UM SISTEMA MULTIPROCESSADOR DE IMAGENS UTILIZANDO PROCESSADORES DIGITAIS DE SINAIS

Humberto Ferasoli Filho (1)
Renê Pegoraro (1)
Marcelo Nicoletti Franchin (1)
José Hiroki Saito (2)

RESUMO

Este trabalho apresenta um sistema multiprocessador voltado ao processamento de imagens, utilizando processadores digitais de sinais TMS320C30 da Texas Instruments, incluindo os elementos de hardware e software necessários para o seu funcionamento. Alguns aspectos de análise de desempenho são apresentados através de resultados obtidos por simulação do sistema executando a transformada de Fourier bidimensional.

ABSTRACT

This work presents a image multiprocessor system, using Texas Instruments's TMS320C30 digital signal processors, including the hardware and software elements needed for its operation. Some aspects on performance analysis are presented through results obtained from system simulation running the bidimensional Fourier transform.

(1)
Mestres em Ciência da Computação - UFSCar
Professores da UNESP - Departamento de Computação
UNESP
Cx.P.473 - CEP 17033-360 - Bauru SP
Tel: (0142) 302111 R.124 Fax: (0142) 344470
Email: uebrudc@brfapesp.ansp.br

(2)
Doutor em Engenharia Elétrica - POLI-USP
Professor da UFSCar - Departamento de Computação
UFSCar
Cx.P.676 - CEP 13565-905 - São Carlos SP
Tel: (0162) 748133 Fax: (0162) 712081
Email: ufscarc@brfapesp.ansp.br

1. INTRODUÇÃO

O processamento de imagens mostra-se uma interessante área de pesquisa, não apenas pela ampla gama de aplicações, mas também pelo desafio de manusear uma grande quantidade de dados com velocidade. O tratamento dos dados de imagens envolvem operações aritméticas específicas, como as transformadas de Fourier que são utilizadas em muitas destas aplicações.

O crescimento da tecnologia VLSI traz processadores cada vez mais poderosos e dotados de recursos de hardware embutidos, sempre com o intuito de melhorar a velocidade e conseqüentemente o desempenho. Entretanto, esses processadores são concebidos para desempenhar tarefas de propósitos gerais, norteados, talvez, pelas necessidades impostas pelo mercado, tornando-os inviáveis em muitas aplicações de fins específicos.

O processamento paralelo apresenta-se como alternativa racional de otimização do desempenho de algoritmos que podem ser operados concorrentemente. As arquiteturas monoprocessadoras, apesar de todo avanço tecnológico, com a crescente introdução de módulos de operações paralelas, internamente ao "chip", não vinham até pouco tempo recebendo dos fabricantes funções que as credenciassem a trabalhar em arquiteturas multiprocessadoras.

As arquiteturas paralelas proliferam-se carregando consigo problemas como otimização de recursos computacionais (processadores, memórias, conexões, etc.), quantização de concorrência, de "pipeline", etc.. As arquiteturas voltadas para fins específicos sofrem dos mesmos problemas acrescidos dos diferentes modelos computacionais, usados no processamento digital de imagem, relacionados ao desenvolvimento tecnológico versus as operações básicas da implementação.

Os processadores digitais de sinais (DSPs) são, essencialmente, microprocessadores/microcomputadores de alta velocidade, projetados especificamente para desempenhar computação intensiva de algoritmos de processamento digital de sinais. Utilizando as vantagens das arquiteturas avançadas, como processamento paralelo, e do conjunto de instruções DSP dedicados, estes dispositivos podem executar milhões de

operações DSP por segundo. Esta capacidade permite que algoritmos DSPs complicados possam ser implementados numa minúscula pastilha de silício, o qual requeria anteriormente o uso de minicomputadores ou até arranjos de processadores.

O fato do processamento de imagem trabalhar com grande quantidade de dados em um tempo razoavelmente pequeno, extrapola os limites de arquiteturas monoprocessadoras, mesmo que estas utilizem-se dos recursos descritos anteriormente em DSPs. Portanto, devido a tais características, a exploração de arquiteturas concorrentes apresenta-se como alternativa única, até o momento, para satisfazer tais exigências.

Estas tendências levaram a diversos desenvolvimentos de arquiteturas paralelas. Entre elas uma arquitetura paralela específica utilizando DSPs por Ferasoli [FERA92], na Universidade Federal de São Carlos (UFSCar).

Este trabalho apresenta esta arquitetura e os recursos de software necessários à sua utilização. Estes recursos representam meios de comunicação inter-processos, comunicação/controle feito pelo hospedeiro, bem como algumas ferramentas que auxiliam no desenvolvimento, verificação e correção de erros de algoritmos a serem implementados.

As ferramentas implementadas incluem um montador para o TMS320C30 com diretivas específicas à comunicação inter-processos, que aplicam exclusão mútua de forma simples e segura através de monitores. Um simulador para a arquitetura foi, também, desenvolvido para permitir a validação tanto dos softwares gerados como também da arquitetura como um todo, e é descrito em [PEGO93].

2. PROCESSADORES DIGITAIS DE SINAIS

Os processadores digitais de sinais são microprocessadores projetados para desempenhar operações matemáticas complexas do domínio analógico, utilizando hardware dedicado, ao invés de rotinas de software, para executar suas funções.

Os processadores digitais de sinais apareceram no final da década de 70 com o DSP1 da Bell Labs e o NEC 7720. Desde então muitos fabricantes entraram no mercado com componentes cada vez mais sofisticados. Uma relação destes fabricantes com as principais características destes componentes é apresentado em [LEE88], mostrando a evolução tecnológica e a seqüência histórica de introdução ao mercado.

Dentro desta evolução percebemos o pouco esforço em projetar DSPs dirigidos a computação paralela. Poucas são as características, a nível de hardware ou software, para facilitar as tarefas de sincronização de processadores ou acessos a recursos compartilhados.

Os processadores mais recentes oferecem algumas características voltadas ao processamento paralelo, como notado no TMS320C30 da TEXAS com a sua função de "interlock" e no seu sucessor, lançado durante o desenvolvimento deste projeto, o TMS320C40 com caracteríticas ainda mais avançadas e com ampla aplicação à arquiteturas paralelas, sem a necessidade de adequações com circuitos adicionais. A tendência dominante nos DSPs é o crescimento da complexidade e não da simplicidade.

A real vantagem advinda dos algoritmos DSP está na estabilidade, programabilidade, reprodutibilidade, eficiência, além de tratamentos que não dispõem de equivalência analógica.

Sempre que discutimos processamento digital de imagens algumas considerações são lembradas: A quantização e a amostragem de uma imagem geram pelo menos uma matriz muito grande (no mínimo da ordem de 10^5 pixels) de reais ou complexos e qualquer processamento sobre esse volume de dados exige um alto desempenho computacional com características DSP.

3. A Arquitetura

O projeto da arquitetura foi desenvolvido para fins específicos de processamento de imagens. Esta arquitetura MIMD, segundo a taxonomia de Flynn, está utilizando o DSP TMS320C30 da Texas Instruments, conectados através de uma barra a um sistema de memória compartilhada e a um microcomputador hospedeiro via

interface, também conectada à barra. A concorrência à barra ficará controlada por um sistema de arbitração que efetuará a manipulação das disputas.

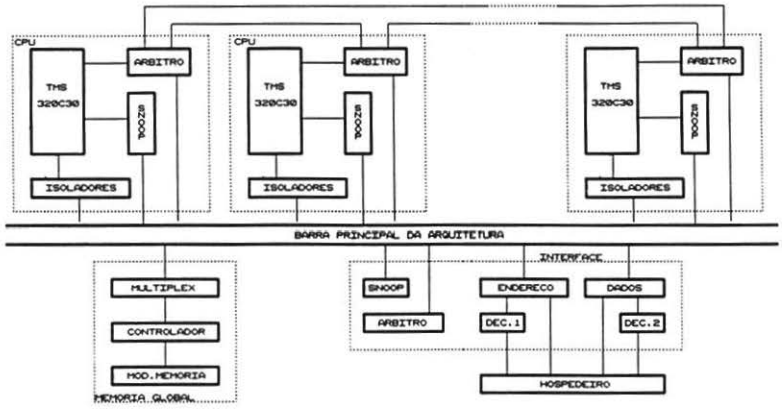

Diagrama em Blocos da Arquitetura

O sistema de arbitração é responsável por uma das tarefas mais nobres dentro de um sistema multiprocessador com memória compartilhada. Dele depende a justiça e a prioridade nas decisões das disputas em situações onde acessos múltiplos à memória global compartilhada ocorrem [LEVI87].

O sistema de arbitração utiliza o "daisy-chain" rotativo, pela integração observada da política oferecida pelo árbitro em relação às características da aplicação desejada. Este sistema é distribuído por cada módulo processador, facilitando a modularidade sobre este aspecto.

Quando o sistema é iniciado, o hospedeiro ganha a maior prioridade frente ao árbitro, devido aos aspectos funcionais. Isto acontece, sincronamente, pelo sinal de RESET propagado pela arquitetura. Após isto as concessões se normalizarão.

Concluído um acesso, o módulo recebe a menor prioridade, implicando em uma maior prioridade ao módulo seguinte. A secção principal do sistema de arbitração gera um laço de passagem de permissão indicando, em caso de contendas, qual o processador tem maior prioridade.

Com o advento do VLSI com componentes em torno de 1 micrômetro, encontramos microprocessadores de alto desempenho que já incorporam memória local interna ao "chip" que não foram desenvolvidos para operar, primariamente, em arquiteturas multiprocessadoras e que normalmente não tem mecanismos de gerenciamento de controle de consistência, como os existentes nas memórias "cache", limitando a troca de dados diretamente através da memória global.

Assim, se considerarmos uma arquitetura em barra compartilhada, com processadores com memória local interna, sem mecanismos de consistência com a memória global, a troca de mensagens poderia tornar-se degradante ao sistema, visto que o processador esperando pelos dados deveria fazer múltiplos acessos à memória, através da barra compartilhada, para verificar a disponibilidade ou não dos dados a serem recebidos, aumentando os acessos ao barramento e portanto, aumentando o tempo de espera dos processadores em acessos a memória global compartilhada.

Para minimizar este problema foi implementado um dispositivo, denominado "snoop", que uma vez programado pelo processador local, verifica um endereço da memória compartilhada, informando a este processador a ocorrência de acesso no endereço especificado.

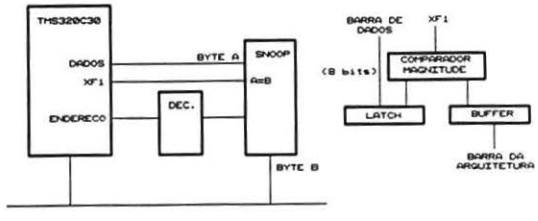

Diagrama em Bloco do Sistema de "snoop"

O sistema de "snoop" está ligado ao processador pela barra de periféricos, mapeado como dispositivo de E/S. Os

comparadores de magnitude comparam as linhas de endereço da arquitetura com o byte armazenado no "latch" deste sistema. Quando uma igualdade é estabelecida, o comparador sinaliza, permitindo a utilização de instruções de intertravamento, disponíveis internamente ao processador.

O TMS320C3(é composto por duas interfaces externas: a barra primária e a barra de expansão. O módulo da CPU está ligado, a barra da arquitetura, pela barra primária. A barra primária contém 32 bits de dados, 24 bits de endereço e um conjunto de sinais de controle. Obviamente, num sistema compartilhado, enquanto um módulo de CPU faz acessos à memória compartilhada os demais módulos da arquitetura permanecem isolados.

O TMS320C30 por não ser projetado para o uso direto em arquiteturas multiprocessadoras fortemente acopladas, não permite a isolação externamente controlada de suas interfaces internas. Esta limitação obrigou o uso de isoladores discretos.

O sistema de "clock" é centralizado para evitar lógicas de sincronização quando da interação entre módulos da arquitetura. O sistema de RESET também é centralizado.

O módulo de interfaceamento está dividido em quatro partes: o sistema de multiplexação de dados, o sistema de decodificação de endereços, o sistema de arbitração e o sistema de "snoop".

O sistema de multiplexação tem como objetivo compatibilizar a barra de 32 bits de dados da arquitetura paralela com a barra de 8 bits de dados do computador hospedeiro.

O sistema de decodificação mapeia a interface dentro de áreas livres como memória e como dispositivos de entrada/saída do computador PC-XT ou compatíveis. O hospedeiro utiliza uma janela de 64K para obter acesso a toda área de memória da arquitetura.

A área de dispositivos de entrada/saída utilizada é um espaço livre para expansões. Este espaço de endereço, no

projeto, serve para preparar as páginas de acesso, o tratamento do "snoop" e arbitração.

O computador hospedeiro, similarmente aos processadores, faz acessos à arquitetura paralela através de um módulo do sistema de arbitração.

O dado monitorado é parte de um endereço de uma variável compartilhada, armazenado num "latch" e comparado à barra de endereços da arquitetura, indicando as igualdades. Para o hospedeiro o sistema de "snoop" também serve para indicar os términos de tarefas pela arquitetura.

4. O SOFTWARE

Na implementação dos softwares utilizou-se a linguagem Pascal para o desenvolvimento no ambiente do microcomputador hospedeiro IBM PC com sistema operacional MS-DOS. Já no ambiente da arquitetura paralela, os códigos são gerados pelo montador e carregados através do hospedeiro.

Os processadores da arquitetura atuam basicamente na memória compartilhada, portanto, para que os dados possam ser processados, devem ser carregados por um outro processador (o hospedeiro), que seja capaz de fazer acessos aos dispositivos de entrada/saída e à memória da arquitetura.

No desenvolvimento de um processamento na arquitetura o hospedeiro pode obter informações, gerar dados complementares, obter resultados intermediários e detectar o fim deste processamento. A arquitetura dispõe de meios que capacitam o hospedeiro, de forma análoga a um dos outros processadores, a fazer acessos ao barramento através do árbitro, permitindo a manipulação da memória durante o processamento em um algoritmo. É possível ainda, a utilização pelo hospedeiro de um dispositivo, já discutido, denominado "snoop", que pode indicar acessos ocorridos a um conjunto de posições específicas de memória, sem que o barramento seja obstruído por acessos, através de um pedido e concessão do árbitro.

O paralelismo da execução (escolha das seções paralelas do software), a implementação dos algoritmos e os meios para que

cada processador "saiba" o que executar, fica totalmente a cargo do programador.

Um dos esquemas de cooperação inter-processos, que menos depende da atenção do programador para gerar um código livre de erros, é o monitor [TANE87]. Para permitir a utilização de monitores, sem a necessidade de implementação de uma linguagem de alto nível, utilizou-se um pré-montador que permite o uso de diretivas especiais para a implementação de monitores.

As diretivas para este pré-montador e que implementam o monitor são:

- **.monitor** - Inicia uma área onde todas as rotinas implementadas com a diretiva ".proc" tenham em substituição, a esta diretiva, um código extra que implemente a exclusão mútua,
- **.endmonitor** - Termina a área de um monitor,
- **.proc** - Inicia uma área de definição de um procedimento e
- **.endproc** - Termina a área de definição de um procedimento. Se esta diretiva é utilizada dentro de um monitor, ela é substituída por códigos indicando o término de utilização de uma região crítica.

Quando em uma linguagem são implementados monitores, ela deve utilizar um método de comunicação inter-processos para controlar o acesso às regiões críticas. Aqui, optou-se pela utilização do "Test-and-SeT" (TST), por ser simples de implementar, seguro e fácil de utilizar em arquiteturas com memória compartilhada. O processador TMS320C30 não tem uma instrução específica de TST, porém pode-se utilizar da instrução paralela **LDI||STI** que apresenta um funcionamento similar [TEXA88][TEXA90].

Para que o barramento seja melhor aproveitado, apenas recebendo acessos úteis, o esquema de "snoop" de barramento, é empregado pelo software. Isto implica em uma espera ocupada, realizada por uma instrução com "interlock", que aguarda a ocorrência de uma operação, por outro processador, no endereço de uma variável de trava, que quando alterada, libera o processador corrente. Após a liberação, este verifica através da instrução TST se a variável está realmente liberada. Se

afirmativo continua a execução da rotina, senão, volta a espera ocupada no "interlock" repetindo o ciclo.

Para implementar os códigos extras, que substituem as diretivas ".proc" e ".endproc" após a pré-montagem, dentro de um monitor será criada uma variável de trava única, para cada diretiva ".monitor" usada.

Implementação de monitores pelo pré-montador

```
Antes do pré-montador:          |       Após o pré-montador:
                                |
Rotina1         .proc           |   rotina1:
        .                       |           PUSH      AR0
        .                       |           PUSH      DP
        .                       |           LDP       $Flag
        ;outras instruções      |           LDI       @$Flag, AR0       ;AR0 recebe o endereço de Flag
        .                       |           CALL      EntraRC
        .                       |           POP       DP
        .                       |           POP       AR0
        .endproc                |           .
                                |           .
                                |           .
                                |           ;outras instruções
                                |           .
                                |           .
                                |           .
                                |           PUSH      AR0
                                |           PUSH      DP
                                |           LDP       $Flag
                                |           LDI       @$Flag, AR0       ;AR0 recebe o endereço de Flag
                                |           CALL      SaiRC
                                |           POP       DP
                                |           POP       AR0
                                |           RETS
                                |
                                |   $Flag   .WORD     Flag
                                |   Flag    .WORD     0                 ;Variáveis trava regiões críticas
```

Note que as rotinas **EntraRC** e **SaiRC** são definidas pelo pré-montador e são responsáveis pelo controle das regiões críticas.

5. Ferramentas Desenvolvidas

Para que a implementação do software básico fosse feita com sucesso, algumas ferramentas foram criadas. A necessidade que levou a isto foi o alto preço da aquisição destas ferramentas e a dificuldade de adaptação destas aos requisitos impostos pelo nosso projeto.

Basicamente, duas ferramentas foram criadas para simplificar a implementação e verificação do software básico: um simulador e um montador, ambos para o TMS320C30.

5.1 O Simulador para o TMS320C30 e para a Arquitetura

Neste trabalho utilizamos um simulador para verificar nossos algoritmos de comunicação inter-processos, para contabilizar as vantagens de ganho de tempo na arquitetura proposta e para simplificar e verificar programas exemplos quanto aos resultados e tempo gasto pela arquitetura para executar estes algoritmos.

Para obtermos resultados comparativos entre a arquitetura em um único processador, implementou-se um simulador a nível de instruções capaz de simular tanto um, quanto a arquitetura com até 9 processadores TMS320C30.

O simulador tem as seguintes características:

- Simulação de todas as instruções do TMS320C30
- Simulação do DMA para a carga e descarga dos blocos de memória de dados internos dos processadores
- Simula a utilização de cache
- Os modos de execução incluem:
  - Instrução a instrução
  - Execução direta até um ponto de parada ou até uma tecla ser pressionada
  - Execução até a posição do cursor
- Para o simulador da arquitetura, temos as limitações impostas pelas contendas ocorridas no barramento
- Contagem dos ciclos, considerando:
  - O estado de espera para memórias externas
  - Conflitos de acessos à memória externa
  - A espera pela liberação do barramento
- Os conflitos de "pipeline" nos processadores não são considerados na contabilização dos ciclos de execução
- Inexistência de dispositivos periféricos ("timers" e portos seriais)

O simulador tem ainda restrições impostas pelo ambiente, sendo as principais limitantes o tempo de execução e a memória disponível nos ambientes onde este é usado.

O simulador foi desenvolvido em três partes funcionais distintas:

- A primeira e mais importante é responsável pela simulação propriamente dita,
- A segunda que implementa a interface com o usuário. Através desta interface as memórias e todos os processadores podem ser vasculhados em seus valores internos e
- A terceira parte é a responsável pela montagem e desmontagem dos códigos de instruções durante uma sessão de simulação.

Em suma, o simulador atua como um processador ou como a arquitetura. Programas em desenvolvimento são executados em um computador IBM-PC simulando as situações que ocorreriam caso aquele código fosse utilizado em um dispositivo real. Execuções com paradas controladas pelo usuário podem ser facilmente utilizadas. Assim, seções que causam maus funcionamentos, laços infinitos ou "dead-locks" podem ser mais facilmente detectadas.

5.2 O Montador para o TMS320C30

O montador transforma arquivos texto em linguagem de montagem para arquivos objeto dos processadores TMS320C30. O arquivo texto pode conter mnemônicos de instruções e diretivas de orientação à montagem.

Como o montador tem características genéricas para a programação de qualquer arquitetura, utilizando TMS320C30, é necessário a implementação de um pré-montador que insira no arquivo, a ser montado, informações necessárias ao funcionamento correto da arquitetura.

O pré-montador, como dito, implementa outras diretivas capazes de criar monitores que controlem de forma ordenada, e mais segura, os acessos às regiões críticas feitos pelos processadores da arquitetura.

6. CASO PRÁTICO: Transformada de Fourier Bidimensional

Para podermos avaliar o desempenho do sistema em uma aplicação prática diversos métodos foram testados. Um algoritmo que nos parece muito importante, pela sua ampla utilização em processamento de imagens, é a transformada de Fourier.

A equação da transformada unidimensional de Fourier, tanto contínua como discreta, é apresentada em muitas publicações, entre elas [BALL82][GONZ87][HIGG90].

Gonzalez [GONZ87] nos mostra um meio para calcular a transformada bidimensional de Fourier. O processo está na aplicação da transformada unidimensional de Fourier a todas as linhas e em seqüência a todas as colunas. O algoritmo destas transformadas unidimensionais foi extraído de [TEXA88][TEXA90].

Nesta implementação utilizou-se imagens de 16x16, 32x32 e 64x64 de pixels complexos. Estas resoluções são baixas porém tornam viável a execução, no simulador, em tempo razoável, sem comprometer os resultados que permanecem representativos apesar desta restrição.

Pela simulação do algoritmo seqüencial nas diversas resoluções obtivemos os ciclos de execução: 16x16: 24395 ciclos; 32x32: 100811 ciclos e 64x64: 428619 ciclos.

O algoritmo paralelo utilizado é semelhante ao seqüencial, tendo como única diferença a forma de controle das linhas e colunas a transformar que, na memória compartilhada, indicam qual o próximo processamento a ser executado.

Tabela de ciclos e "speed-up" na execução da transformada de Fourier paralela

Número de	Ciclos			"Speed-up"		
Processadores	16x16	32x32	64x64	16x16	32x32	64x64
1	27362	106498	439746	0,89	0,95	0,97
2	15367	54853	222145	1,59	1,84	1,93
3	12475	39150	151504	1,96	2,57	2,83
4	12526	33071	117700	1,95	3,05	3,64
5	14256	32856	105598	1,71	3,07	4,06
6	15222	35534	100245	1,60	2,84	4,28
7	18175	39098	102549	1,34	2,58	4,18
8	20286	42539	108542	1,20	2,37	3,95

Os valores apresentados na tabela, obtidos por simulação, mostram o comportamento da arquitetura até a utilização de 8 processadores em várias resoluções, mostrando ainda que com o aumento da resolução consegue-se um maior "speed-up". Esta tabela á representada em forma de histograma do "speed-up".

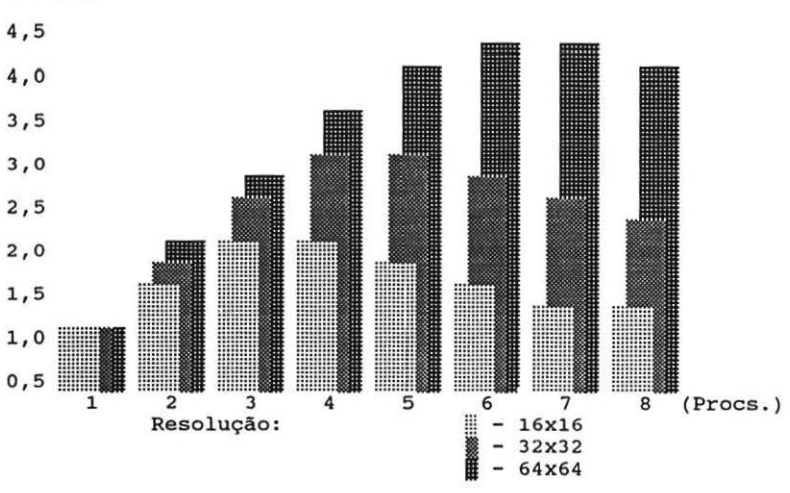

Gráfico de saturação da arquitetura para o algoritmo

O aumento do "speed-up", na realidade não ocorre pelo aumento da resolução e sim pelo aumento da relação entre o número de ciclos para o processamento do algoritmo propriamente dito e o número de ciclos para a carga e armazenamento dos dados na memória interna.

CONSIDERAÇÕES FINAIS

Dos vários métodos propostos, em inúmeros trabalhos técnicos, para a cooperação inter-processos, apresentamos um esquema de monitor simplificado que permite a troca de dados, entre os processadores, de forma simples e segura, implementada em linguagem de montagem através de diretivas especiais do montador.

O simulador e o montador implementados são de grande importância não só na utilização para a arquitetura, mas como uma bancada para experiências em programação paralela e também para auxiliar na criação e depuração de códigos para o TMS320C30 em qualquer outro ambiente monoprocessador.

A arquitetura utilizada traz vantagens principalmente pela sua simplicidade e custos acessíveis. Sobre o ponto de vista de hardware a arquitetura encontra-se em fase de montagem com dois módulos processadores. Testes preliminares foram realizados através do desenvolvimento de uma placa de avaliação com este processador.

Mesmo este sistema tendo sido desenvolvido e dirigido ao processamento de imagens, ele não é de forma alguma restrito, podendo ser utilizado em qualquer tipo de aplicação, desde que admita um algoritmo paralelo na sua implementação.

Este trabalho abrirá uma ampla gama de possibilidades para novas pesquisas na implementação, teste e validação de algoritmos paralelos com uma velocidade de processamento bastante alta, por volta de 120MFLOPS de pico (utilizando 4 processadores). Possibilitará ainda em fases posteriores, outros estudos e melhorias na estrutura e utilização do software básico.

REFERÊNCIAS BIBLIOGRÁFICAS

[BALL82] Ballard, D. H. & Brown, C. M., Computer Vision, Prentice-Hall International, 1982

[FERA92] Ferasoli Fº, Humberto, Projeto de uma Arquitetura Paralela Explorando Processadores Digitais de Sinais (DSPs) Direcionados para Aplicações de Processamento de Imagens, São Carlos, Universidade Federal de São Carlos, 1992. 110p. (Tese)

[FREE87] Freer, J. R., Systems Design with Advanced Microprocessors, Pitman Publishing, 1987

[GONZ87] Gonzalez, Rafael C., Digital Image Processing, Addison-Wesley Publishing Company, 1987

[HIGG90] Higgins, R. J., Digital Signal Processing in VLSI, Prentice-Hall, 1990

[LEE88] Lee, E. A. - Programmable DSP Archtectures: Part 1 and 2 - IEEE ASSP Magazine, october 1988, pp. 4-14-19

[LEVI87] Levialdi, Stefano, Issues on Parallel Algorithms for Image Processing - The Characteristics of Parallel Algorithms, MIT Press, 1987

[PEGO93] Pegoraro, Renê, Suporte de Software Básico para Arquitetura Paralela Específica Destinada ao Processamento de Imagens, São Carlos, Universidade Federal de São Carlos, 1993, 113p. (Tese)

[TANE87] Tanenbaum, Andrew S., OPERATING SYSTEMS: Design and Implementation, Prentice-Hall International, 1987.

[TEXA90] TMS320C3x User's Guide, Texas Instruments, 1990

[TEXA88] Third-Generation TMS320 User's Guide, Texas Instruments, 1988