminuir o valor efetivo de t_{pe} e melhorar o aproveitamento de recursos da máquina, deixados para um trabalho futuro.

BIBLIOGRAFIA

[Fox88] - Fox, Geoffrey C., et al, "Solving Problems on Concurrent Processors. Vol I - General Techniques and Regular Problems " Prentice Hall, Englewood Cliffs, New Jersey, 1988, 592 pp

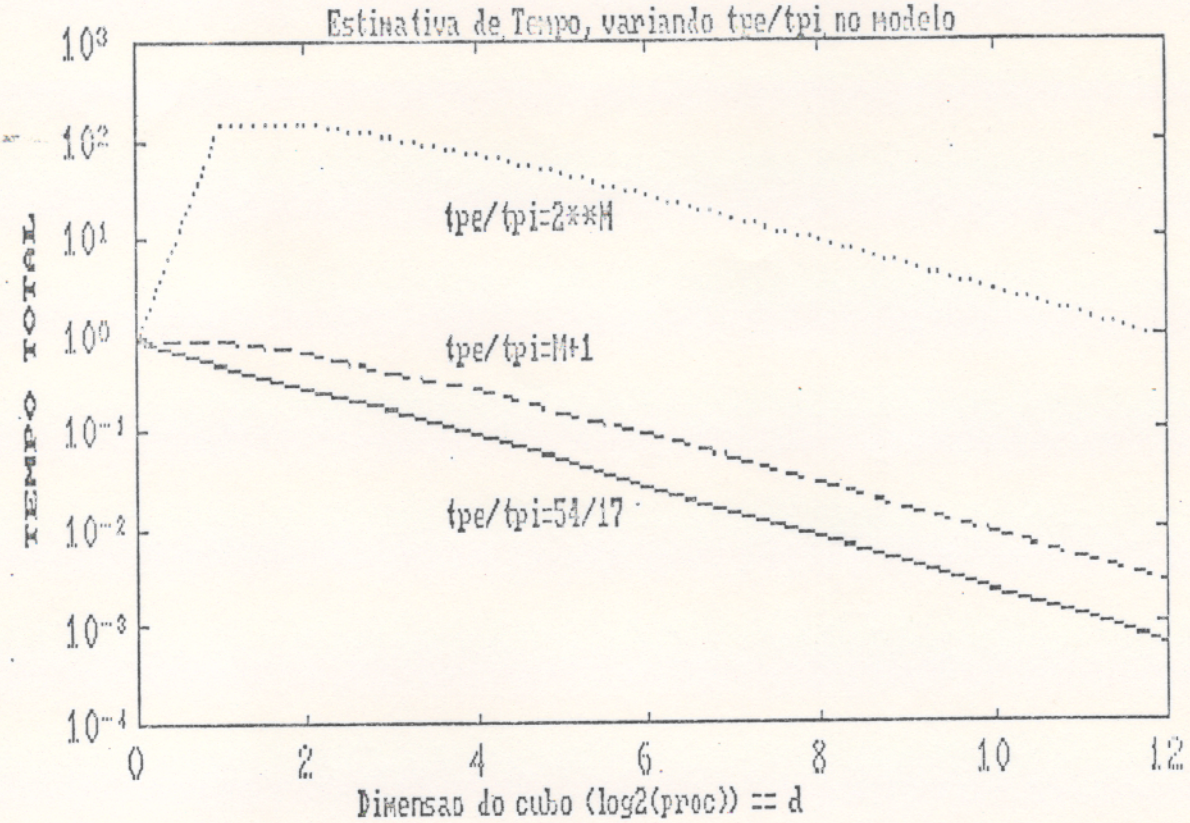
[Opp75] - Oppenheim & Schafer " Digital Signal Processing " Prentice Hall, Englewood Cliffs, New Jersey, 1975

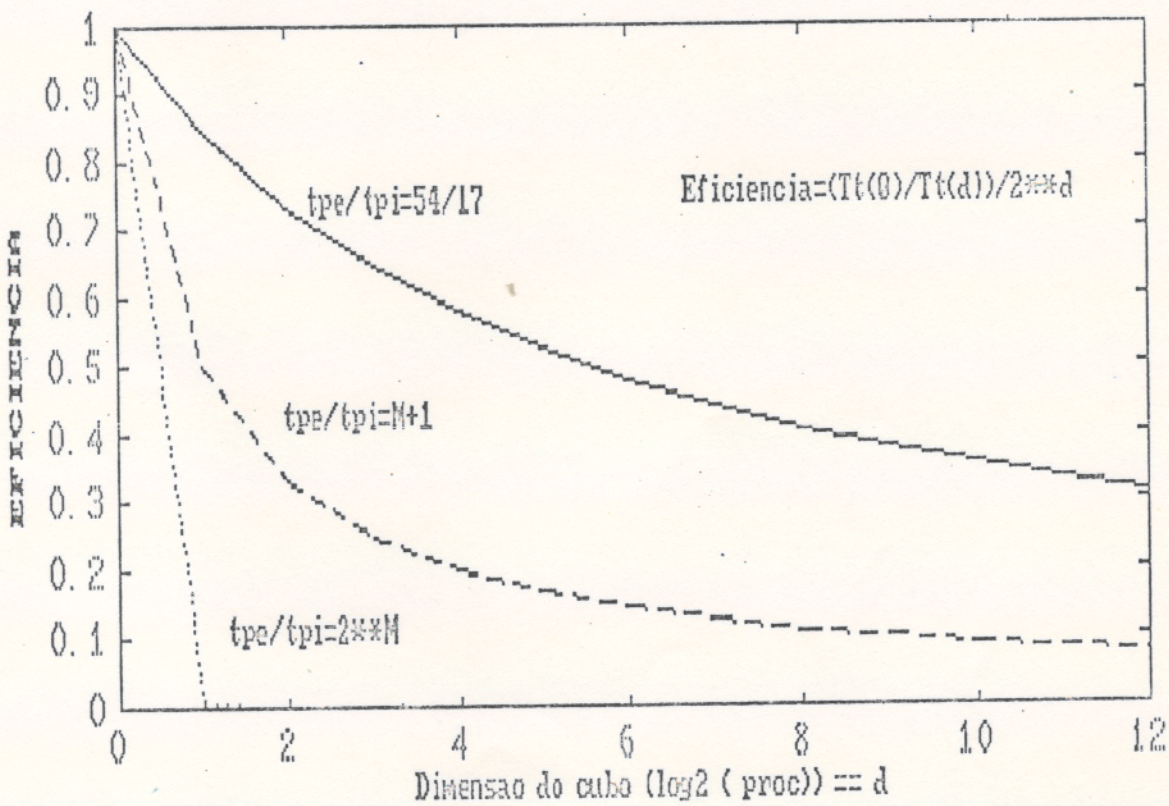
[Cve87a] - Cvetanovic Z. "Best and Worst Mappings for the Omega Network" IBM Journal of R&D, Armonk, NY, 31(4): 452-63, July 1987

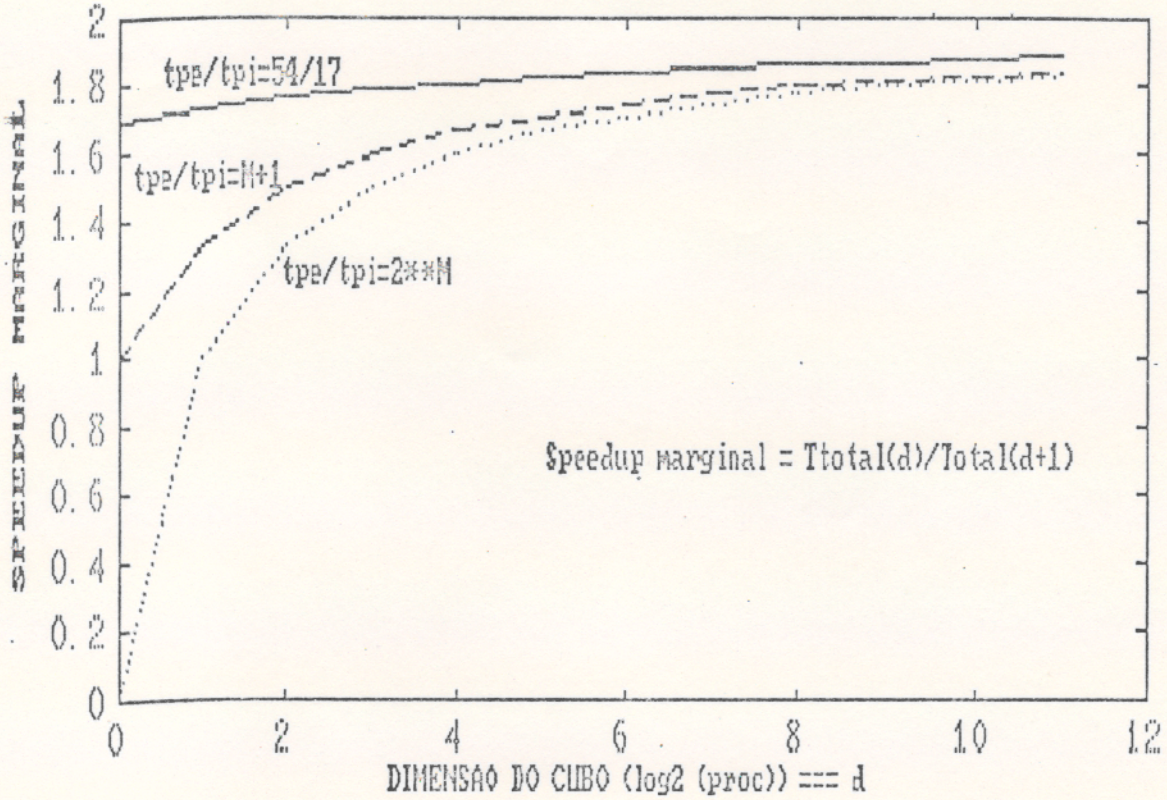
[Cve87b] - Cvetanovic Z. "Performance Analysis of the FFT Algorithm on a shared-memory Parallel Architecture" IBM Journal of R&D, Armonk, NY, 31(4): 435-51, July 1987

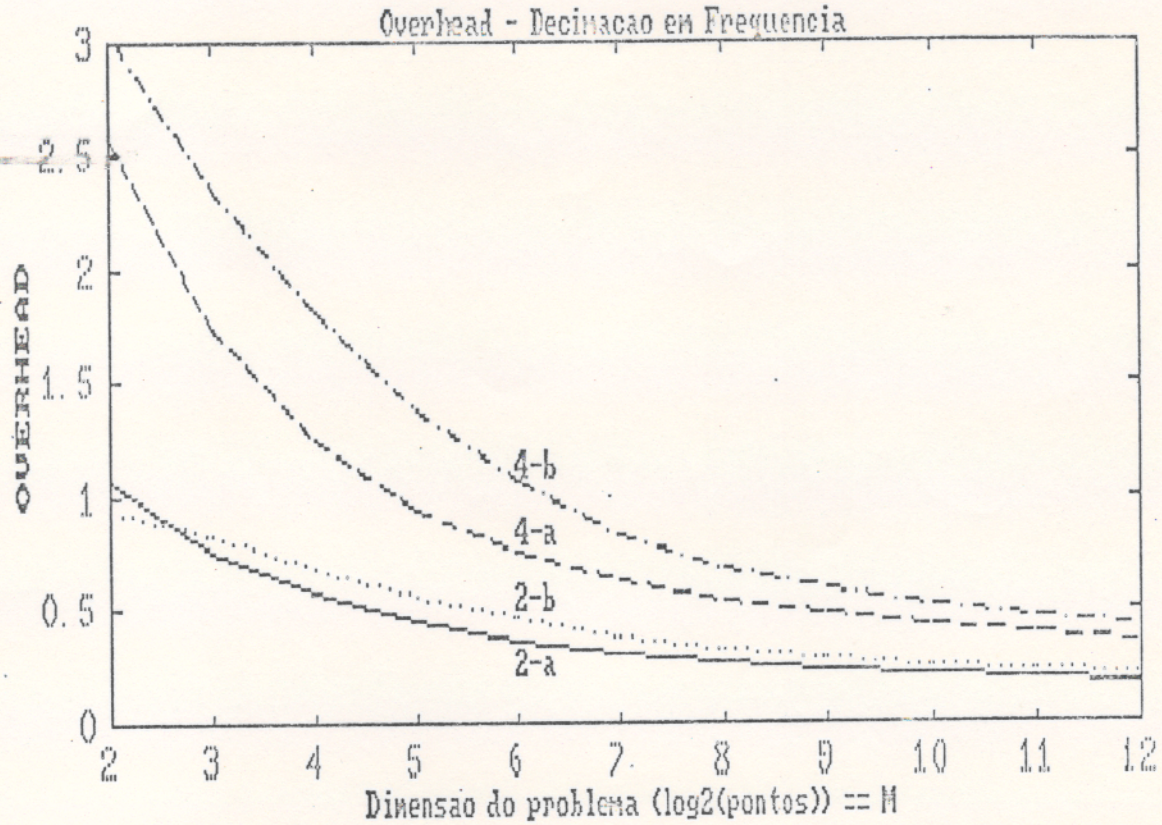
[Sto87] - Stone, H.S. "High Performance Computer Architecture" Addison-Wesley Publishing Co., Reading, MA, 1987, 411 pp

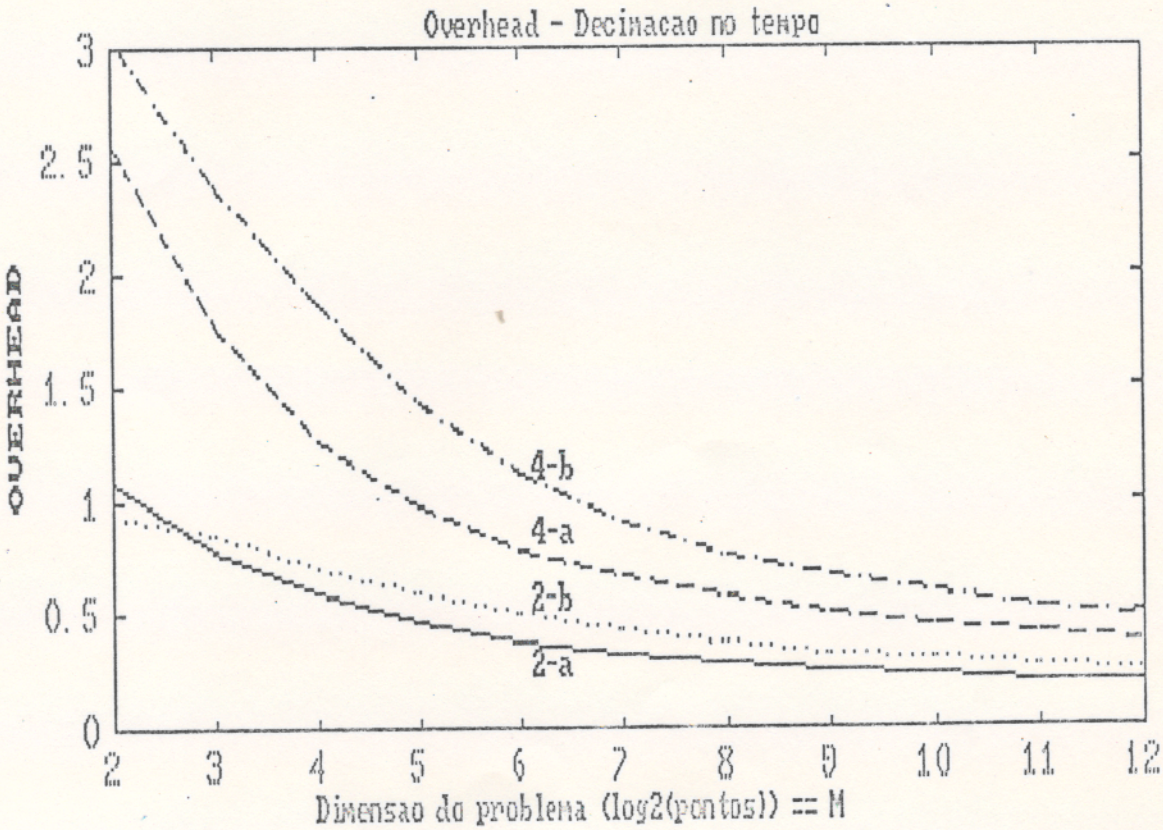
[Par88] - "Parallel C User Guide", 3L Ltd, 1988.

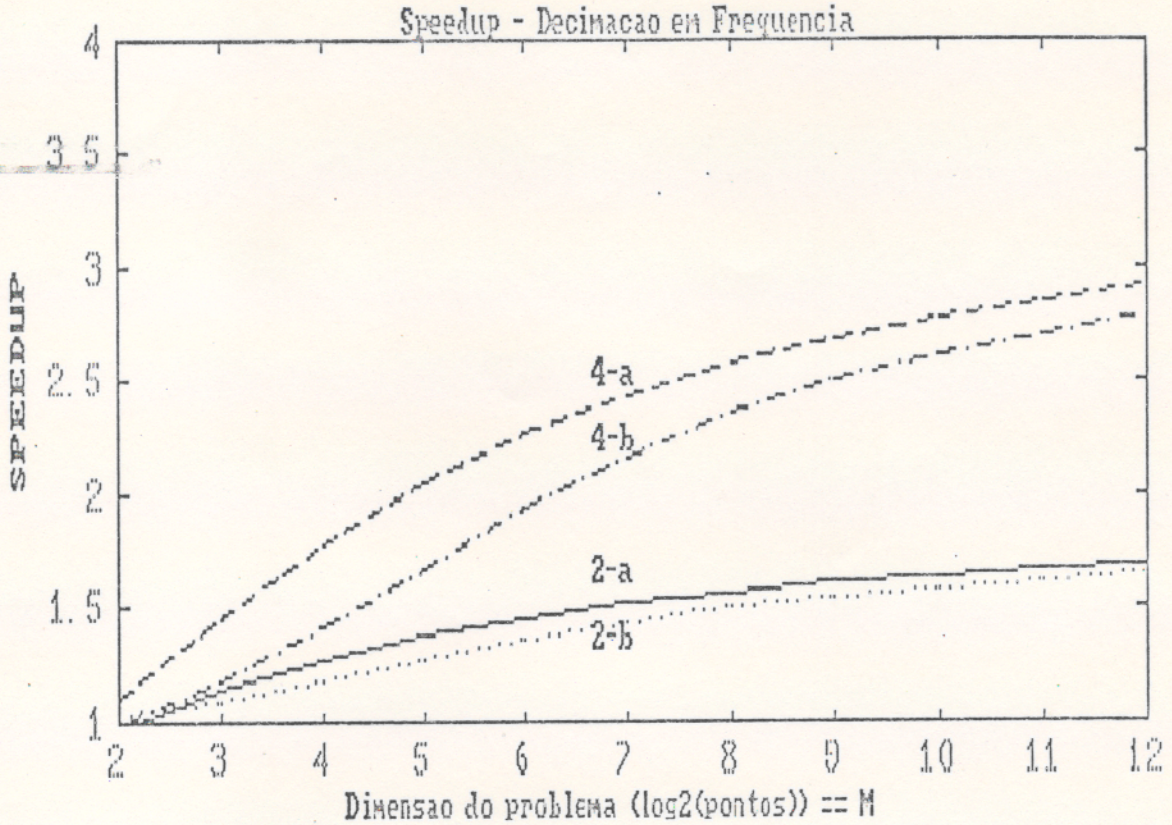

Estimativa de Tempo, variando tpe/tpi no modelo
TEMPO TOTAL
10^3
10^2
10^1
10^0
10^{-1}
10^{-2}
10^{-3}
10^{-4}
tpe/tpi=2**M
tpe/tpi=M+1
tpe/tpi=54/17
0
2
4
6
8
10
12
Dimensao do cubo (log2(proc)) == d

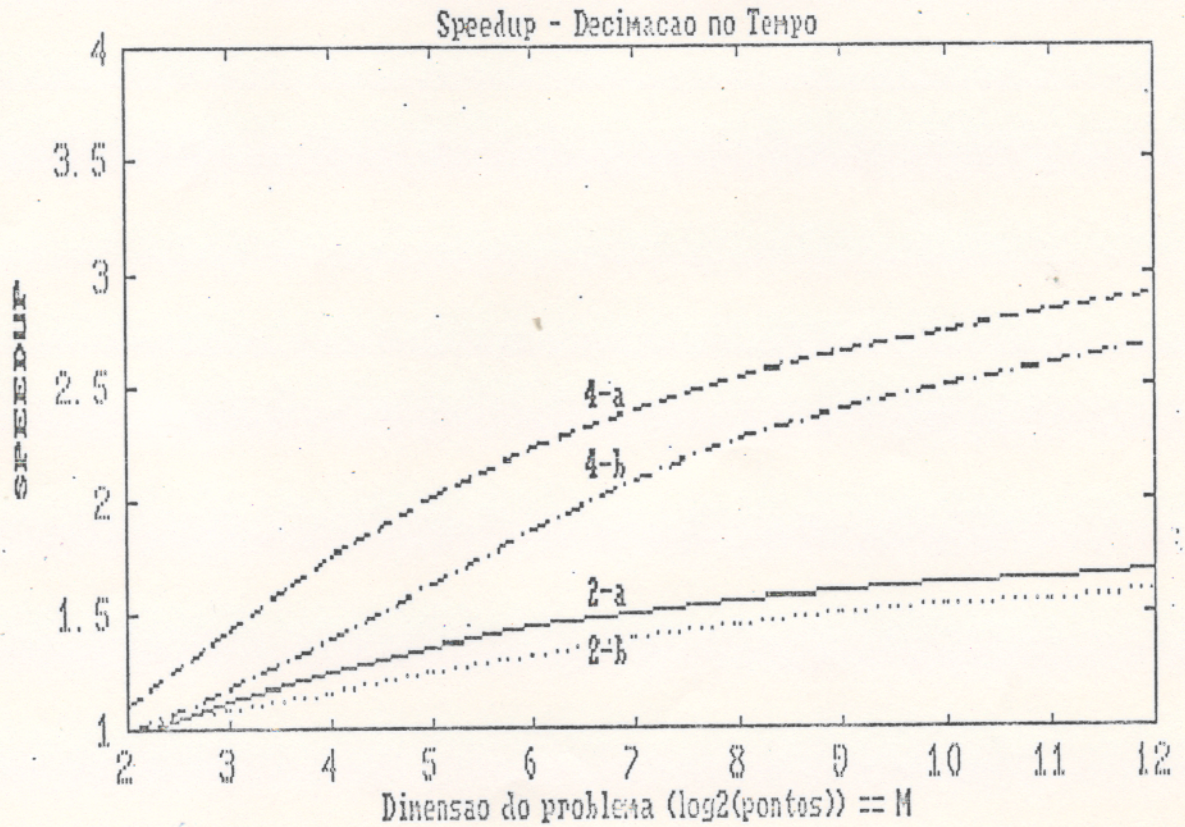

EFICIENCIA
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
tpe/tpi=54/17
Eficiencia=(Tt(0)/Tt(d))/2**d
tpe/tpi=M+1
tpe/tpi=2**M
0
2
4
6
8
10
12
Dimensao do cubo (log2 (proc)) == d


SPEEDUP MARGINAL
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
2
4
6
8
10
12
tpa/tpi=54/17
tpa/tpi=N+1
tpa/tpi=2**N
Speedup marginal = Ttotal(d)/Total(d+1)
DIMENSAO DO CUBO (log2 (proc)) === d


Overhead - Decimacao em Frequencia
OVERHEAD
3
2.5
2
1.5
1
0.5
0
2
3
4
5
6
7
8
9
10
11
12
4-b
4-a
2-b
2-a
Dimensao do problema (log2(pontos)) == M


Overhead - Decimacao no tempo
OVERHEAD
3
2.5
2
1.5
1
0.5
0
2
3
4
5
6
7
8
9
10
11
12
4-b
4-a
2-b
2-a
Dimensao do problema (log2(pontos)) == M


Speedup - Decimacao em Frequencia
SPEEDUP
4
3.5
3
2.5
2
1.5
1
4-a
4-b
2-a
2-b
2
3
4
5
6
7
8
9
10
11
12
Dimensao do problema (log2(pontos)) == M


Speedup - Decimacao no Tempo
SPEEDUP
4
3.5
3
2.5
2
1.5
1
4-a
4-b
2-a
2-b
2
3
4
5
6
7
8
9
10
11
12
Dimensao do problema (log2(pontos)) == M