UM ESTUDO EXPERIMENTAL DO DESEMPENHO
DO SISTEMA DE MEMÓRIA EM MULTICOMPUTADORES

CELSO STAREC

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO EM ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO

Aprovada por:

Prof. Valmir C. Barbosa Ph.D.
(Presidente)

Prof. Edil S. T. Fernandes Ph.D.

Prof. Júlio Salek Aude Ph.D.

RIO DE JANEIRO, RJ - BRASIL
MAIO DE 1992

STAREC, CELSO

Um Estudo Experimental do Desempenho do Sistema de Memória em Multicomputadores [Rio de Janeiro] 1992.

IX, 101 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 1992)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1. Processamento Paralelo

I. COPPE/UFRJ II. Titulo (série)

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.).

UM ESTUDO EXPERIMENTAL DO DESEMPENHO DO SISTEMA DE MEMÓRIA EM MULTICOMPUTADORES

CELSO STAREC

MAIO, 1992

Orientador : Prof. Valmir Carneiro Barbosa.
Programa : Engenharia de Sistemas e Computação.

Os multicomputadores são máquinas paralelas, MIMD, de memória distribuída, cujos nós de processamento se comunicam através da troca de mensagens. Sua eficiência e aplicabilidade dependem de sua capacidade de comunicação em relação à de processamento.

A comunicação, que em essência é a movimentação de um objeto de dados da memória de um nó para a de outro, é limitada, entre outras coisas, pela banda passante da memória. Nesta tese estuda-se o sistema de memória para estas máquinas.

Microprocessadores já são capazes de prover grande desempenho de ponto flutuante, requerido pelas aplicações da máquina, mas também são muito dependentes do sistema de memória.

Deseja-se utilizar memórias dinâmicas (DRAMs) para minimizar o custo. A velocidade destas não tem acompanhado a dos processadores, e o uso das técnicas tradicionais de melhoria da banda passante é dificultado pelas restrições

impostas aos nós e pelos avanços na escala de integração. Estas memórias oferecem a possibilidade de melhoria da banda passante, utilizando ciclos curtos, através da exploração da localidade nos acessos.

São realizados experimentos que avaliam o uso deste mecanismo tanto nos acessos do processador, quanto para a comunicação. O processador usado já suporta estes acessos otimizados, e para a comunicação é simulado um mecanismo de arbitrar a memória por conjuntos (bursts) de acessos, de diversos tamanhos, o que permite melhor aproveitamento da localidade.

Abstract of Thesis presented to COPPE/UFRJ as partial fulfillment of the requirements to the degree of Master of Science (M.Sc.).

AN EXPERIMENTAL STUDY OF THE PERFORMANCE OF MULTICOMPUTER MEMORY SYSTEM

CELSO STAREC

MAY, 1992

Thesis Supervisor : Prof. Valmir Carneiro Barbosa.
Department : Systems Engineering and Computer Science.

Multicomputers are distributed memory MIMD parallel machines whose nodes communicate through message-passing. Their efficiency depend on the node's relative capacity of communication and processing.

Communication, which is in essence a data object movement from one node's memory to another, is limited, among other things, by the memory bandwidth. This work studies the memory system for these machines.

Recent microprocessors are already able to provide the high floating point throughput needed by the applications, but are also heavily dependent on the memory system.

Dynamic memory (DRAMs) use is desired for cost minimization. Their access time, however, is not coping with processor's needs. Restrictions imposed on the nodes and the evolution on the integration scale make difficult the use of traditional technics for memory bandwidth enhancement.

Bandwidth enhancement is possible in DRAMs using

shorter cycles that exploit access locality. Their use is evaluated by experiments with concurrent in-node processing and simulated communication. Optimized memory access is already supported by the processor and is achievable for communication with burst memory accesses. This scheme is simulated for many burst sizes.

ÍNDICE

Esta tese é dedicada à memória de meu inesquecível tio Henrique Rath-Fingerl, Z"L, grande Homem, exemplo de integridade e valor, um Tzadik, que tanto me ensinou no sentido mais amplo da vida e muito me incentivou a concluir este trabalho, tendo sido impedido de ver sua conclusão pela fatalidade da vida. A ele, todo meu mais profundo respeito e carinho.

Meus agradecimentos especiais a minha família e amigos, que tanto ajudaram nas horas mais críticas. A meus pais Nelly e Iloni pelo suporte financeiro e emocional, às minhas irmãs Elaine e Simone pelo apoio constante. Ao amigo Ricardo Gomes Donadio, por toda amizade e companheirismo ao longo de todo o desenvolvimento do trabalho, além das incontáveis "quebradas de galho" e toda a ajuda. Ao dublê de técnico da COPPE e amigo José Duarte de Queiroz e sua noiva Ana Paula, que sempre prestativos, com interesse e auxílio, em muito ajudaram a humanizar a face da instituição e facilitar o desenvolvimento deste trabalho.

E por fim, "last but not least", meus agradecimentos a meu orientador Valmir Carneiro Barbosa por toda paciência, compreensão e auxílio.

CAPÍTULO 1 :
INTRODUÇÃO

1) INTRODUÇÃO

O trabalho aqui apresentado versa sobre o sistema de memória para um nó de uma máquina paralela, MIMD, de memória distribuída, ou seja um multicomputador.

Esta é uma categoria importante no processamento paralelo, que alcançou notoriedade com o "Cosmic Cube" da Caltech, [Seitz85], e na qual já há diversas (16) firmas explorando comercialmente o mercado [Bell89]. Sua aplicabilidade para problemas científicos e tecnológicos já foi comprovada [Fox88] e [Fox88a].

Esta categoria tem um apelo especial pois ela permite o uso de tecnologia similar à utilizada em microcomputadores e "workstations", tanto no que diz respeito ao hardware quanto ao software, e que muitas firmas de fato empregam. Isto traz vantagens de custo devido ao uso de tecnologia já "massificada" e à familiaridade com a mesma.

Na realidade um multicomputador se assemelha a uma rede de computadores independentes. Existem diferenças óbvias como ter um sistema operacional (S.O.) único, e minimizar os custos periféricos como fontes, gabinetes, interfaces com usuários, etc. Além destas, existe uma diferença mais fundamental: as soluções adotadas para criar uma rede de comunicação de alta capacidade e baixa latência que permita um acoplamento maior entre os processadores e seu aproveitamento efetivo, mesmo nos casos em que devam cooperar numa computação.

O conhecimento até agora acumulado demonstra que a comunicação entre os nós é o elemento chave para definir a qualidade destas máquinas, que se expressa por uma fórmula que relaciona a eficiência da computação paralela com a razão entre tempos de comunicação (Tcomm) e procesamento (Tproc) gerados por um algoritmo para processamento paralelo.

Muitos são os fatores que influenciam os tempos de comunicação, como: a topologia da rede (cubos,meshes,...); a técnica de transmissão (store&forward, wormhole-routing); as

primitivas e o suporte do sistema operacional (modelo síncrono de baixa latência para "produção" e/ou assíncrono para suporte); a velocidade dos links e a banda passante de memória.

As redes de comunicações para estas máquinas têm sido objeto de muitos estudos, e aos poucos vai se formando um consenso sobre algumas questões como o uso de "wormhole-routing" para minimizar as latências de comunicação entre nós não vizinhos. Outras questões como a melhor topologia, se um hipercubo-binário ou um K-ary n-cube, ainda são objeto de alguma controvérsia que espera uma definição experimental usando problemas, máquinas e softwares reais.

O termo granularidade é usado livremente para identificar ou o tamanho do elemento computacional de uma máquina, ou o tamanho de uma unidade de execução de um programa concorrente.

Este trabalho identifica a limitação da banda passante de memória como uma questão importante para a categoria dos multicomputadores de granularidade média e grossa das próximas gerações. Isto se dá por vários motivos: primeiro a velocidade das CPUs tem crescido muito mais rapidamente que a das memórias; segundo as CPUs têm se tornado mais dependentes de um sistema de memória otimizado para fornecer o desempenho prometido; terceiro as restrições impostas aos nós da máquina paralela coibem o uso das técnicas tradicionais para superar a limitação da banda passante da memória. Isto provoca uma limitação da capacidade sustentada relativa de comunicação piorando a relação Tcomm/Tcalc, e conseqüentemente a efetividade da máquina.

Este trabalho foi desenvolvido em três etapas distintas. As etapas correspondem às fases seqüenciais de investigação e desenvolvimento do aparato que viabilizou a geração dos resultados experimentais. Estes envolvem vários aspectos do sistema de memória e sua capacidade de suportar processamento e comunicação. Cada etapa permitiu avaliar a

validade e a viabilidade da argumentação da tese e do método experimental adotado.

A primeira etapa consistiu de uma simulação inicial simples que permitiu uma primeira estimativa dos possíves ganhos com o aproveitamento da localidade no acesso às memórias dinâmicas através de acessos mais curtos. Nesta etapa usou-se um modelo determinístico bastante simples para os acessos à memória. Tomando por base este padrão de acessos idealizado fêz-se a projeção de qual seria o ganho de banda passante da memória para diversos valores da relação entre os tempos de um acesso distante, completo e lento, Ciclo Far, e um acesso próximo, curto e rápido, Ciclo Near.

A segunda etapa consistiu na definição dos experimentos, e na realização de um conjunto inicial de testes que não requeriam desenvolvimento substancial de hardware adicional.

Nesta etapa foi usada a placa do i860 do projeto do NCP-I da COPPE/UFRJ, que representa um exemplo válido de sistema para fim dos testes propostos pelas características de sua CPU e do seu sistema de memória dinâmica.

Foi desenvolvido o ambiente de software que permitiu o uso da linguagem de alto nível C, com sua biblioteca matemática. Foram desenvolvidas também rotinas para controlar o hardware e rotinas de I/O com um IBM-PC conectado como terminal que permitiram executar e depurar programas de teste e registrar os resultados dos experimentos.

As rotinas de teste da segunda etapa foram FFTs binárias, transformadas rápidas de Fourier, por decimação no tempo e na freqüência em diferentes versões e algumas rotinas acessórias.

Nesta etapa avaliou-se o desempenho do conjunto placa/compilador, os efeitos das opções de otimização, a efetividade da cache e dos Ciclos Near e fez-se uma estimativa inicial de degradação que o DMA simples para

comunicação causa à capacidade de processamento.

A terceira etapa consistiu do projeto e implementação de uma placa de extensão que é acoplada à placa do i860. Este dispositivo permitiu a análise, com resolução apropriada, da influência do DMA para comunicação no desempenho do processamento; e da eficácia do uso de "bursts" de acessos locais para melhorar a banda passante da memória e a taxa máxima de comunicações suportável.

Tendo em vista que o código gerado pelo compilador faz pouco uso da capacidade super-escalar do processador e tem um desempenho baixo comparado com o de pico ou mesmo o sustentável possível, foram usados programas de teste em assembly que faziam uso destes recursos. Neste caso torna-se mais evidente a dependência com o desempenho do sistema de memória. A aplicação usada foi a multiplicação de matrizes. Usaram-se programas apresentados no Programmer's reference manual do i860, [Intel89c,Cap9], e outros desenvolvidos a partir destes.

No capítulo 2 apresenta-se a demanda por supercomputação e o porquê do processamento paralelo. A seguir discute-se sobre o espectro do processamento paralelo, em especial no que diz respeito ao grau de replicação e à complexidade de cada elemento, e apresentam-se os tipos principais de máquinas e sua classificação. Discute-se também as características da computação científica e sua viabilidade em diversos tipos de máquina. Por fim apresenta-se o multicomputador, a capacidade relativa de comunicação como fator de mérito.

No capítulo 3 dicute-se o nó de processamento. Primeiro alguns princípios, restrições e critérios para o desenvolvimento do nó. A seguir a relação entre VLSI e os componentes do nó. Em especial apresentam-se as características das CPUs modernas que as tornam muito dependentes de um sistema de memória otimizado para manter o desempenho elevado que prometem tomando o i860 da Intel,

[Intel89a,b,c,d], como exemplo principal. Estudam-se também as projeções de especialistas para as DRAMs de megabits, as características da memória que permitem otimizações no seu tempo de acesso, os protocolos de "Page Mode", "Static Column" e "Nibble Mode" já estabelecidos, e novos protocolos e arquiteturas sugeridos em bibliografia recente. Segue-se uma análise das opções de nó e uma exposição precisa do problema tratado na tese.

No capítulo 4 apresenta-se o ambiente experimental, as idéias, o hardware e o software envolvidos nas etapas experimentais.

No capítulo 5 são apresentados os experimentos realizados e os resultados deles obtidos.

No capítulo 6 apresentam-se as conclusões do trabalho.

CAPÍTULO 2 :
MÁQUINAS PARALELAS COM MEMÓRIA DISTRIBUÍDA

2) MÁQUINAS PARALELAS COM MEMÓRIA DISTRIBUÍDA

2.1 A Demanda por Supercomputadores

Existe uma grande demanda por computadores rápidos e eficientes para uso em universidades, na indústria, e em centros de pesquisa governamentais ou privados. Estas máquinas são usadas em muitas áreas, como: científica, de engenharia, de recursos energéticos, médica, militar, inteligência artificial, e pesquisa básica.

Computações em grande escala são realizadas nas áreas citadas. Em [Fox88,pg18] cita-se, como uma ocorrência comum, problemas cujas soluções são conjuntos de equações diferenciais parciais tridimensionais sobre um domínio discretizado em um total de 10^6 a 10^8 pontos, exigindo Gigabytes de memória e Gigaflops de desempenho. Em [Stone87,cap4] citam-se problemas, cujas soluções ao nível de precisão desejado, requerem de 10^{12} a 10^{15} operações de ponto flutuante que, em computadores de alto desempenho convencionais, levariam anos para serem completadas. Existem muitas outras aplicações onde grande capacidade de cálculo é necessária, citadas em [Fox88], [Hwang84], [Fox88a] e em suas referências, além de uma infinidade de outras que seriam possíveis caso existisse capacidade de cálculo a custos acessíveis.

Em [Fox88] e [Hwang84] diz-se que a solução de problemas científicos em grande escala envolve a interação de teorias, experimentos e computação. Cientistas teóricos desenvolvem modelos matemáticos que são resolvidos numericamente para permitir sua análise, enquanto que os dados gerados experimentalmente são processados para modelar processos, ou para sua análise.

A relevância da simulação computacional na substituição de experimentos físicos convencionais tem sido cada vez mais reconheciada. Isto ocorre porque as simulações computacionais podem ser mais rápidas e baratas; elas são limitadas apenas pela velocidade e capacidade de memória

enquanto os experimentos têm muitas limitações práticas; e além disso os computadores podem ser aplicados a um universo de problemas bem maior que equipamentos específicos de laboratório. Em [Fox88] é citada uma dúzia de campos em que estas simulações computacionais são de grande importância.

Aplicações que exigem cálculos em grande escala tem sido o campo dos supercomputadores, vagamente definidos como as máquinas mais rápidas (e caras) a um dado tempo. O desempenho ainda tem sido definido pelas máquinas vetorias como os CRAYs e seus equivalentes da CDC, ETA, NEC, etc. com custos na faixa de 10 a 30 milhões de dólares.

2.2 Processamento paralelo

Muitos são os motivos para a adoção do processamento paralelo na tentativa de realizar computadores mais rápidos e baratos que os convencionais.

Os supercomputadores atuais são máquinas vetorias, e exploram eficientemente este tipo de concorrência pelo uso de "pipelining" e múltiplos bancos independentes de memória para prover a banda passante necessária. A execução simultânea de muitas operações nos vários estágios dos "pipelines" e nas unidades funcionais, junto com o uso dos circuitos mais rápidos disponíveis, permite um "throughput" muito alto.

Um limite básico é a velocidade da luz. Mesmo à velocidade da luz, um sinal só poderia percorrer 30cm em 1ns, o que dá uma idéia das dificuldades técnicas para se construir máquinas reais.

Nos supercomputadores atuais, a dimensão física da máquina tem de ser limitada, pois os sinais se propagam a velocidades menores que a da luz, há problemas de reflexão e estabilização, e há necessidade de margens de tempo por causa do "clock skew". Os circuitos, tipicamente em ECL, têm dissipação elevada de energia, que se torna mais difícil por causa do volume limitado da máquina. O CRAY2 parece com um aquário já que suas placas são imersas em um fluído inerte

que, como a água do radiador de um automóvel, serve para refrigeração, [TIME85,pg39] e [Stone87,pg261]. Uma das maiores dificuldades no desenvolvimento do CRAY1 também estava na refrigeração, no desenvolvimento da "barra fria", [Bell82,pg752].

Mesmo superando todas as dificuldades técnicas em estender os limites de velocidade dos componentes e circuitos, e da montagem física compacta, ainda assim, a velocidade de um processador seqüencial é limitada.

Em [Seitz90] argumenta-se a favor da concorrência e do processamento paralelo. É apresentada uma análise muito interessante sobre o custo de uma operação em relação a sua latência baseada em estudos sobre Teoria da Complexidade em VLSI e Aspectos Computacionais de VLSI citados em suas referências.

É citado que, para certas computações transitivas, em que cada saída pode potencialmente depender de qualquer entrada, a área ocupada no chip A e o tempo de latência de uma computação T são relacionados de tal forma que AT^2 deva superar uma certa função, dependente do problema, do tamanho deste, em que os coeficientes acomodam as características da tecnologia. Assim, $AT^2 > k$, ou $AT > k/T$.

Pode-se interpretar AT como o custo de uma operação, e este custo varia inversamente com T, a latência que é permitida para a operação. Seitz chama a esta hipótese de: "haste-makes-waste", ou a pressa faz o disperdício, e afirma que quando se compra um computador está se pagando por tempo de resposta e não por ciclos.

A análise prossegue mostrando que em determinados casos a energia gasta por operação também poderia ser feita variar com 1/T, empurrando o gasto para fora do chip, onde potencialmente poderia ser reduzido.

Supondo-se uma computação que possa ser desdobrada em N concorrentes, e usando o resultado de que o custo por operação varia com 1/T, chega-se a duas concluões

importantes:

- O custo da computação concorrente pode ser 1/N para um tempo fixo, em relação ao que custaria executá-la seqüencialmente numa máquina N vezes mais rápida; e

- A velocidade da computação concorrente pode ser N vezes maior que a seqüencial a um custo fixo por computação (em termos de AT, ou CPUs-hora).

Esta exposição é valida para problemas limitados pela comunicação dentro do chip (AT^2), porém existem problemas com outros tipos de limitações como: I/O do chip (AT) e armazenamento (A).

Além disso, assume-se implicitamente que a comunicação entre os chips participantes da solução concorrente pode ser negligenciada baseado numa hipótese de granularidade mínima, em que há, no problema, computação seqüencial independente suficiente para cada chip. Esta é uma hipótese importante cujos efeitos sobre a eficiência de uma computação paralela serão abordados adiante.

Seitz, citando Gordon Bell, ainda afirma que o avanço para a taxa do clock dos supercomputadores tradicionais tem sido da ordem de 1.17/ano. Este ganho é menor do que o observado em outras classes de máquinas, [Patterson90,pg4], em especial nas baseadas em microprocessadores.

Mesmo os projetistas destes supercomputadores têm derivado a maior parte do ganho de desempenho do paralelismo. Como exemplo tem-se a comparação do CRAY-YMP com o CRAY-XMP cuja relação de clocks é 1.5 e a do número de CPUs é 2. Ainda segundo Seitz, pode-se interpretar que os projetistas destas máquinas não estão indo bem em aumentar o clock, pois isto significa diminuir as latências dos circuitos individuais contra ganhos decrescentes. Daí conclui-se que mesmo os projetistas de supercomputadores têm retirado seu ganho de desempenho do paralelismo, aumentando N, através do acréscimo de CPUs, e não a velocidade do clock de cada CPU.

É interessante observar que a emergência do

processamento paralelo realça o papel da comunicação/ comutação elevando-o a um eixo de projeto importante nos sistemas de computadores.

São três as primitivas associadas ao tratamento dos fenômenos associados à informação:

-o processamento, ou transformação no espaço e no tempo;

-a memória, o transporte no tempo;

-a comunicação, o transporte no espaço.

Esta abordagem aparece tanto num estudo mais abstrato como na "Filosofia Natural da Informação", em [Mammana81], quanto na notação PMS (Processors, Memory, Switches), [Bell82,pg17], que serve, num determinado nível de abstração, para representar grosseiramente a estrutura de um sistema, os fluxos de informação entre seus componetes, e as quantidades de informação armazenadas neles.

Em essência, a natureza está nos dizendo que transportar informações de um ponto a outro do espaço tem custo, e os recursos que tornam isto possível não devem ser desperdiçados. Sempre que for possível aproveitar localidade pode-se ganhar eficiência, portanto os custos e as opções devem ser tornados visíveis para que possam ser otimizados.

Os processadores e a memória são eixos de projeto tradicionais, e agora a comunicação completa um modelo "Natural" dos fatores de custo, que aponta para as vantagens da localidade nos diversos níveis em que puder ser aproveitada, seja internamente a um chip, uma placa, ou um sistema.

2.3 Espectro do processamento paralelo

Um ponto de acordo entre autores é o de que os compromissos entre custo, desempenho, aplicabilidade a problemas e programabilidade para os diversos tipos de máquinas paralelas ainda estão mal definidos, pois o conhecimento acumulado em aplicações e programação paralela ainda é limitado. Ambiciona-se velocidade e uma boa relação

custo/desempenho.

Em [Seitz90] é apresentado um gráfico que relaciona o grau de replicação N e a complexidade dos nós. O produto destes eixos representa o custo da máquina. A um custo constante teríamos uma parábola, que num gráfico log-log dá uma reta. Tomando uma reta para definir um sistema de custo mínimo e outra para o máximo prático define-se um espaço real de projeto para uma máquina paralela.

Para sistemas práticos, o número de chips é limitado por requisitos de disponibilidade, pois quanto maior o número de chips, maior a propensão a falhas.

Em ordem decrescente de replicação e crescente de complexidade encontram-se as RAMs, Connection Machines (das séries 1 e 2, CM-1 e CM-2), "arrays" sistólicos, multicomputadores e computadores convencionais. O desempenho de pico tende a aumentar com a replicação, enquanto que a generalidade com a complexidade.

Em [Patterson90,pg576] é apresentado um gráfico que relaciona o número de processadores com desempenho de cada processador individual. Esta métrica obviamente guarda uma relação íntima com a anterior. Este autor argumenta que o aumento de velocidade por processador privilegia o desenvolvimento e o aproveitamento de software, enquanto que a replicação facilita o desenvolvimento de hardware.

A Connection Machine (CM-1) e outras máquinas como a descrita em [Optic89] nasceram da idéia de incrementar chips de memória com alguma lógica para permitir paralelismo e processamento simples, rápido, eficiente e em larga escala.

Na faixa dos multicomputadores, a idéia é aproveitar a disponibilidade de microprocessadores e memórias com a tecnologia já massificada para outros mercados e uma excelente relação custo/desempenho. Estes processadores têm a melhor relação de custo/desempenho e seu desempenho nominal cada vez mais se aproxima do das classes maiores de máquinas.

Um i860 da intel, em CMOS, a 40 MHz, pode dar 80 MFlops

de pico em precisão simples, e um processador do CRAY-YMP, em ECL, a quase 170 MHz (6ns), 340 MFlops. A diferença entre os desempenhos sustentados é bem maior, principalmente devido à capacidade de acesso aos operandos na memória, mas estes números já indicam uma mesma ordem de grandeza.

Numa máquina paralela existem diversas opções. Pode ser SIMD ou MIMD, [Flynn1966], ter memória centralizada ou distribuída, compartilhada ou privada.

Máquinas SIMD, "single instruction and multiple data streams", tem o objetivo de minimizar o custo do controle, e é aplicável a máquinas como a Connection Machine (CM-1 e -2) com seus 64K processadores de um bit. Seria mesmo inconcebível, devido ao sobre-custo, replicar unidade de controle complexa e os programas para cada nó de granularidade tão fina. Existem também máquinas SIMD de granularidade grossa como ILLIAC-IV da NASA, [Bell82,pg306], e GF-11 da IBM, [Stone87,pg268].

Máquinas MIMD, "multiple instruction and multiple data streams", tem um processador completo, com unidade de controle, em cada nó. Assim, diferentes nós podem estar executando diferentes seqüências de instruções ao mesmo tempo, o que confere maior generalidade à máquina, apesar do custo maior em comparação a uma SIMD.

É difícil escalar para muitos processadores um sistema com memória fisicamente centralizada, e por isso, quando se propõe máquinas que possam ter mais do que algumas dezenas de nós, usa-se memória distribuída.

A memória pode ter o espaço de endereçamento único, e ser compartilhada ("shared") pelos nós implicitamente, ou múltiplo e ser privada para cada nó, sendo compartilhada explicitamente por troca de mensagens entre os nós.

Dentre a multiplicidade de arquiteturas paralelas existentes, propostas e possíveis, trataremos apenas de três sub-classes: SIMD, MIMD-DM, MIMD-SM, onde, DM significa memória distribuída (e privada) e SM memória compartilhada.

Estas três classes estão em ordem crescente de custo e

gereralidade de aplicação, sendo que a maioria dos multicomputadores se insere na classe MIMD-DM.

2.4 Computação científica em grande escala

A computação científica em grande escala tem como característica o uso intensivo de cálculos de ponto flutuante.

Uma outra característica é a estruturação. Em [Stone87,pg333], argumenta-se que para um número de operações da ordem de 10^{11} deve existir no máximo de 10^4 a 10^6 instruções distintas no código, ou o problema seria humanamente intratável. Assim, deve haver uma repetição de 10^5 a 10^7 embutida em algum lugar, provavelmente num loop central, onde o programa gasta a maior parte do tempo. Neste trecho devem-se concentrar os esforços de otimização e paralelização.

Em [Fox88,pg42] existe um argumento semelhante. No caso mais geral, as computações concorrentes de sistemas complexos em computadores complexos podem ser difíceis ou mesmo impossíveis, mas nos casos particulares que nos interessam, onde tanto os computadores quanto os problemas são produtos da natureza, do homem ou de suas invenções, as computações tendem a ser modulares, e tratáveis diretamente numa máquina concorrente.

Em [Fox88a] é feito um levantamento de muitas (84) aplicações paralelas reais que são efetivamente executadas em máquinas das diversas classes. O objetivo disto é tentar esclarecer qual arquitetura paralela é mais apropriada para que tipo de problema, e qual é o ambiente de programação adequado.

São analisados problemas que rodam nas três classes de máquinas anteriormente mencionadas: SIMD, MIMD-DM (Multi-computador), MIMD-SM (com memória compartilhada).

Aplicações paralelas têm paralelismo de dados, ou seja, é feita uma decomposição do domínio dos dados entre os elementos processadores que operam sobre estes em paralelo.

É desenvolvida uma classificação baseada em uma analogia com espaço-tempo, em que o espaço representa o domínio dos dados e o tempo, a etapa da computação sendo realizada sobre esses dados.

Os problemas são divididos de acordo com a variação da estrutura "temporal" do seu grafo de computação em três classes: S ("synchronous"), LS ("loosely synchronous"), e A ("asynchronous"). Como uma classe engloba a outra, são definidas as categorias S, PLS (LS mas não S), PA (A mas não LS). Existem ainda aplicações cujo grafo é desconexo, (ou quase), não requerendo comunicação. A estas chama-se: Embaraçosamente Paralelas EP, que podem requerer apenas máquinas SIMD (EP-S), ou necessitar de uma MIMD (EP-M), em função da homogeneidade.

Os problemas S, cuja estrutura "temporal" não varia com a evolução da computação, executam eficientemente em qualquer máquina, e permitem a exploração de todas as vantagens da classe SIMD.

Os problemas PLS têm um grafo computacional constante no "tempo" para unidades macroscópicas deste, sendo que entre estes pontos de sincronização, as diferentes partes do "espaço" podem evoluir independentemente, como ocorre com simulações físicas "espacialmente" irregulares, ou não homogêneas. Estes problemas em geral requerem máquinas MIMD para que se possa manter os processadores ocupados e uma eficiência razoável no uso da máquina. Isto ocorre por causa da limitação de uma única instrução igual para todos os processadores, nas SIMDs, o que as obriga a operar com apenas alguns processadores, deixando outros sem uso. Quanto maior for a irregularidade maior a ineficiência.

Os problemas PA não possuem regularidade no "tempo" e cada membro do "espaço" evolui independentemente sem sincronização natural. Estes problemas requerem balanceamento dinâmico de carga, e talvez sejam mais eficiêntes em máquinas MIMD-SM, apesar de que isto ainda não é claro, pois a experiência acumulada ainda é limitada.

O percentual de aplicações por categoria foi de:

S	40%
PLS	36%
EP-S	7%
EP-M	7%
PA	10%

Pode-se concluir que a arquitetura SIMD é aplicável a aproximadamente 50% dos problemas e MIMD-DM a 90%, com elevado grau de paralelismo, desde que satisfeitas as condições implícitas de granularidade mínima e capacidade de comunicação na máquina paralela.

Ainda segundo [Fox88a], a conclusão mais importante é a de que problemas LS grandes são executados eficientemente em máquinas paralelas, e o "speedup" pode ser proporcional ao número de nós.

2.5 O Multicomputador

O interesse pelos multicomputadores como uma opção viável para fazer máquinas de alto desempenho surgiu com o advento dos microprocessadores com capacidade de ponto flutuante, notadamente com o Cosmic Cube, [Seitz85], um hipercubo de 64 nós com processadores Intel 8086/8087 desenvolvido na Caltech.

Uma introdução ao assunto pode ser encontrada em [Hayes89]. Em [Reed87] pode-se encontrar uma análise mais detalhada que inclui a teoria e os mecanismos, em especial das redes de interconexão, aplicações, e a descrição e a análise de desempenho das máquinas da primeira geração. Já há inclusive quem proponha sistemas de "Computer-Aided Programming for Message-Passing Systems", como em [WU89].

Este grande interesse advém do fato de que, com o aumento da escala de integração e da velocidade dos componentes VLSI, se torna possível construir máquinas baseadas em múltiplos microprocessadores com um desempenho agregado muito grande a um custo por operação baixo.

Em [Bell89], Gordon Bell estima que os multi-

computadores só se tornarão uma estrutura aceita se forem capazes de prover uma potência de cálculo não disponível em outro lugar e com uma relação custo/desempenho pelo menos uma ordem de grandeza melhor que os supercomputadores atuais.

O projetista de um multicomputador deve procurar dotar a máquina das características que permitam sua utilização mais eficiente.

2.5.1 Comunicação como fator de mérito

Muitos são os fatores que afetam o desempenho de um multiprocessador, como por exemplo: o desbalanceamento da carga de trabalho entre os processadores, os "overheads" de escalonamento, de sincronização e de comunicação e duplicação de esforço entre os processadores.

O desbalanceamento de carga e a duplicação de esforço devem ser tratados ao nível do algoritmo e do particionamento do problema, de maneira que os processadores sempre tenham alguma computação útil a realizar. No caso do desbalanceamento pode-se mapear diversas tarefas para um mesmo nó, de modo que o tempo ocioso médio seja menor.

Se uma máquina deve operar com granularidade fina, deve ter um custo de escalonamento baixo para não ser inviabilizada por esta fonte de "overhead". Esta é uma preocupação do Transputer, [INMOS87], [INMOS88], e [Nicoud89], que mantém apenas um mínimo de estado e pode chavear de processos rápida e eficientemente. Esta também é uma procupação para a J-Machine em desenvolvimento no MIT, [Dally88] e [Seitz90,pg140].

Num multicomputador, que é baseado no paradigma de passagem de mensagens, a sincronização entre as computações se dá exclusivamente pela troca de mensagens, recaindo no problema mais geral da latência das comunicações. Este é o fator principal a determinar a eficiência da máquina paralela.

Na realidade uma computação pode ser executada

eficientemente numa máquina desde que tenha uma granularidade mínima, [Stone87,cap6] e [Fox88,cap3]. Esta granularidade representa o tamanho de uma computação em relação ao "overhead" que ela gera, no caso: a comunicação.

Em [Fox88] afirma-se que desde que a granularidade mínima seja atingida, ou seja, desde que o problema seja suficientemente grande, o "speedup" pode ser proporcional a N, o número de processadores. A idéia da granularidade mínima baseia-se numa generalização multidimensional da relação área/perímetro (computação/comunicação) para o conjunto de dados em um processador, e supõe uma estrutura de interconexão suficientemente rica para dar o suporte necessário às comunicações, como no caso dos hipercubos.

É apresentado um modelo de desempenho que relaciona speedup (S), overhead (fc), e eficiência (Ef), onde Tseq é o tempo de execução seqüencial e Tconc(x) o tempo de execução concorrente em x processadores. O modelo parte da premissa de que a comunicação e o processamento não se sobrepõem, e portanto, todo tempo de comunicação é overhead.

```
Tseq = Tconc(1)
S(N) = Tseq / Tconc(N)
Ef = S/N
```

ou ainda:

```
Tconc(N) = (Tseq / N) * (1+fc)
S  = N / (1+fc)
Ef = 1 / (1+fc)
```

O overhead fc é função tanto do tamanho do problema por nó quanto da relação entre t_{comm} e t_{calc}, tempos médios básicos da comunicação de uma palavra e de uma computação de ponto flutuante na arquitetura.

O termo do tamanho do problema por nó é quem define a condição de granularidade mínima para uma classe de problemas em uma máquina, reduzindo o overhead e permitindo um speedup linear com N a uma eficiência alta (S >= 0.8N).

Ao projetista de hardware cabe otimizar o termo dependente da arquitetura, que é a relação t_{comm}/t_{calc}.

Nos cubos da Caltech e JPL da primeira geração este valor correspondia aproximadamente a 2, [Fox88,pg31e51]. Em [Fox88,pg444], diz-se que, se comunicação e processamento fossem concorrentes, uma relação t_{comm}/t_{calc} <= 15 ainda seria aceitável, em termos de eficiência.

2.5.2 As diferenças das gerações

Muito esforço tem sido dispendido em melhorar as redes de interconexão, [Reed87]. As redes da primeira geração usavam topologia hipercúbica e store-and-forward.

O esquema de store-and-forward provocava uma degradação muito grande quando a comunicação se dá entre nós não vizinhos, e sua latência é proporcional ao número de nós intermediários da rede ("hops") por onde as mensagens passam.

As máquinas mais modernas da segunda geração se utilizam de um mecanismo chamado "wormhole routing" que reduz muito a latência da comunicação entre nós não vizinhos, quando a rede está sob pouca carga, e permite que a máquina pareça estar totalmente interligada.

Estudos apresentados em [Reed87,cap3] e [Dally90] indicam que nem sempre a topologia de um hipercubo binário é ótima, e sugerem uma variante chamada K-ary n-cube, onde o grau de cada nó (número de links) é diminuido, mas a banda passante de cada um (número de fios) é aumentada.

Estes estudos se baseiam na premissa de que o custo de uma rede depende não do número de chaves usadas, mas da densidade de ligações necessárias para construir a rede, e analisam o desempenho das redes sob uma mesma "bissection width", que é a densidade de ligações que conecta duas metades do multicomputador.

Pelo menos uma máquina comercial já emprega esta tecnologia: a AMETEK série 2010, [Seitz88] e [Seitz90].

Em [Zhang91], "System Effects of Interprocessor Communication Latency in Multicomputers", apresenta-se estudo interessante sobre o desempenho dos sistemas de

comunicação em multicomputadores comerciais.

CAPÍTULO 3 :
UM NÓ DE PROCESSAMENTO

3) UM NÓ DE PROCESSAMENTO

3.1 Otimização do nó da máquina paralela

Na construção de um nó de uma máquina paralela deve-se procurar ter desempenho elevado, mas, também, tentar maximizar a relação custo/benefício. Uma maneira empírica de tentar manter esta relação a melhor possível está em utilizar as técnicas e tecnologias mais eficientes disponíveis, fazendo exceção apenas quando a aplicação localizada de uma tecnologia menos eficiente traga um ganho líquido para o sistema como um todo que a justifique.

O grau de replicação prático máximo da máquina é limitado pelo consumo de energia e pelo número de chips por exigir fontes, refrigeração, área maior, e outros fatores geradores de custos, além de reduzirem a confiabilidade e a disponibilidade.

O ponto apropriado na curva de Replicação x Complexidade para os multicomputadores exige complexidade suficiente para uma CPU com bom desempenho, mas sugere um nó simples que use tecnologias eficientes, que minimize os gastos em área e energia com funções periféricas ou a "glue-logic".

Esta necessidade é ainda mais exacerbada na medida em que cada acréscimo à arquitetura de um nó é replicado N vezes, e que o avanço da escala de integração faz com que qualquer "glue-logic" represente um custo em área e energia nada marginal em relação às funções básicas do processador e da memória.

Quando o processador é projetado especialmente para multicomputadores, um cuidado muito grande é dado a isto, como no caso dos NCUBEs, [Hayes86] e [Colley89], e diversos "Transputer arrays". Ambos contém os elementos da rede de comunicação no chip do processador e otimizam sua interface de memória para requerer pouco, ou nenhum, suporte.

Para projeto em VLSI, altera-se a métrica antiga que tratava do número de chips ativos a cada dado instante para

sua versão mais moderna que aborda a questão do percentual de área ativa de um chip.

Em resumo, deseja-se bom desempenho, mas, devido à replicação dos nós, minimizando os custos em área e energia gastas por nó. Isto pode ser conseguido pela minimização do número de chips usados, e concentrando-os nas funções principais e minimizando as periféricas.

A disponibilidade é um ponto importante. Em aposta recente, descrita em [Patterson90,pg590], entre Gordon Bell e Danny Hillis sobre o tipo de supercomputador que seria mais rápido em 1995, se com N<=100 ou N>=1000, usa-se a métrica de MFLOPs sustentados ao longo de um mês, exatamente para modelar a disponibilidade. Em [Siewiorek82,pg10] diz-se que o CRAY-1 tinha um MTTF ("Mean Time To Fault") de apenas 4 horas, com um MTTR ("Mean Time to Repair") de apenas 25 minutos, devido à existência de uma equipe treinada para dar assistência, no local, e mesmo assim isto representa 10% da capacidade computacional da máquina.

Um outro ponto importante é a flexibilidade, e a generalidade que podem alargar sua aplicabilidade e a base de usuários de uma máquina. O multicomputador já é uma máquina MIMD, e seu principal fator de mérito: a relação t_{comm}/t_{calc} deve ser otimizada. Além disso, a quantidade de memória em um nó também pode ser uma fator limitante importante, [Reed87,pg363].

3.2 O estado atual e as perspectivas de integração VLSI

O avanço da escala de integração dos componentes VLSI provoca mudanças não só quantitativas, mas também qualitativas notáveis, pois alteram o particionamento das funções entre os chips, o número deles, as interfaces entre eles, e seu desempenho relativo. Estas mudanças devem ser consideradas na definição do nó ótimo da máquina.

3.2.1 CPUs

Diz-se que os ganhos com a escala de integração em VLSI derivam em parte da miniaturização, e parte de "designer's wizardry", [Weste85]. O avanço da tecnologia permite a redução das dimensões, e esta a redução das capacitâncias e atrasos, e o aumento do número de dispositivos por chip.

O fator referente ao "designer's wizardry" advém da exploração que os projetistas fazem da maior liberdade que o aumento da escala de integração oferece. Aproveitando esta liberdade é possível eliminar atrasos inter-chip, interfaces, níveis de hierarquia, e usar a melhor opção de implementação das funções na tecnologia de microeletrônica usada.

Em meados da década de 70, os microprocessadores eram unidades simples com palavras de 8-bits e freqüências de 1MHz, como o 6502 dos AppleII, ou o 8080 dos micros com CPM, enquanto que o CRAY-1, contemporâneo, a 80MHz realizava operações de ponto flutuante de 64-bits à taxa de 160MFlops.

A evolução dos microprocessadores acompanhou a da microeletrônica. As técnicas arquiteturais desenvolvidas anteriormente para as classes de computadores maiores foram sendo absorvidas, logo que seu uso se tornava factível, a cada novo limiar da tecnologia. Como um exemplo, segundo [Patterson90,pg16], quando a tecnologia MOS chegou a um ponto em que era possível colocar entre 25000 e 50000 transistores em um único chip, viabilizou-se um microprocessador de 32-bits e um ganho dramático na relação custo/desempenho.

O aumento do número de transistores tem sido usado para diminuir as latências das operações individuais e aumentar o número de funções oferecidas. O ponto importante a se notar é o esgotamento do ciclo de mover funcionalidades, tradicionalmente encontradas nas CPUs dos mainframes, para os microprocessadores.

Hoje, diversos microprocessadores dispõem de unidades

de ponto flutuante como o IMS-T800, Intel i486, e Motorola 68040, e trabalham entre 20 e 50MHz. Como os tempos de acesso das memórias dinâmicas não melhoraram na mesma proporção, muitos processadores se utilizam de caches para limitar o impacto em seu desempenho. Alguns como o i486, o i860, e o MC68040 têm caches internas, pois, mesmo com o uso de memória estática externa, a interface do chip já cria um gargalo e provoca uma limitação do desempenho.

Além destes, surgem CPUs como o i860, a 40MHZ, com a latência de uma operação escalar de ponto flutuante de 3 ticks do clock, e throughput pipelined de até duas operações por clock, ou 80MFlops. Este desempenho é da ordem de metade do de um CRAY-1 e 1/5 de uma CPU do CRAY-YMP, a um custo pelo menos duas ordens de grandeza mais baixo.

Numa CPU como o i860 já se usam técnicas superescalares e de pipelining, o que requer diversos operandos por clock. Sem auxílio de algum recurso arquitetural não seria possível suprir esta demanda. Caches de código e de dados são implementadas dentro do chip, ocupando mais de 30 % de sua área, [Spectrum89], para tentar superar esta limitação.

No modelo vetorial de computação, por exemplo, a banda passante da memória é um fator muito limitante. Estudos como em [Dongarra88] mostram que reorganizando as computações pode-se usar a hierarquia de memória para melhorar a relação entre as quantidades de cálculos e de referências aproveitando da localidade nestas. Novos pacotes para cálculo numérico linear são desenvolvidos sobre este conceito, como o LAPACK, [Bischof88], e as extensões do BLAS para cálculo matricial.

No i860, por exemplo, sugere-se em [Kohn89] usar a cache como registrador vetorial, através do uso de instruções de "load" distintas a normal que usa sempre a cache ou a pipelined que evita seu uso, para minimizar os acessos à memória externa ao chip. Mesmo assim, o desempenho fica limitado pela interface do chip com o sistema de memória (I/O do chip), e o desempenho deste.

Num futuro previsível esta tendência tende a se agravar, pois a velocidade das CPUs pode continuar aumentando junto com a escala de integração, mas seu desempenho sustentado fica inevitavelmente limitado pela capacidade de I/O do chip, mesmo com auxilio de caches, ou registradores vetoriais e do software, para problemas de porte.

Em resumo pode-se dizer que as CPUs dos micro-processadores já oferecem desempenho de pico na mesma ordem de grandeza das CPUs dos supercumputadores, mas devido as limitações de I/O do chip (e dos sistemas de memória) seu desempenho sustentado fica bastante reduzido.

Como o gargalo destes processadores está no I/O do chip para acesso à memória, e não pode ser retirado pelo acréscimo de transistores, a interface deve ser otimizada.

De fato, a otimização crescente das interfaces tem sido uma tendência observável nestes dispositivos. Tomando a linha de microprocessadores da Intel, por exemplo:

- o i286 adianta o endereço do próximo ciclo à memória ("address pipelining");

- o i386 faz o mesmo quando requisitado pelo sistema de memória;

- o i486 tem ciclos de "burst" onde se pode presumir a seqüência de endereços e encurta-se o tamanho dos ciclos, muito útil em conjunção com sua cache interna;

- o i860 tem três níveis de pipelining na memória e um sinal de "mesma página" (NeNe#) para otimizar o acesso à memória dinâmica.

3.2.2 Memória

As memórias semicondutoras podem ser estáticas (SRAMs) ou dinâmicas (DRAMs), que diferem em custo, densidade, e tempo de acesso devido ao tipo de célula que empregam.

As memórias estáticas utilizam circuitos com realimentação positiva em cada célula, o que mantém o seu estado, enquanto que nas dinâmicas o estado é armazenado

como uma carga em capacitores. Devido a inevitáveis correntes de fuga dos capacitores, torna-se necessário reestabelecer seu valor periodicamente, o que é conhecido como "refresh".

Segundo [Patterson90,pg426], para memórias usando tecnologias comparáveis, a capacidade das DRAMs é aproximadamente 16 vezes maior que a das SRAMs, enquanto o tempo de ciclo das SRAMs é de 8 a 16 vezes mais rápido que o das DRAMs.

Dentro do critério de procurar a tecnologia mais eficiente em custo, privilegia-se o uso de DRAM no nó da máquina paralela, pois seu custo por bit é muito menor. Procura-se, também, estudar as formas de otimizar o uso das DRAMs e minimizar os efeitos do ciclo lento.

As memórias dinâmicas, por questões de custo do encapsulamento, multiplexam as linhas de endereço reduzindo seu número à metade. A parte alta do endereço é fornecida primeiro, durante o "Row Address Strobe", ou RAS, e a parte baixa durante o "Column Address Stobe", ou CAS. Estes nomes têm ligação com a organização interna dos chips em um ou mais blocos matriciais retangulares de células.

Para as DRAMs, convém distinguir duas métricas para a latência da memória. A primeira é o tempo de acesso, que é o tempo gasto entre o pedido de leitura de uma palavra e sua chegada. A segunda é o tempo de ciclo, ou o tempo entre dois pedidos distintos à memória. Nas SRAMs o tempo de acesso é igual ao de ciclo, mas não nas DRAMs, pois certas linhas internas do chip tem de ser precarregadas antes de uma nova operação dos circuitos, o que é conhecido como tempo de "precharge".

As DRAMs são commodities de mercado, fabricadas através de processos especialmente desenvolvidos e ajustados para este fim e são consideradas "technology drivers".

A densidade das DRAMs tem quadruplicado a cada três anos, desde a introdução das DRAM de 1-Kbit no início da década de 1970 até a geração de 16-Mbit no início da década

de 1990. O custo tem caído de acordo, chegando sempre a um mesmo valor monetário por chip, ou seja, com uma redução do custo por bit equivalente ao aumento da densidade.

A velocidade das DRAMs não tem aumentado na mesma proporção que a dos processadores, [Patterson90,427]. Em [Itoh90], onde é apresentado um pequeno retrospecto e também as tendências de evolução das DRAMs, não se prevê melhorias dramáticas, mesmo com o uso de BiCMOS.

Como mencionado anteriormente a memória é composta de blocos retangulares de células. Cada célula contém o capacitor que armazena a carga C_{bit}, e um transistor de passagem. Este transistor funciona como uma chave e permite acessar C_{bit}, quando acionado. Dois conjuntos de linhas se cruzam perpendicularmente nos blocos: as linhas de "word" e as de "bit" ou "data".

O endereço fornecido no RAS é usado para selecionar uma linha ("page") do bloco ativando uma linha de "word". As linhas de "word" são ligadas aos "gates" dos transistores de passagem e selecionam todas as células que são ativadas num ciclo, abrindo as chaves e conectando os C_{bit} às suas respectivas linhas de "data", que são previamente carregadas.

Quando isto acontece, há uma divisão de carga entre C_{bit} e C_d, a capacitância das linhas de "data", que altera a tensão desta linha e chega-se a um novo equilíbrio. Esta alteração permite, aos sensores diferenciais ligados às linhas de "data", ler e restaurar a carga de C_{bit}, ou seja, o conteúdo de todas as colunas do bloco de memória, que são acessadas em paralelo. A partir do endereço fornecido em CAS, seleciona-se uma das colunas acessadas e refrescadas em paralelo para ser lida ou escrita. Este paralelismo será explorado na diminuição dos tempos de acesso.

3.2.2.1 Localidade e otimização

Existem diversas otimizações baseadas na exploração do paralelismo no acesso às colunas da memória. Os mais simples

são os que, através de alguns acréscimos ao protocolo de sinais de controle das DRAMs, permitem acessos mais curtos para refêrencias dentro de uma mesma linha ou página, especificada no RAS. Estes acessos podem ser realizados sem perder os tempos de "precharge" das linhas de "data", de RAS, da decodificação da linha, da abertura das chaves, da troca de cargas entre C_{bit} e C_d, e no sensoreamento e amplificação e restauração dos estados das células das colunas.

Ao modo de acesso normal chama-se de acesso de linha, e a um acesso curto dentro da mesma página de acesso de coluna. Existem três modos bastante difundidos de acessos de coluna:

Nibble mode -Para cada acesso de linha, a DRAM fornece três bits extras de uma seqüência de posições;

-Page mode -O conjunto sensor/amplificador/buffer age como uma SRAM, que em se alterando o endereço de coluna, pode-se acessar aleatoriamente qualquer posição dentro de uma mesma página até que seja necessário realizar um ciclo de refresh;

-Static Column -É um modo semelhante ao page mode, em que não é necessario ciclar o sinal de CAS quando se muda o endereço da coluna.

Segundo [Patterson90,pag431], a partir das DRAMs de 1Mbit a maioria dos "dies", chips não encapsulados, suportam as três versões do protocolo, e a otimização só é definida no momento do encapsulamento pela escolha dos "pads" conectados.

Os tempos de acesso por linha apresentam uma variação da ordem de 20% entre as DRAMs mais rápidas e as mais lentas. Os tempos de acesso por colunas representavam cerca de 50% do tempo de acesso por linha mais rápido até a geração de 256Kbit, que empregava tecnologia NMOS. Com a mudança para CMOS, a partir da geração 1Mbit, os tempos dos acessos por colunas passaram a representar apenas cerca de

25%, (1/4), dos tempos por linha.

A relação entre os ciclos curtos, ou Ciclos Near, dentro da mesma página, e os ciclos completos, ou Ciclos Far, apartir do RAS, mantém relação equivalente ao dos tempos de acesso: 1/2 até a geração de 256Kbit e 1/4 a partir da de 1Mbit. Em [Patterson90,pg432] é apresentada uma tabela reproduzida aqui.

Geração da DRAM	Acesso de Linha lento	rápido	Acesso Coluna	Ciclo Far	Ciclo Near
64K	180ns	150ns	75ns	250ns	150ns
256K	150ns	120ns	50ns	220ns	100ns
1M	120ns	100ns	25ns	190ns	50ns
4M	100ns	80ns	20ns	165ns	40ns
16M	85ns	65ns	15ns	140ns	30ns

Tabela 3.1

Existem outras otimizações baseadas no paralelismo do acesso por colunas, a base de muitas idéias de incrementar chips de RAM com alguma função extra que através do paralelismo possa dar capacidade computacional elevada a custo baixo, como em [Optic89], por exemplo. As vezes estas RAMs incrementadas são chamadas de "SMART-RAMs".

Um dispositivo já estabelecido comercialmente é a VRAM, ou vídeo RAM, e sua aplicação principal, mas não a única é para "frame buffers". São dispositivos com dois "ports", um aleatório normal e outro serial otimizado. Basicamente toda uma linha é transferida para um registrador em um único ciclo do "port" aleatório. No outro é feito acesso serial muito rápido a este registrador, que pode fornecer os dados para o "raster" de vídeo por exemplo, sem conflito com o "port" mormal. Em [Nicoud88] é apresentado bastante material sobre "Estruturas e Aplicações de VRAMs".

Existem, também, dispositivos semelhantes com três "ports", um aleatório e dois serias independentes, para suportar I/O de disco por exemplo, [Micron90] e [ED90].

Existem ainda novos protocolos e organizações de chips sendo propostos. Muitas incluem deteçào e correção de error internamente aos chips, como em [Arimoto90] e [Asakura90].

Um protocolo interessante é o "Nibbled-Page", apresentado em [Numata89]. Este protocolo mescla uma extensão do "nibble mode" para 8 bits com o "page mode" para aproveitar as vantagens de ambos. O "nibble mode" permite o acesso mais rápido pois o chip pode "presumir" o próximo endereço, e o "page mode" permite acessos otimizados enquanto se permanecer na mesma página.

Uma arquitetura nova interessante é a "Cache DRAM", apresentada em [Asakura90] e [Hidaka90]. Nesta arquitetura parte do chip da DRAM é usada para implementar memória estática que funcionará como uma cache. O endereçamento não é multiplexado, e os acessos podem ser para a cache, para a DRAM, com ou sem transferência de uma para a outra. A transferência de todo um bloco da cache é feita simultaneamente usando o paralelismo de coluna citado.

Existem propostas para organização hierárquica dentro do chip de memória, como uma árvore por exemplo, em [Jarwala88], com o objetivo de melhorar, o desempenho e a testabilidade dos componentes.

Talvez, no futuro, se possa refinar ainda mais as interfaces dos chips para oferecer múltiplos módulos independentes dentro de um único chip, com pipelining de acesso, para permitir sistemas de memória com poucos chips, e grande banda passante.

A tendência de alterar as interfaces de memórias para acoplá-las melhor aos processadores já é perceptível como em [CD90,pg81].

3.2.3 Comunicação

Não é o objetivo deste trabalho dissecar os sistemas de comunicações para multicomputadores, área que tem sido objeto de muita pesquisa internacionalmente. Em [Reed87] existe uma boa fundamentação para o assunto. Em [Seitz90,pg140] um trabalho abrangente e atualizado sobre o assunto: "Network and Processor Architecture for Message Driven Computers", por William Dally do MIT.

Do ponto de vista do trabalho aqui apresentado interessam os aspectos associados ao sistema de memória. Nas primeiras gerações que usavam store&forward, havia um grande desperdício de banda passante no nó, que, as vezes, obrigava às máquinas a ter módulos de memória específicos para a comunicaçào já que cada palavra que fluia pelos links requeria um ciclo de memória, pelo menos.

Com o uso de "wormhole routing" e uma espécie de circuito chaveado, isto não mais acontece. Os acessos à memória do nó são apenas para as comunicações que o tem como origem ou destino.

O que é relevante observar é que com todos os avanços, o fator limitante da banda passante das comunicações é a banda passante da memória. Charles Seitz da Caltech afirma, em [Seitz90,pg31], que já se poderia dispor de uma banda passante de comunicações maior, a um custo modesto, mas como a memória já limita a banda passante não há incentivo para realizá-la.

O caso real de que ele trata é a máquina que tinha o sistema de comunicação mais moderno de então, o AMETEK 2010. Esta máquina implementa um mesh, K-ary 2-cube, com "mesh-routing chips" (MRCs), um para cada nó. Estes têm capacidade de comunicação bruta nos nós de 225Mb/s, ou 28MB/s. Cada nó se conecta a estes MRCs, que ficam no "backplane", por dois canais, de entrada e saída, cada um com 28MB/s.

3.2.4 Opções de Nó

O particionamento das funções pelos chips ainda deve separar a memória do processador, pois estas requerem processos muito específicos e otimizados, e quando se junta os dois há grandes perdas na relação custo/desempenho.

Eventualmente, com o avanço da escala de integração, digamos na geração de 1ou 4 Gbit, mesmo em processador complexo, digamos 16 vezes mais que um i860 com 16Mega transistores, será apenas uma pequena fração, um "cantinho", do chip, permitindo a implementação de máquinas MIMD complexas como são hoje as "smart-rams".

O elemento de comunicações pode ser integrado conjuntamente com o processador ou não, apresentando um compromisso entre o acoplamento das comunicações e o processamento e a banda passante de comunicações.

Quando juntos, as latências para controle e reconhecimento das comunicações diminuem, mas há a disputa pelo pinos do chip para o barramento de memória e para os links de comunicação, que é um recurso escasso e de pouca elasticidade.

3.3 A exposição do problema

Tanto o processamento quanto a comunicação estão limitados pelo acesso à memória. Queremos utilizar DRAMs para baixar o custo, mas para que isto seja prático, é necessário otimizar seu uso. Técnicas como "interleave" de memória não são práticas ou factíveis devido ao reduzido número de chips e bancos necessários, e restrições ao tamanho do nó.

Neste trabalho estuda-se a degradação imposta pelo DMA para realizar comunicação sobre o desempenho do processamento no nó. Esta pode ser muito grande por causa da interferência na localidade do padrão de acessos conjugado do DMA e da interface otimizada dos processadores.

Deseja-se também ser capaz de suportar uma banda passante de comunicações compatível com a capacidade de

processamento, o que é difícil com um único banco de DRAMs e acessos de linha normais.

Pode-se usar acessos em "bursts" (rajadas) para comunicação com o objetivo de aproveitar os acessos por coluna nas DRAMs para comunicaçào também, aumentando a banda passante de memória disponível, e assim, a taxa máxima de comunicações suportável, e diminuindo a quantidade do tempo da memória usada, liberando mais para o processador.Além disso, o efeito da perda de localidade no padrão de acessos conjugado é substancialmente reduzido.

A Figura 3.1 mostra uma projeção para o ganho de eficiência nos acessos à memória para comunicação para cada tamanho de burst (Tamburst) em função da relação entre os tempos de um Ciclo Far (Tf) e um Ciclo Near (Tn), onde o ganho de eficiência é dado por :

(Tamburst * Tf) / (Tf + (Tamburst-1) * Tn).

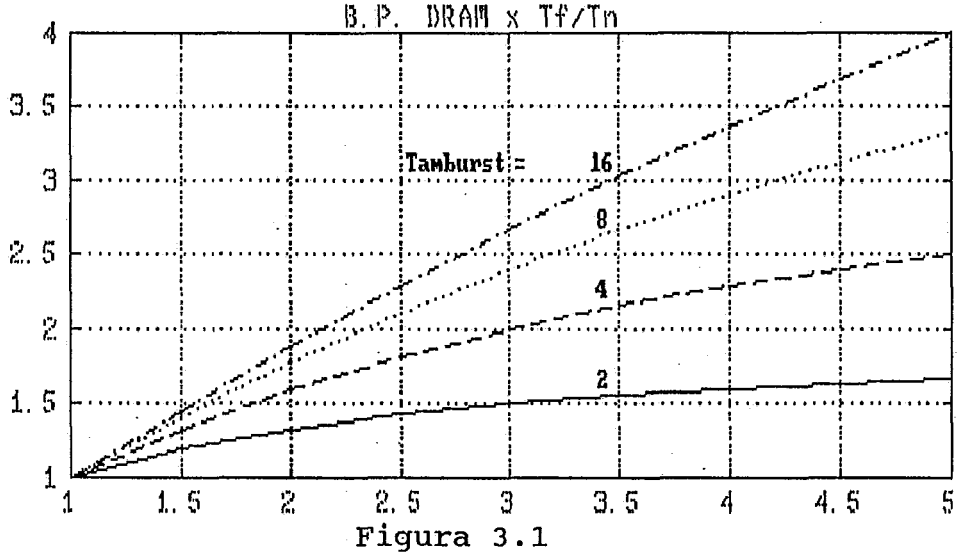


Figura 3.1

Cabe observar que não se trata o cruzamento de página em meio a um burst. Este cruzamento pode ser evitado pela alocação dos buffers para as mensagens, e mesmo que não o fosse, seu efeito seria pequeno dado o tamanho das páginas (512 words em um chip de 256Kwords).

CAPÍTULO 4 :
O AMBIENTE EXPERIMENTAL

4) O AMBIENTE EXPERIMENTAL

4.1 A idéia do que se busca medir

A contribuição maior deste trabalho reside no estudo da questão dos processadores modernos, suas unidades de ponto flutuante pipelined, sua capacidade de pico muito superior à sustentada; e a grande demanda por Banda Passante de memória para manter o desempenho sustentado. A isto soma-se que nos multicomputadores a comunicação inter-nó é um dos principais fatores de mérito e deve ser rápida e eficiente. Esta comunicação envolve a transferência de um objeto de dados da memória de um nó para a de outro, demandando B.P. de memória.

A taxa de comunicação sustentada estará limitada pela B.P. máxima da memória enquanto que a capacidade de processar concorrentemente à comunicação será fortemente afetada pelo conflito no uso da memória acima descrito.

Abstraindo-se da rede e modelando a comunicação inter-nó como um DMA sobre buffers, pode-se por simulação chegar a resultados relativamente precisos no que diz respeito a B.P. de memória e taxa de comunicações suportável. Isto foi feito e será relatado no próximo capítulo.

Dois trabalhos se impõem como experimentais devido às dificuldades em modelar e validar o resultado de hipotéticas simulações :

1 - Avaliar a degradação causada ao desempenho do processador pelo DMA da comunicação;

2 - Avaliar a melhoria possível pelo uso de um protocolo de arbitração do uso da memória em bursts de ciclos.

Este trabalho experimental foi realizado usando uma placa protótipo com a CPU i860 da Intel desenvolvida na COPPE/UFRJ e uma pequena placa de extensão desenvolvida pelo autor para viabilizar a experiência, que são descritas a seguir juntamente com o software e temas associados.

4.2 A Placa do i860 da COPPE

A Placa da COPPE/UFRJ faz parte do projeto do NCP-I e é descrita em detalhes em [CITRO91]. Ela é baseada no microprocessador de 64 bits i860 da Intel e projetada logicamente para uma freqüência nominal de 33.3MHz, porém, sendo um protótipo em "wire-wrap", é usada com metade do clock nominal, ou seja, a 16.6MHz. O dempenho do sistema a 33.3MHz dobraria em relação ao usado, mas para 40MHz teriam de ser usados dispositivos mais rápidos no projeto ou mudar os números de clocks para os acessos à memória.

Esta placa pôde ser usada independentemente ligada a um IBM-PC através de uma interface serial padrão RS-232C e um programa emulador de terminais. Para usá-la carrega-se programas no formato de saída do montador para i860 da COPPE (.HEX). Este formato é suportado pela ROM da placa que aceita a carga do código e depois lhe passa o controle.

A placa dispõe de muitos recursos, entre eles : interface para o módulo do Transputer T800, [Inmos88], usado no NCP-I; interface para um barramento VME (como slave); memória estática dual-ported real accessível pelo i860 ou pelo módulo do T800; hardware para interrupção entre os módulos; etc... Entretanto, somente alguns de seus recursos foram utilizados para realizar os experimentos propostos e serão aqui brevemente descritos, a saber : o i860; os sistemas de memória dinâmica (DRAM); e de temporização.

4.2.1 O i860

O processador i860 da Intel é descrito em suas diversas facetas no "Data Sheet" [Intel89a], no "Hardware Design Guide" [Intel89b], no "Programmers Reference Manual" [Intel89c], e no "Assembler and Linker Reference Manual" [Intel89d]. Pode-se referenciar ainda [Spectrum89], [Micro89], [Dobb's90] e [Prog91] para a história de seu desenvolvimento, suas características, funcionalidade, desempenho esperado e programação.

O chip tem uma unidade RISC inteira e lógica ("core"),

unidades de ponto flutuante para soma, multiplicação e gráficos que podem ser usadas em modo escalar (non-pipelined) e vetorial (pipelined por software). Também é capaz de executar operações no "core" e nas unidades de ponto flutuante simultaneamente num modo especial chamado "Dual Instruction Mode". Além disso pode usar duas unidades de ponto flutuante simultaneamente nas chamadas "Dual Operation Instructions" perfazendo três operações por ciclo; sem contar no recurso do endereçamento com auto-incremento. Quando não há contenção nem "miss" nas caches, as intrucões do "core" e as "pipelined" de ponto flutuante executam em um ciclo e as escalares em três ciclos.

Para dar suporte a este "throughput", o chip utiliza metade de sua área de silício para implementar as caches de código (4KB) e de dados (8KB), ambas organizadas em blocos de 32 bytes (4 words), "two-way, set-associative" sendo a de dados do tipo "write-back". Tem barramentos internos de 64 bits para código, e 128 bits para dados e barramento externo de 64 bits.

A interface do barramento externo é otimizada para o uso de DRAMs. Permite o pipelining da memoria em até 3 níveis (endereços sendo processados concorrentemente) e apresenta um sinal /NeNe (Next Near) gerado por um comparador interno que indica se um determinado endereço pode ser acessado dentro da mesma linha da mémoria dinâmica que o último. Assim, pela combinação de pipelinig de memória com Ciclos Near na DRAM é possível ter um tempo efetivo de acesso próximo ao das SRAMs a uma fração do custo e manter o desempenho o mais próximo possível do máximo sustentável, limitado em geral pelo acesso a uma word em no mínimo dois ciclos pelas limitações da interface do chip.

4.2.2 Temporização

A temporização da placa é realizada através de um timer programável i8253 da Intel, [Intel87b,pg 2-14]. Este chip contém três timers que estão alocados para gerar: a temporização dos pedidos de refresh da memória dinâmica (Timer 0); a interrupção de tempo real da CPU (Timer 1); e a base de tempo para a comunicação da placa com o ambiente externo através de uma i8251A (Timer 2). Todos recebem um sinal de "CLK" de 2 MHz, e têm o sinal "GATE" constantemente habilitado.

4.2.3 O sistema de memória dinâmica (DRAM)

O sistema de memória dinâmica da placa do i860 forma uma memória three-ported accessível pelo i860, T800 e pelo VME. Nos experimentos porém, apenas o acesso do i860 é usado. Para os acessos do i860 estão implementados os Ciclos Near capazes de suportar o pipelining de memória definido na interface do i860.

O sistema é formado por : três conjuntos de oito buffers 74AS646; um banco de 2MB de memória em 16 chips de 256Kx4 (514256); e um conjunto de PALs de controle.

4.2.3.1 Buffers/pipelining

Os buffers 74AS646 são bidirecionais (transceivers) e registrados e têm por função isolar eletricamente os três "ports" de acesso e suportar o modelo de pipelining de memória usado pelo i860.

4.2.3.2 A máquina de estados na PAL (DMSTATE)

O sistema de memória dinâmica é controlado por uma PAL22V10 central que define uma máquina de estados: DMSTATE. Esta máquina de estados dá prioridade absoluta ao refresh da memória, realizado sempre que o sinal /RFSHREQ (ou /RFSH) é mantido ativo. Implementa também os Ciclos Near quando apropriado usando o sinal /NeNe (Next Near) do i860. Este sinal indica um acesso com mesmo valor de linha (ROW) e

permite omitir a seqüência de /RAS e evitar a operação de pré-carga da memória dinâmica que constitui o ciclo completo, ciclando apenas o /CAS para realizar os acessos em "Page Mode" num tempo menor.

4.2.3.3 Refresh

O pedido de refresh é gerado pelo timer 0 da i8253A, sincronizado por dois flip-flops (7474) e injetado na PAL22V10 que implementa DMSTATE (a máquina de estados de controle da memória dinâmica). Este pedido tem precedência sobre qualquer outra operação na memória dinâmica. Os acessos eventualmente pendentes (já iniciados) são completados e o ciclo de refresh é prontamente iniciado.

O acesso seguinte após um ciclo de refresh é sempre longo, realizando toda a seqüência de /RAS e /CAS.

4.3 O Software

4.3.1 O compilador e o ambiente de desenvolvimento

O compilador usado foi o HighC da MetaWare, [HighC90], um compilador ANSI C para UNIX da AT&T para i860 que roda em IBM-PC com CPUs 80386 ou compatíveis e bastante memória, o único disponível.

Este compilador tem, além do modo normal, dois níveis de otimização (-O e -O -Sched). No primeiro nível faz melhoras genéricas no código, mas no segundo (-Sched) o compilador tenta rearrumar a ordem das instruções de maneira a coreografar a execução no processador e aproveitar cada lacuna de tempo disponível com trabalho útil.

No modo em que foi usado, o compilador gera arquivos em assembly (.S) que são montados e ligados pelos programas da própria Intel, [Intel89d], fornecidos com o compilador.

4.3.2 Biblioteca e rotinas de suporte

A biblioteca original do compilador para ambiente UNIX foi editada de maneira a extrair um módulo apenas com as funções matemáticas (sin,cos,exp,log,...) e com "string I/O" (sprintf e sscanf). O arrazoado por trás disso é de que estas rotinas dependem apenas da CPU, não fazem chamadas ao sistema, não dependendo do ambiente UNIX, inexistente.

Isso visava de imediato dar suporte ao software científico que se pretendia rodar nesta placa e também ao seu I/O. A mais longo prazo, permitiria que se criasse um conjunto de rotinas para fazer, se não sistema, pelo menos um ambiente operacional a exemplo do descrito em [Fox88].

A estratégia funcionou no que diz respeito às funções matemáticas usadas por exemplo no cálculo dos coeficientes das FFTs; não tendo dado resultado no entanto para as rotinas de "string I/O". Supõe-se que os "bugs" da versão (SX221) dos primeiros "engineering samples" do chip i860 não sejam tratados ou suportados pela biblioteca como descrito em [HighC90]. Infelizmente não foi possível ter uma versão comercial ("bug-free") do chip em tempo hábil para este trabalho.

Foi criada uma série de rotinas de suporte para:

- Converter números inteiros e reais em strings;
- Suportar a comunicação com o terminal;
- Temporizar a execução dos programas;
- Controlar o hardware:
  - mascarar e desmascarar interrupções;
  - dar flush, ligar e desligar a cache;
  - configurar parâmetros dos testes.

4.3.3 Conversor de formatos (out2hex)

O arquivo gerado pelo ligador (linker,.OUT) é dado num formato definido pela AT&T (COFF). Este arquivo é transformado para o formato (.HEX) da COPPE/UFRJ antes de poder ser carregado na placa do i860. Esta conversão é feita pelo programa (Out2Hex) desenvolvido pelo autor.

4.3.4 Programas de teste

A idéia original para avaliar a degradação causada pelo conflito pela memória previa um "suite" de testes variado, baseado num conjunto de rotinas e programas com padrões de acessos diferentes para tornar a pesquisa mais rica, robusta e representativa.

Para tornar isto factível, com uma carga de trabalho realista, baseada em aplicações númericas consagradas, dotou-se o sistema da capacidade de rodar programas em C.

Na segunda etapa, foi usado um conjuto de rotinas para transformadas rápidas de Fourier, FFTs, binárias, complexas, baseadas em [Oppe75,cap6]. As rotinas serão brevemente descritas adiante.

Ficou demostrado na segunda etapa que o dempenho das rotinas compiladas estava muito aquém da capacidade da CPU. Isto ocorre por ineficiência do compilador, que não usa a maioria das potencialidades do i860, em especial:

- as instruções de ponto flutuante em modo pipeline que permitem o "throughput" de uma por clock;
- o "dual instruction mode" que permite a uma instrução escalar do "core" do i860 e uma de ponto flutuante serem executadas em paralelo;
- as "dual operation instructions" que permitem o uso das unidades de adição e soma de ponto flutuante em paralelo.

Para a terceira etapa, optou-se por usar rotinas em assembly que fizessem uso de tais recursos e representassem um exemplo mais realista para o código da aplicação. Por simplicidade e conveniência adotou-se a multiplicação de matrizes como aplicação, para este desenvolvimento e depuração em assembly. Tal desenvolvimento baseou-se em exemplos da próría Intel, [Intel89c,cap9], que forneceu as rotinas básicas de precisão simples.

Uma breve descrição das rotinas e suas características é apresentada adiante.

4.3.4.1 FFTs - Rotinas da segunda etapa

O conjunto de rotinas compreende as transformadas de Fourier binárias de seqüências complexas, de precisão dupla, por decimação no tempo (_dt) e na freqüência (_df), implementadas em três diferentes versões (Normal, ww, S), diretas e inversas (fft, ifft); e um conjunto de rotinas acessórias (bininv, gera_ww, preparainv).

A seguir, apresenta-se uma pequena descrição das rotinas usadas.

As rotinas acessórias :

bininv -realiza o "binary reversal" requerido pelo algoritmo das FFTs binárias, trocando pares de pontos de posição no vetor. Tem complexidade O(N).

gera_ww -cria uma tabela de coeficientes para o cálculo com ffts (ww e S). A partir de um ponto inicial definido e um passo calculado com uma função trigonométrica gera-se cada um dos coeficientes recursivamente, ou seja, fazendo a multiplicação complexa do coeficiente anterior pelo passo. São N/2 coeficientes, logo O(N).

preparainv(/) -rotina que permite o uso do código das transformadas diretas para realizar as inversas, aproveitando uma das propriedades das FFTs. Isto é feito através da aplicação dos complexos conjugados normalizados da seqüência que se quer inverter à FFT direta. Neste caso (/) cada normalização é feita pela divisão por N. São 2N divisões e N mudanças de sinal reais.

preparainv(*) -rotina analoga a preparainv(/), só que neste caso (*) cada normalização é feita pela multiplicação do recíproco de N, previamente calculado.

As FFTs diretas, O(NlogN), têm as seguintes versões:

fft_dt -transformada direta por decimação no tempo, com cálculo dos coeficientes, "in place".

fft_df -transformada direta por decimação na

freqüência, com cálculo dos coeficientes, "in place".

fft_ww_dt -transformada direta por decimação no tempo, com uso de coeficientes tabelados, "in place".

fft_ww_df -transformada direta por decimação na freqüência, com uso de coeficientes tabelados, "in place".

fft_S_dt -transformada direta por decimação no tempo, com uso de coeficientes tabelados, seqüencial.

fft_S_df -transformada direta por decimação na freqüência, com uso de coeficientes tabelados, seqüencial.

As IFFTs, são composições de preparainv(*) com as FFTs.

4.3.4.2 Multiplicação de Matrizes (terceira etapa)

Foram usadas rotinas para multiplicação de matrizes escritas em assembly do i860 e que usam seus recursos para alto desempenho. Estes recursos incluem o "dual instruction mode", "dual operation instruction", instruções de ponto flutuante na versão "pipelined", "multi-word fetch", autoincremento das referências, etc.

As rotinas foram derivadas de exemplos de programação da própria Intel, em [Intel89c,cap9]. Lá são fornecidas rotinas de precisão simples em duas versões com diferentes estratégias de uso da cache. As rotinas de precisão dupla foram desenvolvidas com base nestes exemplos, também em duas versões de uso da cache, que são referenciadas neste texto como **dp** e **dc**.

Sendo a multiplicação das matrizes do tipo C=AxB, as estratégias diferem no uso da cache para as matrizes A e B na versão **dc**, ou somente para A na versão **dp**. Isto é feito com o uso de dois tipos de instrução de "load" do i860: fld a instrução normal que faz uso da cache, e pfld a versão e acesso "pipelined" na memória que não faz. A estratégia **dc** é melhor para problemas pequenos em que ambas as matrizes cabem na cache, enquanto que **dp** é melhor nos outros casos, pois faz a cache simular um registrador vetorial para as

linhas de A minimizando os acessos à memória externa ao chip que é o gargalo da CPU.

Usou-se também as rotinas **dummydp/dummydc** similares em quase tudo à multiplicação de matrizes. No entanto, estas rotinas usam no "loop" principal um registrador especial (f0), que não causa interlocks no pipeline por dependência de dados. Este registrador quando lido retorna zero e quando escrito não tem efeito algum. Este artifício permite averiguar o quanto a execução das rotinas é dependente dos interlocks, e o quanto da latência da memória junto com a especificação da computação.

4.4 Ciclos Near

Uma maneira de comprovar a necessidade e a eficácia dos Ciclos Near é justamente comparar os tempos de execução de um mesmo programa com e sem o uso de tais ciclos.

Uma solução engenhosa e simples de realizar tais testes consiste em dotar o hardware de um jumper que permite selecionar se o sinal que é apresentado à PAL que implementa DMSTATE como /NeNe é o sinal real gerado pelo i860 ou um sinal falso desativo, ou seja, que implica em que todos os ciclos sejam completos (/RAS e /CAS) e longos.

4.5 DMA para comunicação

Um dos objetivos principais deste trabalho é avaliar o efeito do DMA para comunicação inter-nó num multicomputador. Deseja-se entretanto fazer isto a um custo mínimo, sem ter a necessidade de construir um protótipo para isto.

Uma vez que temos apenas um nó para comunicar, e que não estamos tentando modelar nem prever o comportamento da rede, mas tão somente a capacidade de comunicação suportável pelo sistema de memória e a capacidade de processar do nó sob alguma carga de comunicação; é possível abstrair o DMA para um simples acesso à memória.

Tanto no caso de nosso estudo, onde no nó proposto a memória é "single-ported" e com um banco, quanto na placa

usada nos testes, os acessos do processador e do DMA são excludentes. Sendo assim qualquer mecanismo que roube ciclos da CPU a uma taxa determinada pode simular os efeitos do DMA para comunicação deste estudo.

Novamente uma solução simples e engenhosa é usar o mecanismo do refresh da memória dinâmica, pois este já está "programado" na DMSTATE, com prioridade maior que a do i860, e com sua taxa no tempo programável pela simples reprogramação de um timer; no caso o timer 0 da i8253.

A resolução na geração dos pedidos de refresh expressa em número de clocks da CPU é dada pela razão entre a freqüência da CPU e a do timer (16.6MHz/2MHz), ou seja, aproximadamente 8.34. Esta resolução mostrou-se insuficiente para prover curvas de boa qualidade e gerou a necessidade de um mecanismo de temporização mais preciso. Este mecanismo foi implementado na placa de extensão descrita mais adiante neste capítulo.

4.6 Arbitramento em bursts

Um outro objetivo importante proposto foi o de avaliar a possibilidade de arbitrar o barramento em busrts (rajadas) de ciclos de maneira a preservar a localidade nos acessos do DMA e da CPU melhorando a B.P. efetiva da memória e também a máxima para comunicação permitindo uma taxa de comunicações maior.

Baseado na idéia da simulação do DMA simples descrita acima, pode-se simular um burst de acessos à memória pelo mecanismo de comunicação por um "ciclo de refresh" com duração prolongada, representando o primeiro acesso (Far) e os seguintes (Near). Isto claramente requer alterações no hardware pois a máquina de estados DMSTATE não comporta este tipo de ciclo.

Seria interessante também averiguar os efeitos deste DMA com vários tamanhos de burst, e para que isto seja possível o tamanho do burst deve ser programável e não fixo pelo hardware.

4.7 A Placa de Extensão

A placa de extensão foi concebida como um apoio de hardware aos experimentos propostos. Teve como uma das diretivas de desenvolvimento o objetivo de minimizar os custos em tempo de montagem, depuração e componentes, estes pelo valor monetário e pelo consumo de energia que deveria ser fornecida pela placa do i860. Disto resultou uma montagem de 6 chips (i8254,74LS74,74LS374,PAL16L8,PAL16R8, 74LS00), exposta na figura 4.1, cujas funções são descritas a seguir.

4.7.1 Interface com a placa do i860

A interface é constituída de uma série de sinais de entrada (para a placa de extensão), um barramento bidirecional de I/O (I/OD0..I/OD7) e dois sinais de saída (/RH e OUT08254). A alimentação vem diretamente das barras de distribuição da placa do i860.

A i8254, os flip-flops (74LS74), e o latch (74LS374) são mapeados no espaço de I/O da placa do i860, sendo os flip-flops no espaço de I/O rápido e o resto no lento.

4.7.2 Temporização

A temporização com maior resolução requerida pelo refresh é realizada por um circuito de clock de 7.15MHz e uma i8254, [Intel87b,pg 2-25], que nada mais é que um "super-set" da i8253 que admite freqüências mais altas, na versão usada até 8MHz. Desta maneira, a resolução na geração dos pedidos de refresh expressa em número de clocks da CPU é de (16.6MHz/7.15MHz), ou seja, aproximadamente 2.3.

Os outros dois timers (1 e 2) são cascateados para montar um cronômetro de 32 bits em hardware. O "GATE" destes timers é controlado pelo estado de um dos flip-flops (U2A) e o "CLK" é o mesmo de 2MHz da i8253 da placa do i860.

Figura 4.1

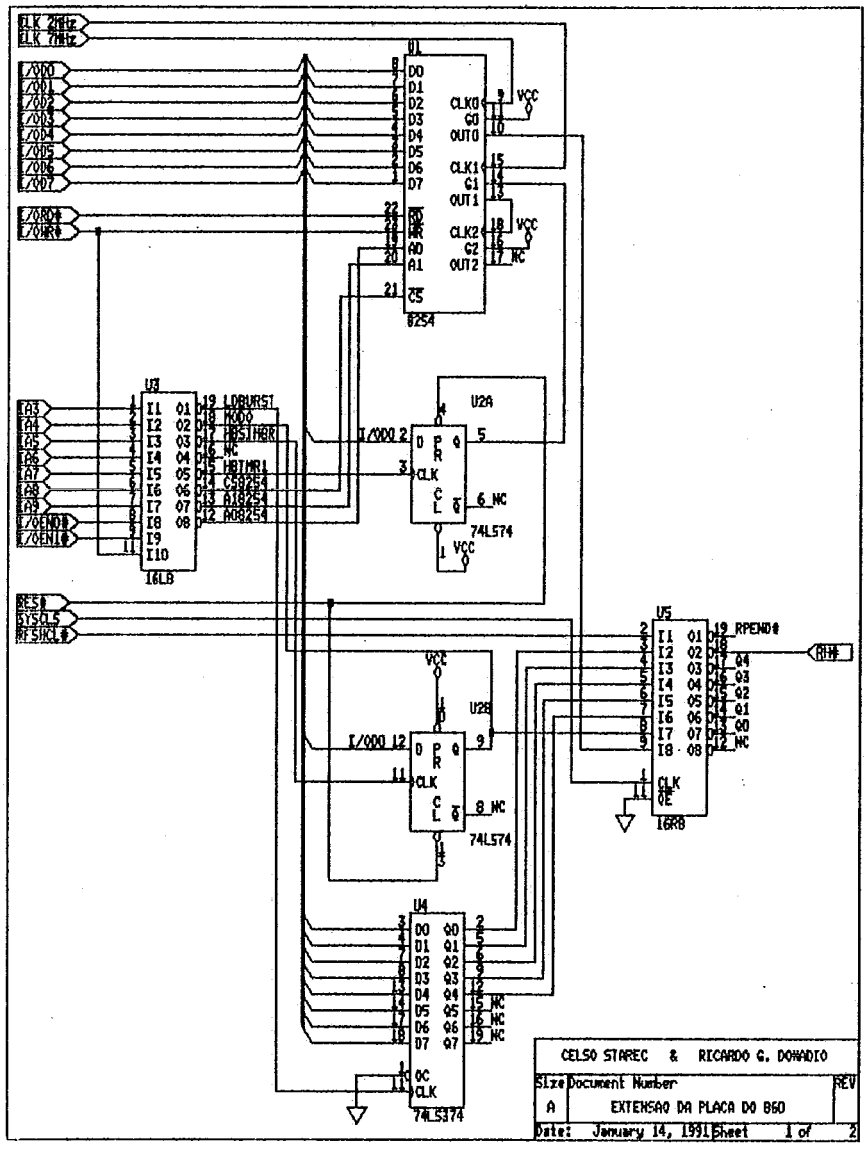
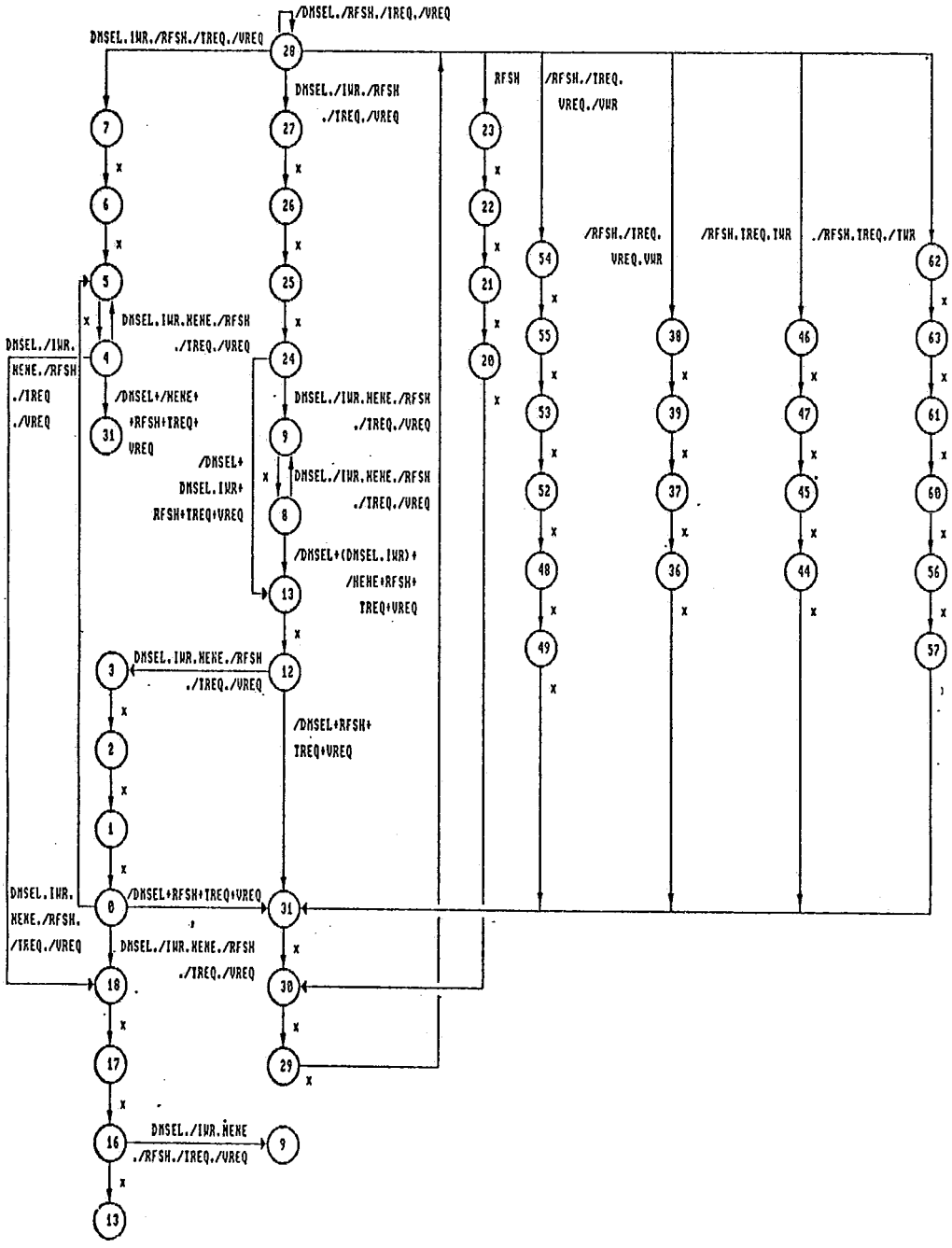

CELSO STAREC & RICARDO G. DOWADIO
EXTENSAO DA PLACA DO 860
January 14, 1991
74LS74
74LS374
8254
16LB
16R8

Figura 4.2


/DMSEL./RFSH./IREQ./VREQ
DMSEL.IWR./RFSH./IREQ./VREQ
28
DMSEL./IWR./RFSH
./IREQ./VREQ
RFSH
/RFSH./IREQ.
VREQ./VWR
/RFSH./IREQ.
VREQ.VWR
/RFSH.IREQ.IWR
./RFSH.IREQ./IWR
DMSEL./IWR.
HENE./RFSH
./IREQ
./VREQ
DMSEL.IWR.HENE./RFSH
./IREQ./VREQ
/DMSEL+/HENE+
+RFSH+IREQ+
VREQ
/DMSEL+
DMSEL.IWR+
RFSH+IREQ+VREQ
/DMSEL+(DMSEL.IWR)+
/HENE+RFSH+
IREQ+VREQ
/DMSEL+RFSH+
IREQ+VREQ
/DMSEL+RFSH+IREQ+VREQ
DMSEL./IWR.HENE
./RFSH./IREQ./VREQ

4.7.3 Simulacao de burst

A simulação de DMA com acessos em burst apóia-se na idéia discutida anteriormente de usar ciclos de refresh prolongados. Para isto são necessárias pequenas alterações na placa do i860. Estas alterações deveriam ter um mínimo impacto já que a placa em questão é compartilhada.

As alterações da placa do i860 se restringem ao seguinte:

1- A introdução de um jumper que permite escolher o modo de operação da placa, se original ou simulação, selecionando o sinal apresentado a PAL22V10 da DMSTATE. Pode-se optar pelo sinal original de pedido de refresh (/RFSHREQ) ou por um novo sinal de Refresh e Hold (/RH) gerado na placa de extensão.

2- Uma pequena mudança na máquina de estados da memória DMSTATE, Fig 4.2, que permite a esta permanecer num determinado estado inócuo do ciclo de refresh (S20) enquanto a linha de pedido de refresh estiver ativa. Isto é feito tornando a transição, antes incondicional, do estado S20 para S30, condicinada a /RFSH. Assim, no modo "original" permanece tudo como antes pois o sinal de desativação de pedido de refresh (/RFSHCL) é gerado antes, em S22, e /RFSH está sempre desativado em S20. No modo de simulação a duração de um ciclo de refresh depende do padrão do sinal /RH gerado pela máquina de estados da placa de extensão exatamente para este fim.

3- O acréscimo da programação do timer 0 da i8254 da placa de extensão na EPROM com o código de "power-up" da placa do i860. Este timer é responsável pelo refresh quando o jumper está em modo de simulação.

4.7.3.1 Máquina de estados da placa de extensão

A máquina de estados da placa de extensão é a responsável pela geração do sinal de refresh e hold (/RH) apresentado à DMSTATE no modo de simulação. Esta máquina de estados foi implementada completamente em uma PAL16R8. Esta

PAL registrada é síncrona com DMSTATE e usa o mesmo clock. Seu diagrama de tempo pode ser visto na Fig 4.3.

O flip-flop (U2B) controla o modo de simulação habilitando ou não os bursts (refreshes prolongados). O latch (U4) armazena o valor de contagem em clocks da CPU de que o refresh deve ser prolongado.

O funcionamento desta máquina de estados pode ser descrito da seguinte maneira:

Quando chega um pedido de refresh (/OUT0) este é registrado em /RPENDL (refresh pending). No ciclo seguinte /RH será ativado e mantido enquanto durar /RPENDL (refresh) ou a contagem for diferente de zero (hold). /RPENDL permanecerá ativado até que eventualmente DMSTATE entre em um ciclo de refresh. Quando isto ocorre o sinal /RFSHCL é ativado em S21, e tem o efeito de desativar /RPENDL e carregar o valor de contagem de D0..D5 para Q0..Q5, se o MODO for 1. O sinal /RH será mantido ativo e DMSTATE permanecerá em S20 enquanto a contagem em Q0..Q5 é decrementada. Quando esta contagem chega a zero /RH é desativado e DMSTATE prossegue normalmente.

Quando MODO for 0 a contagem Q0..Q5 será sempre 0, e o ciclo de refresh ocorre normalmente sem ser prolongado.

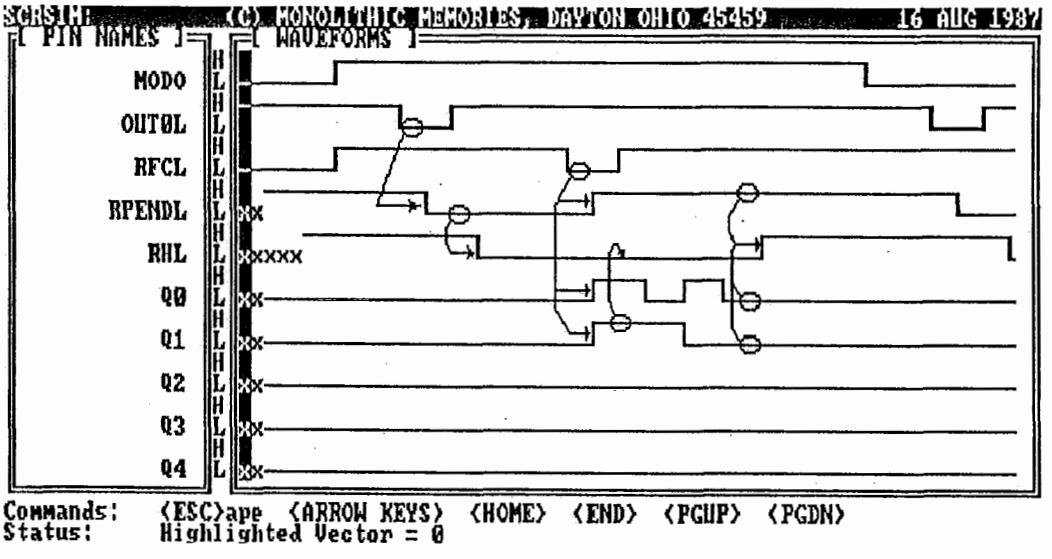


Figura 4.3

CAPÍTULO 5 :
EXPERIMENTOS E RESULTADOS

5) EXPERIMENTOS E RESULTADOS

Os experimentos foram realizados na segunda e na terceira etapas descritas anteriormente, e serão relatados junto com seus resultados por etapa.

Para expressar os resultados são usados, sempre que possível, valores relativos. Em geral, o significado de tais resultados é óbvio, entretanto, convém definir e analisar melhor algumas das métricas usadas.

A taxa de DMA é a taxa com que um suposto elemento de comunicação toma posse do barramento da memória. Fisicamente seria expressa como um número de "bursts" por segundo, sendo reduzido ao número de palavras por segundo no caso em que o burst é de uma palavra.

A taxa de comunicações (COM), representa a quantidade de informação que pode fluir por unidade de tempo para comunicação, e equivale à taxa de DMA multiplicada pelo tamanho do burst, que se expressa como o número de palavras por segundo que são acessadas para esse fim.

Seria desejável que a unidade adotada para representar estas taxas pudesse abstrair dos resultados o tamanho da palavra no acesso, e a freqüência dos clocks da CPU e do refresh. Isto tornaria o resultado facilmente conversível para MW/s, MB/s, Mb/s, para freqüências de clock de 16.6MHz, 33.3MHz e, caso fosse possível levar o sistema de memória a esta freqüência, 40MHz. Tudo isto é válido desde que se mantenha a relação entre os tempos dos ciclos longos e curtos.

A unidade adotada para expressar as taxas de DMA e de comunicações é a fração da banda passante far, (B.P.Far). Este é um valor relativo à banda passante da memória caso todos os acessos fossem distantes.

Para o DMA, esta fração indica a quantidade de "bursts" em relação ao número máximo de acessos possíveis por unidade de tempo caso todos os acessos fossem longos.

Para a COM, esta fração indica a quantidade de palavras

acessadas em relação ao número máximo de palavras acessíveis por unidade de tempo caso todos os acessos fossem longos.

Uma outra métrica usada é a degradação esperada (DEG); esta pode ser calculada a partir da taxa de DMA e do tamanho do "burst" como o inverso do total de tempo que resta à CPU para acessar à memória, descontado o tempo gasto para comunicação.

Esta métrica indica a degradação no tempo de processamento que seria visível caso o processador dispusesse de sua fração da banda passante da memória contígua no tempo, o que permite analisar o efeito do DMA sobre o desempenho do sistema de memória.

Na placa do i860 usada para os testes, um acesso Far gasta 7 clocks e um acesso Near 2 clocks, enquanto um Ciclo Far gasta 10 clocks e um Ciclo Near 2 clocks. A relação entre estes tempos é influenciada pelo uso do mecanismo de "pipelining" de memória que torna os acessos e Ciclos Far mais lentos do que seriam num sistema simples.

Para a escala de B.P.Far usou-se um Ciclo Far idealizado de 7 clocks, como seria possível com um sistema de memória simples, para dar um resultado mais realista, comparável diretamente com o desempenho dos chips de memória num sistema simples. A escala B.P.Far com Ciclo Far de 10 clocks é proporcional à de 7. A conversão (de 7 para 10) pode ser feita multiplicando-se o eixo por 0.7.

A escala DEG usa o Ciclo Far real de 10 clocks.

5.1 Experimentos da segunda etapa

Na segunda etapa mediram-se os tempos de execução do conjunto de rotinas da FFT para diversas situações diferentes. Estas situações compreendem as combinações do uso de:

- cache de dados;
- Ciclos Near;
- otimizações pelo compilador;
- diferentes valores para as taxas de refresh simulando a demanda por comunicação.

Os dados são relacionados em oito tabelas encontradas no apêndice A. Cada tabela representa uma situação testada, para diversas taxas de refresh, sendo todas porém para um problema com M=10, ou seja, FFT de 1024 pontos. Usando (+) para representar o uso de um recurso e (-) para representar o não uso, as situações testadas são:

Tab 1:	+cache	+near	-optm
Tab 2:	-cache	+near	-optm
Tab 3:	+cache	-near	-optm
Tab 4:	-cache	-near	-optm
Tab 5:	+cache	+near	+optm(-O -sched)
Tab 6:	+cache	-near	+optm(-O -sched)
Tab 7:	+cache	+near	+optm(-O)
Tab 8:	+cache	-near	+optm(-O)

Estas medidas foram feitas usando-se temporização por software, ou seja, com interrupções habilitadas. A resolução da medição de tempos é de um milissegundo. As rotinas mais rápidas (bininv, gera_ww e preparainv) foram chamadas cem vezes, enquanto as mais lentas apenas dez. Isto garantiu que todas as contagens de tempo excedessem cem unidades de tempo ficando o erro relativo da medição limitado a menos de um por cento (1%).

A demanda por comunicação foi simulada através do aumento da taxa de refresh, como já explicado. Esta taxa era programada no timer 0 da i8253 da placa do i860, com o valor em tics do clock de 2MHz, aplicado ao componente, que deveria corresponder ao período entre um ciclo de refresh e o subseqüente. A este valor nos referimos aqui como : **rr**. Cada tic representa meio (0.5) milissegundo, ou 8.34 clocks da CPU.

Os testes foram feitos programando-se **rr**, a taxa de refresh, para valores de 15 a 2.

5.1.1 Desempenho do conjunto processador/compilador

A Tabela 5.1 a seguir apresenta as relações entre os tempos de execução das rotinas na placa do i860 a 16.6MHz na memória dinâmica, com um PC-AT 386/387 a 20MHz sem cache e com 0 "wait-states".

Observam-se ganhos expressívos em relação ao AT-386/387, mesmo este sendo usado sem wait states a 20MHz, enquanto o i860 trabalha a 16.6MHz. A vantagem do i860 dobraria com a freqüência de clock de 33.3MHz para a qual a placa foi projetada.

Para uma FFT binária tem-se N*log(N)/2 butterflies. Uma FFT com o número de pontos N=1024 realiza 5120 butterflies. Cada butterfly é composta de uma multiplicação e duas somas complexas, o que corresponde a aproximadamente 10 operaçãos reais.

Daí, tomando-se a fft_ww_dt como exemplo, tem-se 51200 operações reais executadas em 30.8ms, ou seja, um desempenho de 1.66 MFlops. Apesar da vantagem impresionante sobre o AT-386/387 o desempenho em MFlops fica muito aquém do esperado, já que os desempenhos de pico a esta freqüência são 33 MFlops e 24.7 MFlops para as precisões simples e dupla respectivamente. Isto se deve ao compilador, conforme descrito no capítulo anterior.

Desempenho do conjunto processador/compilador

(A) Tempos no PC 386/387DX @ 20 MHz sem cache 0 -wait states
(B) Tempos na Placa COPPE i860 @ 16.6 MHz c/cache c/nene c/pipe de mem.

Rotina(s) chamadas	Tempo/chamada(A)	Tempo/chamada(B)	T(A)/T(B)
bininv	26.37 ms	2.60 ms	10.1
gera_ww	26.37 ms	1.16 ms	22.7
fft_df ifft_dt	857.14 ms	73.2 ms	11.7
fft_df	445.05 ms	39.6 ms	11.2
fft_dt	395.60 ms	31.6 ms	12.5
fft_df ifft_dt (ww)	802.20 ms	75.6 ms	10.6
fft_ww_df	417.58 ms	42.0 ms	9.94
fft_ww_dt	368.13 ms	30.8 ms	11.9
fft_df ifft_dt (S)	741.76 ms	65.2 ms	11.4
fft_S_df	362.64 ms	33.2 ms	10.9
fft_S_dt	362.64 ms	29.6 ms	12.3
ifft_df	461.54 ms	41.6 ms	11.1
ifft_dt	412.09 ms	33.6 ms	12.3
preparainv(/)	32.97 ms	2.00 ms	16.5
preparainv(*)	22.53 ms	1.76 ms	12.8

obs1 : as ffts sao de 1024 pontos, 5120 butterflies, 51200 flops.
obs2 : as iffts sao feitas usando preparainv(*).
obs3 : as rotinas em (B) foram compiladas com hc860 -O -sched.

Tabela 5.1

5.1.2 Efeitos das otimizações do compilador

A Tabela 5.2 a seguir apresenta as relações entre os tempos de execução das rotinas com e sem o uso das opções de otimização disponíveis no compilador.

Nos casos típicos a melhora de desempenho não ultrapassou 6%, tendo às vezes até mesmo piorado em 2.7%; mas em alguns casos patológicos com a fft_S_df, que causava "thrashing" na cache, e em preparainv(/), que não aproveitava a vantagem em tempo da multiplicação sobre a divisão, a melhora foi substancial.

Efeitos dos switches de otimização do compilador

T0==)(-optm) T1==)(+optm(-O)) T2==)(+optm(-O -sched))				
T2/T0	T1/T0	T0/T2	T0/T1	
1.0111	1.0000	0.9890	1.0000	bininv
0.8108	0.8176	1.2333	1.2231	gera_ww
0.9576	0.9651	1.0443	1.0362	fft_df ifft_dt
0.9474	0.9725	1.0556	1.0282	fft_df
0.9677	0.9560	1.0333	1.0460	fft_dt
1.0102	0.9987	0.9899	1.0013	fft_df ifft_dt (ww)
1.0277	0.9977	0.9730	1.0023	fft_ww_df
0.9819	0.9879	1.0185	1.0122	fft_ww_dt
0.6927	0.8798	1.4436	1.1367	fft_df ifft_dt (S)
0.5545	0.8120	1.8034	1.2315	fft_S_df
0.9624	1.0094	1.0391	0.9907	fft_S_dt
0.9454	0.9760	1.0577	1.0246	ifft_df
0.9696	0.9586	1.0313	1.0432	ifft_dt
0.3709	0.3691	2.6961	2.7094	preparainv(/)
0.9701	1.0050	1.0308	0.9950	preparainv(*)

Tabela 5.2

5.1.3 Efeitos do DMA para comunicação

A Tabela 5.3 a seguir apresenta as relações entre os tempos de execução das rotinas com rr=2 (1.0us) e rr=15 (7.5us), o que representa um DMA de .42 B.P.Far (1.786 DEG) e .056 B.P.Far (1.059 DEG) respectivamente.

Considerando que com a taxa de DMA em .056 B.P.Far os tempos da CPU são quase contíguos, a degradação que se poderia esperar é de 1.786/1.059, ou seja, 1.686. Isto ocorreria apenas se cada ciclo de memória roubado da CPU produzisse uma perda líquida de desempenho, o que não é o caso pois o compilador aproveita pouco a CPU e não chega a estressar o sistema de memória.

Ainda assim, tomando as relações de tempo para as seqüências de fft_df-ifft_dt e fft_ww_df-ifft_ww_dt pode-se observar que:

- há um acréscimo de 50/60% no tempo de execução com cache e Ciclos Near por causa do roubo de ciclos para DMA;
- o valor da degradação se mantém na mesma faixa, aproximadamente 50%, mesmo quando não se usa a cache, apesar do uso da cache acelerar em duas vezes (100%) a execução;
- sem o uso de Ciclos Near, mas com a cache ligada, o que gera ciclos de "cache-flush" e "cache-fill", a degradação é da ordem de 35%, ou seja, menor do que nas outras circunstâncias;
- sem cache ou Ciclos Near, apesar de os tempos serem substancialmente maiores, o padrão de degradação permanece próximo do original.

Observando gera_ww vê-se que a degradação é muito dependente da presença ou não da cache, o que era de se esperar na medida em que esta rotina repete um "loop", que faz uma multiplicação complexa sempre com os mesmos elementos e armazena seqüencialmente o resultado numa tabela. Assim, a cache reduz muito o tráfego da memória.

Estimativa inicial da degradação do desempenho

T(rr=1.0us)/T(rr=7.5us)				
(+cache) (+near)	(-cache) (+near)	(+cache) (-near)	(-cache) (-near)	
1.60	1.43	1.37	1.44	bininv
1.03	1.46	1.02	1.52	gera_ww
1.49	1.49	1.33	1.46	fft_df ifft_dt
1.52	1.50	1.35	1.49	fft_df
1.47	1.48	1.31	1.43	fft_dt
1.60	1.50	1.37	1.47	fft_df ifft_dt (ww)
1.70	1.51	1.36	1.50	fft_ww_df
1.53	1.49	1.35	1.44	fft_ww_dt
2.10	1.43	1.48	1.44	fft_df ifft_dt (S)
2.52	1.45	1.54	1.46	fft_S_df
1.29	1.42	1.30	1.43	fft_S_dt
1.50	1.50	1.36	1.48	ifft_df
1.46	1.47	1.31	1.42	ifft_dt
1.01	1.18	1.04	1.05	preparainv(/)
1.51	1.20	1.51	1.26	preparainv(*)

Tabela 5.3

5.1.4 Efeitos da Cache

A Tabela 5.4 a seguir apresenta as relações entre os tempos de execução das rotinas com e sem uso da cache de dados.

Cabe notar que a cache de código foi mantida em uso todo o tempo por ser um recurso arquitetural de grande valor prático e validade comprovada. Esta cache tem um "hit-rate" muito elevado, possivelmente acima de 90%, o que minimiza muito o número de acessos à memória principal em busca de código, e a consequente influência destes no desempenho das aplicações.

Nota-se que o uso da cache (de dados) melhora significativamente o desempenho para os testes combinados, da ordem de 2.15, ou (115%).

Existe uma excessão importante em fft_S_df que provoca "trashing" na cache. Existe ainda o caso de bininv onde o padrão de acessos irregular e sem reaproveitamento diminui a eficiência da cache, mas ainda assim o ganho de desempenho é de 50%. No caso de preparainv, o que ocorre é que o tempo de execução desta rotina é dominado pelos cálculos que a CPU executa escalarmente numa grande seqüência de instruções para cada ponto acessado do vetor.

Nota-se também, que a eficácia da cache é maior quando empregada junto com os Ciclos Near, assunto que será tratado a seguir.

Efeitos da cache

T(-cache)/T(+cache)				
(+near)		(-near)		
7.5us	1.0us	7.5us	1.0us	
1.50	1.34	1.13	1.19	bininv
2.24	3.16	2.44	3.63	gera_ww
2.15	2.15	1.60	1.76	fft_df ifft_dt
2.26	2.23	1.64	1.81	fft_df
2.08	2.10	1.60	1.76	fft_dt
2.14	2.00	1.50	1.62	fft_df ifft_dt (ww)
2.24	1.99	1.48	1.62	fft_ww_df
2.06	2.01	1.56	1.66	fft_ww_dt
1.55	1.06	.88	.85	fft_df ifft_dt (S)
1.18	.68	.59	.56	fft_S_df
2.30	2.53	1.75	1.91	fft_S_dt
2.22	2.21	1.63	1.78	ifft_df
2.03	2.04	1.58	1.72	ifft_dt
1.18	1.37	1.34	1.36	preparainv(/)
1.37	1.10	1.39	1.15	preparainv(*)

Tabela 5.4

5.1.5 Efeitos dos Ciclos Near

A Tabela 5.5 a seguir apresenta as relações entre os tempos de execução das rotinas com e sem o uso de Ciclos Near.

O uso dos Ciclos Near provoca respostas bastante diferentes, dependendo se são usados conjugadamente com a cache ou não. No primeiro caso dão ganho de 40% a 50%, aproximadamente, enquanto que no segundo apenas 6%. O motivo será visto a seguir.

5.1.6 A cache como "packetizing filter"

A cache atua entre o processador e a memória como um "packetizing filter".

O processador gera uma série de requisições ao sistema de memória que especificam endereços dentro de um "working set". Estes endereços aparecem desordenadamente trocando de set, o que, visto da perspectiva do sistema de memória dinâmica, torna a seqüência de acessos não local, exigindo o uso de acessos longos.

Quando a cache é usada, além de a demanda por acessos à memória principal ser reduzida aos "cache misses", o protocolo de acesso à memória dinâmica passa a ser composto dos "cache flush" e "cache fill", escritas e leituras "multi-word" respectivamente. Agora, para cada bloco (ou packet) de acessos, os endereços são locais, admitindo o uso de Ciclos Near para otimizar o acesso. Este efeito é bem visível, e recomenda o uso conjugado destes recursos: a cache e Ciclos Near.

Este resultados são semelhantes aos obtidos em [Goodman84].

Efeitos dos Ciclos Near

T(-near)/T(+near)				
(+cache)		(-cache)		
7.5us	1.0us	7.5us	1.0us	
1.46	1.34	1.09	1.10	bininv
.99	.98	1.08	1.13	gera_ww
1.42	1.27	1.06	1.06	fft_df ifft_dt
1.44	1.28	1.05	1.04	fft_df
1.38	1.23	1.06	1.03	fft_dt
1.52	1.29	1.06	1.04	fft_df ifft_dt (ww)
1.60	1.28	1.05	1.07	fft_ww_df
1.43	1.25	1.04	1.04	fft_ww_dt
1.80	1.28	1.02	1.02	fft_df ifft_dt (S)
2.06	1.26	1.02	1.03	fft_S_df
1.33	1.35	1.01	1.02	fft_S_dt
1.43	1.30	1.06	1.05	ifft_df
1.38	1.23	1.07	1.04	ifft_dt
.99	1.01	1.12	1.01	preparainv(/)
1.25	1.25	1.26	1.32	preparainv(*)

Tabela 5.5

5.2 Experimentos da terceira etapa

Na terceira etapa mediram-se os tempos de execução de rotinas de multiplicação de matrizes, descritas no capítulo anterior, para fazer um uso mais digno das possibilidades da CPU em relação a suas potencialidades, e retratar mais realisticamente as aplicações.

Nestes testes mediram-se os tempos de execução das rotinas. Os testes foram feitos multiplicando uma matriz A de dimensão (1xN) por uma B, (NxN), variando os seguintes parâmetros:

- o uso da cache apenas para A ou para A e B;
- o tipo de dados float (REAL*4) ou double (REAL*8);
- tamanho do problema, N=(8,16,32,64,128);
- a taxa de DMA de comunicação: **br** (burst rate);
- e o tamanho do burst: **bz** (burst size), **bz**=(1,2,4,8,16).

Estes testes foram realizados em duas baterias, que diferem basicamente na região de variação de **br**, a taxa de DMA. A exemplo de **rr** da segunda etapa de testes, **br** é um valor de contagem para um timer da i8254 da placa de extensão. Esta, entretanto, tem um clock de aproximadamente 7.15MHz, ou seja 140ns, ou ainda 2.33 clocks da CPU, e é a resolução com que se pôde variar o **br**.

Na primeira bateria variou-se **br** de um máximo de 24 até um mínimo que depende do tamanho do burst: **bz**. Assim, os dados do teste mostram o comportamento do sistema na região de maior interesse, em que o desempenho do processamento é significativamente afetado pela comunicação (omitindo a parte assintótica das curvas). Esta bateria de testes esgotou todas as combinações possíveis dos parâmetros acima mencionados.

Na segunda bateria variou-se **br** de 56 até um valor mínimo igual ao da primeira bateria. Assim, ganhou-se pouco mais que uma oitava extra nas medições, o que permite uma

visualização clara do comportamento assintótico das curvas. Os dados desta bateria englobam os da primeira; entretanto, o conjunto de dados não está completo para todos os valores citados dos parâmetros.

Nesta bateria foram medidos também os tempos de execução das rotinas **dummydp/dummydc**, descritas no capítulo quatro, onde não ocorrem interlocks no pipeline por dependência de dados, no "loop" principal. Este artifício permite averiguar o quanto a execução das rotinas é dependente dos interlocks ou da latência da memória junto com a especificação da computação.

As medidas de tempo foram feitas em hardware com as interrupções, incluindo a de tempo real, desabilitadas. A resolução das medidas é de 2us, o que corresponde a 4 ticks de um clock de 2MHz. A precisão relativa das medidas depende do tamanho do problema. Para N=128, a situação em que se concentra a maior parte da análise, a precisão obtida é sempre melhor do que 0.1%. Lista-se abaixo a precisão relativa mínima por tamanho de problema:

- N=128 ==> .1%
- N=64 ==> .33%
- N=32 ==> 1.1%
- N=16 ==> 3.3%
- N=8 ==> 10%

As rotinas usadas são referidas através de siglas. Estas siglas indicam precisão dupla (**d**) ou simples (**s**), presença de Ciclos Near (**n**) ou ausência (**f**), o tipo da rotina se **dp** ou **dc**, o tamanho do problema, e, nas medidas da segunda bateria, se era multiplicação de matrizes (**m**) ou "**dummy**" (**d**). O tamanho é expresso por um número igual a $\log_2$ de N.

Assim, por exemplo, na primeira bateria:

-dndp7 dupla, com Ciclos Near, tipo **dp**, N=128.

-sfdc5 dupla, sem Ciclos Near, tipo **dc**, N=32.

-dndpX dupla, com Ciclos Near, tipo **dp**, vários Ns.

Analogamenta, na segunda bateria:

-dnmdp7 dupla, com Ciclos Near, multiplicação normal, tipo **dp**, N=128.

-sfddc5 dupla, sem Ciclos Near, multiplicação **dummy**, tipo **dc**, N=32.

Convém observar, também, que a seqüência de medições foi feita em loops por DMA (**br**), o mais interno da menor taxa para a maior, e por tamanho do burst (**bz**), o mais externo e também do menor para o maior. O miolo dos loops executava primeiro **dp** e depois **dc**, na primeira bateria, e **mdp**, **mdc**, **ddp** e **ddc**, nesta ordem na segunda bateria. Como a cache não foi esvaziada, alguns resultados podem parecer tendenciosos devido à ordem dos testes.

Em algumas medições, quando a taxa de DMA é muito elevada, aparecem pequenas anomalias nas curvas apresentadas (como na Fig. 5.1a). Estas anomalias são causadas pela incapacidade do hardware de atender a "todos os pedidos de DMA", perdendo alguns, o que torna algumas curvas não monotônicas em situações extremas.

5.2.1 Caracterização do software

5.2.1.1 Desempenho e o tamanho do problema

As Figuras 5.1a, 5.1b, 5.1c e 5.1d apresentadas a seguir relacionam o desempenho do processador (MFlops) com a taxa de DMA (B.P.Far) para precisão dupla e simples e para os dois tipos de rotina **dp** e **dc**.

O desempenho atingido para o problema com N = 128 nas rotinas **dp**, Figuras 5.1a e 5.1b, é quase metade do de pico (24.7 MFlops) para esta freqüência de clock.

As rotinas **dc**, Figuras 5.1c e 5.1d, têm o desempenho máximo em problemas com N=16 para precisão dupla e N=32 para precisão simples. Isto era esperado porque estas rotinas usam a cache para armazenar a matriz B. Assim, a partir destes valores de N, em que a matriz B ocupa metade dos 8KB da cache de dados, os acessos à matriz B provocam "thrashing" na cache.

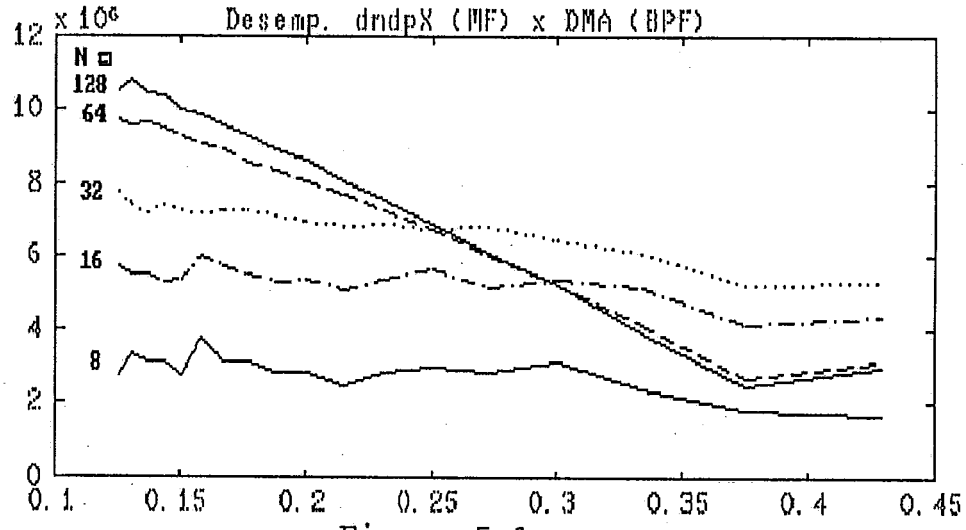


Figura 5.1a

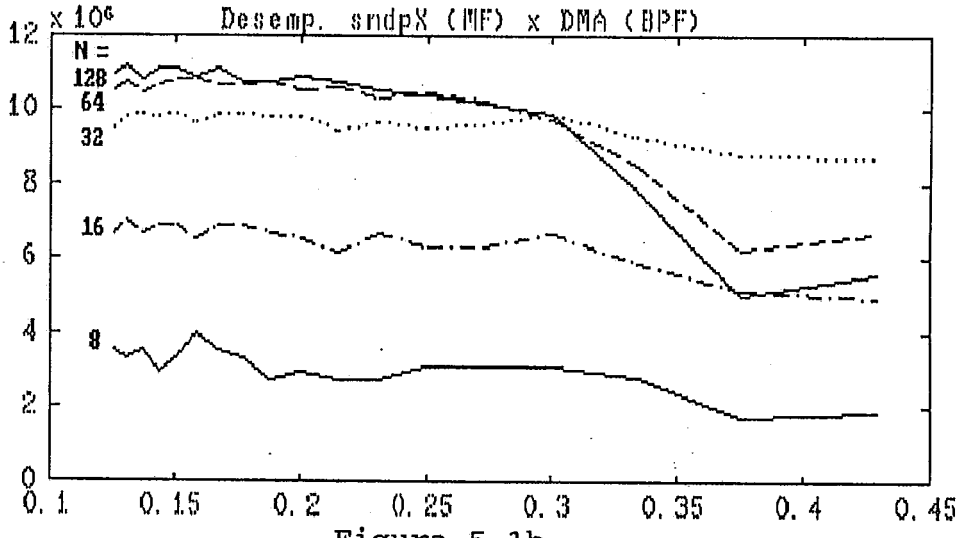


Figura 5.1b

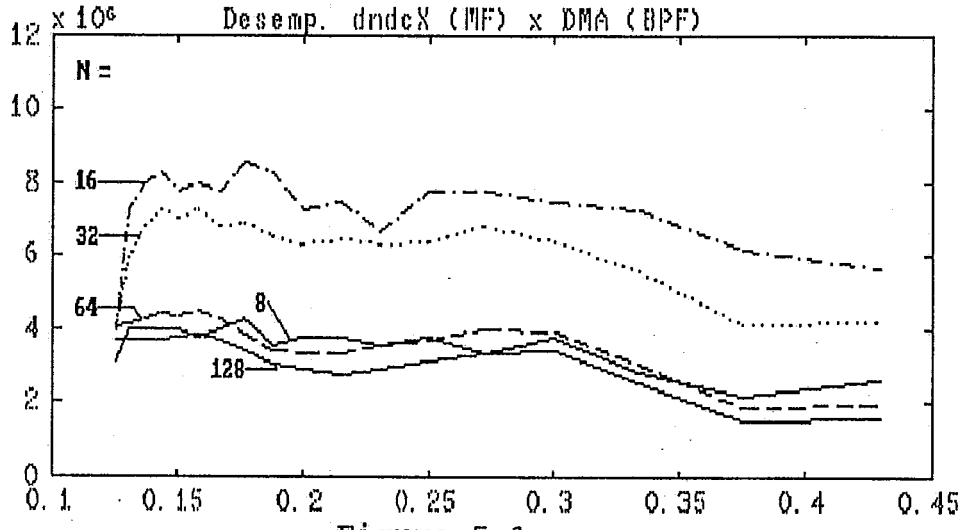


Figura 5.1c

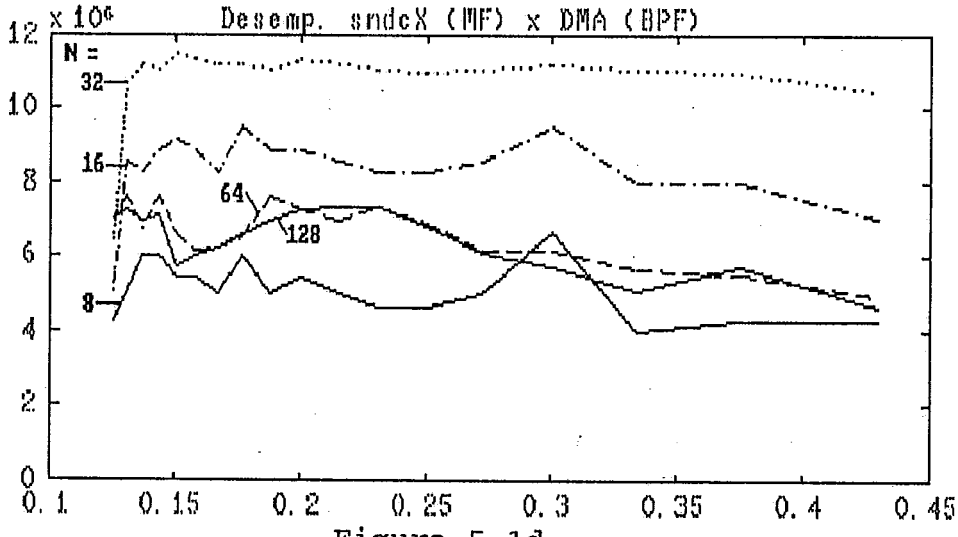


Figura 5.1d

5.2.1.2 Ciclos Near, DMA e o tamanho do problema

As Figuras 5.2a, 5.2b, 5.2c e 5.2d apresentadas abaixo relacionam tempo de execução sem Ciclos Near e com eles. Assim pode-se avaliar a utilidade de tais ciclos, e como ela se mantém na presença do DMA, expresso em B.P.Far.

Observa-se na Figura 5.2a que a falta dos Ciclos Near pode provocar uma perda de desempenho de até 4 (quatro) vezes. Isto se deve ao ótimo aproveitamento da localidade através do esquema de "Page Mode" da memória, perdido sem os ciclos curtos, ou minimizado pelo DMA da comunicação.

Para precisão dupla, com N<=32, a partir da segunda medida os dados de B, e boa parte das outras variáveis, já se encontram na cache devido à ordem dos testes. Isto explica a queda abrupta da eficácia aparente dos Ciclos Near na Figura 5.2a para estas curvas, pois o tráfego da memória externa ao chip diminui. Para N>32, a matriz B se torna maior que a cache, e este efeito não se verifica, sendo que, na prática, o acesso à matriz B pela rotina **dc** provoca um expurgo da cache.

Para a rotina de precisão simples, o limiar é N=64.

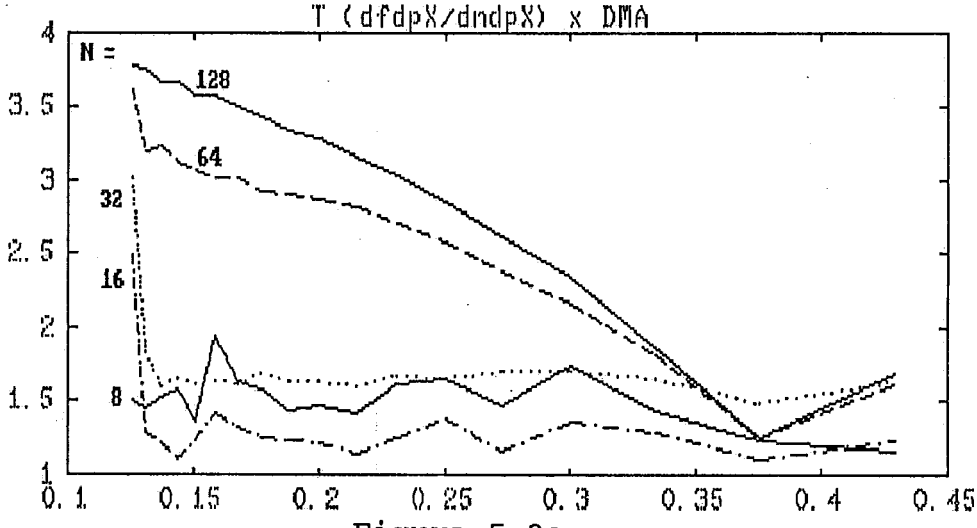


Figura 5.2a

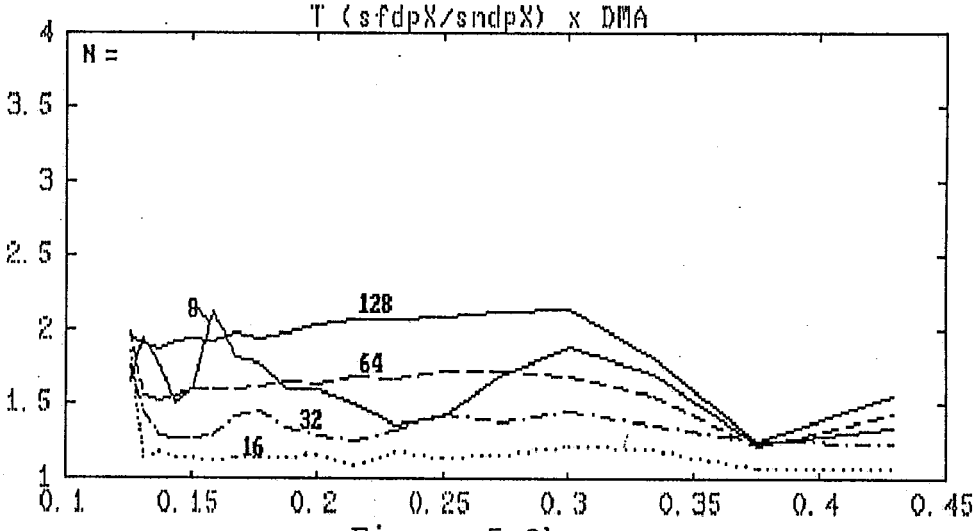


Figura 5.2b

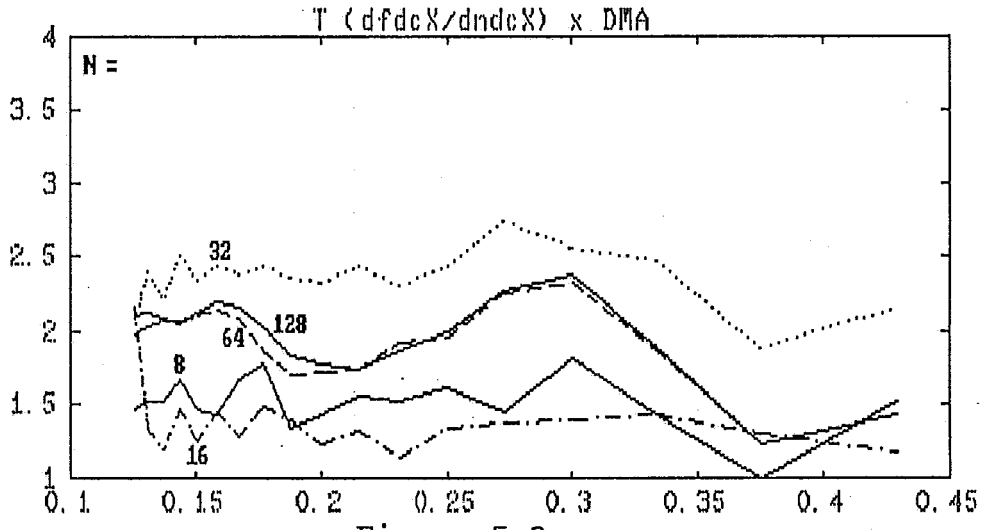


Figura 5.2c

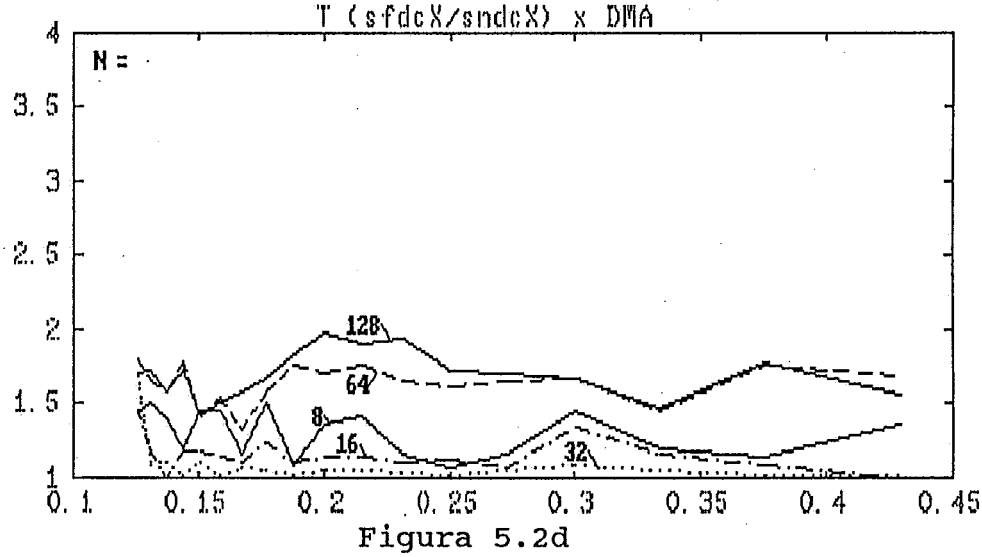


Figura 5.2d

5.2.1.3 Dependência da memória e dos Interlocks

A Figura 5.3a apresentada a seguir relaciona o tempo de execução das rotinas de multiplicação de matrizes com as do código sem interlocks, **"dummy"**, para saber quanto do tempo de execução normal depende do atraso da CPU e quanto do acesso à memória.

A velocidade da rotina **dp** de precisão dupla não se altera, o que significa que seu desempenho é limitado pela memória. As outras têm alguma folga no uso da memória, pois sua velocidade pode ser aumentada mudando os tempos de execução na CPU.

A Figura 5.3b compara os tempos de execução da rotina de precisão dupla com a simples. Estes são aproximadamente iguais, a princípio, mas quando a taxa de comunicação COM aumenta, o gargalo de ambas passa a ser a memória, e como a rotina de precisão dupla requer o dobro de banda passante, fica limitada à metade da velocidade em relação à de precisão simples.

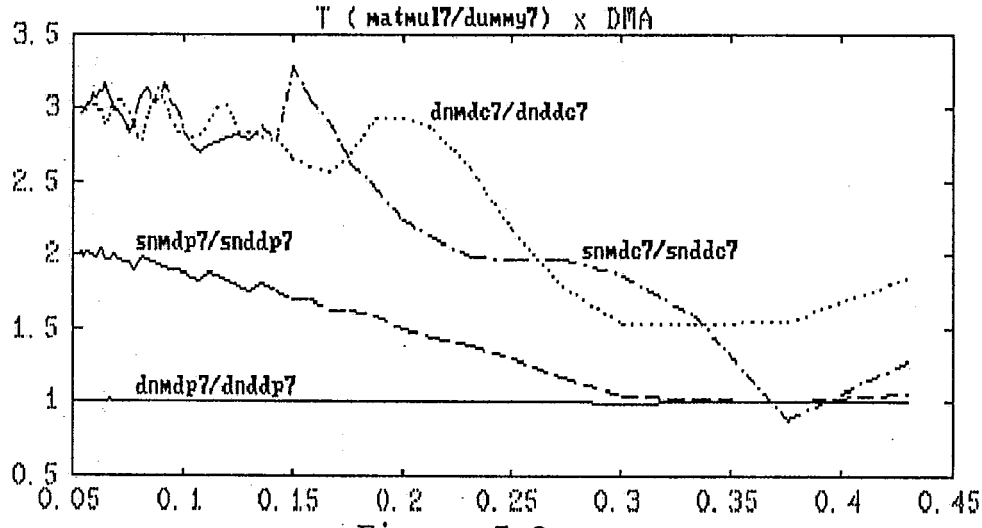


Figura 5.3a

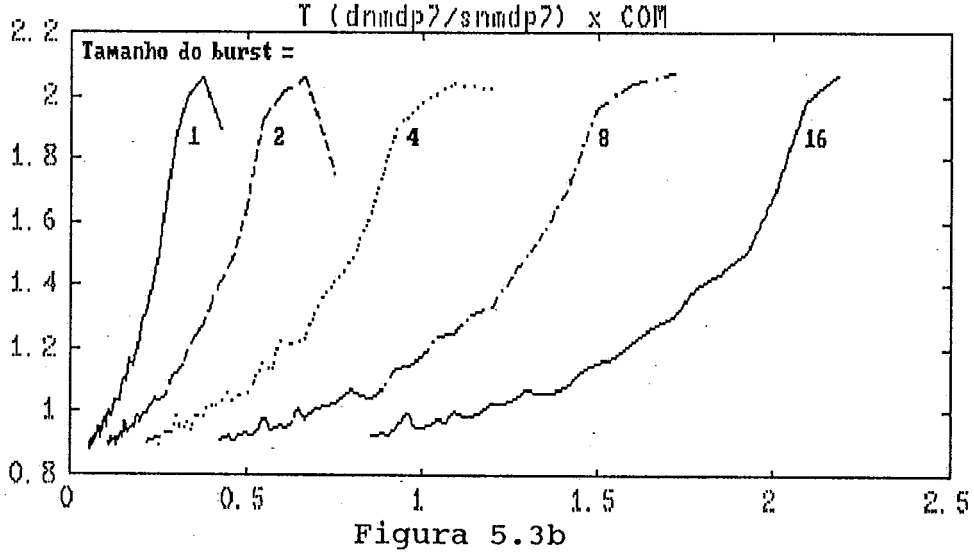


Figura 5.3b

5.2.2 Ganhos com os Ciclos Near

As Figuras 5.4a, 5.4b, 5.4c e 5.4d apresentadas a seguir relacionam os tempos de execução com e sem o uso de ciclos curtos. Isto permite avaliar o ganho que o uso de tais ciclos traz, e como ele se relaciona com a comunicação.

A rotina **dp** de precisão dupla, da Figura 5.4a, cuja velocidade está limitada pela memória, tem um aproveitamento ótimo da localidade. Seus acessos são seqüências e concentrados na matriz B, o que permite o uso ótimo do "Page Mode" da memória. Quando a taxa de comunicações aumenta, entretanto, diminui a localidade vista pelo sistema de memória e cai o rendimento.

Nas outras rotinas, das Figuras 5.4b, 5.4c e 5.4d, o efeito dos ciclos curtos é menor, mesmo assim, da ordem de duas vezes, ou seja, nada desprezível.

As curvas em 5.4a e 5.4b são mais "lisas" porque o padrão de acessos das rotinas é mais regular. Em 5.4c e 5.4d o padrão de acessos sofre o efeito do uso da cache para a matriz B, o que o torna mais complexo.

Os resultados para precisão simples em 5.4b e 5.4d são equivalentes. Os resultados para precisão dupla, em 5.4a e 5.4c, são diferentes pois os ciclos de "write-back", que aparecem em função da disputa da matriz B pela cache, tornam o padrão de acessos sub-ótimo, do ponto de vista da localidade, mas o ganho com os Ciclos Near ainda é maior que dois.

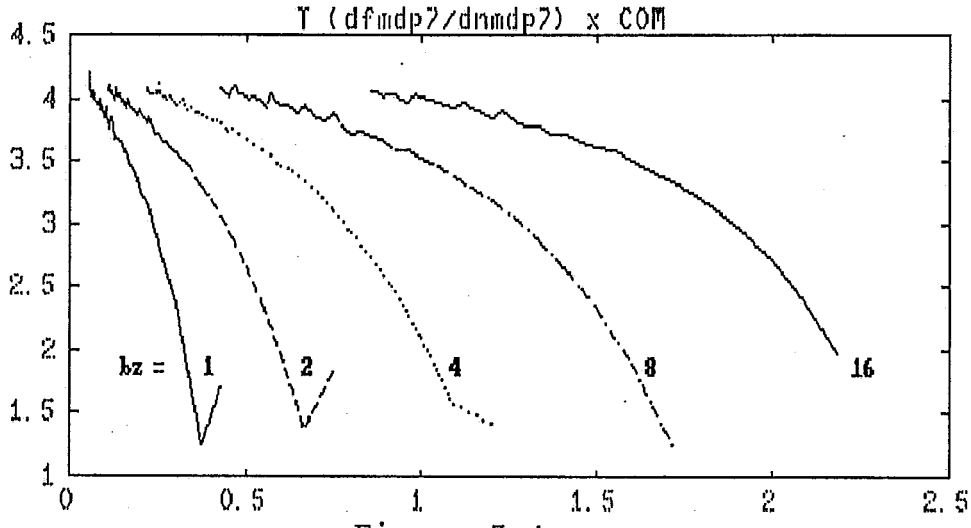


Figura 5.4a

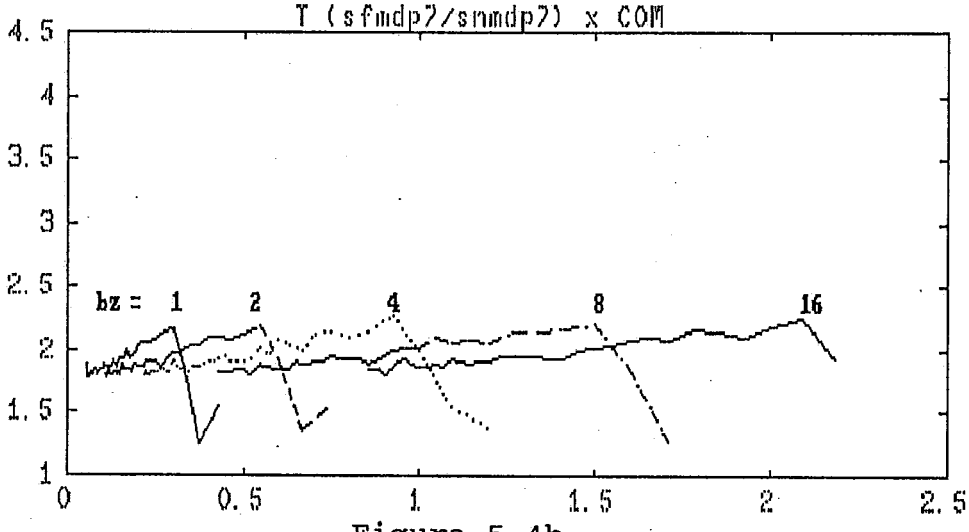


Figura 5.4b

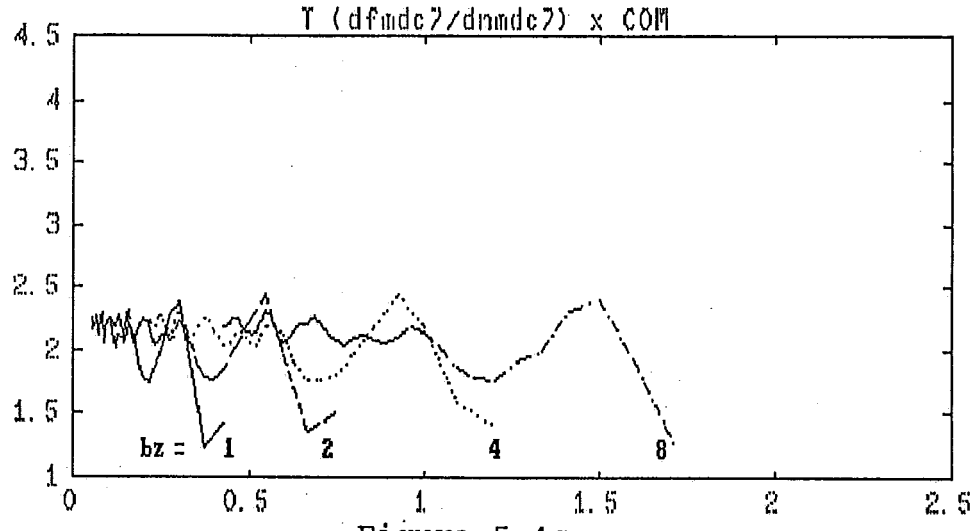


Figura 5.4c

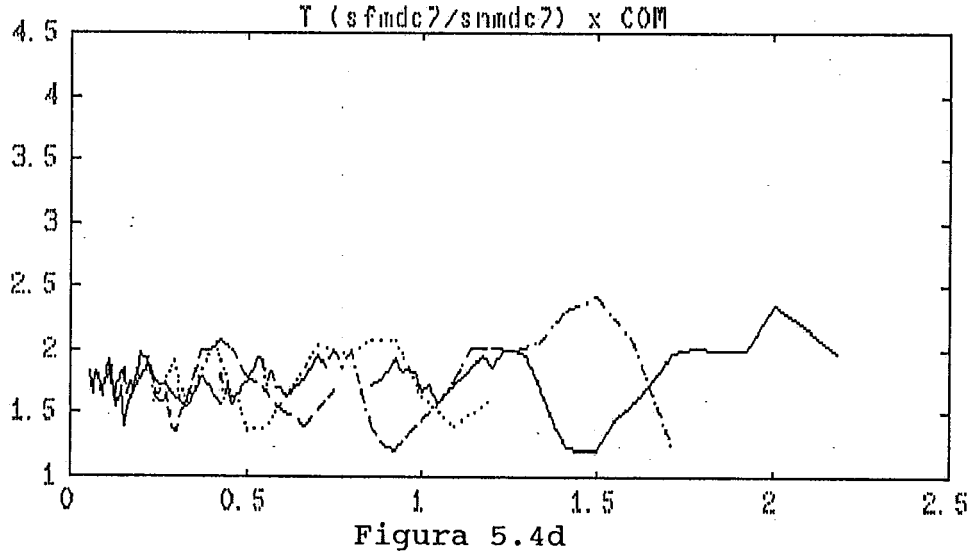


Figura 5.4d

5.2.3 Degradação causada pela comunicação

As Figuras 5.5a, 5.5b, 5.5c e 5.5d apresentadas a seguir relacionam os tempos normalizados de execução com a taxa de comunicação, expressos em B.P.Far, para diversos tamanhos de "burst". Isto permite avaliar qualitativa e quantitativamente como se dá a degradação e como esta se relaciona com o tamanho do "burst".

As Figuras 5.6a, 5.6b, 5.6c e 5.6d apresentadas adiante relacionam os tempos normalizados de execução, que equivalem à degradação real observada, com a degradação esperada (DEG), métrica descrita anteriormente que é função da taxa de DMA, do tamanho do burst, e das relações entre as latências dos Ciclos Far e Near.

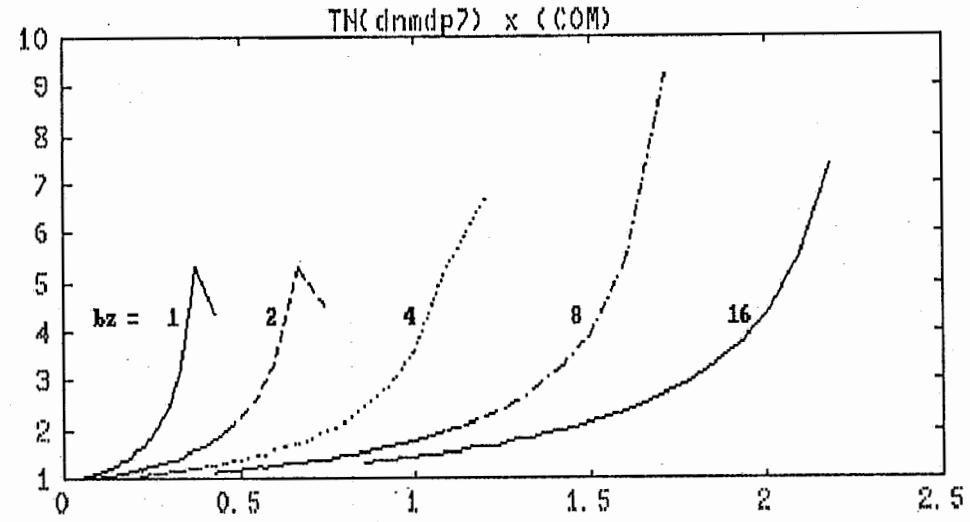


Figura 5.5a

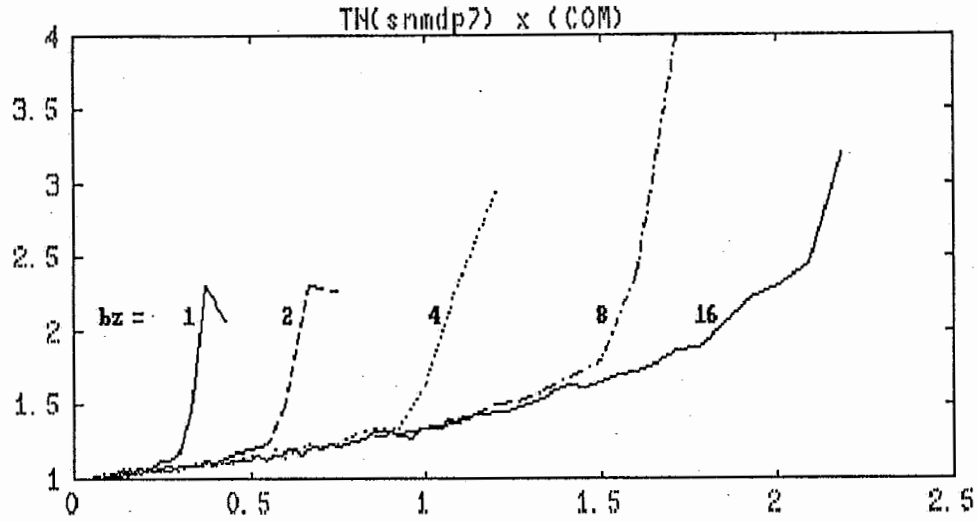


Figura 5.5b

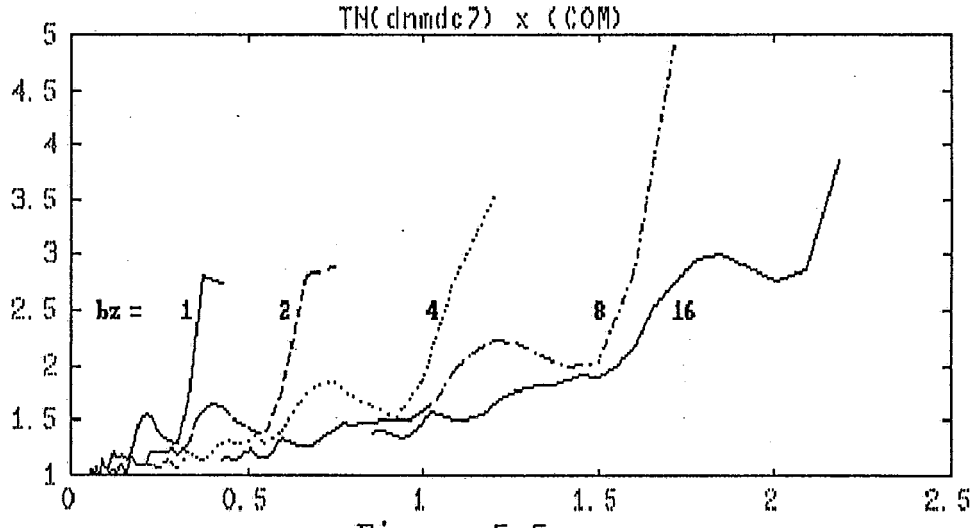


Figura 5.5c

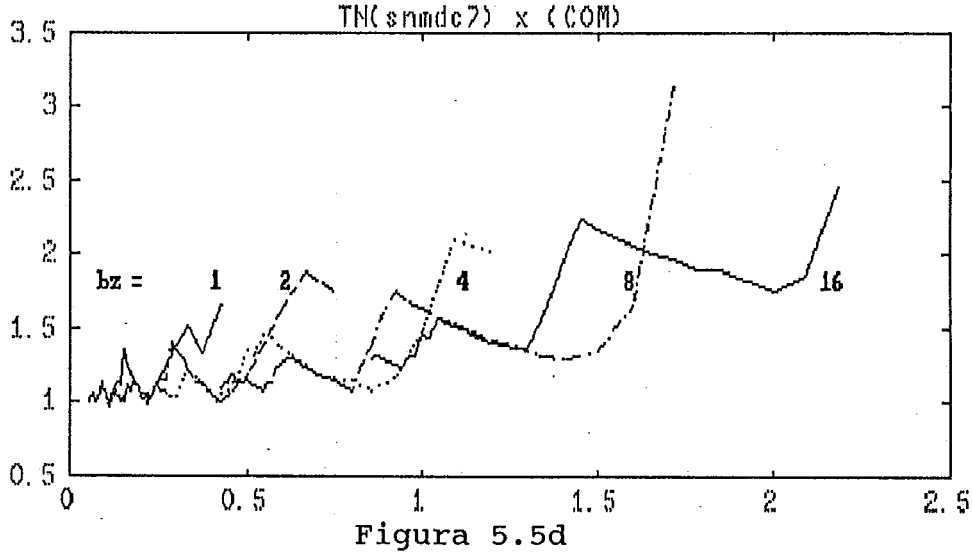


Figura 5.5d

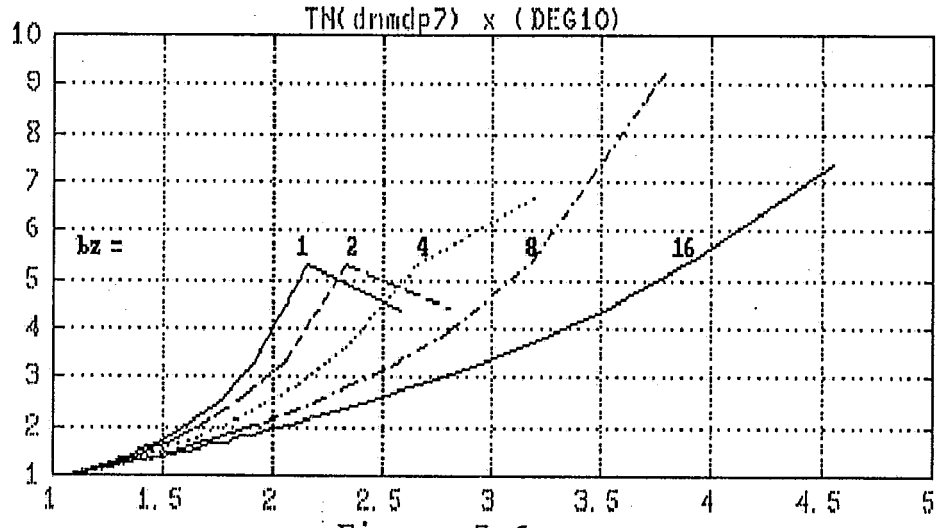


Figura 5.6a

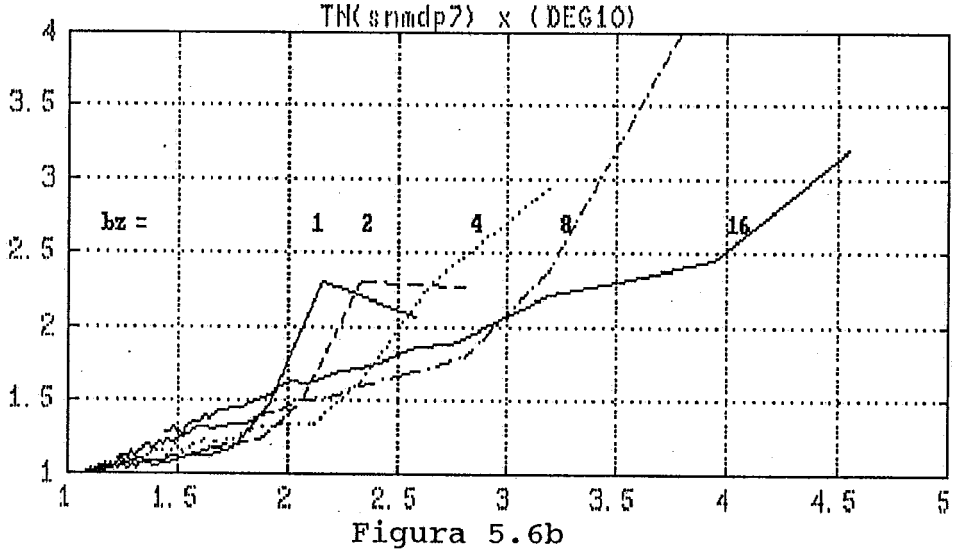


Figura 5.6b

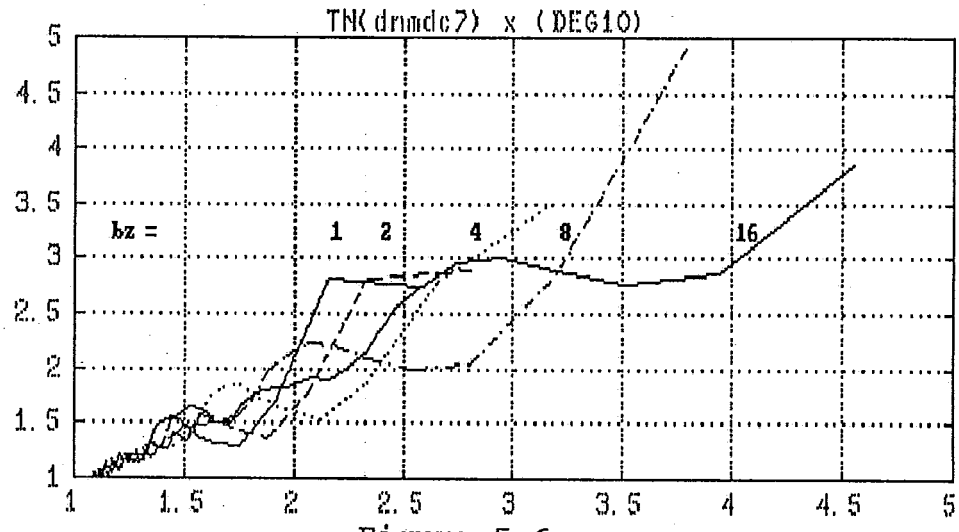


Figura 5.6c

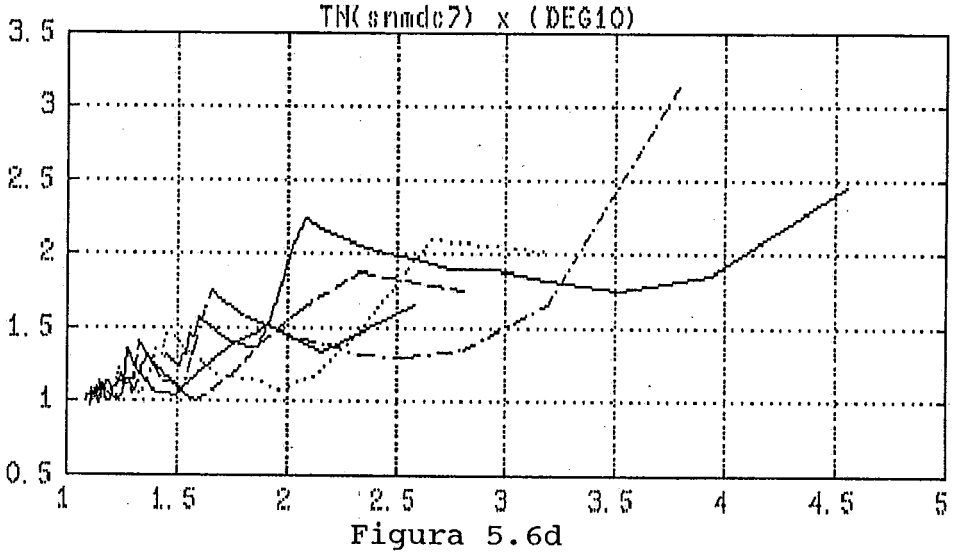


Figura 5.6d

5.2.4 Ganhos com o Burst

As Figuras 5.7a, 5.7b e 5.7c apresentadas a seguir relacionam os tempos de execução em função da taxa de DMA, da taxa de comunicação suportável e da degradação esperada para diversos tamanhos de "burst".

A realização do DMA da comunicação em "burst" de acessos diminui a taxa de interferência do DMA no sistema de memória ao custo de alongar a duração de cada uma, para uma taxa de comunicações fixa. Uma vez que o burst usa os ciclos curtos, os acessos tornam-se mais eficientes. Além de ser possível uma taxa de comunicação absoluta maior, também diminui a degradação causada ao processador.

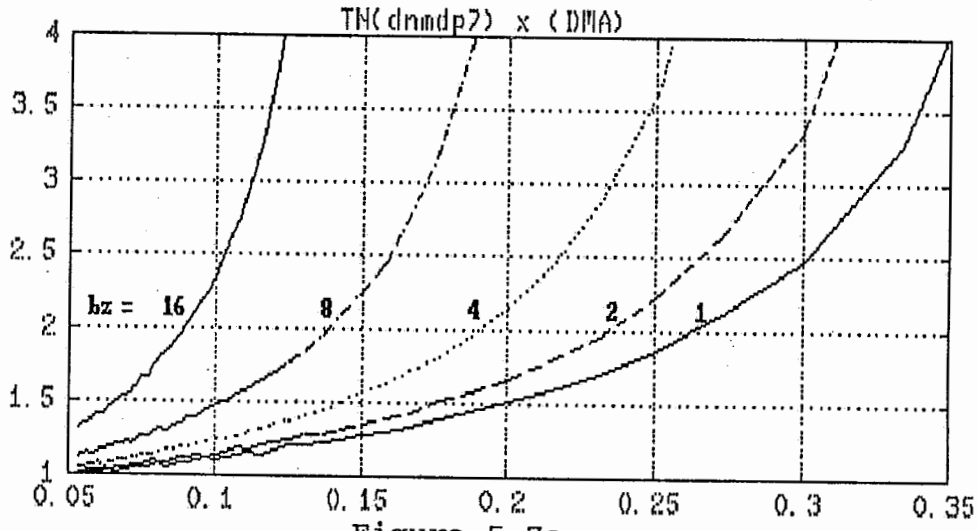


Figura 5.7a

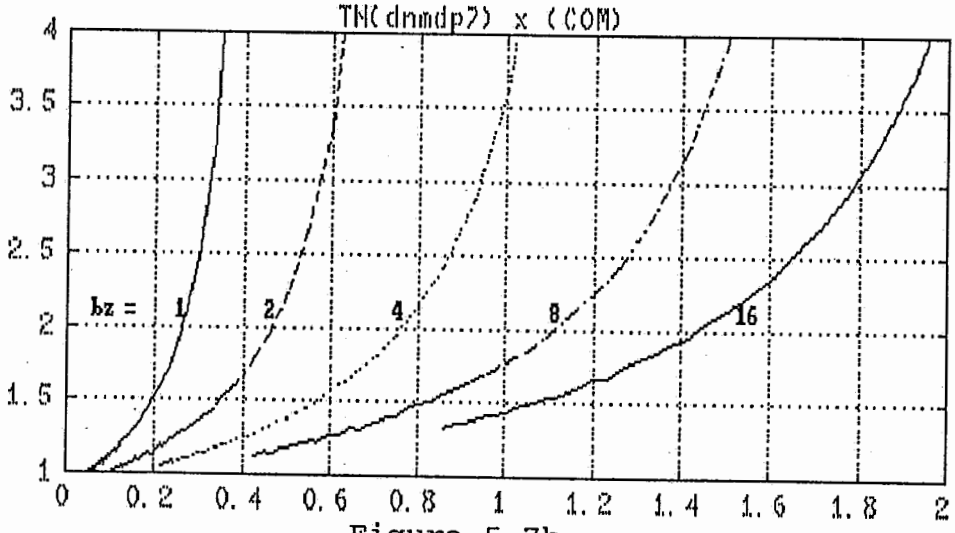


Figura 5.7b

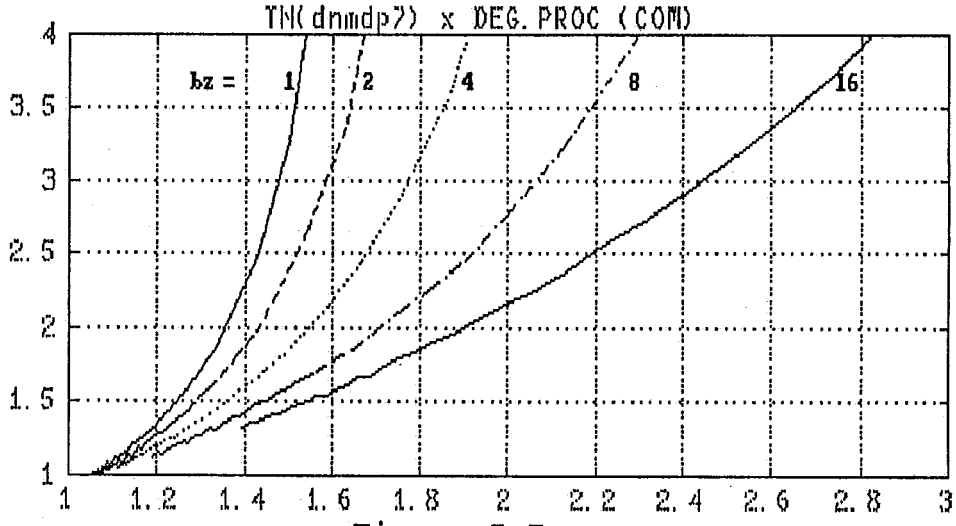

TN(dnmdp7) x DEG.PROC (COM)
bz =
1
2
4
8
16
4
3.5
3
2.5
2
1.5
1
1
1.2
1.4
1.6
1.8
2
2.2
2.4
2.6
2.8
3

Figura 5.7c

CAPÍTULO 6 :
CONCLUSÕES E SUGESTÕES

6) CONCLUSÕES E SUGESTÕES

Aqui são apresentadas conclusões baseadas no estudo realizado:

- Quando um microprocessador com capacidade de alto desempenho de ponto flutuante como o i860 é usado, seu desempenho sustentado fica limitado pela sua capacidade de acesso à memória.

- Sendo o sistema de memória baseado em memórias dinâmicas (DRAMs), este pode ser otimizado para aproveitar a localidade nos acessos com ciclos mais curtos, como no conjunto: pipelining de memória, Ciclos Near e "Page Mode" do exemplo testado.

- Os resultados demonstram que o uso de Ciclos Near por parte do processador é de grande valia, em especial quando combinado com o uso de cache, pois esta funciona como um "packetizing filter" aumentando a localidade percebida pelo sistema de memória. Nos exemplos tratados, para um sistema de memória onde a relação entre os tempos de um acesso longo e de um curto é de 3.5 (7:2) e entre os tempos de um ciclo longo e curto é de 5 (10:2), o tempo de execução é aumentado em até 4 (quatro) vezes, quando se impede a ocorrência de acessos curtos.

- A degradação do tempo de execução é maior do que linear com a quantidade de clocks roubados pelo DMA para comunicação. Na verdade, esta degradação também é afetada pela taxa de DMA em si. Isto ocorre, pois, a cada arbitragem do uso do barramento da memória, segue-se um ciclo longo, afetando a localidade nos acessos do processador.

- Soluções para aumentar a banda passante de um sistema de memória, em especial quando dinâmica, são de grande valia já que este é o fator limitante para a taxa de transmissão em projetos recentes, [Seitz90,pag31].

- A implementação do DMA para suportar a comunicação em "bursts" de acessos permite que também se aproveite a localidade, nos acessos para esse fim. Supondo-se a

comunicação baseada em mensagens armazenadas seqüencialmente em buffers, o ganho de eficiência pode ser expressivo.

- O uso de acessos em "burst" para comunicação faz com que seja aumentada a banda passante da memória, em especial, a percebida para este fim, permitindo uma maior taxa de comunicação. Esta maior taxa melhora a relação Tcomm/Tproc e permite melhorar a eficiência do multicomputador.

- O ganho de eficiência pelo uso de "bursts" nos acessos para comunicação também permite uma capacidade maior de processamento concorrente à comunicação. Esta concorrência permite manter a eficiência da computação paralela mesmo fazendo uso de uma relação Tcomm/Tproc pior, quando comparada a nenhuma sobreposição entre comunicação e processamento, [Fox88,pg444].

Este trabalho atingiu seus objetivos:

- ao identificar e analisar a questão do sistema de memória para multicomputadores;

- ao propor soluções eficientes para um nó simples com o uso de memória dinâmica, o que permite manter o custo por nó baixo, e o volume reduzido;

- ao quantificar os efeitos dos mecanismos para o sistema de memória, permitindo uma tomada de decisões racionais em futuros projetos.

6.1 Transputer como processador de comunicações

Sugere-se, ainda, que no desenvolvimento do NCP-II seja dada grande atenção à questão do sistema de mémoria, em especial no caso do projeto seguir o caminho cogitado, baseado nas novas versões do transputer: o T9000, [Byte91,pg265], e do i860; usando o T9000 como processador de comunicações.

Ambos os processadores prometem grande desempenho de ponto flutuante e interfaces otimizadas para uso eficiente

de memória dinâmica. O T9000 também promete uma cache com "4 32-bit word lines" e acessos curtos para o "flush" e "fill" da cache, em 2 acessos de 64 bits, 4 de 32 ou até 16 de 8 bits.

Como o T9000 deve suportar roteamento de mensagens em hardware e "worm-hole routing", um projeto baseado neste componente pode ter as características de comunicação da segunda geração de multicomputadores.

Com uma filosofia de projeto adequada, e um projeto cuidadoso, talvez se possa realizar uma evolução do nó do NCP-I, resultando num nó de excelente desempenho, boa capacidade de comunicação, compacto, e com um sistema de memória simplificado porém eficiente.

Deve-se todavia averiguar se ainda será conveniente ou necessário o uso do i860, pois, se a diferença no desempenho sustentado em relação ao T9000 não for muito grande, a complexidade extra do nó pode não compensar, sendo preferível uma máquina com um maior número de nós mais simples e baratos.

BIBLIOGRAFIA

[Annaratone87]- "The Warp Computer: Architecture, Implementation, and Performance"; Marco Annaratone, et al.; IEEE Transactions on Computers, Vol 36 Num 12, Dec.1987, pg 1523.

[Arimoto90]- "A Speed-Enhanced DRAM Array Architecture with Embedded ECC"; Kazutami Arimoto; IEEE Journal on Solid State-Cicuits, Vol 25 Num 1, Dec.1990, pg 11.

[Asakura90]- "An Experimental 1-Mbit Cache DRAM with ECC"; Mikio Asakura, et al.; IEEE Journal on Solid State-Cicuits, Vol 25 Num 1, Feb.1990, pg 5.

[Bell82] - "Computer Structures: Priciples ans Exemples"; Daniel P. Siewiorek; C. Gordon Bell; Allen Newell; 1982; McGraw-Hill.

[Bell89] - "The Future of High Performance Computers in Science and Engeneering"; C. Gordon Bell; Communications of the ACM; Sep.1989; Vol 32, Num 9, pg1091.

[Bischof88]- "LAPACK Working Note#5, Provisional Contents"; Chris Bischof, et al.; Argonne National Laboratory, ANL8838, Sep. 1988.

[Byte91]- "The Transputer Strikes Back"; BYTE, August 1991, pg265.

[CD90]- "Will the search for the ideal memory architecture ever end?"; Computer Design; July 1, 1990, pg 78.

[Colley89]- "Memory Design for a Supercomputer"; High Performance Systems; April 1989, pg 66.

[Dally88]-"Fine-Grain Message Passing Concurrent Computers"; William J. Dally; The Third Conference on Hypercube Concurrent Computers and Applications, Vol 1, 1988, pg 2.

[Dally90]- "Performance Analysis of K-ary n-cube Interconnection Networks"; William J. Dally; IEEE Transactions on Computers, Vol 39 Num 6, Jun.1990, pg 775.

[Dobbs90] -"Programing RISC Engines"; Dr. Dobb's Journal; Feb.1990, pg 116.

[Dongarra88]- "Algorithm Design for High Performance Computers"; Jack J. Dongarra, Danny C. Sorensen; Parallel Systems and Computation, North Holand, 1988, pg83.

[ED89]- "Design A DRAM Memory Bank For The 80386"; Electronic Design International, February 1989, pg 65.

[ED90]- "Triple-Port Dynamic RAM Accelerates Data Movment"; Electronic Design, May 24,1990, pg 37.

[Fazzari88]-"The Second Generation FPS T Series: An Enhanced Parallel Vector Supercomputer"; Rodney J. Fazzari, et al.; The Third Conference on Hypercube Concurrent Computers and Applications, Vol 1, 1988, pg 61.

[Flynn66]- "Very High Speed Computing Systems"; M. J. Flynn; Proceeding of the IEEE, vol 54, Num 12, Dec.1966, pg 1901.

[Fox88] - "Solving Problems On Concurrent Processors"; Geoffrey C. Fox, et al.; 1988; Prentice Hall

[Fox88a] - "What Have We Learnt from Using Real Paralel Machines to Solve Real Problems ?"; Geoffrey C. Fox, et al.; 1988; Third Conference on Hyprcube Concurrent Computers and Applications; pg897.

[Frey88]-"Problems and Approaches for a Teraflop Processor"; Alexander H.Frey, Geoffrey C. Fox; The Third Conference on Hypercube Concurrent Computers and Applications, Vol 1, 1988, pg 21.

[Goodman84]-"The Use of Static Column RAM as a Memory Hierarchy"; James R. Goodman, Men-chow Chiang; Proc. 11th Annual Symposium on Computer Architecture (june 5-7), Ann Arbor, Mich.; (0194-7111/84/0000/0167$01.00 c 1984 IEEE), 1984, pg167.

[Hayes89] -"Hypercube Supercomputers"; John P. Hayes, Trevor Mudge; Proceedings of the IEEE, Vol 77, Num12, Dec. 1989, pg 1829.

[Hidaka90]- "The Cache DRAM Architecture: A DRAM with an On-Chip Cache Memory"; Hideto Hidaka, et al.; IEEE MICRO, April 1990, pg 14.

[Hwang84] - "Computer Architecture and Paralel Processing"; Kai Hwang, Faye A. Briggs; 1984; McGraw-Hill.

[INMOS88] - "IMS T800 TRANSPUTER (Data Sheet)"; INMOS; Mar.1988.

[INMOS87] - "The TRANSPUTER Instruction Set - A Compiler Writers' Guide"; INMOS; Feb.1987.

[Intel89a]- "i860™ 64-BIT MICROPROCESSOR (Data Sheet)"; INTEL; Feb.1989; Order Number 240296-001.

[Intel89b]- "i860™ 64-BIT MICROPROCESSOR HARDWARE DESIGN GUIDE"; INTEL; 1989; Order Number 240330-001.

[Intel89c]- "i860^{TM} 64-BIT MICROPROCESSOR PROGRAMMER'S REFERENCE MANUAL"; INTEL; 1989; Order Number 240329-002.

[Intel89d]- "i860^{TM} 64-BIT MICROPROCESSOR ASSEMBLER AND LINKER REFERENCE MANUAL"; INTEL; Feb.1989; Order Number 240436-001.

[Intel89e]- "i486^{TM} MICROPROCESSOR (Data Sheet)"; INTEL; Apr.1989; Order Number 240440-001.

[iPSC2a]-"iPSC/2 System: A Second Generation Hypercube"; Ramune Arlauskas; The Third Conference on Hypercube Concurrent Computers and Applications, Vol 1, 1988, pg 38.

[iPSC2b]-"iPSC/2 Node Architecture"; Paul Close; The Third Conference on Hypercube Concurrent Computers and Applications, Vol 1, 1988, pg 43.

[iPSC2c]-"iPSC/2 Direct-Connect Communications Technology"; Steven F. Nugent; The Third Conference on Hypercube Concurrent Computers and Applications, Vol 1, 1988, pg 51.

[Itoh90]- "Trends in Megabit DRAM Circuit Design"; Kiyoo Itoh; IEEE Journal on Solid State-Cicuits, Vol 25 Num 3, Jun.1990, pg 778.

[Jarusek86]- "A Multiprocessor Design in Custom VLSI"; VLSI Systems Design; June 1986, pg 26.

[Jarwala88]- "TRAM: A Design Methodology for High-Performance, Easily Testable, Multimegabit RAMs"; Najmi T. Jarwala, et al.; IEEE Transactions on Computers, Vol 37 Num 10, Oct.1988, pg 1235.

[Kohn89]-"Introducing the Intel i860 64 Bit Microprocessor"; Leslie Kohn, Neal Margulies; IEEE MICRO, August 1989, pg15.

[Mammana81]- "A Filosofia natural da Informação"; Claudio Zamitti Mammana, Rev. Bras. Tecn., Brasília 12(1):54-65, jan./mar 1981.

[Micron90]- "MT43C4257/8 Triple-Port DRAM (Data sheet)"; Micron Technologies, May 1990.

[NCUBE89]- "Ncube Ups the Ante in Hypercube Performance, the new line hits 100 Gflops and 27 scalar Gflops"; High Performance Systems; July 1989, pg 97.

[Nicoud88]- "Video RAMs: Structure and Applications"; Jean-Daniel Nicoud; IEEE MICRO, February 1988, pg 8.

[Nicoud89]- "The Transputer T414 Instruction Set"; Jean-Daniel Nicoud, et al.; IEEE MICRO, June 1989, pg 60.

[Numata89]- "New Nibbled-Page Architecture for High-Density DRAMs"; Kenji Numata, et al.; IEEE Journal on Solid State-Cicuits, Vol 24 Num 4, Aug.1989, pg 900.

[Oppenheim75]- "Digital Signal Processing"; Alan V. Oppenheim, Ronald W. Shafer; 1975; Prentice Hall International, Inc.; Cap 6.

[Optic89] -"Inteligent Memory Chip Competes With Optical Computing" Optical Computing Newsletter, Vol 2, Num 3, Apr.1989, pg163.

[Patterson90]- "Computer Architecture A Quantitative Approach"; John L. Hennessy, David A. Patterson; 1990; Morgan Kaufmann Publishers Inc.

[Peterson91]- "iWarp: A 100-MOPS LIW Microprocessos for multicomputers"; Craig Peterson, et al.; IEEE MICRO, June 1991, pg 26.

[Prog91] -"Loop Unrolling A RISCy Business", Programmer's Journal; July/August 1991, pg66.

[Reed87] - "Multicomputer Networks: Message Based Parallel Processing"; Daniel A. Reed, Richard M. Fujimoto; 1987; MIT Press.

[Seitz85]- "The Cosmic Cube"; Charles L. Seitz; Communications of the ACM; Jan.1985; Vol 28, Num 1, pg22.

[Seitz88]-"The Architecture and Programming of the Ametek Series 2010 Multicomputer"; Charles L. Seitz, et al.; The Third Conference on Hypercube Concurrent Computers and Applications, Vol 1, 1988, pg 33.

[Seitz90]- "VLSI and Parallel Computation"; Charles L. Seitz, Bill Dally, et al.; 19 ; Morgan Kaufmann Publishers INC.

[Siewiorek82]- "The Theory and the Practice of Reliable System Design"; Daniel P. Siewiorek, Robert Swartz; 1982; Digital Press.

[Spectrum89]- "Intel's Secret is Out"; IEEE SPECTRUM, April 1989, pg22.

[Stone87]- "High-Performance Computer Architecture"; Harold S. Stone; 1987; Addison-Wesley Publishing Company.

[Texas86a]- "TMS44C251 256Kx4 VRAM (Data sheet)"; Texas Instruments, CMOS, 1986, pg 4-79.

[Texas86b]- "TMS44C256/7 256Kx4 DRAM (Data sheet)"; Texas Instruments, CMOS, 1986, pg 4-79.

[Time85] -"A Sleek, Superpowered Machine"; Time magazine International Edition; june 17,1985.

[Tuazon88]-"Mark IIIfp Hypercube Concorrent Processor Architecture"; J. Tuazon, et al.; The Third Conference on Hypercube Concurrent Computers and Applications, Vol 1, 1988, pg 71.

[VLSI86]- "Wait-State Remover Improves System Performance"; VLSI, November 1986, Vol VII, Num 11.

[Weitek87a]- "XL-Series Overview"; Weitek; may 1987.

[Weitek87b]- "XL-Series Hardware Designer's Guide"; Weitek; September 1987.

[Weitek87c]- "XL-8136 32-Bit Program Sequencing Unit"; Weitek; July 1987.

[Weitek87d]- "XL-8137 32-Bit Integer Processing Unit"; Weitek; July 1987.

[Weitek88]- "WTL3164/XL-3164, WTL3364/XL-3364 64-Bit Data Path Units"; Weitek; april 1988.

[Weste85] - "Principles of CMOS VLSI Design A systems Perspective"; Neil H. E. Weste, Kamram Eshraghian; Addison-Wesley; 1985.

[Wu89] - "Computer-Aided Programming for Message-Passing Systems: Problems and a Solution"; Min-You Wu, Daniel D. Gajski; Proceedings of the IEEE, Vol 77, Num 12, Dec. 1989, pg 1983.

[Zhang91]- "System Effects of Interprocessor Communication Latency in Multicomputers"; Xiaodong Zhang; IEEE MICRO, April 1991, pg 12.

APÊNDICE A :
MEDIDAS DA SEGUNDA ETAPA

Tabela 1 : Tempos (ms) X Taxa de I/O (rfsh rate) (+cache +near)

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	<--rfsh rate
270	270	271	272	273	275	274	276	277	277	280	290	297	305	433	bininv
148	148	148	148	147	147	148	148	149	149	149	148	150	150	153	gera_ww
807	802	801	812	803	806	811	817	831	834	825	846	861	885	1197	fft_df ifft_dt
437	437	432	438	438	439	441	445	454	448	453	467	476	509	666	fft_df
342	341	346	348	343	342	346	345	350	363	348	354	360	354	501	fft_dt
788	786	779	800	791	783	800	804	820	822	816	827	861	908	1260	fft_df ifft_dt (ww)
441	433	434	434	435	435	442	446	458	442	456	466	490	532	736	fft_ww_df
326	331	327	342	335	328	335	335	342	358	339	342	346	353	507	fft_ww_dt
970	973	967	985	1000	986	993	1018	1001	1046	1075	1073	1187	1190	2040	fft_df ifft_dt (S)
630	633	632	642	638	643	647	672	649	703	709	722	816	813	1594	fft_S_df
319	316	317	322	337	321	320	324	329	322	343	335	350	354	411	fft_S_dt
463	458	455	460	458	461	463	467	480	469	478	490	500	531	686	ifft_df
362	362	366	370	364	362	367	368	371	383	368	375	381	378	530	ifft_dt
550	550	550	549	550	550	549	550	551	558	551	551	551	564	557	preparainv(/)
208	201	204	206	208	203	200	222	209	203	205	201	210	234	303	preparainv(*)

Tabela 2 : Tempos (ms) X Taxa de I/O (rfsh rate) (-cache +near)

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	<--rfsh rate
404	406	405	403	406	409	411	416	419	427	427	443	461	514	579	bininv
331	332	333	346	329	328	338	341	345	352	350	360	378	406	484	gera_ww
1721	1724	1733	1733	1748	1758	1770	1783	1803	1795	1827	1897	1969	2109	2569	fft_df ifft_dt
987	988	989	993	1006	1005	1017	1024	1034	1030	1051	1104	1131	1229	1485	fft_df
706	708	715	712	713	722	725	729	741	735	744	766	807	848	1050	fft_dt
1682	1682	1699	1697	1709	1724	1730	1745	1760	1748	1783	1854	1938	2053	2516	fft_df ifft_dt (ww)
967	970	978	979	988	987	999	1010	1013	1008	1027	1082	1109	1196	1462	fft_ww_df
686	683	691	690	693	707	704	707	718	711	723	744	797	824	1020	fft_ww_dt
1501	1507	1507	1505	1526	1524	1531	1559	1587	1600	1621	1665	1638	1887	2161	fft_df ifft_dt (S)
744	747	748	740	755	757	759	769	791	795	802	833	803	928	1080	fft_S_df
729	733	729	737	742	738	743	760	768	777	787	805	803	926	1047	fft_S_dt
1015	1015	1019	1021	1034	1036	1045	1054	1062	1059	1083	1132	1163	1262	1519	ifft_df
735	736	745	740	742	751	753	759	769	764	776	793	839	881	1083	ifft_dt
649	648	662	701	648	660	676	649	650	669	649	676	650	651	762	preparainv(/)
288	276	296	281	279	297	278	293	289	285	324	278	325	326	332	preparainv(*)

Tabela 3 : Tempos (ms) X Taxa de I/O (rfsh rate) (+cache -near)

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	<--rfsh rate
393	393	394	396	397	399	401	401	404	409	414	423	436	461	538	bininv
147	147	147	147	147	147	147	147	148	147	148	147	149	149	150	gera_ww
1143	1141	1149	1148	1153	1160	1161	1165	1167	1177	1194	1205	1258	1300	1516	fft_df ifft_dt
638	631	639	637	640	648	643	651	648	655	675	673	702	742	854	fft_df
469	470	470	472	473	476	480	477	477	484	488	497	514	530	614	fft_dt
1193	1191	1197	1205	1206	1212	1220	1218	1225	1237	1259	1284	1324	1379	1626	fft_df ifft_dt (ww)
695	691	697	701	701	704	709	712	713	715	732	741	774	800	943	fft_ww_df
471	472	473	474	478	479	483	479	481	489	494	502	518	538	635	fft_ww_dt
1757	1752	1764	1763	1776	1783	1797	1810	1831	1847	1887	1931	2010	2169	2601	fft_df ifft_dt (S)
1306	1303	1311	1313	1325	1327	1341	1352	1367	1384	1413	1450	1517	1655	2007	fft_S_df
421	424	423	419	423	424	427	429	432	434	440	449	463	482	553	fft_S_dt
663	657	665	662	667	664	665	682	674	683	702	692	738	764	892	ifft_df
496	498	497	501	503	504	509	505	506	513	517	526	546	563	651	ifft_dt
544	543	544	544	545	545	545	546	546	547	547	547	548	553	563	preparainv(/)
251	251	250	250	252	258	255	252	256	260	262	268	282	300	380	preparainv(*)

Tabela 4 : Tempos (ms) X Taxa de I/O (rfsh rate) (-cache -near)

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	<--rfsh rate
443	443	445	446	448	450	455	457	461	465	472	483	503	537	639	bininv
358	359	362	359	360	363	366	370	375	385	385	411	419	446	545	gera_ww
1811	1825	1824	1835	1833	1841	1857	1887	1912	1931	1956	2011	2031	2286	2667	fft_df ifft_dt
1027	1036	1036	1043	1037	1047	1060	1068	1085	1095	1113	1147	1161	1298	1544	fft_df
749	753	753	756	762	758	762	784	791	798	807	826	827	952	1079	fft_dt
1782	1791	1790	1800	1802	1809	1826	1853	1871	1901	1925	1973	1987	2240	2626	fft_df ifft_dt (ww)
1016	1020	1022	1025	1022	1033	1044	1055	1065	1083	1097	1128	1139	1272	1526	fft_ww_df
732	735	732	739	745	739	746	763	770	781	792	808	806	931	1056	fft_ww_dt
1536	1540	1544	1553	1566	1555	1572	1586	1614	1623	1661	1704	1698	1917	2215	fft_df ifft_dt (S)
768	764	773	775	783	780	786	783	802	799	829	856	850	948	1114	fft_S_df
734	740	736	742	750	740	749	766	774	786	796	811	806	931	1058	fft_S_dt
1062	1071	1072	1078	1072	1083	1096	1103	1121	1133	1148	1184	1203	1335	1588	ifft_df
784	789	788	791	797	793	798	819	827	836	843	864	870	988	1122	ifft_dt
723	727	757	718	725	732	723	732	724	759	741	764	761	762	766	preparainv(/)
347	348	351	352	347	353	362	348	361	380	353	373	416	368	438	preparainv(*)

obs1 : dados da tab. 1 obtidos com cache e com ciclos near (12/4/91).
obs2 : dados da tab. 2 obtidos sem cache e com ciclos near (15/4/91).
obs3 : dados da tab. 3 obtidos com cache e sem ciclos near (16/4/91).
obs4 : dados da tab. 4 obtidos sem cache e sem ciclos near (16/4/91).
obs5 : refresh rate em tics, cada tic de .5 us.
obs6 : tempo (em hexa) de n chamadas (n=10 ou 100) em tics de 1 ms.

Tabela 5 : Tempos X rr (+cache +near +optm(-O-sched))

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	<--rfsh rate
272	273	273	273	275	277	276	277	279	280	281	292	301	307	434	bininv
120	120	121	120	120	121	120	120	120	120	121	121	121	122	122	gera_ww
765	768	764	777	768	765	773	777	794	807	801	805	837	864	1223	fft_df ifft_dt
411	414	414	417	415	412	415	418	431	429	439	440	468	495	689	fft_df
328	330	331	338	333	331	333	333	339	352	337	340	344	343	485	fft_dt
785	794	779	799	794	789	798	803	816	822	817	833	866	901	1311	fft_df ifft_dt (ww)
440	445	437	440	445	441	449	447	460	444	457	476	509	528	800	fft_ww_df
322	325	323	337	326	322	329	330	335	356	334	334	338	348	488	fft_ww_dt
677	674	679	682	702	684	683	694	693	697	714	705	746	769	996	fft_df ifft_dt (S)
348	351	349	351	350	350	354	358	359	357	361	370	391	397	566	fft_S_df
307	307	308	310	329	312	307	311	313	311	333	317	337	342	401	fft_S_dt
432	433	434	436	436	432	437	443	453	450	461	462	488	519	718	ifft_df
350	351	352	358	354	352	355	357	361	373	358	361	366	367	517	ifft_dt
208	204	204	206	210	205	203	229	211	203	205	202	210	231	302	preparainv(/)
185	195	184	182	183	191	183	183	207	185	186	196	205	207	302	preparainv(*)

Tabela 6 : Tempos X rr (+cache -near +optm(-O-sched))

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	<--rfsh rate
395	395	396	398	399	401	403	403	406	411	416	424	437	463	539	bininv
121	122	122	121	121	121	121	121	122	121	122	122	122	122	123	gera_ww
1083	1087	1091	1096	1102	1109	1117	1109	1121	1128	1157	1173	1221	1285	1472	fft_df ifft_dt
598	601	603	604	608	614	616	618	628	626	645	661	689	733	831	fft_df
451	453	452	454	455	458	461	457	459	464	471	478	495	515	590	fft_dt
1168	1173	1171	1171	1181	1191	1196	1191	1198	1206	1227	1252	1299	1370	1611	fft_df ifft_dt (ww)
661	666	670	673	672	676	679	679	693	682	698	711	746	783	922	fft_ww_df
468	468	468	470	474	477	480	480	478	487	488	502	520	542	632	fft_ww_dt
923	927	929	923	929	936	937	942	950	953	966	986	1020	1064	1227	fft_df ifft_dt (S)
482	483	485	485	486	490	489	494	496	502	506	517	531	561	642	fft_S_df
412	414	416	409	413	416	417	419	423	420	429	440	453	465	537	fft_S_dt
625	628	631	633	639	642	643	644	656	655	677	686	720	760	870	ifft_df
478	480	479	482	482	487	488	485	488	493	497	506	526	547	624	ifft_dt
250	250	253	253	254	261	254	252	255	260	262	268	283	303	375	preparainv(/)
250	253	252	253	256	258	255	255	254	263	261	270	280	307	377	preparainv(*)

Tabela 7 : Tempos X rr (+cache +near +optm(-O))

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	<--rfsh rate
270	270	271	272	272	275	274	275	277	279	280	290	297	304	433	bininv
121	121	121	120	121	120	121	121	121	120	121	120	121	122	122	gera_ww
779	774	771	787	781	776	788	792	804	808	803	809	833	876	1213	fft_df ifft_dt
430	425	423	427	428	427	431	436	446	439	443	451	467	498	688	fft_df
326	326	329	336	331	327	331	330	335	344	337	332	342	345	488	fft_dt
788	785	779	799	791	788	797	804	817	829	819	826	858	908	1235	fft_df ifft_dt (ww)
439	432	428	437	433	435	440	445	454	444	461	466	490	526	694	fft_ww_df
322	327	325	339	329	324	330	331	336	356	335	339	340	350	494	fft_ww_dt
850	856	860	858	878	858	861	880	883	898	913	941	998	1039	1624	fft_df ifft_dt (S)
510	514	517	516	523	516	523	533	532	553	545	586	628	664	1186	fft_S_df
318	322	319	323	336	322	319	322	329	323	344	332	349	353	410	fft_S_dt
451	447	445	449	448	449	453	459	465	461	465	471	490	524	719	ifft_df
347	347	349	356	352	348	351	353	356	364	358	353	364	368	518	ifft_dt
208	203	204	206	210	203	200	223	211	202	207	202	211	228	304	preparainv(/)
208	202	204	205	206	203	205	218	211	200	212	203	212	223	304	preparainv(*)

Tabela 8 : Tempos X rr (+cache -near +optm(-O))

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	<--rfsh rate
392	394	395	395	396	398	401	401	404	408	414	423	436	460	539	bininv
121	121	122	122	122	122	121	121	121	122	122	122	122	122	124	gera_ww
1114	1109	1110	1116	1124	1126	1137	1129	1138	1149	1166	1182	1225	1277	1488	fft_df ifft_dt
633	620	624	627	636	631	643	633	648	651	667	670	701	733	867	fft_df
450	450	451	452	454	457	461	457	458	460	465	478	492	514	587	fft_dt
1164	1169	1174	1173	1182	1187	1194	1202	1211	1220	1233	1265	1326	1367	1627	fft_df ifft_dt (ww)
663	661	666	668	674	674	678	689	697	697	709	729	772	787	944	fft_ww_df
468	470	471	472	475	477	482	479	479	485	489	500	519	540	638	fft_ww_dt
1334	1341	1343	1344	1349	1362	1368	1372	1391	1402	1423	1465	1526	1623	1920	fft_df ifft_dt (S)
885	889	891	894	897	906	910	913	928	940	953	989	1029	1106	1328	fft_S_df
422	424	423	420	423	425	427	428	433	434	440	449	463	483	552	fft_S_dt
662	657	654	655	659	655	671	667	679	675	699	690	724	762	898	ifft_df
478	478	479	482	483	485	489	484	487	490	495	508	522	545	623	ifft_dt
250	253	251	253	256	259	253	254	260	258	262	265	281	303	379	preparainv(/)
251	253	253	250	256	259	253	252	257	259	261	268	280	304	375	preparainv(*)

obs7 : Testes com opcoes de otimizacao do compilador -O e -sched.
obs8 : dados da tab. 5a obtidos com cache e com ciclos near (23/4/91).
obs9 : dados da tab. 7a obtidos com cache e sem ciclos near (23/4/91).
obs10: Testes com opcoes de otimizacao do compilador -O.
obs11: dados da tab. 9 obtidos com cache e com ciclos near (23/4/91).
obs12: dados da tab. 10 obtidos com cache e sem ciclos near (23/4/91).