XIV ERI-PR 2007
ISBN: 978-85-7669-115-0

Uso da Computação Paralela Distribuída para Melhoria no Tempo de Processamento de Imagens Médicas

Priscila T. M. Saito, Ricardo J. Sabatine, Fátima L. S. Nunes, Kalinka R. L. Jaquie Castelo Branco

Univem – Centro Universitário Eurípides de Marília
Fundação de Ensino "Eurípides Soares da Rocha"
Programa de Pós-Graduação em Ciência da Computação
Bacharelado em Ciência da Computação
Av. Hygino Muzzi Filho 529, CEP 17509-901, Marília, SP

{psaito, sabatine, fatima, kalinka}@univem.edu.br

Abstract. *This paper aims at demonstrating the viability in the use of parallel distributed computing to improve the image processing. Image processing techniques were implemented in the sequential and parallel way using the Java language and the parallel virtual libraries mpiJava and JPVM. A algorithm that executes the median filter was implemented and processed with different size of masks. From the results it was possible to make a comparison between the sequential and parallel implementation, and show that the gain using parallel distributed computing depending on the amount of data to be processed.*

Resumo. *Este artigo tem como objetivo demonstrar a viabilidade da melhoria no tempo de execução de algoritmos utilizados para o processamento de imagens através do uso da computação paralela distribuída. Técnicas de processamento de imagens foram implementadas de forma seqüencial e paralela, utilizando a linguagem Java e as bibliotecas de trocas de mensagens mpiJava e JPVM. Foi implementado o algoritmo filtragem mediana, sendo executado com diferentes tamanhos de máscaras. A partir dos resultados obtidos foi possível construir uma base de comparação entre a implementação seqüencial e a paralela, e mostrar que o ganho de desempenho obtido com o paralelismo dependendo depende da quantidade de dados a ser processada.*

ISBN: 978-85-7669-115-0

1. Introdução

O processamento de imagens é amplamente utilizado em muitas aplicações. Entretanto essa área de conhecimento exige intenso poder computacional, visto que as imagens apresentam, em sua grande maioria, tamanho elevado, constituindo matrizes enormes de pontos a serem processados, tornando-se inviável para quem não dispõe de tal poder de processamento.

Em se tratando de imagens médicas, esta questão é ainda mais crítica, uma vez que aplicações médicas requerem curto tempo de resposta e não se pode permitir armazenamento com perda de dados caso a imagem seja usada para diagnóstico. Exigindo precisão na sua aquisição, essa classe de imagens gera um conjunto ainda maior de dados [Barbosa 2000]. O tamanho elevado aliado à necessidade de passagem de algum filtro, seja para suavização, atenuação ou realce, aumenta o tempo de processamento dessas imagens, prejudicando a avaliação das mesmas [Nunes, 2001].

A necessidade de alto desempenho e alto poder computacional para o processamento desse tipo de imagem pode ser resolvida através do uso de sistemas distribuídos e de bibliotecas de passagem de mensagens, que viabilizam a computação paralela sobre sistemas distribuídos (computação paralela distribuída).

Este artigo está estruturado da seguinte forma: a seção 2 faz uma descrição da computação paralela distribuída. A seção 3 descreve o processamento de imagens médicas. A seção 4 apresenta um modelo de paralelismo para o processamento de imagens. Na seção 5 encontram-se os testes e resultados obtidos e a seção 6 conclui o artigo.

2. Computação Paralela Distribuída

Os sistemas computacionais distribuídos aplicados à computação paralela permitem uma melhor relação custo/benefício para a computação paralela. Estes sistemas oferecem a potência computacional adequada às aplicações paralelas que não necessitam de uma máquina maciçamente paralela, porém necessitam de uma potência computacional maior que uma máquina seqüencial pode oferecer [Branco, 1999].

Para a realização da computação paralela sobre sistemas distribuídos, é necessária uma camada de *software* que possa gerenciar o uso paralelo, pois existe a necessidade da passagem de informações entre as várias máquinas que compõe a plataforma.

Existem bibliotecas especializadas para o tratamento da comunicação entre processos e a sincronização dos processos concorrentes. Segundo Seinstra & Koelma (2004), a programação paralela com base na troca de mensagens requer do programador controle sobre a distribuição e troca de dados, além da especificação explícita da execução paralela do código entre os diferentes processadores, o que impõe um alto grau de dificuldade quando comparada à programação seqüencial.

Os sistemas baseados na troca de mensagens mais utilizados são o MPI (*Message Passing Interface*) [Snir, 1996] e o PVM (*Parallel Virtual Machine*) [Geist, 1994]. Estas bibliotecas provêem rotinas para iniciar e configurar o ambiente bem como enviar e receber mensagens de dados entre os elementos de processamento do sistema. Existem muitas implementações de MPI e PVM em aplicações desenvolvidas em linguagens como Fortran, C e C++. Com o surgimento de Java, inúmeras propostas foram apresentadas para a utilização dessas bibliotecas nessa linguagem, entre as quais

pode-se citar o mpiJava [Baker, 1998] e o JPVM (*Java Parallel Virtual Machine*) [Ferrari, 1998], ambientes de passagem de mensagens utilizados mais especificamente neste trabalho.

A escolha de ambos como ambiente para processamento paralelo dos algoritmos ocorre em decorrência do uso da linguagem Java, que contém fatores como portabilidade, permitindo a independência de plataforma, simplicidade e clareza nos códigos, além da existência de APIs especializadas que permitem o uso cada vez maior desta linguagem em processamento de imagem. Além disso, a linguagem vem apresentando uma inserção crescente nas organizações, o que aumenta a possibilidade de aplicações práticas dos algoritmos desenvolvidos.

2.1. mpiJava

mpiJava é uma interface que permite fazer uso da orientação a objetos em Java juntamente com a biblioteca MPI, amplamente usada na computação paralela e distribuída [Baker 1998]. Para tanto, as chamadas dos métodos obedecem à estrutura das funções definidas em MPI, o que torna a programação menos flexível. Também a portabilidade é atingida, uma vez que a chamada das funções MPI não é específica para uma determinada arquitetura. Em um nível de abstração mais baixo, mpiJava executa as funções nativas de uma implementação MPI, conforme a definição feita no momento da instalação.

Dentro das formas de comunicação possíveis em Java, a biblioteca apresenta bons resultados [Baker, 1999]. O uso de mpiJava já foi validado em diversas implementações e avaliações de desempenho [Taboada, 2003].

2.2. JPVM

JPVM é uma API implementada totalmente em Java que permite a troca de mensagens explícita, combinando as vantagens da linguagem Java como portabilidade e interoperabilidade com as técnicas de troca de mensagem entre processos paralelos em ambientes distribuídos, sendo uma implementação em Java da biblioteca *Parallel Virtual Machine*. A interface semelhante do JPVM em relação ao PVM permite a programadores acostumados ao PVM padrão a migração de um ambiente para o outro com uma maior facilidade.

O JPVM é composto por duas partes [Ferrari, 1998]: *JpvmDaemon*, processo que é executado em todos os nós a fim de formar uma máquina paralela virtual, realizar a comunicação entre os processos criados e coordenar as tarefas em execução; e o *JpvmEnvironment*, biblioteca que trata funções básicas para geração do paralelismo como troca de mensagens, criação e eliminação de processos, sincronização de tarefas, modificação da máquina virtual, envio e recebimento de mensagens.

3. Processamento de Imagens Médicas

A finalidade das imagens médicas é auxiliar na composição do diagnóstico de anomalias e fornecer material para acompanhamento de terapias, sendo estas imagens provenientes de diversos tipos de modalidades como Radiografia, Ultra-sonografia e Ressonância Magnética Nuclear. [Nunes, 2001]

As técnicas de processamento de imagem, reconhecimento de padrões, entre outras, são aplicadas com o objetivo de melhorar as imagens e extrair delas informações úteis aos diagnósticos. Existem inúmeros métodos de processamento de imagens. A

ISBN: 978-85-7669-115-0

escolha de procedimentos a serem aplicados depende do objetivo que se deseja em relação a uma determinada categoria de imagem.

De forma geral, as operações com imagens podem ser classificadas em baixo nível (pré-processamento), nível médio (segmentação) e nível alto (reconhecimento de padrões).

Neste trabalho o objetivo é verificar o desempenho de técnicas implementadas de forma seqüencial e paralela. Inicialmente foi escolhida uma técnica de suavização a fim de verificar a relação custo benefício decorrente da aplicação das tecnologias de computação paralela distribuída.

3.1. Suavização

Filtros de suavização são utilizados em uma etapa de pré-processamento para a redução de ruídos e para a remoção de pequenos detalhes de uma imagem antes da extração de objetos, assim como para a conexão de pequenas descontinuidades em linhas e curvas [Gonzales, 1992].

Entre as técnicas de suavização conhecidas existem as de suavização conservativa e técnicas de redução de ruídos. Entre as técnicas mais comuns de suavização estão os filtros de média e mediana. Em decorrência do processamento, efeitos colaterais podem ser observados nas imagens processadas. As técnicas de suavização não conservativas, como os filtros de média, por exemplo, muitas vezes, geram borramento, podendo eliminar detalhes como linhas finas e curvas agudas (Nunes, 2006).

A filtragem mediana, usada neste artigo, cujo algoritmo é apresentado na Figura 1, consiste em substituir o valor de um determinado *pixel* pelo valor mediano da sua vizinhança. O valor mediano é o valor central obtido quando se ordena os *pixels* da vizinhança. Com essa técnica, obtém-se a redução do ruído e diminui-se o borramento obtido por outros filtros de suavização, como o filtro de média, porém o filtro de mediana pode não ser tão eficaz na remoção de ruídos gaussianos. Um outro problema desse filtro é o seu custo relativamente complexo e caro. Como citado, para encontrar a mediana é necessária a ordenação de todos os valores da vizinhança, o que é relativamente lento, dependente principalmente do método de ordenação utilizado [Gonzales, 1992]. Para avaliação neste artigo foi utilizado o cálculo da mediana fazendo uso do algoritmo de ordenação *shellsort*.

Na Figura 2 são apresentados exemplos da aplicação da filtragem mediana em imagens médicas.

```
defina tamanho_template
defina vetor_elementos_mediana com tamanho_template^2 elementos
para linha = 1 até quantidade_linhas
    para coluna = 1 até quantidade_colunas
         vetor_elementos_mediana ← pixels da vizinhança
         ordena vetor_elementos_mediana
        pixel_linha,coluna ← vetor_elementos_mediana_Tamanho_template/2
    fim para
 fim para
```

Figura 1 – Algoritmo de Filtro de Mediana (Nunes, 2006)

ISBN: 978-85-7669-115-0

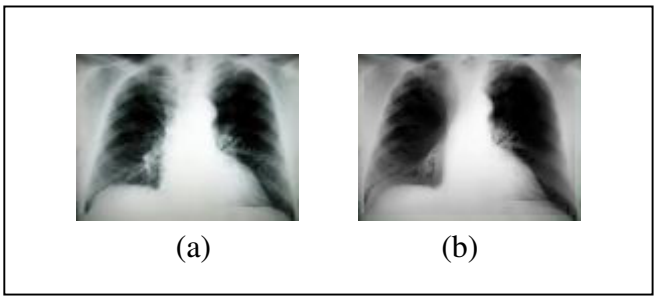

Figura 2 - Exemplos de suavização utilizando o filtro de mediana em imagens médicas. (a) imagem de Raio-X – original; (b) imagem suavizada. Os processamentos usaram *template* com tamanho 7x7 *pixels*.

4. Um Modelo de Paralelismo para o Processamento de Imagens

O requisito básico de um sistema de processamento paralelo de imagens consiste em uma infra-estrutura que permita a execução eficiente de algoritmos neste domínio, sejam de nível baixo - que executam alterações globais na imagem; de nível médio - que identificação de estruturas importantes na imagem ou de nível alto – relacionados com o reconhecimento de padrões.

Essa infra-estrutura é composta essencialmente de funções de comunicação e distribuição de dados adequados ao processamento de imagem [Barbosa, 2000].

Os filtros de processamento de imagens podem ser executados tanto no domínio da freqüência (considerando a definição matemática da imagem) quanto no domínio espacial, considerando o conjunto de *pixels* que formam a imagem. No entanto, o domínio espacial é o mais utilizado devido principalmente à facilidade de implementação, mas em contrapartida, exige alto poder de processamento, visto que as imagens constituem, na maioria das vezes, matrizes enormes de pontos a serem processados. No caso deste trabalho, a escolha do algoritmo é devida ao objetivo de processamento, buscando executar um filtro que exigisse uma quantidade elevada de operações a fim de que a diferença entre a implementação seqüencial e paralela pudesse ser evidenciada. É neste contexto que as técnicas de processamento de imagens e, em especial, aquelas desenvolvidas para aplicação em imagens médicas, podem se beneficiar dos conceitos de paralelização de imagens.

Quando se trata de imagens médicas a importância do paralelismo é ainda maior, visto que esta classe de imagens não pode permitir armazenamento com perda de dados e, muitas vezes, exige precisão na sua aquisição, gerando um conjunto ainda maior de dados.

Uma proposta de paralelização eficiente 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 neste caso, pois se pode 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 a ser vencido, neste caso e, mais especificamente em imagens médicas, é a divisão da imagem em blocos e a posterior junção desses blocos sem perdas de processamento.

A Figura 3 ilustra uma estratégia de paralelização em que uma imagem médica é dividida em blocos pelo mestre. Os blocos são subseqüentemente transmitidos por

ISBN: 978-85-7669-115-0

meio das bibliotecas mpiJava e JPVM aos escravos. Os escravos têm a incumbência de processar os blocos e devolvê-los já processados ao mestre, o qual os une, reconstituindo a imagem. Pode-se perceber também que os blocos apresentam tamanhos variados (como percebido pelo número de linhas em cada um dos blocos 1, 2 e 3), dependendo da característica de cada algoritmo.

O algoritmo abordado neste artigo faz uso de máscaras coeficientes que operam sobre uma "vizinhança" de pontos da imagem. Sendo assim, há necessidade de alguma redundância nos blocos para o processamento (destacada em vermelho na Figura 3).

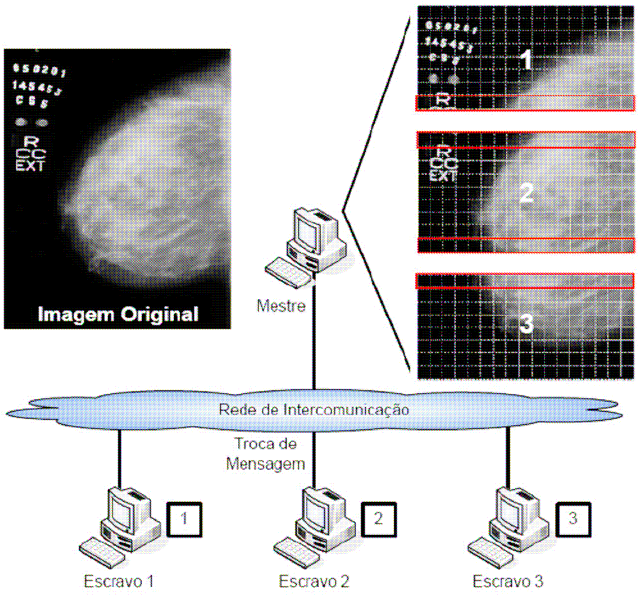

Figura 3 - Estratégia de paralelização para o processamento das imagens médicas.

5. Análise de Desempenho dos Resultados Obtidos

A análise de desempenho do algoritmo de filtro de mediana foi realizada através de diferentes testes reais em um ambiente paralelo distribuído composto inicialmente por 3 máquinas homogêneas (Pentium IV de 2.7GHz com 512Mbytes, interligadas por uma rede *ethernet* de 100Mb/s) e posteriormente foram acrescidas máquinas idênticas para compor um ambiente de 4 e 5 máquinas.

Foram utilizadas imagens de elevado tamanho (11 MB), característica típica das imagens médicas. O filtro de mediana foi avaliado com máscaras de tamanhos diferentes 3x3, 5x5 e 7x7, que definem a quantidade de pontos vizinhos que serão considerados no processamento de cada *pixel* da imagem.

Após a realização dos testes obteve-se uma média dos 30 tempos de processamento tanto para aplicação seqüencial quanto para a paralela (fazendo uso do filtro de mediana utilizando o algoritmo de ordenação *shellsort*) 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 sendo iniciados em paralelo, sendo possível, assim, realizar a comparação de desempenho de cada uma das possíveis combinações. Todos os testes foram efetuados fazendo uso das bibliotecas JPVM e mpiJava. Todos os resultados obtidos foram avaliados estatisticamente para comprovar seu grau de significância.

ISBN: 978-85-7669-115-0

As Figuras 4 e 5 apresentam os resultados da execução seqüencial do filtro de mediana bem como as execuções em paralelo do mesmo algoritmo em 3, 4 e 5 máquinas respectivamente. Pelas figuras é possível observar que o tempo de execução do algoritmo seqüencial, quando considerada a vizinhança de tamanho 3x3, é significativamente melhor (comprovação dada a partir das Tabelas 1 e 2) que o tempo do mesmo em paralelo fazendo uso tanto do JPVM quanto do mpiJava independentemente do número de máquinas e da quantidade de processos iniciados em paralelo. Uma vez que a quantidade de cálculo efetuada pelo filtro com máscara 3x3 é relativamente pequena, a paralelização do mesmo não impõe melhoria uma vez que 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 subvetor é maior que o tempo gasto pelos escravos para ordenar cada um desses subvetores, o que a torna uma aplicação mais voltada para comunicação do que para processamento.

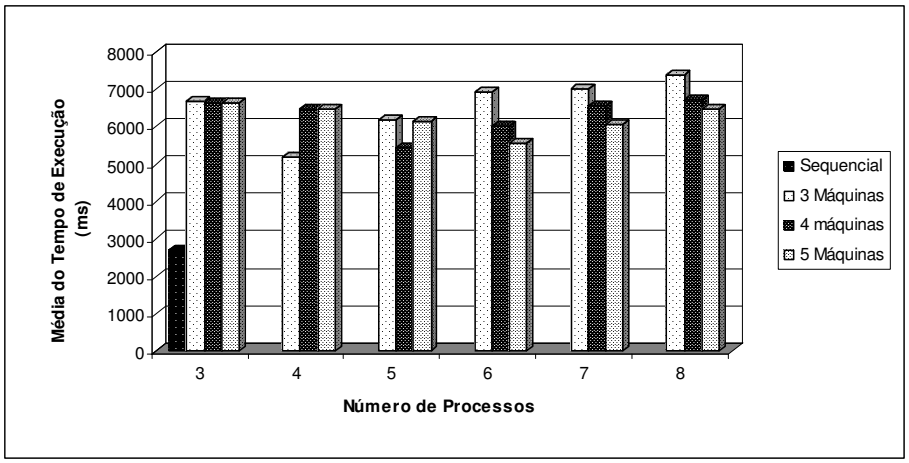

Figura 4 - Gráfico da execução da aplicação seqüencial comparada com a aplicação paralela do filtro de mediana de máscara 3x3 fazendo uso do ambiente de troca de mensagem mpiJava.

Tabela 1 - Valores calculados para o desempenho obtido com a execução de máscaras 3x3 das aplicações seqüencial e paralela do filtro de mediana com a utilização de 4 máquinas e 5 processos utilizando mpiJava.

3x3	Sequencial	Paralelo
Média(ms)	2674,533	5416,467
Desvio Padrão	45,233	129,263
Variância	2046,049	16709,049
Hipótese α= 0,01		109,662

As Figuras 6, 7, 8 e 9 apresentam os resultados das mesmas execuções anteriores levando em consideração a máscara 5x5 e 7x7, respectivamente. Pelas figuras é possível observar que o tempo de execução do algoritmo seqüencial a partir do uso dessas máscaras é significativamente pior (comprovação dada a partir das Tabelas 3, 4, 5 e 6) que o tempo de execução em paralelo, tanto fazendo uso do JPVM quanto do mpiJava. Isso pode ser observado principalmente para situações onde o número de processos equivale ou é menor que o número de máquinas. Em caso onde o número de processos ultrapassa uma unidade em cada máquina o tempo paralelo fica um pouco prejudicado

ISBN: 978-85-7669-115-0

no caso da máscara 5x5. Mais uma vez isso acontece devido a quantidade de processamento não sobrepor a quantidade de comunicação envolvida.

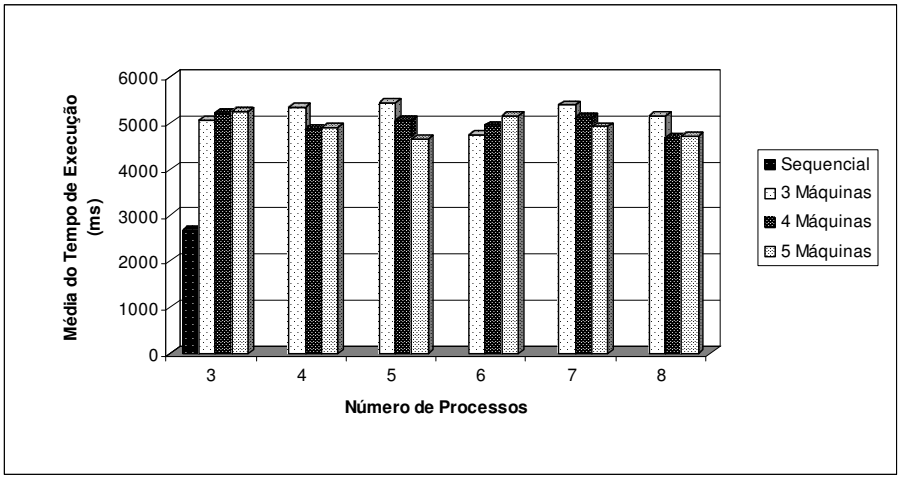

Figura 5 - Gráfico da execução da aplicação seqüencial comparada com a aplicação paralela do filtro de mediana de máscara 3x3 fazendo uso do ambiente de troca de mensagem JPVM.

Tabela 2 - Valores calculados para o desempenho obtido com a execução de máscaras 3x3 das aplicações seqüencial e paralela do filtro de mediana com a utilização de 4 máquinas e 5 processos utilizando JPVM.

3x3	Sequencial	Paralelo
Média(ms)	2615,767	5057,800
Desvio Padrão	31,199	139,621
Variância	973,379	19494,160
Hipótese α= 0,01		93,493

À medida que se aumenta o número de processos para certa quantidade de máquinas, o desempenho diminui, tornando-se maior que o tempo médio seqüencial. Isso acontece porque quando se aumenta o número de processos em alguns casos é evidente a queda de desempenho.

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 melhor que o tempo seqüencial. Isso acontece porque o processamento é alto. Essa diferença e significativa melhora de desempenho, quando se faz uso de uma arquitetura paralela distribuída, dá-se pelo fato dos cálculos efetuados pelas máscaras serem volumosos, o que garante uma melhoria do seu uso em paralelo uma vez que a comunicação imposta se torna menos relevante quando comparada ao ganho em relação aos cálculos efetuados. Isso significa que a quantidade de comunicação exigida exerce menor influência na avaliação, visto que o volume de dados a serem transmitidos é menor que o volume de dados a ser processado.

ISBN: 978-85-7669-115-0

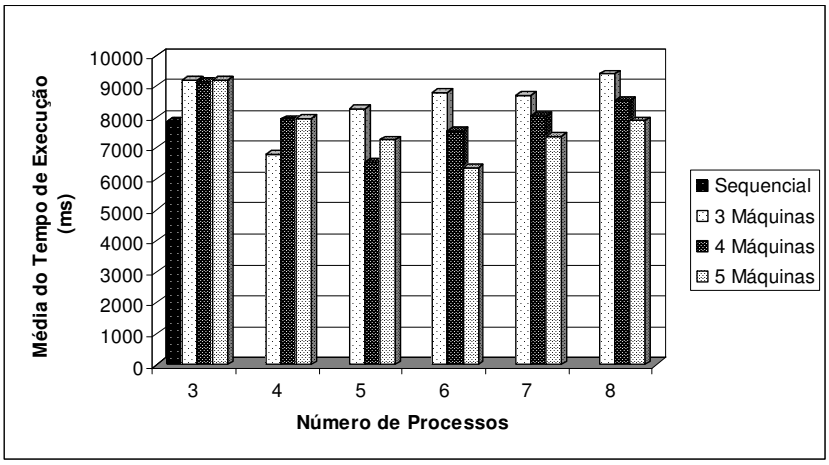

Figura 6 - Gráfico da execução da aplicação seqüencial comparada com a aplicação paralela do filtro de mediana de máscara 5x5 fazendo uso do ambiente de troca de mensagem mpiJava.

Tabela 3 - Valores calculados para o desempenho obtido com a execução de máscaras 5x5 das aplicações seqüencial e paralela com a utilização do filtro de mediana de 4 máquinas e 5 processos utilizando mpiJava.

5x5	Sequencial	Paralelo
Média(ms)	7871,067	6532,233
Desvio Padrão	15,431	203,036
Variância	238,129	41223,446
Hipótese α= 0,01		-36,013

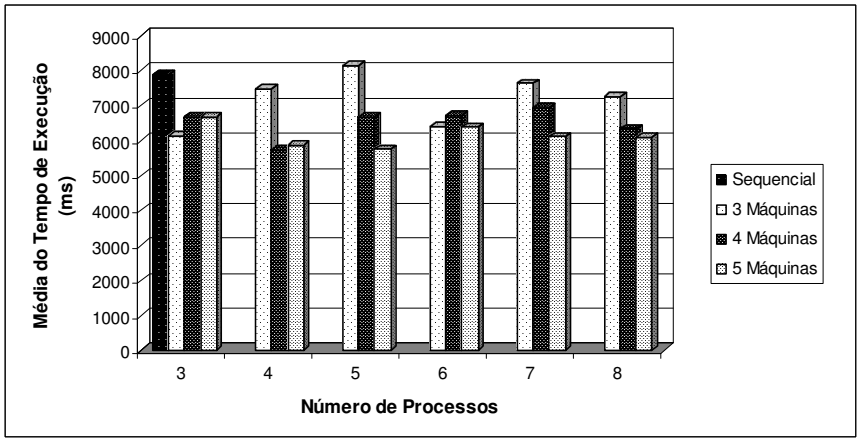

Figura 7 - Gráfico da execução da aplicação seqüencial comparada com a aplicação paralela do filtro de mediana de máscara 5x5 fazendo uso do ambiente de troca de mensagem JPVM.

Tabela 4 - Valores calculados para o desempenho obtido com a execução de máscaras 5x5 das aplicações seqüencial e paralela do filtro de mediana com a utilização de 4 máquinas e 5 processos utilizando JPVM.

5x5	Sequencial	Paralelo
Média(ms)	6956,767	6656,733
Desvio Padrão	155,886	203,732
Variância	24300,512	41506,596
Hipótese α= 0,01		-6,406

ISBN: 978-85-7669-115-0

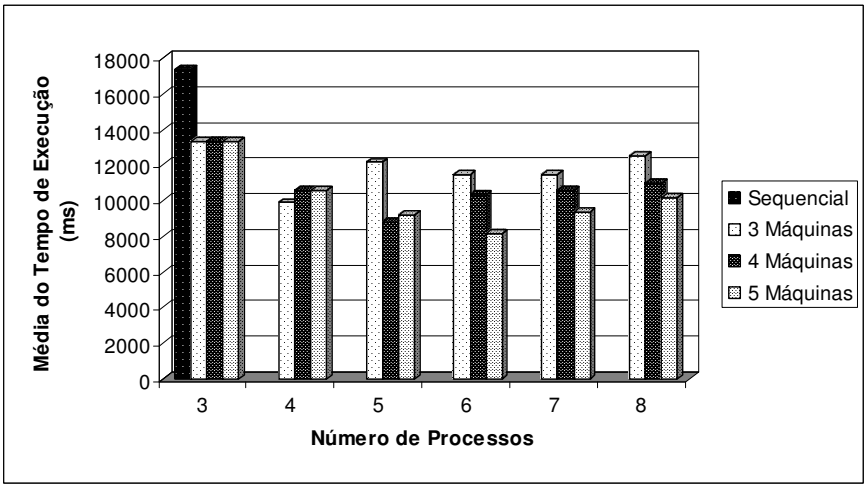

Figura 8 - Gráfico da execução da aplicação seqüencial comparada com a aplicação paralela do filtro de mediana de máscara 7x7 fazendo uso do ambiente de troca de mensagem mpiJava.

Tabela 5 - Valores calculados para o desempenho obtido com a execução de máscaras 7x7 das aplicações seqüencial e paralela do filtro de mediana com a utilização de 4 máquinas e 5 processos.

7x7	Sequencial	Paralelo
Média(ms)	17360,500	8818,033
Desvio Padrão	17,068	132,085
Variância	291,317	17446,366
Hipótese α= 0,01		-351,314

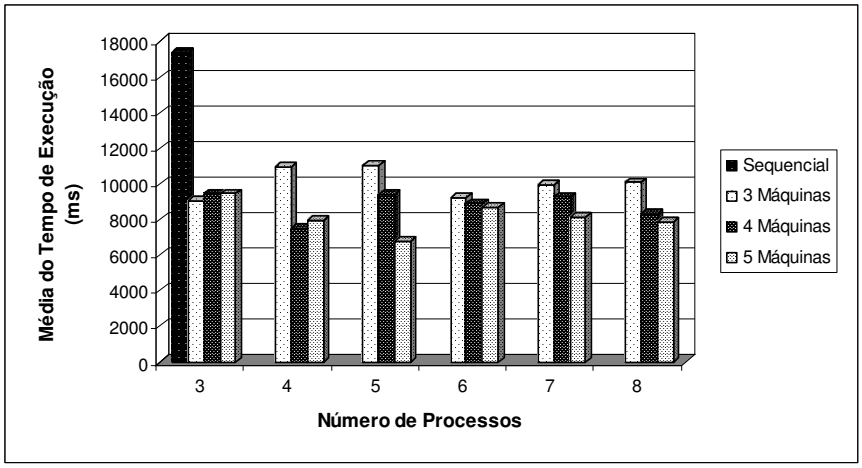

Figura 9 - Gráfico da execução da aplicação seqüencial comparada com a aplicação paralela do filtro de mediana de máscara 7x7 fazendo uso do ambiente de troca de mensagem JPVM.

Na Figura 10, pode-se observar os tempos de execução seqüencial e paralela 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 principalmente quando se inicia um número maior de processos do que o número de máquinas. Para poder avaliar as diferenças entre as bibliotecas novos estudos devem ser efetuados.

Tabela 6 - Valores calculados para o desempenho obtido com a execução de máscaras 7x7 das aplicações seqüencial e paralela do filtro de mediana com a utilização de 4 máquinas e 5 processos utilizando JPVM.

7x7	Sequencial	Paralelo
Média(ms)	14963,200	9418,467
Desvio Padrão	438,515	87,743
Variância	192295,693	7698,782
Hipótese α= 0,01		-67,910

De qualquer forma, seja considerado JPVM ou mpiJava, em ambos os casos o desempenho médio da execução em paralelo é significativamente melhor que o da seqüencial.

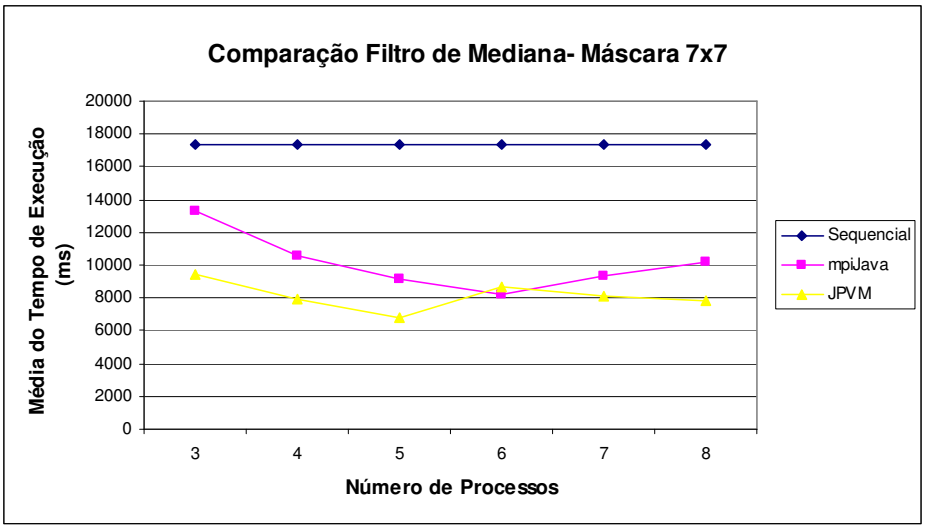

Figura 10 - Gráfico da execução da aplicação seqüencial comparada com as aplicações paralelas do filtro de mediana de máscara 7x7 fazendo uso dos ambientes de troca de mensagem mpiJava e JPVM.

6. Conclusões

Partindo-se 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 também possa prover melhora significativa de desempenho para a versão paralela quando comparada à execução seqüencial.

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

Com base nisso tem-se como trabalhos futuros o desenvolvimento de novos algoritmos de processamento de imagens bem como a avaliação de outros algoritmos de ordenação presentes nestes algoritmos de processamento. Acredita-se também que a paralelização desses algoritmos de ordenação possa prover diminuição ainda mais significativa nos resultados obtidos com o processamento paralelo, uma vez que já se tem comprovado na literatura [Cortés, 1999] que a paralelização de algoritmos de ordenação permite um aumento de desempenho quando comparado a sua versão seqüencial.

Testes adicionais devem ser executados para se avaliar também a superioridade ou não da biblioteca JPVM sobre a biblioteca mpiJava, para isso serão efetuados, também como trabalhos futuros, a execução de outros algoritmos de processamento de imagens tanto em uma quanto em outra biblioteca, o que permitirá a obtenção de um conjunto maior de dados a partir do qual tais informações poderão ser extraídas.

7. Referências

Baker, M. et al. (1998) mpiJava: A Java Interface to MPI. Submitted to First UKWorkshop on Java for High Performance Network Computing, Europar.

Baker, M. et al. (1999) mpiJava: An Object-Oriented Java Interface to MPI.

Barbosa, Jorge M. G. (2000) Paralelismo em Processamento e Análise de Imagem Médica. Tese (Doutorado) apresentada ao Departamento de Engenharia Electrotécnica e de Computadores – Faculdade de Engenharia da Universidade do Porto, p.240.

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

Cortés, O. A. C. (1999) Desenvolvimento e Avaliação de Algoritmos Numéricos Paralelos. Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação - USP, São Carlos, São Paulo.

Ferrari, A. J.; (1998) 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.

Geist, A.; Beguelin, A.;Dongarra, J.; Jiang, W.; Mancheck, B.; Sunderam, V. (1994) PVM: Parallel Virtual Machine – A User´s Guide and Tutorial for Network Parallel Computing, The MIT Press.

Gonzales, R. F.; Woods, P. (1992) Digital Image Processing, Addison-Wesley.

Nunes, F. L. S. (2001) Investigações em Processamento de Imagens Mamográficas para Auxílio ao Diagnóstico de Mamas Densas. São Carlos, p.230. Tese (Doutorado) apresentada ao Instituto de Física de São Carlos – Universidade de São Paulo.

Nunes, F. L. S. (2006) Introdução ao processamento de imagens médicas para auxílio ao diagnóstico. Breitman, K.; Anido, R. (Org). Atualizações em Informática. 1 ed. Rio de Janeiro: PUC-Rio, 2006, v. 1, p. 73-126.

Seinstra, F.J.; Koelma, D. (2004) User Transparency: A Fully Sequential Programming Model for Efficient Data Parallel Image Processing, Concurrency and Computation: Practice and Experience, vol. 16, no. 6, pp. 611-644, May 2004.

Snir, M. Otto, S. M., Huss-Lederman, S., Dongarra, J. (1996) MPI: The Complete Reference, The MIT Press.

Taboada, G. L.; Tourino, J.; Doallo, R. (2003) Performance Modeling and Evaluation of Java Message-Passing Primitives on a Cluster. Lecture Notes in Computer Science, v.2840, p.29-36.