ISSN 1981-2841

OTIMIZAÇÃO DE ALGORITMOS DE PROCESSAMENTO DE IMAGENS MÉDICAS UTILIZANDO A COMPUTAÇÃO PARALELA

MEDICAL IMAGE PROCESSING ALGORITHMS OPTIMIZATION USING PARALLEL COMPUTING

Priscila T. M. Saito[1], Ricardo J. Sabatine[1], Kalinka R. L. J. Castelo Branco[1] e Fátima L. S. Nunes[1]

RESUMO

Neste artigo, objetiva-se demonstrar a viabilidade da melhoria no tempo de execução de algoritmos utilizados para o processamento de imagens médicas por meio do uso da computação paralela distribuída. Implementaram-se técnicas de processamento, de forma sequencial e paralela, utilizando a linguagem Java e as bibliotecas de trocas de mensagens mpiJava e JPVM. Executaram-se algoritmos de suavização e de detecção de bordas no domínio espacial, fazendo uso de diferentes tamanhos de máscaras. Após a prática, foi possível construir uma base de comparação entre a aplicação sequencial e a paralela, o que permitiu a avaliação do ganho de desempenho obtido com o paralelismo.

Palavras-chave: computação paralela, processamento de imagens médicas, sistemas distribuídos, mpiJava, JPVM.

[1] Centro Universitário Eurípides de Marília - UNIVEM.
Fundação de Ensino Eurípides Soares da Rocha.
Bacharelado em Ciência da Computação.
Av. Hygino Muzzi Filho 529, CEP 17509-901, Marília, SP.
{priscilasaito, sabatine}@gmail.com, {kalinka, fatima}@univem.edu.br

ABSTRACT

This paper aims at demonstrating the viability in the use of parallel distributed computing to improve the execution time of algorithms used for medical image processing. Some image processing techniques were implemented in the sequential and parallel way by using the program language Java and the parallel virtual libraries mpiJava and JPVM. Smoothing and edge detection algorithms were implemented in the spatial domain, using different mask sizes. After the implementation, it was possible to make a comparison between the sequential and parallel application, that permitted to evaluate and to demonstrate the performance gain using parallelism.

Keywords: *Parallel Computing. Medical Image Processing. Distributed Systems. mpiJava. JPVM.*

INTRODUÇÃO

Imagens médicas têm por finalidade o auxílio na composição do diagnóstico de anomalias e o fornecimento de material para acompanhamento de terapias. O avanço na aquisição, no processamento e no armazenamento dessas imagens vem permitindo o aperfeiçoamento dos diagnósticos e dos tratamentos de doenças de naturezas diversas.

Sistemas de diagnóstico auxiliado por computador, *computer-aided diagnosis* (CAD), ajudam o radiologista a decidir sobre um diagnóstico (obtido por resultados de uma análise computadorizada de imagens médicas) (GIGER, 2000). Sendo assim, esses sistemas são objetos de pesquisa em várias instituições, nas quais diversos autores destacam a importância da utilização de esquemas CAD na melhoria do desempenho de radiologistas no diagnóstico médico.

No entanto, sistemas desse tipo, aplicados no dia a dia da prática médica, são poucos pela necessidade do alto desempenho exigido por essa classe de sistema, tanto em nível de velocidade de execução quanto em nível de acerto nos resultados (NUNES, 2006). Isso acontece pois alguns tipos de erros não são admitidos, uma vez que esses resultados são utilizados na tomada de decisão para diagnósticos ou para escolha de tratamento.

O uso de sistemas distribuídos e de bibliotecas de passagem de mensagens, que viabilizam a computação paralela sobre sistemas distribuídos, pode resolver essa necessidade de alto desempenho exigida para o processamento dessas imagens. Desse modo, neste artigo, apresenta-se a implementação de algoritmos

de suavização e detecção de bordas de imagens de forma sequencial e paralela, aplicados a imagens mamográfica. Compreende, ainda, comparações entre o desempenho obtido com as duas formas de processamento e as bibliotecas de passagem de mensagens utilizadas.

COMPUTAÇÃO PARALELA DISTRIBUÍDA

Os sistemas computacionais distribuídos, aplicados à computação paralela, permitem melhor relação custo/benefício, pois oferecem potência computacional adequada a aplicações paralelas que não necessitam de uma máquina maciçamente paralela, porém precisam de uma potência computacional maior que uma máquina sequencial pode oferecer (BRANCO, 1999).

Para a realização da computação paralela sobre sistemas distribuídos necessita-se de uma camada de *software* que gerencie o uso paralelo, uma vez que existe a necessidade da passagem de informações entre as várias máquinas que compõem a plataforma. Os sistemas baseados na troca de mensagens mais utilizados são o MPI (*Message Passing Interface*) e o PVM (*Parallel Virtual Machine*).

Com o surgimento da linguagem Java apresentam-se inúmeras propostas para a utilização das bibliotecas, como o mpiJava (BAKER, 1998) e o JPVM (*Java Parallel Virtual Machine*) (FERRARI, 1998), ambientes de passagem de mensagens utilizados neste trabalho.

A escolha de ambos, como ambiente para processamento paralelo dos algoritmos, vem em decorrência do uso da linguagem *Java* que possibilita: portabilidade, permitindo a independência de plataforma, simplicidade e clareza nos códigos, além da existência de API's especializadas, as quais admitem o reuso de código e facilitam a construção de aplicações nos mais diversos domínios, como é o caso do processamento de imagem.

MPIJAVA E JPVM

MpiJava é uma interface amplamente usada na computação paralela distribuída, que permite fazer uso da orientação a objetos em *Java*, com a biblioteca MPI (BAKER, 1998). O uso de MpiJava já foi validado em diversas implementações e avaliações de desempenho (TABOADA, 2003).

JPVM é uma API implementada em *Java* que possibilita a troca de mensagens explícitas, combina as vantagens da linguagem *Java*, como portabilidade e interoperabilidade, com as técnicas de troca de mensagens entre

processos paralelos em ambientes distribuídos. A interface, semelhante em relação ao PVM, admite a programadores, acostumados com o PVM padrão, a migração de um ambiente para outro com uma maior facilidade.

UM MODELO DE PARALELISMO PARA O PROCESSAMENTO DE IMAGENS

O requisito básico de um sistema de processamento paralelo de imagens consiste em uma infraestrutura que consente a execução eficiente de quaisquer algoritmos, sejam eles de nível baixo (pré-processamento), médio (segmentação) ou alto (reconhecimento de padrões). Neste trabalho, a escolha do algoritmo deveu-se ao objetivo do processamento, buscando executar um filtro que exigisse uma quantidade elevada de operações, a fim de que a diferença entre a implementação sequencial e paralela pudesse ser evidenciada. Inicialmente, selecionou-se as técnicas de suavização e de detecção de bordas.

Os filtros de suavização empregam-se na etapa de pré-processamento para a redução de ruídos e para a remoção de pequenos detalhes de uma imagem antes da extração de objetos (GONZALEZ, 2002). Entre as técnicas mais comuns de suavização estão os filtros de média e mediana filtragem. A filtragem mediana, usada neste artigo, consiste em substituir o valor de um determinado *pixel* pelo valor mediano da sua vizinhança, que é o valor central obtido quando se ordenam os *pixels* da vizinhança.

A detecção de bordas representa outro exemplo de algoritmo que usa operações baseadas em vizinhança. Representar uma imagem por meio de suas bordas pode ser vantajoso, pois se reduz, sensivelmente, a quantidade de dados guardados na imagem, sem perder muita informação. Para avaliação, neste trabalho, utilizou-se o algoritmo de detecção de bordas, fazendo uso dos operadores de Sobel (GONZALEZ, 2002). Eles calculam o valor absoluto aproximado do gradiente em cada ponto da imagem analisada, deixando, em maior evidência, as áreas cuja frequência espacial possui um valor alto e que correspondem às bordas da imagem.

Uma proposta de paralelização eficiente desses algoritmos seria a divisão da imagem em blocos distribuídos pelos processadores, de forma a processar, ao mesmo tempo, vários blocos de uma imagem. Definir o tipo de paralelismo que melhor se adapte ao processamento proposto é tarefa que permite obtenção de melhor desempenho. Desse modo, o paralelismo de dados é o que melhor se enquadra, tornando-se o mais

interessante nesse caso, pois pode-se definir que o processador execute as mesmas tarefas sobre diferentes dados, aproximadamente do mesmo tamanho, tendo um único fluxo de controle – SPMD *(Single Process Multiple Data)*.

O desafio proposto, em imagens médicas, incidi na divisão da imagem em blocos e a posterior junção desses blocos sem perdas de processamento.

ANÁLISE DE DESEMPENHO DOS RESULTADOS OBTIDOS

A análise de desempenho dos algoritmos de filtro mediana e de detecção de bordas realizou-se por meio de diferentes testes reais, em um ambiente paralelo, distribuído, controlado e composto, inicialmente, por 3 máquinas homogêneas (Pentium IV de 2.7GHz com 512Mbytes de RAM, interligadas por uma rede *ethernet* de 100Mb/s). Posteriormente, foram acrescidas máquinas idênticas para comporem um ambiente de 4 e 5 máquinas.

Valeu-se de imagens mamográficas de elevado tamanho (aproximadamente, 11 MB – matrizes de tamanho médio 2500X3000, com formato TIFF e possuindo uma resolução de 16 bits), característica típica das imagens médicas. O filtro mediana foi avaliado com máscaras de tamanhos diferentes (3x3, 5x5 e 7x7), utilizando o algoritmo de ordenação *shellsort*. O filtro de detecção de bordas foi executado com máscara de tamanho 11x11. Observa-se que o tamanho da máscara consiste no que define o tamanho da vizinhança a ser considerada. Desse modo, dependendo do tipo da imagem, uma maior vizinhança indica um melhor resultado de processamento.

Após a realização dos testes, obteve-se uma média dos 30 tempos de processamento tanto à aplicação sequencial quanto à paralela, para os diferentes tipos de máscaras. Quando da avaliação da execução em paralelo foram efetuados testes com até cinco máquinas e, para cada uma delas, testes com até 8 processos iniciados em paralelo, isso possibilitou a realização da comparação de desempenho de cada uma das possíveis combinações.

As figuras 1(a) e 1(b) apresentam os resultados da execução sequencial do filtro mediana, bem como as execuções em paralelo do mesmo algoritmo em 3, 4 e 5 máquinas, utilizando mpiJava e JPVM, respectivamente. Por meio delas, é possível observar que o tempo de execução do algoritmo sequencial, quando considerada a vizinhança de tamanho 3x3, torna-se significativamente melhor que o tempo do mesmo em paralelo, tanto usando mpiJava quanto JPVM, independentemente do número de máquinas e da quantidade de processos iniciados em paralelo.

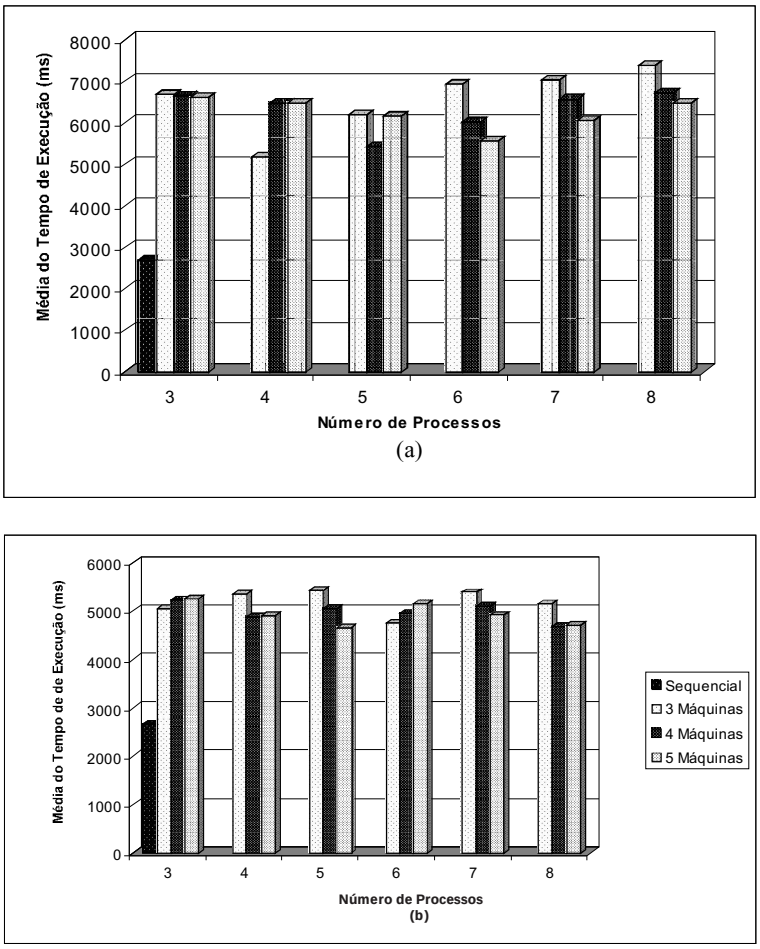

Figura 1 – Média do tempo de execução do filtro mediana de máscara 3x3 fazendo uso dos ambientes de troca de mensagens. (a) mpiJava. (b) JPVM.

Uma vez que a quantidade de cálculos efetuados pelo filtro com máscara 3x3 é relativamente pequena, a paralelização desse não impõe melhoria, pois se consome mais tempo, em termos de comunicação em rede do que no cálculo da máscara propriamente dita, ou seja, o tempo utilizado para o envio de cada parte da imagem (subvetor) é maior que o tempo gasto pelos "escravos" para realizar o processamento (formação e ordenação do vetor), o que a torna uma aplicação mais voltada para a comunicação do que para o processamento.

Nas figuras 2 e 3 são apresentados os resultados das mesmas execuções anteriores, levando-se em consideração as máscaras 5x5 e 7x7, respectivamente. Pelas figuras, pode-se observar que o tempo de execução do algoritmo sequencial, a partir do uso dessas máscaras, é significativamente pior que o tempo de execução em paralelo, tanto fazendo uso do mpiJava quanto do JPVM, mas nas situações em que o número de processos equivale (no caso do JPVM) ou é maior em uma unidade (no caso do mpiJava) que o número de máquinas o escalonamento é do tipo *round-robin*. Dependendo do número de máquinas e do número de processos o tempo médio paralelo fica um pouco prejudicado no caso da máscara 5x5. Mais uma vez isso acontece pela quantidade de processamento ser relativamente menor que a quantidade de comunicação envolvida. Desse modo, à medida que se aumenta o número de processos para certa quantidade de máquinas o desempenho diminui, tornando-se maior que o tempo médio sequencial. Isso acontece porque quando se aumenta o número de processos, em alguns casos, é evidente a queda de desempenho.

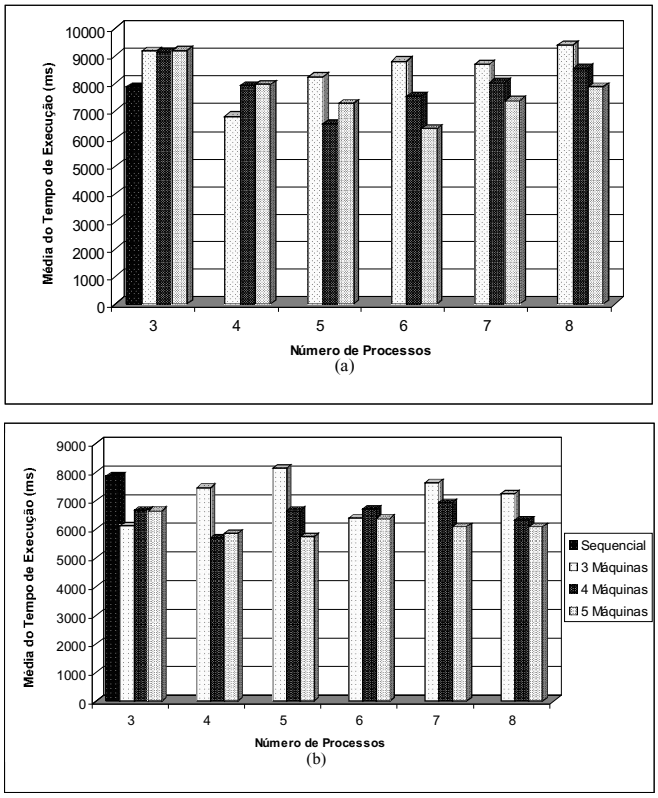

Figura 2 – Média do tempo de execução do filtro mediana de máscara 5x5 fazendo uso do ambiente de troca de mensagens. (a) mpiJava. (b) JPVM.

Diferentemente das máscaras 3x3 e 5x5, quando utilizada a máscara 7x7, independentemente do número de processos iniciados e do número de máquinas, o tempo médio paralelo é sempre menor que o tempo sequencial, uma vez que o processamento é alto e o aumento no número de processos ainda implica em ganho de desempenho.

Essa diferença e significativa melhora de desempenho, quando se faz uso de uma arquitetura paralela distribuída, dá-se pelo fato de os cálculos efetuados pelas máscaras serem volumosos, garantindo uma melhoria do uso em paralelo, uma vez que a comunicação imposta se torna desprezível, quando comparada ao ganho em relação aos cálculos efetuados.

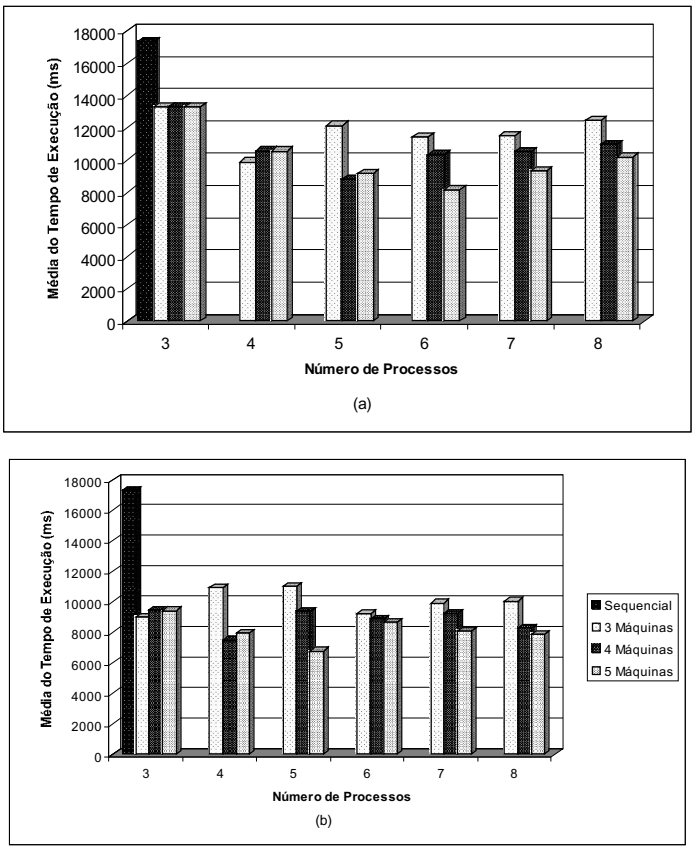

Figura 3 – Média do tempo de execução do filtro mediana de máscara 7x7 fazendo uso do ambiente de troca de mensagens. (a) mpiJava. (b) JPVM.

Na figura 4, pode-se observar os tempos de execução sequencial e paralelo do mpiJava e do JPVM. Observa-se que o JPVM apresenta um melhor desempenho, se comparado com o mpiJava, o que pode ser um indicativo que a biblioteca JPVM possui um comportamento relativamente melhor que a mpiJava. De qualquer forma, em ambos os casos, o desempenho médio da execução em paralelo é significativamente melhor que o da sequencial.

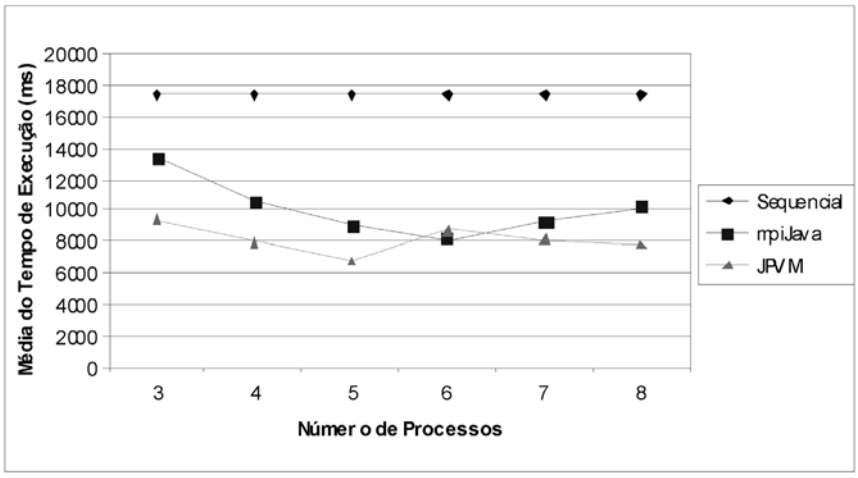

Figura 4 – Média do tempo de execução do filtro mediana de máscara 7x7 fazendo uso dos ambientes de troca de mensagem mpiJava e JPVM, utilizando-se 5 máquinas.

O número de processos mostrou-se um fator muito importante. No caso da biblioteca JPVM, o melhor número de processos é o mesmo que o número de máquinas que forma a máquina virtual. Já no mpiJava, o melhor resultado obtém-se quando o número de processos é maior do que o número de máquinas participantes. Isso ocorre pois o mpiJava faz uso de um processo mestre que é contabilizado, mas não efetua a tarefa "escrava" propriamente dita.

As figuras 5(a) e 5(b) apresentam os resultados da execução sequencial do filtro de detecção de bordas, bem como as execuções em paralelo do mesmo algoritmo em 4 e 5 máquinas utilizando máscara de tamanho 11x11 com as bibliotecas mpiJava e JPVM, respectivamente. É possível observar que, independentemente do número de processos iniciados, o tempo médio paralelo, tanto fazendo uso do mpiJava quanto do JPVM, é sempre melhor que o sequencial.

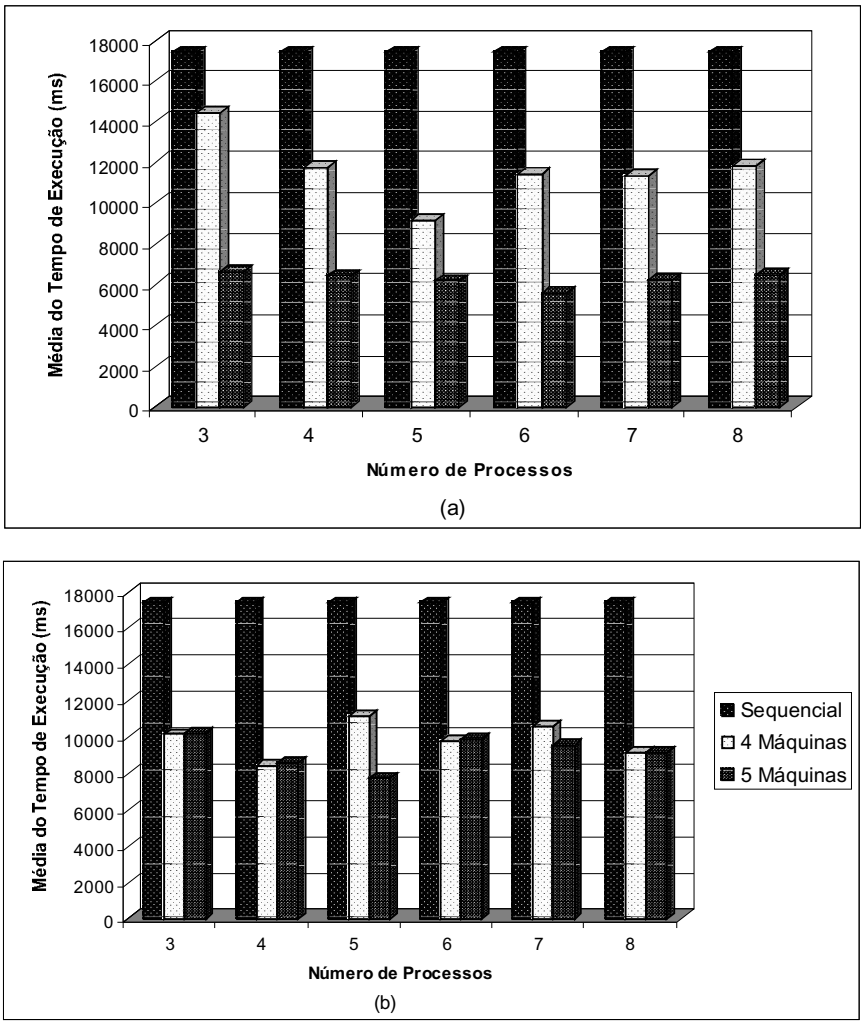

Figura 5 – Média do tempo de execução do filtro de detecção de bordas de máscara 11x11 fazendo uso dos ambientes de troca de mensagens. (a) mpiJava. (b) JPVM.

Nas figuras 6(a) e 6(b), apresentam-se os tempos de execução sequencial e paralelo do mpiJava e do JPVM do filtro de detecção de bordas em 5 máquinas, com a utilização de máscara de tamanho 9x9 e 11x11, respectivamente. Pode-se observar que, da mesma forma que o filtro mediana, JPVM ou mpiJava, o desempenho médio da execução em paralelo é significativamente melhor que o da sequencial.

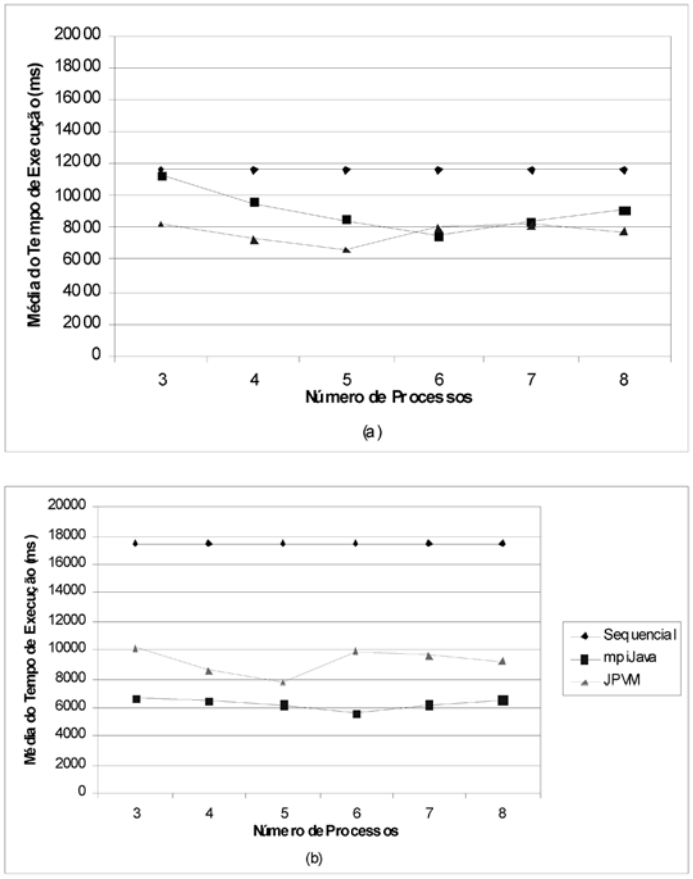

Figura 6 – Comparação da média do tempo de execução do filtro de detecção de borda fazendo uso dos ambientes de troca de mensagens mpiJava e JPVM. (a) máscara 9x9. (b) máscara 11x11.

Com a utilização de máscara de tamanho 9x9, observa-se que o JPVM apresenta um melhor desempenho, se comparado com o mpiJava. O mpiJava consegue melhor resultado que o JPVM quando o número de processos passa a ser maior, em uma unidade, que o número de máquinas utilizadas, ou seja, 5 máquinas e 6 processos.

Com a utilização de máscara de tamanho 11x11, percebe-se que o mpiJava apresenta um melhor desempenho, se comparado com o JPVM. Isso implica que, para se avaliar corretamente as bibliotecas, a fim de determinar suas diferenças e definir qual é a melhor, de forma geral ou específica, para alguns tipos de aplicações, novos estudos devem acontecer.

CONCLUSÃO

A partir dos resultados obtidos, pode-se verificar que existe um ganho em se fazer uso do processamento paralelo distribuído, quando se pensa em processamento de imagens médicas. Acredita-se que a utilização de outros filtros no domínio espacial também possa prover melhora significativa de desempenho para a versão paralela, se comparada à execução sequencial.

Apesar de o resultado não ser favorável à execução paralela do filtro mediana, quando aplicada à máscara 3x3 e, em alguns casos, à máscara 5x5, observou-se que, para processamentos intensos, o uso do processamento paralelo é bastante vantajoso.

Com base nesses resultados, deseja-se, como trabalhos futuros, o desenvolvimento de novos algoritmos de processamento de imagens e a avaliação de outros algoritmos de ordenação (excetuando-se o *shellsort* aqui utilizado), presentes nesses algoritmos de processamento de imagens. Acredita-se, também, que a paralelização desses algoritmos de ordenação ocasione melhora significativa no desempenho dos mesmos.

Além disso, testes adicionais devem ser executados para avaliar qual das duas bibliotecas utilizadas (JPVM e mpiJava) apresenta melhor desempenho, uma vez que para um mesmo filtro, dependendo da máscara utilizada, houve um comportamento diferente para as duas bibliotecas. Mediante os resultados define-se, ainda, como trabalhos futuros, a execução de outros algoritmos de processamento de imagens em ambas as bibliotecas, pois permitirá a obtenção de um conjunto maior de dados, a partir do qual, tais informações poderão ser extraídas com maior fidelidade, a fim de se construir uma base de comparação efetiva.

AGRADECIMENTOS

Os autores gostariam de agradecer à agência de financiamento FAPESP pelo apoio dado aos projetos do Laboratório de Arquitetura de Sistemas (LAS) da UNIVEM, principalmente, pelos processos nº 2006/06671-0 e nº 2007/00868-0.

REFERÊNCIAS

BAKER, M. et al. **MpiJava**: A Java Interface to MPI. In: FIRST UKWORKSHOP ON JAVA FOR HIGH PERFORMANCE NETWORK COMPUTING, Europar, 1998.

BARBOSA, Jorge M. G. **Paralelismo em Processamento e Análise de Imagem Médica.** p. 240. 2000. Tese (Doutorado) - Depto de Engenharia Electrotécnica e de Computadores – Faculdade de Engenharia da Universidade do Porto, Porto, 2000.

BRANCO, K. R. L. J. C. **Extensão da Ferramenta de Apoio à Programação Paralela (F.A.P.P) para Ambientes Paralelos Virtuais**. São Carlos, p. 152. 1999. Dissertação (Mestrado) - Instituto de Ciências Matemáticas e de Computação de São Carlos – Universidade de São Paulo, 1999.

FERRARI, A. J. **JPVM:** Network parallel computing in Java, In: ACM 1998 WORKSHOP ON JAVA FOR HIGH-PERFORMANCE NETWORK COMPUTING, Palo Alto, February 1998. Concurrency: Pactice and Experience, 1998.

GIGER, M. L. **Computer-Aided Diagnosis of Breast Lesions in Medical Images**. In: COMPUTING IN SCIENCE & ENGINEERING, v. 2, n. 5, p. 39-45, 2000.

GONZALEZ, R. C.; WOODS, R. E. **Digital Image Processing**. In: ADDISON-WESLEY PUBLISHING COMPANY, Massachusetts, 2. ed., 2002.

NUNES, F. L. S.; RODELLO, I., A.; BREGA, J.R.F.; BRANCO, K.R.L.J.C. (Org). **Processamento de Imagens Médicas para Sistemas de Auxílio ao Diagnóstico**. Escola Regional de Informática São Paulo/Oeste. 1. ed. SP: Marília/Bauru, 2006, cap. 4, p. 83-135, 2006.

TABOADA, G. L.; TOURINO, J.; DOALLO, R. **Performance Modeling and Evaluation of Java Message-Passing Primitives on a Cluster**. Lecture Notes in Computer Science, v. 2840, p. 29-36, 2003.