Análise do desempenho da transformada de Hough paralela em arquiteturas de memória compartilhada

CHRISTIAN CESAR BONES[1]
ODEMIR MARTINEZ BRUNO[1]

USP - Universidade de São Paulo
ICMC - Instituto de Ciências Matemática e de Computação
SCC - Departamento de Ciências de Computação
Cx Postal 668 - CEP 13560-970 São Carlos (SP)
[1](chris,bruno)@icmc.usp.br

Resumo. Este trabalho busca avaliar o desempenho da Transformada de Hough(TH), método utilizado em processamento de imagens. A TH foi paralelizada para máquinas de arquitetura MIMD (Multiple Instruction Multiple Data) com memória compartilhada. O objetivo da paralelização da TH é permitir que esta técnica possa ser utilizada em sistemas com tempo crítico ou mesmo tempo-real. São realizados experimentos com duas estratégias de exploração de paralelismo da TH na qual foram feitas implementações com 3 ferramentas de programação paralela: MPI, OpenMP e pThreads. São apresentados resultados que levam a análises e comparações, indicando qual é a melhor abordagem para desenvolver uma TH de alta performance em arquitetura MIMD memória compartilhada.

Palavras-Chave: transformada de Hough, computação paralela, arquitetura de memória compartilhada

Performance analysis of the parallel Hough transform in shared memory architectures

Abstract. This work evaluates the performance of the parallel Hough transform (HT) under MIMD (Multiple Instruction Multiple Data) shared memory architectures. The objective of to develop a TH parallel algorithm is to allow the uses of this method in real time systems. There are compared two TH algorithms implemented using three parallel programming tools: MPI, OpenMP and Pthreads. There are presented experiments, discussions and results, that shows the best TH parallel approach for shared memory architectures.

Keywords: Hough transform, parallel computation, shared memory architectures

(Received October 11, 2006 / Accepted January 03, 2007)

1 Introdução

Este trabalho tem como objetivo descrever os resultados obtidos na paralelização da transformada de Hough (TH) [10, 8, 4, 12, 13, 7] para retas. A transformada de Hough é um método de processamento de imagens utilizado para detectar curvas parametrizadas, sendo que, sua mais conhecida utilização é na detecção de retas.

O objetivo de implementar a paralelização da transformada de Hough é para que o método consiga realizar a detecção de retas em um menor tempo de execução, podendo, até mesmo, ser utilizado em sistemas de tempo real.

Este trabalho avalia o desempenho da transformada de Hough paralela para máquinas MIMD de memória compartilhada [1, 16] utilizando as ferramentas para pro-

gramação paralela Message Passage Interface MPI [15, 9, 17], OpenMP[6, 17] e POSIX Thread [11, 5].

Nas próximas seções são descritas a transformada de Hough, as técnicas de paralelização da transformada de Hough, as ferramentas utilizadas para a paralelização, os resultados obtidos em cada uma das estratégias para cada ferramenta e as conclusões alcançadas através da análise dos resultados obtidos.

2 Transformada de Hough

A transformada de Hough (TH), foi inicialmente proposta como um método para detecção de padrões complexos em imagens binárias, patente que foi concedida a P. V. C. Hough em 1962 [10], como o nome de "Method and Means for Recognizing Complex Patterns". Duda e Hart [8] adaptaram a idéia de Hough para imagens digitais utilizando coordenadas polares para definir uma reta, trabalhando com os parâmetros Ângulo-Raio ao invés de Inclinação-Intersecção. O método de Duda e Hart envolve o mapeamento de retas do espaço imagem, para conjuntos de pontos num espaço de parâmetros Ângulo-Raio.

2.1 Transformada de Hough para retas

O método desenvolvido por Hough consistia em detectar pontos colineares ou quase colineares numa imagem. Para saber se os pontos são colineares, deve-se calcular o coeficiente das retas que passam pelo ponto. Em um dado ponto da imagem podem passar infinitas retas. Uma reta pode ser definida por dois parâmetros, utilizando-se coordenadas polares (ρ,θ) [8], onde ρ indica a distância mínima da reta a origem do plano cartesiano e, θ o coeficiente angular que o segmento de reta r faz com o eixo x das ordenadas. Os pontos desta reta podem ser representados por:

$$\rho = x\cos(\theta) + y\sin(\theta) \quad (1)$$

Os novos parâmetros utilizados para representar o espaço são definidos agora por ρ e θ. Logo, o problema de detectar pontos colineares pode ser convertido no problema de se encontrar curvas concorrentes [4, 12, 13]. Segundo Duda e Hart [8] as propriedades para se transformar pontos em curvas são:

- Um ponto no plano imagem, corresponde a uma curva senoidal no plano de parâmetros.
- Um ponto no plano de parâmetros corresponde a uma reta no plano da imagem.
- Pontos pertencentes a mesma linha reta no plano imagem, correspondem a curvas através de um ponto no plano de parâmetros.
- Pontos pertencentes a mesma curva no plano de parâmetros correspondem a linhas através do mesmo ponto no plano imagem.

Na Fig. 1 (a) são apresentados os pontos no espaço imagem, em (b) o mapeamento dos pontos no espaço de Hough (ρ, θ) e (c) ilustra as propriedades de colinearidade, onde o ponto "A" denota a intersecção entre as curvas correspondentes aos pontos 1, 3 e 5 no plano $x-y$ e similarmente o ponto "B" corresponde aos pontos colineares 2, 3 e 4. A Fig. 1 (d) ilustra a propriedade de reflexão, onde "A", "B" e "C" se repetem quando a senóide inicia um novo ciclo, entretanto com sinal negativo.

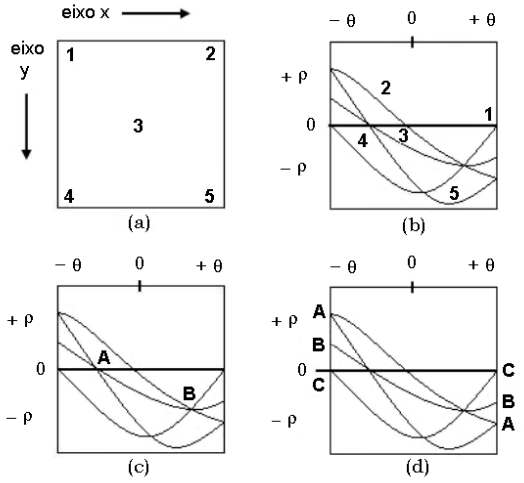

Figura 1: (a) pontos no espaço imagem, (b) mapeamento dos pontos no espaço de Hough, (c) detecção das propriedades de colinearidade e em (d) a propriedade de reflexão

A construção do arranjo acumulador bidimensional é definida dentro de um erro aceitável, em (ρ, θ) formando uma grade. Esta grade pode ser limitada no intervalo $0 \leq \theta < \rho$ e $-R \leq \rho < R$, onde R é o tamanho da maior diagonal da imagem. A Fig. 2 ilustra um exemplo de um arranjo acumulador.

A detecção de retas consiste em localizar as células da matriz do espaço de Hough, calculada pela equação 1, que apresentem os maiores valores, ou seja determinar os pontos que possuem máximos locais.

3 Estratégia de paralelização da TH

A transformada de Hough seqüencial é muito morosa, já que para cada coordenada da imagem, devem-se fazer os cálculos de ρ variando o θ de 0 a 180 graus, o

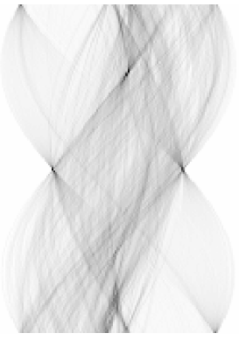

Figura 2: Exemplo de um arranjo acumulador (espaço de Hough)

que exige grande poder computacional e uma grande repetição de tarefas [7, 14]. Observando esses problemas surgiu a idéia de transformar o método de Hough seqüencial em um método que possa ser executado em paralelo [3]. Neste trabalho serão demonstradas duas abordagens: a primeira consiste na divisão da imagem [7, 14], onde cada processo escravo calcula um pedaço da imagem, a segunda abordagem consiste na divisão do arranjo acumulador [13] e cada processo escravo calculará um determinado pedaço do arranjo acumulador. As duas estratégias são descritas a seguir.

3.1 Divisão da imagem

Nesta estratégia a imagem é dividida em pedaços. Um computador central, chamado de mestre, é o responsável por dividir a imagem em tamanhos iguais e enviar os pedaços da imagem para os processadores ou computadores escravos, para serem processados [3]. O escravo irá calcular os valores de ρ variando os valores de θ de 0 a 180 graus, para o pedaço da imagem que foi a ele designado.

Após calculados os valores de ρ e incrementado o arranjo acumulador na posição correspondente, é passado para o processo mestre somente o arranjo acumulador, para que ele faça a soma de todos os arranjos acumuladores, vindos dos escravos. A Fig. 3 mostra o envio dos pedaços da imagem aos computadores escravos, o processamento do arranjo acumulador pelos escravos, a devolução para o mestre do arranjo acumulador e a soma dos arranjos acumuladores resultantes.

3.2 Divisão do arranjo acumulador

Neste método, em vez dos escravos receberem pedaços da imagem, eles receberam toda a imagem. A divisão para a exploração de paralismo dos dados é feita com o arranjo acumulador [4, 3]. No eixo do ângulo θ, é dividido o ângulo para realização dos cálculos, de acordo com a quantidade de escravos. Por exemplo, se existirem 4 escravos o ângulo será divido em 4 partes, cabendo ao primeiro escravo calcular para toda a imagem o ângulo θ que irá variar de zero a 44 graus. O segundo escravo também irá realizar o cálculo para toda a imagem contudo o ângulo θ irá variar de 45 a 89 graus, e assim subseqüentemente até completar 180 graus. A Fig 4 ilustra esta estratégia de paralelização da TH.

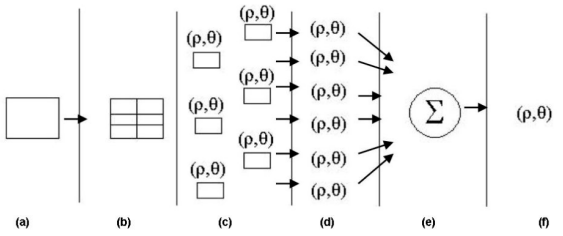

Figura 3: Estratégia de paralelismo da divisão da imagem para a transformada de Hough. As etapas (a), (b), (c), (d), (e), (f) representam respectivamente: a imagem original, divisão da imagem e sua distribuição, processamento de cada fragmento da imagem, arranjos acumulador resultante do processamento de cada fragmento, somatório dos elementos dos arranjos acumulador e arranjo acumulador resultante.

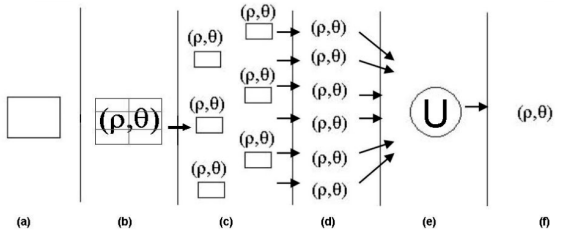

Figura 4: Estratégia de paralelismo da distribuição do arranjo acumulador. (a) imagem original, (b) divisão do arranjo acumulador, (c) processamento para toda a imagem na porção do arranjo acumulador que couber ao escravo, (d) arranjo acumulador resultante, (e) união dos arranjos acumuladores e (f) arranjo acumulador resultante.

4 Ferramentas de troca de mensagens

As duas estratégias de paralelização da TH foram implementadas em 3 ferramentas distintas de exploração de paralelismo: MPI, OpenMP e Posix Threads. Nas próximas subseções será realizada uma breve discussão sobre cada uma delas.

4.1 Message Passage Interface

O Message Passage Interface (MPI) [15, 17] é um padrão para a comunicação entre processos, cuja implementação fornece uma biblioteca de funções ou subrotinas para C/C++ ou Fortran. O MPI provem a comunicação entre os processos de máquinas diferentes numa arquitetura MIMD de memória distribuída, ou entre processos de uma mesma máquina numa arquitetura MIMD de memória compartilhada [1]. Embora seja voltado para sistemas distribuídos, algumas implementações do MPI, como a MPICH [9], apresentam estratégias para a troca de mensagens em memória compartilhada. Nestes casos as trocas de mensagens são realizadas diretamente pela memória não sendo necessário utilizar os dispositivos de redes, como acontece nas implementações MPI padrão.

4.2 OpenMP

O OpenMP [2, 17] é uma Application Program Interface (API) que permite a programação em arquiteturas MIMD de memória compartilhada Symmetric Multiple Processor (SMP) [1] com múltiplos threads. O OpenMP consiste de diretivas de compilação, bibliotecas de rotinas e variáveis de ambientes para diversas plataformas, dentre elas estão o Windows NT/XP e o Linux. Deste modo, ele atua como uma ferramenta de programação paralela, que permite desempenho tanto para granularidade fina ou grossa em arquiteturas de memória compartilhada.

4.3 POSIX Threads

O POSIX Threads [11, 5], assim como o OpenMP, é uma API que permite o desenvolvimento de aplicações paralelas com múltiplos threads em arquiteturas MIMD de memórias compartilhadas. Diferente do OpenMP, que é uma ferramenta específica para a computação paralela, as POSIX Threads são voltadas para uso genérico. Por um lado isto as tornam mais flexíveis entretanto compete ao programador o gerenciamento completo das threads e bem como da comunicação entre elas.

5 Experimentos e resultados

Foram realizados diversos testes para avaliar o desempenho de cada uma das implementações paralelas da TH em relação a versão seqüencial. Os experimentos foram realizados com uma imagem de uma cena real e também com imagens geradas artificialmente. Os tamanhos das imagens adotadas foram 512x512, 1024x1024 e 2048x2048. A imagem real, mostrada na Figura 5 apresenta o canal da piracema, construído para que os peixes possam subir o rio Iguaçu, transpondo a barreira física da Itaipu Binacional, na época da desova. As imagens artificiais, foram utilizadas para testar a TH em sua atuação crítica, ou seja imagens com muitos pontos co-lineares. Foram adotadas duas imagens artificiais, a primeira gerada com retas horizontais e a segunda com retas de diferentes ângulos.

Figura 5: Imagem real adotada no experimento. Canal da piracema, binacional Itaipu.

A máquina utilizada, possuia 2 processadores Intel Xeon Pentium 4 com 2,7 Giga Hertz cada um e memória compartilhada de 4 Giga Bytes. Uma vez que os processadores apresentam núcleo duplo esta configuração equivale a uma máquina com 4 processadores. O sistema operacional adotado no experimento foi o Linux SuSe 8.2.

Foram feitas comparações entre as 2 estratégias de paralelismo da TH em 3 implementações distintas: MPI, OpenMP e Pthreads, sendo que na última foram utilizados dois compiladores, gcc padrão e o OpenMP, que é otimizado para threads. Ainda para avaliar o desempenho dos compiladores, a versão TH seqüencial foi também compilada em gcc e OpenMP.

As Figuras 6, 7 e 8 apresentam os gráficos do tempo de execução da estratégia de divisão do arranjo. A Fig. 6 é referente à imagem artificial constituída por retas horizontais, a Fig. 7 é referente à imagem artificial de retas mistas e a Fig. 8 à imagem real. Para cada uma dos casos foram utilizadas imagens com três tamanhos: 512x512, 1024x1024 e 2048x2048, representadas nas figuras por (a), (b) e (c) respectivamente.

As Figuras 9, 10 e 11 mostram os gráficos do tempo de execução relativos a estratégia de replicação da imagem, referentes a imagens de retas horizontais, retas mistas e real respectivamente. no experimento foram utilizadas imagens de tamanhos: 512x512, 1024x1024 e 2048x2048, representadas respectivamente nas figuras por (a), (b) e (c) .

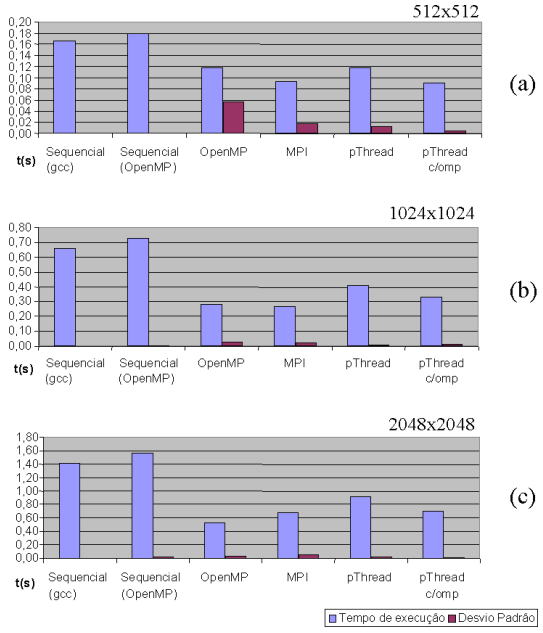

Figura 6: Tempo de execução da TH implementando a divisão do arranjo acumulador para imagem artificial formada de retas horizontais. Tamanho das imagens: (a) 512x512, (b) 1024x1024 e (c) 2048x2048

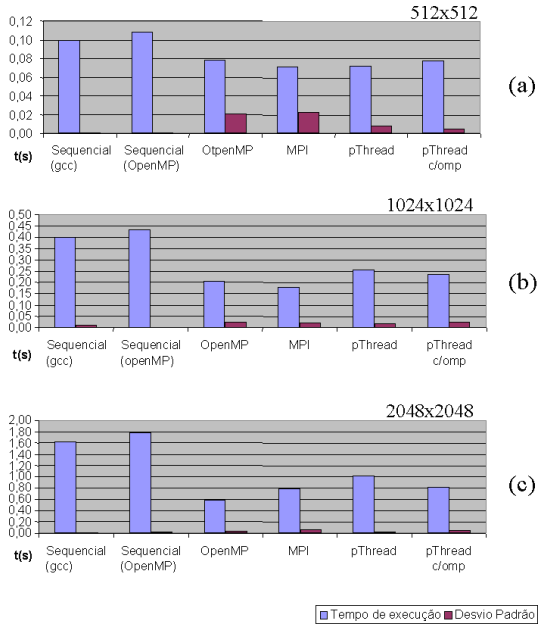

Figura 7: Tempo de execução da TH implementando a divisão do arranjo acumulador para imagem artificial formada de retas mistas. Tamanho das imagens: (a) 512x512, (b) 1024x1024 e (c) 2048x2048

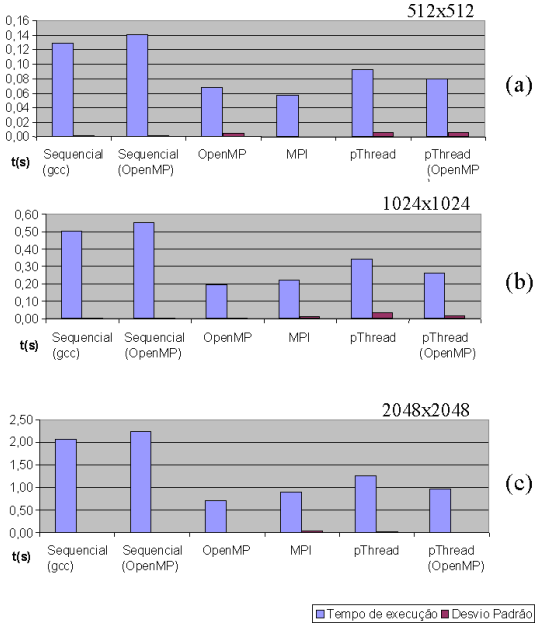

Figura 8: Tempo de execução da TH implementando a divisão do arranjo acumulador para imagem real. Tamanho das imagens: (a) 512x512, (b) 1024x1024 e (c) 2048x2048

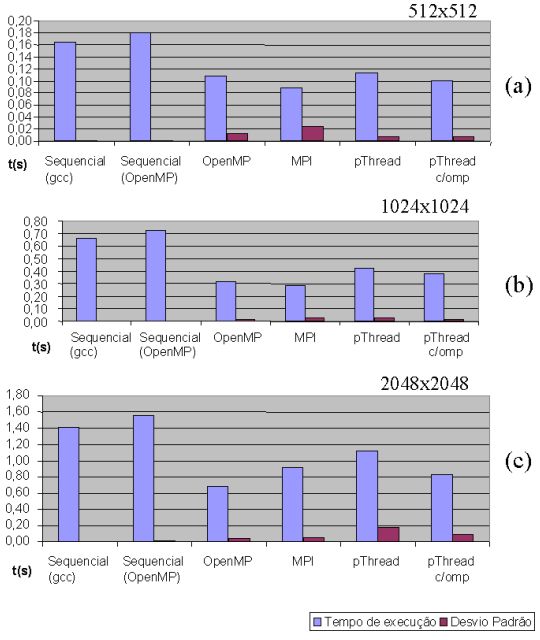

Figura 9: Tempo de execução da TH implementando a replicação da imagem para imagem artificial formada de retas horizontais. Tamanho das imagens: (a) 512x512, (b) 1024x1024 e (c) 2048x2048

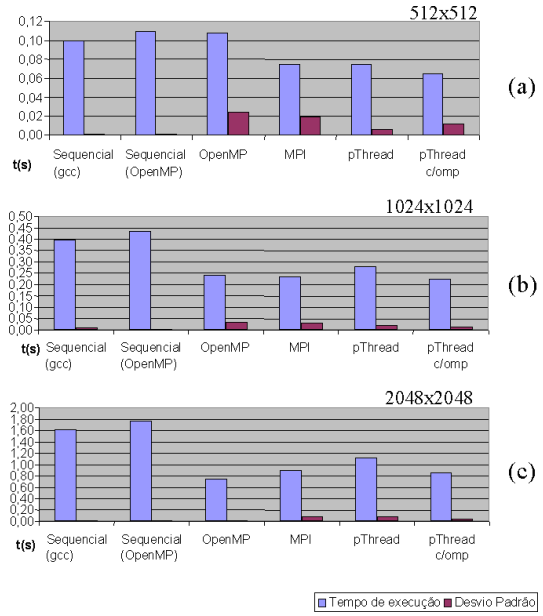

Figura 10: Tempo de execução da TH implementando a replicação da imagem para imagem artificial formada de retas mistas. Tamanho das imagens: (a) 512x512, (b) 1024x1024 e (c) 2048x2048

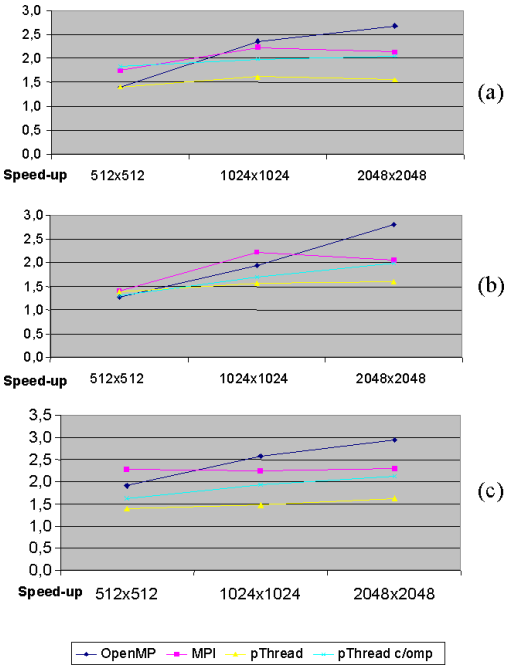

Figura 12: Gráficos de speed-up para a estratégia divisão do arranjo acumulador. (a) retas horizontais, (b) retas mistas e (c) imagem real

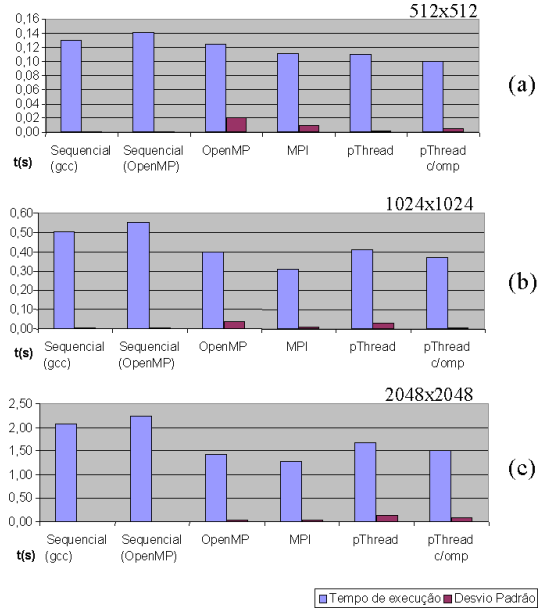

Figura 11: Tempo de execução da TH implementando a replicação da imagem para imagem real. Tamanho das imagens: (a) 512x512, (b) 1024x1024 e (c) 2048x2048

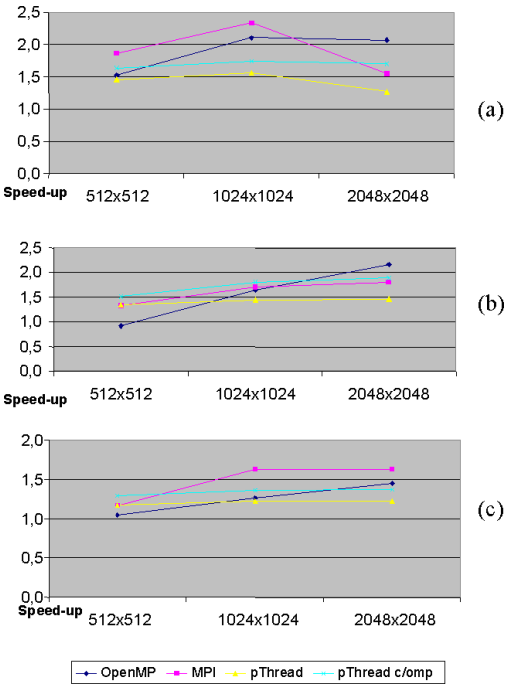

Figura 13: Gráficos de speed-up para a estratégia replicação da imagem. (a) retas horizontais, (b) retas mistas e (c) imagem real

Observando as figuras de tempo de execução (Figs. 6, 7, 8, 9, 10, 11), é possível chegar a algumas conclusões sobre o experimento. A primeira delas é em relação a independência/dependência dos resultados relativos em função da natureza das imagens. Observe que o tempo de execução das implementações paralelas em relação às seqüenciais, se manteve estável para as imagens artificiais e para a imagem real, indicando que ganho de desempenho do paralelismo é independente do conteúdo da imagem no caso da estratégia da divisão do acumulador. O mesmo não ocorreu para a estratégia de replicação da imagem, que obteve oscilação do tempo relativo paralelo em relação ao seqüencial. Observe que para retas mista (Fig. 10 (a)) a implementação paralela OpenMP teve um tempo de execução maior que a seqüencial.

Do ponto de vista de desempenho de compilador, foram avaliados o gcc e o omp (compilador comercial voltado para desenvolvimento OpenMP). Os resultados mostraram que o compilador gcc se saiu melhor na versão seqüencial, apresentando um tempo de execução menor, por outro lado, na exploração de paralelismo com PThreads, o compilador omp obteve os melhores resultados. Este resultado seria esperado, uma vez que o omp é voltado para a exploração de paralelismo.

Em relação as implementações MPI, OpenMP e PThreads, os resultados permitiram chegar algumas conclusões. No que se refere ao desempenho em função do tamanho das imagens, observe que o tempo de execução relativo ao tempo seqüencial manteve constante para o Pthread, indicando que o tamanho dos dados não afeta esta abordagem. Entretanto o tamanho das imagens apresentou uma alteração no desempenho das implementações de MPI e OpenMP. O tempo de execução do MPI foi menor para pequeno volume de dados, mas a medida que o volume de dados aumente e conseqüentemente o fluxo de troca de mensagens, o desempenho do OpenMP melhora chegando a apresentar o menor tempo de execução do experimento. Este padrão se manteve nos dois experimentos (divisão do arranjo acumulador e replicação da imagem) e não dependeu da natureza das imagens.

As Figuras 12 e 13 apresentam os gráficos de speed-up que demonstram os desempenhos dos experimentos. Os gráficos de speed-up indicam quantas vezes a implementação paralela foi superior a seqüencial. A Figura 12 apresenta gráficos referentes aos 3 experimentos com a estratégia de divisão do arranjo acumulador e a Figura 13 é a estratégia de replicação da imagem.

Os resultados demonstram que a estratégia de divisão do arranjo acumulador apresenta um desempenho melhor que a replicação das imagens, ficando no melhor caso (Figura 12 (c)) aproximadamente 3 vezes mais rápida que a implementação seqüencial. Vale a pena ressaltar que este speed-up é um excelente resultado para uma arquitetura de 4 processadores.

Quanto as ferramentas de exploração de paralelismo, o melhor desempenho foi do OpenMP e o pior foi das PThreads. A implementação com MPI superou o OpenMP para as imagens com menor tamanho. Além do desempenho, um elemento importante é a facilidade de desenvolvimento de programas paralelos. Quanto este requisito, o OpenMP também apresenta vantagens. Ele apresenta uma política de facilitar a programação paralela. Isto é feito por meio da delimitação de blocos, combinados com comandos para a exploração do paralelismo. Além do baixo desempenho, as PThreads, também apresenta desvantagens com relação a programação. Uma vez que é uma ferramenta genérica, quando utilizada para exploração de paralelismo, ela requer que o programador implemente rotinas de controle e concorrência, não necessárias em ferramentas para computação paralela como por exemplo o OpenMP, tornando a implementação e a depuração do código mais difíceis.

6 Conclusões

Neste artigo foi apresentado um estudo sobre a paralelização da transformada de Hough (TH) em ambientes MIMD com memória compartilhada. Duas estratégias de exploração de paralelismo foram abordadas e comparadas, uma baseada na divisão do arranjo acumulador e a outra na replicação da imagem. Com base nelas, foi realizado um estudo comparando 3 ferramentas voltadas para a computação paralela em arquiteturas de memória compartilhada: OpenMP, MPI e Pthreads. Este mostrou que o OpenMP apresentou os melhores desempenhos para as imagens de tamanho maior, ao passo que o MPI foi a melhor para imagens menores.

Entre as estratégias de paralelismo, a divisão do arranjo acumulador apresentou o melhor desempenho. Os experimentos realizados neste trabalho, mostraram que a TH paralela pode apresentar um ótimo desempenho, apresentando um speed-up médio entre 2,5 a 3 em uma arquitetura com 4 processadores.

A TH é uma técnica largamente difundida e muito utilizada em processamento de imagens e visão computacional, entretanto seu longo tempo de execução muitas vezes impede sua adoção em sistemas de tempo real. O estudo do paralelismo da TH, apresentado neste trabalho, permite a redução de seu tempo de execução e conseqüentemente sua utilização em aplicações com tempo crítico.

As novas tecnologias que permitem placas mãe com múltiplos processadores (SMT) e processadores com nú-

cleos duplos (Dual Core), estão reduzindo os custos e popularizando as arquiteturas MIMD com memória compartilhada. Este tipo de máquina está cada vez mais presente nos microcomputadores, e a tendência é de que num futuro próximo, a maioria das máquinas possuam esta arquitetura. Neste contexto, surge a necessidade de avaliar ferramentas de exploração de paralelismo, bem como estratégias de desenvolvimento paralelo, voltados para este tipo de arquitetura.

O presente trabalho realizou um estudo comparativo entre ferramentas para programação paralela. A metodologia, experimentos e resultados apresentados neste trabalho, além de contribuir para o emprego de TH paralelas de alto desempenho, fornecem importantes informações técnicas, que podem auxiliar na escolha de ferramentas para desenvolvimento de programas paralelos em máquinas MIMD compartilhadas.

Agradecimentos

Os autores agradecem ao CNPq (Proc. 303746/2004-1 e Proc. 132121/2003-4) pelo apoio à pesquisa.

Referências

[1] Almasi, G. S. and Gottlieb, A. *High Parallel Computing*. The Benjamin Communings Publishing Company, 2 edition, 1994.

[2] Board, O. A. R. Openmp c and c++ application program interface. Technical report.

[3] Bones, C. C. Paralelização da transformada de hough. Dissertação de mestrado, ICMC - Universidade de S. Paulo, 2004.

[4] Bruno, O. M. *Paralelismo em visão natural e artificial*. Tese de doutorado, IFSC - Universidade de S. Paulo, 2000.

[5] Butenhof, D. *Programming with Posix Threads*. Addison-Wesley, 1997.

[6] Chandra, R., Menon, R., Dagum, L., and Kohr, D. *Parallel Programming in OpenMP*. Morgan Kaufmann Publishers, 2000.

[7] Chutatape, O. and Guo, L. A modified hough transform for line detection and its performance. *Pattern Recogniton*, 32:181–192, 1999.

[8] Duda, R. O. and Hart, P. E. Use of the hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15:11–15, 1972.

[9] Gropp, W., Lusk, E., and Skjellum, A. *Using MPI - Portable Parallel Programming with the Message Passing Interface*. The MIT Press, 2 edition, 1999.

[10] Hough, P. V. C. U.s. patent 3.069.654, dec. 18, 1962: Method and means for recognizing complex patterns, 1962.

[11] IEEE and Group, T. O. A backgrounder on ieee std 1003.1. Technical report.

[12] Illingworth, J. and Kittler, J. The adaptative hough transform. *IEEE Trans. Pattern Anal. Machine Intel*, 9(5):11–15, 1987.

[13] Jin, L. and Yang, L. Parallel solution of hough transform and convolution problems a novel multimodal approach. In *ACM/SIGAPP symposium on Applied computing: technological challenges of the 1990's*, pages 775–781, 1992.

[14] Krishnaswamy, D. and p. Banerjee. Exploiting task and data parallelism in parallel hough and randon transforms. In *International Conference on Parallel Processing*, pages 441–444, Bloomington, IL, USA, 1997.

[15] Pacheco, P. S. *Parallel programming with MPI*. Morgan Kaufmann Publishers, 1996.

[16] Quinn, M. J. *Parallel Computing: theory and practice*. McGraw-Hill, 2 edition, 1994.

[17] Quinn, M. J. *Parallel Programming in C with MPI and OpenMP*. McGraw-Hill, 2003.