Artigo

SIBGRAPI'91
IV Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens

PROCESSAMENTO DE IMAGENS EM MÁQUINAS BASEADAS À TRANSPUTER

Roberto de Alencar Lotufo
(DCA-FEE-UNICAMP)

Página em branco na versão original impressa.

PROCESSAMENTO DE IMAGENS EM MÁQUINAS BASEADAS À TRANSPUTER

Roberto de Alencar Lotufo
DCA - FEE - Unicamp
e-mail: Lotufo@bruc.ansp.br

Abstract. This work presents seven parallel programming models for image processing operations. The algorithms uses an implementation based on a virtual concurrent machine with a message passing system. Two experiments are described using up to 65 transputers: the Abingdon Cross Benchmark and the Sobel Edge Detection-Hough Transform algorithm.

1. Introdução

Processamento de Imagens têm sido uma das áreas de aplicações de supercomputadores que utilizam processamento paralelo. O transputer é um microprocessador modular projetado para ser o elemento básico na construção de máquinas MIMD de memória distribuída de granularidade média. Com o transputer, é possível a construção de uma rede de qualquer topologia com o máximo de quatro conexões por nó. A praticidade do uso de transputer para aplicações de processamento de imagens já foi amplamente investigada [1].

Ao contrário de outras máquinas supercomputadores, que normalmente requerem uma estrutura e configuração fixa, as máquinas baseadas em transputer são normalmente disponíveis em uma grande variedade de configurações que vão desde uma instalação consistindo em um computador pessoal com 4 ou 5 processadores com uma topologia restrita, até supercomputadores com mais de 256 processadores com topologia programável, como o Computing Surface da Meiko ou SN10000 da Parsys Ltd. Como o software de algoritmos paralelo está ainda num estágio prematuro, envolvendo altos custos no seu desenvolvimento, é desejável que um mesmo software possa rodar em qualquer máquina multiprocessadora baseada à transputer, independentemente de sua configuração particular.

Uma solução que permite essa facilidade consiste desenvolver os programas na rede de transputers modelando-a como uma máquina concorrente virtual baseada em sistema de passagem de mensagens. Neste ambiente, os processadores são vistos pelo programador como uma rede conectada completamente. É responsabilidade do sistema de passagem de mensagens a comunicação de informações entre quaisquer nós independentemente da topologia física da rêde. Sempre haverá uma topologia física ótima para cada programa, mas isto não impede que o mesmo programa possa rodar em qualquer outra configuração em condições sub-ótimas. Este tipo de solução vem sendo usada, entre outros lugares, no Instituto Caltech [2] e na Universidade de Edinburgo [3]. É importante notar que historicamente o transputer foi feito para ser programado na linguagem OCCAM sob um ambiente de desenvolvimento que não possui sistema de passagem de mensagens.

Este trabalho relata o uso de um sistema de passagem de mensagens configurado em tempo de execução (chamado HiVe) desenvolvido pelo autor para aplicações de processamento de imagens e visão computacional. Os objetivos deste trabalho são:

- apresentar um estilo de programação paralela baseado em sistemas de passagem de mensagens;
- introduzir diversos padrões de comunicações relacionados com algoritmos paralelos de processamento de imagens;

- expor o desempenho de diversos experimentos envolvendo o benchmark *Abingdon Cross Benchmark* [6] e a transformada de Hough [7].

2. Processamento Paralelo utilizando o Sistema de Passagem de Mensagens HiVe

A maioria dos sistemas de passagem de mensagens provê duas funções básicas de comunicação para o usuário: recepção e transmissão de uma mensagem para um nó específico. Estas operações podem ser bloqueantes ou não-bloqueantes. O modo bloqueante requer sincronização explícita entre o nó receptor e o nó transmissor, no estilo de programação requerido pela linguagem OCCAM. No modo não-bloqueante, a mensagem é despachada para o sistema de passagem de mensagem e pode ser testada no nó receptor caso a mensagem tenha chegado ou não.

HiVe foi implementado com o mecanismo não-bloqueante para transmissão e bloqueante para recepção. A decisão de utilizar modo não-bloqueante na transmissão de mensagens foi para se conseguir maior eficiência nas aplicações de processamento de imagem que requerem vastas quantidades de comunicação entre os nós. Com esta configuração, esquemas de comunicação eficientes podem ser implementados utilizando-se apenas código sequencial a nível de programação pelo usuário. A implementação do modo não-bloqueante exige um número de *buffers* no sistema de passagem de mensagens. HiVe permite comunicação nó a nó e comunicação tipo *broadcast*.

Um programa aplicativo que utilize HiVe é carregado em tempo de execução. Existe um arquivo de configuração, que é lido no ínicio da execução da aplicação. Este arquivo de configuração contém o número de processadores na rede, o nome do programa a ser carregado em cada nó e a topologia da rede. A aplicação quando em execução tem acesso ao número total de processadores na rede. Assim é possível escrever programas paralelos cujo código executável possa rodar em diversas topologias e diversos número de processadores.

As duas funções básicas de comunicação em HiVe são (HiVe é programado na linguagem C):

`hive_tx(int` *tag*`, int` *destination*`, char *`*buffer*`, int` *size*`)`

`hive_rx(int` *tag*`, int *`*from*`, char *`*buffer*`, int *`*size*`)`

hive_tx transmite uma mensagem de etiqueta *tag* apontada por *buffer* de tamanho *size* ao processador de número *destination*. Esta função despacha a mensagem no modo não-bloqueante ao sistema de passagem de mensagem e retorna.

hive_rx espera pela chegada de uma mensagem de etiqueta *tag*. A mensagem é retornada apontada por *buffer* com *size* bytes e indica o processador onde a mensagem foi originada em *from*. Existe um *tag* especial que permite a recepção de mensagens de qualquer etiqueta.

O uso de etiquetas associadas à mensagens simplifica a programação de alguns padrões de comunicação, como por exemplo troca de bordas sobrepostas entre processadores vizinhos e combinações globais descritas na próxima seção.

3. Algoritmos Paralelos de Processamento de Imagens Para Máquinas de Grão Médio.

A programação de máquinas distribuídas de grão médio não podem ser feita automaticamente a partir de algoritmos sequenciais já existentes. O programador deve ter em mente as características particulares da máquina de modo a projetar um programa que faça uso eficiente dos processadores

disponíveis na rede. Uma solução que facilita a tarefa do programador destas máquinas paralelas consiste no desenvolvimento de pacotes de decomposição aplicados às várias classes de problemas [2,4].

Este trabalho identifica sete padrões de comunicações comumente usados em processamento de imagem:

- computação local
- operador de vizinhança
- concorrência entre processamento e entrada/saída de dados
- combinação global
- dividir para conquistar
- *processor farm*
- anel *pipeline*

Segue uma breve descrição de cada um desses padrões:

computação local Este é o caso mais simples onde cada nó individual não requer comunicação entre outro nós. O processamento envolve apenas os dados contidos dentro do nó. Operações de *threshold* e transformações de pixels são exemplos típicos de computação local.

operador de vizinhança Este operador processa dados locais e dados pertencentes à processadores vizinhos. Algoritmos cujo cálculo do pixel resultante depende do valor dos pixels da sua vizinhança pertencem à essa classe de operadores. A fase de comunicação que existe entre essas operações é denominada **troca de sobreposição**, em inglês *exchange overlap*. Exemplo de operações que utilizam esse modo de comunicação são entre outros: convoluções, deteção de bordas e operadores morfológicos.

concorrência entre processamento e entrada/saída de dados O carregamento e o display dos pixels de uma imagem em uma rede de processadores sempre constitui um componente de overhead no cálculo do tempo de computação total, apesar de ser muitas vezes não considerada na avaliação das arquiteturas voltadas para processamento de imagens. Este padrão de comunicação permite a redução desse overhead, através da execução simultânea da comunicação de dados e seu processamento. O transputer possue oito interfaces d.m.a que permitem uma concorrência eficiente entre e/s e processamento. Os processos que permitem uma concorrência com comunicação são principalmente dados pelos operadores de computação local e de vizinhança.

combinação global Algumas operações requerem cálculos envolvendo dados armazenados em todos os nós. Um exemplo típico consiste na criação de histograma: um histograma local é criado em cada nó que por sua vez são adicionados globalmente, resultando um histograma final em cada nó.

dividir para conquistar Este padrão de comunicação requer uma divisão recursiva do problema até que ele possa ser resolvido em um único processador. A parte final da operação recursiva envolve a combinação dos resultados até que uma resposta final seja obtida. Dividir para conquistar pode ser usado na determinação da intensidade média de uma imagem ou na rotulação de componentes conectados.

processor farm Este modo de computação é comandado pela demanda, exibindo concorrência entre entrada/saída e processamento, e um balanceamento dinâmico automático de carga. O modo processor farm pode ser usado em todas as operações relacionadas aos modos anteriores descritos, porém seu desempenho será maior na sequência carregamento-processamento-display com processamento local ou de vizinhança.

anel pipeline Existem alguns algoritmos sequenciais de varredura linha a linha onde o processamento de um pixel necessita do resultado dos pixels previamente processados (por ex. os da esquerda e os superiores na varredura direita-esquerda e de cima para baixo). Tais algoritmos são mais apropriados à implementações em processadores sequenciais. Existem situações em que a versão sequencial do algoritmo, implementado numa máquina de memória distribuída de grão médio seguindo o modêlo de um anel pipeline exibe um alto nível de eficiência. Miguet [5] descreve uma implementação utilizando este modelo com um speedup de 26 utilizando uma máquina com 32 transputers.

Para ilustrar o uso do sistema de passagem de mensagens na implementação de alguns dos padrões descritos, dois exemplos são mostrados a seguir: o primeiro contém o corpo do procedimento de processor farm e o segundo o de exchange overlap utilizado em operadores de vizinhança:

Processor Farm

Processor Farm é um dos modelos computacionais mais bem sucedidos em aplicações utilizando transputers [6]. Ele tem sido empregado com sucesso em aplicações de processamento de imagens que se beneficiam de suas características de balanceamento dinâmico automático de carga.

Nas figuras 1 e 2 são dados o esqueleto do algoritmo processor farm utilizando HiVe. Assume-se que a imagem é sub-dividida em `number_packets` pacotes de dados. Por simplicidade de descrição, `number_packets` é múltiplo do número de processadores na rede (sem contar o processador Root). O processador Root é identificado pelo processador de número 0. O programa no Root primeiramente envia um pacote inicial para cada processador Worker. A seguir ele envia outro pacote para cada processador que devolver o seu resultado processado. Finalmente, depois de esgotado todos os pacotes a serem processados, o Root espera pela execução final dos pacotes ainda de posse dos Workers. Do ponto de vista dos Workers, eles simplesmente esperam por um pacote, processam-no e transmitem o pacote processado de volta ao Root (Fig. 2).

A topologia ótima para esse tipo de algoritmo consiste no processador Root conectado à quatro árvores ternárias. Nesta topologia a banda de passagem entre o Root e a rede de processadores é maximizada e a distância média entre o Root e cada worker é minimizada.

Existe uma melhoria no desempenho deste algoritmo se o processador Root enviar inicialmente mais de um pacote à cada Worker. Isto garante que os processadores Workers não fiquem esperando para a chegada de novo pacote ao final da transmissão do pacote processado anteriormente.

```
/* exchange vertical boundaries */
 hive_tx(TAG_UP,    up(proc_ID),upper_border,size_border); /* send     */
 hive_tx(TAG_DOWN,down(proc_ID), down_border,size_border);
 hive_rx(TAG_UP  ,&from, down_overlap,&size);              /* transmit */
 hive_rx(TAG_DOWN,&from,upper_overlap,&size);
/* exchange horizontal + corner boundaries */
 hive_tx(TAG_RIGHT,right(proc_ID),right_side,size_side);  /* send     */
 hive_tx(TAG_LEFT , left(proc_ID), left_side,size_side);
 hive_rx(TAG_RIGHT,&from, left_overlap,&size);             /* transmit */
 hive_rx(TAG_LEFT ,&from,right_overlap,&size);
```

Figura 3: Exchange Overlap

As funções **up()**, **down()**, **right()** e **left()** retornam o número de identificação dos processadores vizinhos calculados em tempo de execução. A eficiência na execução deste algoritmo será ótima se a grade virtual for mapeada exatamente sobre uma topologia física. Se a topologia física for diferente da virtual, o programa terá sua execução correta, porém com desempenho sub-ótimo.

4. Experimentos e Medidas de Desempenho

O código usado nos experimentos foi desenvolvido em C utilizando-se 5 transputers: um Root e 4 processadores modelados como uma grade virtual com 4 x 1, 2 x 2 ou 1 x 4 processadores para fins de testes do código de execução. Uma vez testado e desenvolvido, o código executável era carregado remotamente ao Supercomputador Concorrente de Edinburgo (ECS), onde experimentos utilizando até 65 processadores eram executados utilizando-se diferentes arquivos de configuração de topologia.

Desempenho do Sistema HiVe

O sistema de passagem de mensagens HiVe pode ser aproximadamente caracterizado pela equação do atraso envolvido na transmissão e recepção de uma mensagem de comprimento w bytes entre dois nós distantes de n ligações intermediárias:

$$T = a + n(b + cw) \tag{1}$$

Onde a e b representam os tempos que não dependem do tamanho da mensagem transmitida. Eles representam as operações necessárias para a transmissão e a recepção de uma mensagem: passagem de parâmetros, transmissão do cabeçalho de tamanho fixo, tabela de roteamento, etc.. c é o coeficiente relacionado ao tempo de transferência que depende to tamanho da mensagem. Ele reflete a banda de passagem do canal de comunicação e o movimento interno de dados da mensagem, como por exemplo a cópia de buffers.

Os parâmetros a, b e c foram medidos em máquinas diferentes que utilizam o transputer T800 de 20 MHz:

máquina	a	b	c
BOO3 10Mbps	128 us	64 us	1.16 us/byte
ECS 10Mbps	128 us	64 us	1.4 us/byte
ECS 20Mbps	128 us	40 us	0.75 us/byte

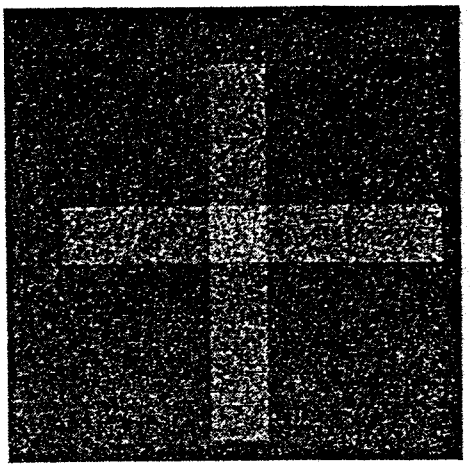

Figura 4: A Image Abingdon Cross (512 x 512)

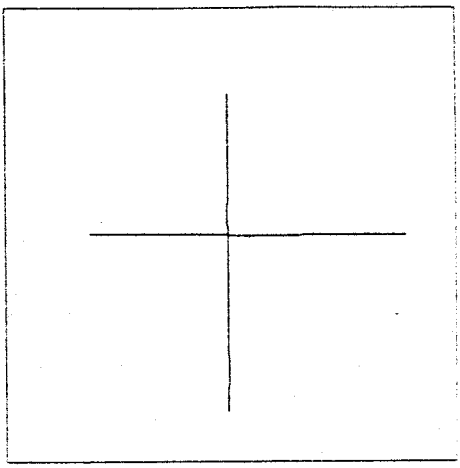

Figura 5: Eixos Medianos após processamento

A razão do computador de Edinburgo (ECS) apresentar uma taxa de transmissão menor que o cartão B003 (parâmetro c: 1.4 us/byte x 1.16 us/byte) se deve ao fato do ECS utilizar chaves programáveis para a ligação entre os canais de comunicação dos transputers.

O Benchmarck Abingdon Cross

O benchmark Abingdon Cross é um benchmark para processamento de imagens que consiste em achar o eixo médio de uma cruz numa imagem com ruído. A imagem de entrada possue 512 x 512 pixels de 8 bits (Fig 4). Este benchmark, ao contrário da maioria, não especifica o método a ser usado para resolver o problema. A maioria das soluções utiliza inicialmente uma filtragem de baixa frequência seguida de um threshold pré-fixado. Após, são feitas algumas operações de dilatação e erosão e finalmente é feito o refinamento da cruz para reconstruir seus eixos medianos (Fig 5). O benchmark Abingdon Cross testa operações de entrada/saída, matriciais e vetoriais, operações lógicas, filtragem e lógica celular [7].

Na implementação feita neste trabalho, o benchmark Abingdon Cross é avaliado seguindo os seguintes passos:

1. Carregamento concorrente com filtragem de média 8 x 1
2. Troca de sobreposição e filtragem de média 1 x 8
3. Threshold ao nível 145
4. Troca de sobreposição e dilatação conectado-4
5. 22 passos de troca de sobreposição e erosão conectado-8
6. 13 passos de troca de sobreposição e afinamento
7. descarregamento da imagem

Estas operações utilizam os seguintes padrões de comunicações descritos: computação local, de vizinhança e concorrência entre entrada/saída e processamento.

As medidas de tempo de execução para cada passo foram feitas para: i) um único transputer rodando a versão sequencial do algoritmo; ii) um processador Root com uma grade de 2 x 2 processadores; iii) um Root com uma grade de 4 x 4; e finalmente iv) um Root com uma grade de 8 x 8 processadores. O transputer Root é conectado à grade por intermédio de três links seriais de 10 Mbps. A tabela abaixo mostra essas medidas com valores de tempo em segundos. Nos passos 5 e 6 é apresentado valores médios de execução por ciclo.

passo	sequencial	2x2	4x4	8x8
1	9.8	2.7	0.7	0.22
2	10.2	2.6	0.6	0.17
3	1.1	0.29	0.075	0.026
4	2.0	0.54	0.153	0.038
5	1.6	0.44/ciclo	0.115/ciclo	0.038/ciclo
6	3.5	0.88/ciclo	0.309/ciclo	0.059/ciclo
7	-	0.17	0.17	0.19
total	103.8	27.42	8.24	2.24

Operações de Processamento de Imagens utilizando Processor Farm

A experiência envolvendo o modelo Processor Farm foi feita com um algoritmo de detetor de bordas Sobel seguido da Transformada de Hough para deteção de linhas retas utilizando a posição e o gradiente da borda detetada [6]. A imagem de entrada possui 256 x 256 x 8 bits (Fig 6). Essas experiências foram feitas no mesmo hardware utilizado na avaliação do benchmark Abingdon Cross. O tempo é medido a partir da imagem estar disponível no frame-buffer do processador Root até a imagem final, neste caso representando o espaço Hough (Fig 7), ser carregada completamente no frame-buffer novamente. A imagem foi testada com um valor no valor de threshold utilizado no detetor de bordas que resulta em 8 % de pontos com contribuição para a construção do espaço Hough. A tabela abaixo mostra o resultado dos tempos de execução em segundos.

	sequencial	2x2	4x4	6x6	8x8
Hough T.	2.23	0.6	0.22	0.18	0.15

5. Comentários e Conclusões

Este trabalho apresenta sete padrões típicos de comunicação envolvidos em algoritmos paralelos de processamento de imagens aplicados à máquinas de memória distribuídas de grão médio. Estes padrões de comunicação podem ser colecionados em uma biblioteca na qual o programador seleciona o padrão apropriado ao seu algoritmo de processamento de imagem. Este é um trabalho introdutório que requer uma continuação para explorar um número maior de algoritmos de processamento de imagens e visão computacional.

Este trabalho utiliza uma rede de processadores transputers programados como uma rede virtual sob um sistema de passagem de mensagens. O uso de um sistema de passagem de mensagens

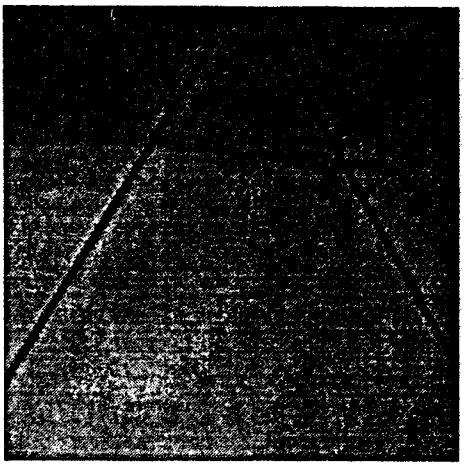

Figura 6: Imagem usada no experimento da Transformada de Hough (256 x 256)

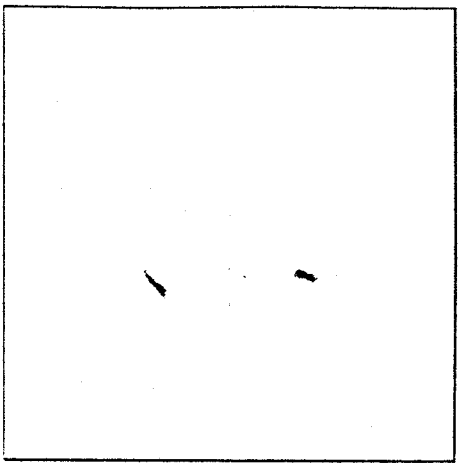

Figura 7: Espaço de Hough, com 5.200 características de contribuições

simplifica a codificação de programas paralelos aplicados à processamento de imagens apresentando um desempenho igual ou superior quando comparado com implementações que utilizam sistema explícito de roteamento de mensagens.

A solução usada nos experimentos descritos possui uma vantagem adicional sobre outras técnicas de programação pela sua flexibilidade de execução do mesmo código executável em diversas topologias. Este argumento aponta favoravelmente para o uso de topologias fixas de hardware, diminuindo custos e overhead impostos pelos sistemas programáveis de chaveamento de ligações.

Bibliografia

1. G.Harp, S.Baker adn H.Webber - "Image Processing", In G.Harp (ed.) "Transputer Applications". Pitman, 1989.

2. G.Fox, M.Johnson, G.Lyzenga, S.Otto, J.Salmon and D.Walker - "Solving Problems on Concurrent Processors". Prentice-Hall International Editions, 1988.

3. L.J.Clarke - "Communication in networks of parallel processors". Edinburgh Concurrent Supercomputer Newsletter, Number 7, April 1989.

4. H.T.Kung - "Computational Models for Parallel Computers". In R. Elliot and C.A.R.Hoare (eds.) "Scientific Applications of Multiprocessors". Prentice Hall pp.1-16 (1989).

5. S.Miguet - "Distance Transform on a Ring of Processors". Working Conference on Decentralized Systems".

6. Lotufo R A, Dagless E L, Milford D J, Morgan A D, Morrissey J F and Thomas B T - "Hough Transform for Transputer Arrays" Third International Conference on Image Processing and its Applications" IEE Conf. Publication N.307, University of Warwick, July, (1989).

7. K.Preston Jr. - "The Abingdon Cross Benchmark Survey". IEEE Computer. July 1989. pp.9-18.

Agradecimentos

Este trabalho foi desenvolvido durante o período de doutoramento do autor com bolsa do CNPq na Universidade de Bristol, Inglaterra. O sistema de passagem de mensagens HiVe foi desenvolvido pelo autor com base em sistema semelhante chamado Tiny desenvolvido por Lyndon Clarke e Michael Norman do Centro de Computação Paralela de Edinburgo. O computador paralelo de Edinburgo faz parte de projeto científico-educacional britânico financiado pelo SERC.