FUNDAÇÃO DE ENSINO “EURÍPIDES SOARES DA ROCHA”
CENTRO UNIVERSITÁRIO “EURÍPIDES DE MARÍLIA” – UNIVEM
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

RICARDO JOSÉ SABATINE

IMPLEMENTAÇÃO DE ALGORITMO PARALELO PARA APOIAR O PROCESSAMENTO DE IMAGENS UTILIZANDO JPVM

MARÍLIA
2007

RICARDO JOSÉ SABATINE

IMPLEMENTAÇÃO DE ALGORITMO PARALELO PARA APOIAR O PROCESSAMENTO DE IMAGENS UTILIZANDO JPVM

Monografia apresentada ao Curso de Graduação em Ciência da Computação, do Centro Universitário Eurípides de Marília, mantido pela Fundação de Ensino Eurípides Soares da Rocha, como requisito para obtenção do grau de Bacharel em Ciência da Computação.

Orientadora:
Profª. Drª. Kalinka R. L. J. C. Branco.

MARÍLIA
2007

SABATINE, Ricardo José

Implementação de Algoritmo Paralelo para Apoiar O Processamento de Imagens Utilizando JPVM / Ricardo José Sabatine; orientadora: Profª. Drª. Kalinka R. L. J. C. Branco. Marília, SP: [s.n.], 2007.

127 f.

Trabalho de Conclusão de Curso (TCC) - Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha.

1. JPVM 2. Imagens Médicas 3. Processamento Paralelo

CDD: 005.2

AGRADECIMENTOS

A Deus, Nossa Senhora, São Franscico de Assis, São Miguel, ao meu anjo da guarda.

À minha mãe, meu pai, minha irmã, meu irmão, minha avó, aos meus mais de 30 tios e tias, aos meus mais de 70 primos, a minha sobrinha, a minha ex-cunhada e aos meus vizinhos.

Ao Thiago, Sono, Mendonça pela amizade nestes quatro anos e a Saito pela ajuda durante estes dois últimos anos.

Ao restante da sala por não ter nem ajudado e nem atrapalhado.

À Michele Noda (Yoda) pessoa pelo qual tenho enorme admiração e carinho. Agradeço pelos cookies, pelo prejuízo financeiro (brincadeira), pelos sermões, pela psicologia, pela preocupação, pelos cachorros entre eles a falecida Nina.

Ao pessoal do mestrado, Cleber, Silvia (Figurinha, pessoa a qual tenho enorme admiração e carinho) Rodrigo (Lost), Rafael (Rio Verde), Dinaldo, Ivair entre outros.

Ao Google e ao Yahoo pela ajuda diária.

À Giulianna Marega (Marques, 2007) pessoa pelo qual tenho enorme admiração, carinho e preocupação. Qualquer forma de agradecimento será insignificante ao que ela realmente merece. Minha orietadora de mentirinha.

À minha orientadora Prof[a]. Dr[a]. Kalinka R. L. J. Castelo Branco, muito mais que uma simples orientadora, uma verdadeira mãe. Nestes dois últimos anos sempre se posicionou de maneira examplar, digna e verdadeira com todos os seus orientados. Serei sempre grato.

À minha "co-orientadora" a menininha Prof[a]. Dr[a]. Fátima alguma coisa (nunca lembro o sobrenome dela, já basta o da minha orientadora!), pelos ensinamentos e confiança depositada em mim. Por ser exemplo tanto para a vida pessoal quanto profissional (parece o Faustão falando).

Aos professores da computação (Beto, Maria Istela, Valter, Delamaro, Sementille, Remo, Raul, Rodolfo, Elton, Galante, Maria Cristina, Celso, Bugatti, Juliana (Inglês), Perozim, Dino, Edward, de Lucca, Marcelo (Direito), Mucheroni, ...), às faxineiras, ao pessoal da coordenação e aos vigias do Univem.

À FAPESP pelo apoio financeiro.

SABATINE, Ricardo José. Implementação de Algoritmo Paralelo para Apoiar O Processamento de Imagens Utilizando JPVM. 2007. 127f. Monografia (Bacharelado em Ciência da Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2007.

.

RESUMO

Este trabalho tem como objetivo 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. Técnicas de processamento de imagens foram implementadas de forma seqüencial e paralela utilizando a linguagem Java e a biblioteca de troca de mensagem JPVM. Foram implementados algoritmos de suavização e de detecção de bordas no domínio espacial, fazendo uso de diferentes tamanhos de máscaras. Após a implementação foi possível construir uma base de comparação entre a aplicação seqüencial e a paralela, o que permitiu avaliar o ganho de desempenho obtido com o paralelismo.

Palavras-Chave: Computação Paralela, Sistemas Distribuídos, Processamento de Imagem, Imagens Médicas.

SABATINE, Ricardo José. Implementação de Algoritmo Paralelo para Apoiar O Processamento de Imagens Utilizando JPVM. 2007. 127f. Monografia (Bacharelado em Ciência da Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2007.

ABSTRACT

This work aims at demonstrating the viability in the use of parallel distributed computing to improve the execution time of algorithms used to the medical image processing. Image processing techniques were implemented in the sequential and parallel way by using the program language Java and the parallel virtual library JPVM. Smoothing and edge detection algorithms were implemented in the special domain, using different size of masks. After the implementation, it was possible to make a comparison between the sequential and parallel application, that permitted to evaluate and to demonstrate the gain using distributed parallel computing.

Keywords: Parallel Computing, Distributed Computing System, Image Processing, Medical Images.

LISTA DE ILUSTRAÇÕES

Figura 1 - Processos em um Sistema Fisicamente Paralelo (STALLINGS, 2003). 24

Figura 2 - Processos em um Sistema Logicamente Paralelos (pseudo-paralelismo) (STALLINGS, 2003). 24

Figura 3 - Taxonomia de Flynn (TANENBAUM, 2001b). 25

Figura 4 - Arquitetura multiprocessador (TANENBAUM, 2001a)..... 27

Figura 5 - Arquitetura Multicomputador (TANENBAUM, 2001a). 28

Figura 6 - Taxonomia para máquinas de arquitetura paralela (TANENBAUM, 2001b) 29

Figura 7 - Exemplo de Vizinhança N4(P) 44

Figura 8 - Exemplo de Vizinhança ND(P) 45

Figura 9 - Exemplo de Vizinhança N8(P) 45

Figura 10 - Exemplo de uma máscara 3x3 genérica 47

Figura 11 - Exemplo filtro mediana máscara 3x3 48

Figura 12 - Algoritmo para implementação do filtro da mediana (NUNES, 2006). 48

Figura 13 - Região e máscaras para detecção de bordas: (a) máscara para detecção de bordas horizontais; (b) máscara para detecção de bordas verticais; (c) máscara para detecção de bordas 45 graus; (c) máscara para detecção de bordas 45 graus negativos. 50

Figura 14 - Organização dos dados Blocos (BARBOSA, 2000)..... 52

Figura 15 - Estratégia de paralelização para algoritmos baseado em máscara (NICOLESCU e JONKER 2002) 53

Figura 16 - Estratégia de paralelização de imagens médica..... 57

Figura 17 - Implementação genérica do algoritmo paralelo utilizado para o processamento de imagens 58

Figura 18 - Diagrama de classe referente ao programa mestre 59

Figura 19 - Construtor da classe Image 60

Figura 20 - Constutor classe MasterProcess..... 61

Figura 21 - Altura do bloco 62

Figura 22 – Inicializando Ambiente Paralelo..... 62

Figura 23 – Geração do bloco 63

Figura 24 – Empacotamento e Envio para Escravos. 64

Figura 25 – Redimensiona altura do último bloco..... 64

Figura 26 – Trecho de código referente a divisão da imagem 65

Figura 27 – Recebimento das Mensagens Enviadas pelos Escravos 65

Figura 28 - Define posição inicial correspondente do bloco na imagem final..... 66

Figura 29 – Eliminação da redundância nos blocos 66

Figura 30 – Cópia dos dados do bloco para imagem original 67

Figura 31 - Diagrama de classe referente ao programa mestre 67

Figura 32 – Estrutura interna do construtor da classe Slave 68

Figura 33 – Método *ReceiveMessage* classe *Slave*..... 69

Figura 34 – Método *ImagProcesing* classe *Slave*..... 69

Figura 35 – Método SendMessage classe Slave..... 70

LISTA DE GRÁFICOS

Gráfico 1 - Média do tempo de execução em paralelo do filtro de mediana com máscaras 3X3, imagem de 11MB 74

Gráfico 2 - Média do tempo de execução em paralelo do filtro de mediana com máscaras 5X5, imagem de 11MB 78

Gráfico 3 - Média do tempo de execução em paralelo do filtro de mediana com máscaras 7x7, imagem de 11MB 81

Gráfico 4 - Média do tempo de execução em paralelo do filtro de mediana com máscaras 3X3, imagem de 21MB 84

Gráfico 5 - Média do tempo de execução em paralelo do filtro de mediana com máscaras 5X5, imagem de 21MB 87

Gráfico 6 - Média do tempo de execução em paralelo do filtro de mediana com máscaras 7x7, imagem de 21MB 90

Gráfico 7 - Média do tempo de execução em paralelo do filtro de detecção de borda com máscaras 9x9, imagem de 11MB 93

Gráfico 8 - Média do tempo de execução em paralelo do filtro de detecção de borda com máscaras 11x11, imagem de 11MB 97

Gráfico 9 - Média do tempo de execução em paralelo do filtro de detecção de borda com máscaras 9x9, imagem de 21MB 100

Gráfico 10 - Média do tempo de execução em paralelo do filtro de detecção de borda com máscaras 11x11, imagem de 21MB 103

LISTA DE TABELAS

Tabela 1. Funções de controle do PVM 41

Tabela 2. Funções de passagem de mensagem do PVM 42

Tabela 3. Métodos da classe Image 60

Tabela 4. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 3x3 em imagens de 11MB com 3 máquinas 76

Tabela 5. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 3x3 em imagens de 11MB com 4 máquinas 76

Tabela 6. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 3x3 em imagens de 11MB com 5 máquinas 76

Tabela 7. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 3x3 em imagens de 11MB com 6 máquinas 77

Tabela 8. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 3x3 em imagens de 11MB com 7 máquinas 77

Tabela 9. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 3x3 em imagens de 11MB com 8 máquinas 77

Tabela 10. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 5x5 em imagens de 11MB com 3 máquinas 79

Tabela 11. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 5x5 em imagens de 11MB com 4 máquinas 79

Tabela 12. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 5x5 em imagens de 11MB com 5 máquinas 79

Tabela 13. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 5x5 em imagens de 11MB com 6 máquinas 80

Tabela 14. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 5x5 em imagens de 11MB com 7 máquinas 80

Tabela 15. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 5x5 em imagens de 11MB com 8 máquinas 80

Tabela 16. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 7x7 em imagens de 11MB com 3 máquinas 82

Tabela 17. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 7x7 em imagens de 11MB com 4 máquinas 82

Tabela 18. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 7x7 em imagens de 11MB com 5 máquinas 82

Tabela 19. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 7x7 em imagens de 11MB com 6 máquinas 83

Tabela 20. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 7x7 em imagens de 11MB com 7 máquinas 83

Tabela 21. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 7x7 em imagens de 11MB com 8 máquinas 83

Tabela 22. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 3x3 em imagens de 21MB com 3 máquinas 85

Tabela 23. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 3x3 em imagens de 21MB com 4 máquinas 85

Tabela 24. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 3x3 em imagens de 21MB com 5 máquinas 85

Tabela 25. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 3x3 em imagens de 21MB com 6 máquinas 86

Tabela 26. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 3x3 em imagens de 21MB com 7 máquinas 86

Tabela 27. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 3x3 em imagens de 21MB com 8 máquinas 86

Tabela 28. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 5x5 em imagens de 21MB com 3 máquinas 88

Tabela 29. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 5x5 em imagens de 21MB com 4 máquinas 88

Tabela 30. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 5x5 em imagens de 21MB com 5 máquinas 88

Tabela 31. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 5x5 em imagens de 21MB com 6 máquinas 89

Tabela 32. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 5x5 em imagens de 21MB com 7 máquinas 89

Tabela 33. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 5x5 em imagens de 21MB com 8 máquinas 89

Tabela 34. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 7x7 em imagens de 21MB com 3 máquinas 91

Tabela 35. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 7x7 em imagens de 21MB com 4 máquinas 91

Tabela 36. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 7x7 em imagens de 21MB com 5 máquinas 91

Tabela 37. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 7x7 em imagens de 21MB com 6 máquinas 92

Tabela 38. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 7x7 em imagens de 21MB com 7 máquinas 92

Tabela 39. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 7x7 em imagens de 21MB com 8 máquinas 92

Tabela 40. Teste hipótese, *Speedup* e Eficiência filtro de detecção de borda para máscara 9x9 em imagens de 11MB com 3 máquinas 95

Tabela 41. Teste hipótese, *Speedup* e Eficiência filtro de detecção de borda para máscara 9x9 em imagens de 11MB com 4 máquinas 95

Tabela 42. Teste hipótese, *Speedup* e Eficiência filtro de detecção de borda para máscara 9x9 em imagens de 11MB com 5 máquinas 95

Tabela 43. Teste hipótese, *Speedup* e Eficiência filtro de detecção de borda para máscara 9x9 em imagens de 11MB com 6 máquinas 96

Tabela 44. Teste hipótese, *Speedup* e Eficiência filtro de detecção de borda para máscara 9x9 em imagens de 11MB com 7 máquinas 96

Tabela 45. Teste hipótese, *Speedup* e Eficiência filtro de detecção de borda para máscara 9x9 em imagens de 11MB com 8 máquinas 96

Tabela 46. Teste hipótese, *Speedup* e Eficiência filtro de detecção de borda para máscara 11x11 em imagens de 11MB com 3 máquinas 98

Tabela 47. Teste hipótese, *Speedup* e Eficiência filtro de detecção de borda para máscara 11x11 em imagens de 11MB com 4 máquinas 98

Tabela 48. Teste hipótese, *Speedup* e Eficiência filtro de detecção de borda para máscara 11x11 em imagens de 11MB com 5 máquinas 98

Tabela 49. Teste hipótese, *Speedup* e Eficiência filtro de detecção de borda para máscara 11x11 em imagens de 11MB com 6 máquinas 99

Tabela 50. Teste hipótese, *Speedup* e Eficiência filtro de detecção de borda para máscara 11x11 em imagens de 11MB com 7 máquinas 99

Tabela 51. Teste hipótese, *Speedup* e Eficiência filtro de detecção de borda para máscara 11x11 em imagens de 11MB com 8 máquinas 99

Tabela 52. Teste hipótese, *Speedup* e Eficiência filtro de detecção de borda para máscara 9x9 em imagens de 21MB com 3 máquinas 101

Tabela 53. Teste hipótese, *Speedup* e Eficiência filtro de detecção de borda para máscara 9x9 em imagens de 21MB com 4 máquinas 101

Tabela 54. Teste hipótese, *Speedup* e Eficiência filtro de detecção de borda para máscara 9x9 em imagens de 21MB com 5 máquinas 101

Tabela 55. Teste hipótese, *Speedup* e Eficiência filtro de detecção de borda para máscara 9x9 em imagens de 21MB com 6 máquinas 102

Tabela 56. Teste hipótese, *Speedup* e Eficiência filtro de detecção de borda para máscara 9x9 em imagens de 21MB com 7 máquinas 102

Tabela 57. Teste hipótese, *Speedup* e Eficiência filtro de detecção de borda para máscara 9x9 em imagens de 21MB com 8 máquinas 102

Tabela 58. Teste hipótese, *Speedup* e Eficiência filtro de detecção de borda para máscara 11x11 em imagens de 21MB com 3 máquinas 104

Tabela 59. Teste hipótese, *Speedup* e Eficiência filtro de detecção de borda para máscara 11x11 em imagens de 21MB com 4 máquinas 104

Tabela 60. Teste hipótese, *Speedup* e Eficiência filtro de detecção de borda para máscara 11x11 em imagens de 21MB com 5 máquinas 104

Tabela 61. Teste hipótese, *Speedup* e Eficiência filtro de detecção de borda para máscara 11x11 em imagens de 21MB com 6 máquinas 105

Tabela 62. Teste hipótese, *Speedup* e Eficiência filtro de detecção de borda para máscara 11x11 em imagens de 21MB com 7 máquinas 105

Tabela 63. Teste hipótese, *Speedup* e Eficiência filtro de detecção de borda para máscara 11x11 em imagens de 21MB com 8 máquinas 105

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Program Interface*
CAD	*Computer-Aided Diagnosis*
CPU	*Central Processing Unit*
JAI	*Java Advanced Image*
JPVM	*Java Parallel Virtual Machine*
LAN	*Local Area Network*
MB	*Mega Bytes*
MIMD	*Multiple Instruction Stream/ Multiple Data Stream*
MISD	*Multiple Instruction Stream/ Multiple Data Stream*
MPI	*Message Passage Inteface*
mpiJava	*Message Passage Inteface JAVA*
MPP	*Massively Parallel Processing*
PVM	*Parallel Virtual Machine*
SIMD	*Single Instruction Stream/ Multiple Data Stream*
SISD	*Single Instruction Stream/ Single Data Stream*
SPMD	*Single Program - Multiple Data*

SUMÁRIO

AGRADECIMENTOS iv

RESUMO v

ABSTRACT vi

LISTA DE ILUSTRAÇÕES vii

LISTA DE GRÁFICOS ix

LISTA DE TABELAS x

LISTA DE ABREVIATURAS E SIGLAS xvi

SUMÁRIO xvii

INTRODUÇÃO 20

Objetivos 22

Organização do Documento 22

CAPÍTULO 1. COMPUTACÃO PARALELA DISTRIBUÍDA 23

1.1. Computação Paralela 23

1.1.1. Arquiteturas Paralelas 25

1.2. Sistemas Distribuídos 30

1.2.1. Modelos de Arquitetura 31

1.3. Computação Paralela Distribuída 32

1.4. Considerações Finais 33

CAPÍTULO 2. AMBIENTES DE PASSAGEM DE MENSAGENS 34

2.1. *Parallel Virtual Machine* (PVM) 35

2.2. *Message Passing Interface* (MPI) 37

2.3. Java e Ferramentas para Troca de Mensagem 37

2.3.1. *Message Passing Interface Java* (mpiJava) 38

2.3.2. *Java Parallel Virtual Machine* (JPVM) 38
2.3.2.1. Modelo computacional 39
2.3.2.2. Comunicação no JPVM 40
2.3.2.3. Interface de Programação 41
2.4. Considerações Finais 43
CAPÍTULO 3. PROCESSAMENTO DE IMAGENS MÉDICAS 44
3.1. Filtragem Espacial 47
3.1.1. Suavização 47
3.2. Segmentação 49
3.2.1. Detecção de Bordas 49
3.3. JAI (Java Advanced Imaging) 50
3.4. Um Modelo de Paralelismo para o Processamento de Imagens 51
3.4.1. Trabalhos Correlatos 54
3.5. Considerações Finais 56
CAPÍTULO 4. IMPLEMENTAÇÃO DE ALGORITMO PARALELO PARA APOIAR O PROCESSAMENTO DE IMAGENS UTILIZANDO JPVM 57
4.1. Desenvolvimento do Algoritmo Paralelo 58
4.1.1. Implementação do Mestre 59
4.1.1.1. Classe Image 59
4.1.1.2. Classe MasterProcess 60
4.1.2. Implementação do Escravo 67
4.1.2.1. Classe Slave 68
4.2. Considerações Finais 70
CAPÍTULO 5. TESTES E RESULTADOS 71
5.1. Análise Estatística Considerada 71

5.2. Resultados do Filtro de Mediana 74
5.2.1. Imagem de Tamanho 2048x2751 74
5.2.2. Imagem de Tamanho 2855X3835 84
5.3. Resultados Filtro de Detecção de Borda 93
5.3.1. Imagem de Tamanho 2751x2021 93
5.3.2. Imagem de Tamanho 2751x2321 100
5.4. Considerações Finais 106
CAPÍTULO 6. Conclusão 107
6.1. Trabalhos futuros 107
6.2. Produção Bibliográfica 108
6.2.1. Artigos Completos Publicados em Periódicos 108
6.2.2. Trabalhos Completos Publicados em Anais de Congressos 108
6.2.3. Resumos Publicados em Anais de Congressos 109
REFERÊNCIAS BIBLIOGRÁFICAS 111
Apêndice A 116
Apêndice B 117
Apêndice C 119
Apêndice D 121
ANEXO 126

INTRODUÇÃO

O processamento de imagens para aplicações médicas tem se mostrado um campo importante de pesquisa científica e tecnológica. Está entre as tecnologias computacionais que mais crescem atualmente, uma vez que a imagem médica é um dos recursos mais utilizados no auxílio ao diagnóstico médico.

O processamento das imagens médicas requer uma quantidade grande de recursos de *hardware* devido ao grande volume de dados a serem processados a um curto tempo de resposta, tornando-se inviável para quem não dispõe de tais recursos. Exige-se precisão na sua aquisição e processamento, pois não se permite armazenamento com perdas de dados (BARBOSA, 2000).

O tamanho elevado da imagem 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).

O processamento paralelo se apresenta como uma solução para obter a potência de processamento necessário (DOWNTON e CROOKES, 1998). Uma solução atraente é a utilização dos sistemas computacionais distribuídos aliados aos conceitos da computação paralela, originando o que passou a ser conhecido como computação paralela distribuída (BRANCO, 2004).

A crescente demanda por processamento tem motivado a evolução dos computadores desde a origem dos mesmos, viabilizando implementações de aplicações que envolvem um intenso processamento e grandes volumes de dados em máquinas que não sejam intrinsecamente paralelas, uma vez que essas últimas possuem preço elevado quando comparadas aos computadores pessoais. A utilização de computadores pessoais autonomamente pode se tornar inviável por não suprir todas as necessidades do usuário.

A computação paralela distribuída se iniciou no final dos anos 80 com a convergência entre as áreas de computação paralela e sistemas distribuídos, apresentando-se como uma solução viável que oferece recursos com baixo custo a fim de se obter melhor desempenho (JAQUIE, 1999).

Segundo Hamdi e Lee (HAMDI e LEE, 1997) os sistemas baseados em *cluster* de computadores em plataforma distribuída apresentam melhor relação

custo/desempenho para a execução de um amplo número de aplicações de alto desempenho, incluindo o processamento de imagem. Isso devido ao fato de que é possível considerar o processamento de imagem um excelente candidato para uso em paralelo, visto o amplo volume de dados (DOWNTON e CROOKES, 1998).

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 compõem a plataforma.

Existem bibliotecas especializadas para o tratamento da comunicação entre processos e a sincronização dos processos concorrentes. De acordo com Seinstra e Koelma (SEINSTRA e KOELMA, 2004), a programação paralela com base na passagem de mensagem (*message passing*) 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 *et al,* 1996) e o PVM (*Parallel Virtual Machine*) (GEIST *et al.,* 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 diversas aplicações desenvolvidas em linguagens como Fortran, C e C++ que utilizam tanto MPI e PVM. Com o surgimento de Java, inúmeras propostas foram apresentadas para a utilização dessas bibliotecas nessa linguagem as quais se pode citar o mpiJava (BAKER *et al.*, 1998) e o JPVM (*Java Parallel Virtual Machine*) (FERRARI, 1998), sendo este último o ambiente de passagem de mensagem utilizado neste trabalho.

A escolha do JPVM como ambiente para processamento paralelo dos algoritmos vem em decorrência do uso da linguagem Java que contém fatores como portabilidade (permite a independência de plataforma), simplicidade e clareza nos códigos, além da existência de *APIs* especializadas que permitem explorar mais a linguagem para efetuar processamento de imagens.

Existem inúmeros métodos de processamento de imagens. A escolha de procedimentos a serem aplicados depende do objetivo que se deseja em relação a uma determinada categoria de imagem.

Objetivos

O objetivo deste trabalho foi selecionar algumas técnicas de processamento de imagens a fim de verificar a relação custo/benefício do processamento paralelo distribuído comparado com o processamento seqüencial.

Buscou-se identificar a melhor estratégia para decomposição do problema, para proporcionar ao usuário uma execução eficiente e transparente. Foram exploradas técnicas de suavização, segmentação, a fim de se obter um melhor tempo de processamento utilizando computação paralela distribuída quando comparada às demais alternativas. Após os resultados obtidos, foi possível avaliar o desempenho da biblioteca JPVM.

Organização do Documento

Estruturalmente esta monografia é composta por cinco capítulos.

No capítulo 1 é feita uma revisão bibliográfica sobre computação paralela e sistemas distribuídos e posteriormente, são abordadas as vantagens e desvantagens da convergência dos sistemas distribuídos e da computação paralela.

No capítulo 2 é feita uma análise dos principais ambientes de passagem de mensagem, em especial a API JPVM.

No capítulo 3 é apresentada uma introdução ao processamento de imagens, além de detalhar a estratégia de desenvolvimento dos algoritmos paralelos.

No capítulo 4 são apresentados os resultados obtidos após uma série de casos de teste.

No capítulo 5 é apresentada a conclusão do trabalho.

CAPÍTULO 1. COMPUTACÃO PARALELA DISTRIBUÍDA

A busca por mais processamento tem motivado a evolução dos computadores desde o surgimento dos mesmos. Mesmo com os avanços obtidos por meio dos uniprocessadores tradicionais, tais ainda demonstram deficiência quando utilizados com aplicações que demandam grande potência computacional. Estas aplicações computacionais estão presentes desde áreas científicas até comerciais, e podem ser grandes em relação a volume de dados manipulados; e complexas referentes ao processamento.

Com o objetivo principal de oferecer suporte às aplicações que necessitam de grande poder computacional surge à computação paralela (ALMASI e GOTTLIEB, 1994; QUINN, 1994) com a capacidade de aumentar o processamento de uma única máquina (NAVAUX, 1989).

Por algum tempo foram utilizadas somente máquinas multiprocessadoras paralelas (supercomputadores paralelos), as quais oferecem alto desempenho, contudo, seu alto custo representava um obstáculo para sua disseminação, além de não oferecerem flexibilidade e escalabilidade.

No final da década de 80, surgiu uma alternativa aos supercomputadores paralelos, a chamada computação paralela sobre sistemas distribuídos.

1.1. Computação Paralela

A computação paralela é uma área da ciência da computação que tem como objetivo principal incrementar o desempenho de aplicações específicas. Na literatura existem diversas definições para computação paralelas entre elas se destaca a de Almasi e Gottlieb (ALMASI e GOTTLIEB, 1994). Na computação paralela são utilizados múltiplos elementos de processamento (EPs ou nós) capazes de se comunicar e cooperar para resolver grandes problemas de forma mais rápida. Na definição de Quinn (QUINN, 1994), a computação paralela é o processamento que enfatiza a manipulação concorrente dos dados que pertencem a um ou mais processos com o objetivo de resolver um único problema.

Para compreender a computação paralela são necessários conhecimentos básicos de concorrência, paralelismo e granulosidade (ou nível de paralelismo).

A concorrência pode ocorrer em sistemas com uma única unidade de processamento ou em sistemas multi-processados (ALMASI e GOTTLIEB, 1994) quando se tem a disputa para execução de dois ou mais processos.

Para que se ocorra o paralelismo é necessário que dois ou mais processos executem no mesmo intervalo de tempo para resolver uma determinada tarefa (ALMASI e GOTTLIEB, 1994; QUINN, 1994), como ilustrado na Figura 1.

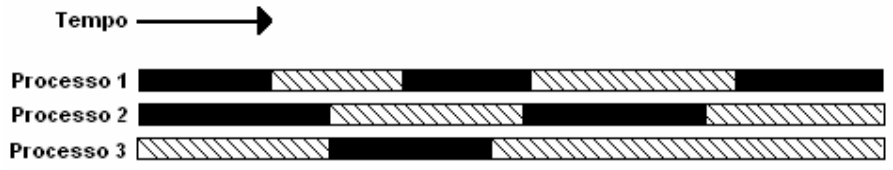

Figura 1 - Processos em um Sistema Fisicamente Paralelo (STALLINGS, 2003).

Nos sistemas uniprocessados tem-se um pseudo-paralelismo o qual se refere à rápida alternância da CPU entre os processos, cada processo executa em pequenos intervalos de tempo dando a impressão de que os processos executam simultaneamente (TANENBAUM, 2001a). Este sistema é ilustrado na Figura 2.

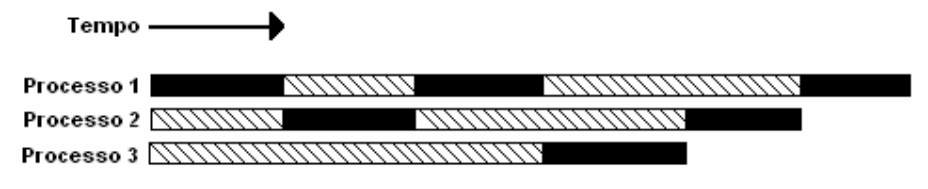

Figura 2 - Processos em um Sistema Logicamente Paralelos (pseudo-paralelismo) (STALLINGS, 2003).

O termo granularidade diz respeito à razão entre o tempo necessário ao cálculo de uma determinada operação e os custos envolvidos nas trocas de mensagem entre esta operação e as demais existentes, em linhas gerais, refere-se ao tamanho das unidades de trabalho submetidas aos processadores (CAVALHEIRO, 2004; KIRNER, 1991). Granularidade está relacionada diretamente com os níveis de paralelismo, quanto mais baixo o nível, mais fina é a granularidade do processamento (NAVAUX, 1989).

A partir desses conceitos pode-se classificar a granulação em três tipos: fina, média e grossa (ALMASI e GOTTLIEB, 1994; QUINN, 1994). Na granularidade grossa cada processo contém um grande número de instruções seqüências e complexas,

considerada paralelismo de alto nível. Na granularidade fina cada processo contém processos com um conjunto simples de poucas instruções, e na granularidade média representa um grupo intermediário.

1.1.1. Arquiteturas Paralelas

Cada arquitetura apresenta determinadas características visando melhor desempenho sob um dado enfoque. Para acompanhar o desenvolvimento das arquiteturas paralelas e agrupar os equipamentos com características comuns, foram propostas algumas taxonomias, dentre elas a de Flynn (FLYNN, 1972; FLYNN e RUDD, 1996), e a de Duncan (DUNCAN, 1990).

A taxonomia introduzida por Flynn (FLYNN, 1972) é a forma mais utilizada para classificar sistemas de processamento paralelo, mesmo sendo proposta em meados dos anos 60, é muito utilizada nos dias atuais (NAVAUX, 1989; DUNCAN, 1990; ALMASI e GOTTLIEB, 1994; QUINN, 1994; BRANCO, 2004).

Por conveniência foram adotadas duas definições: **Fluxo de Instruções**, seqüência de instruções executadas (Simples ou Múltiplos) e **Fluxo de dados**, seqüência de dados que serão chamados pelo fluxo de instruções para ser processados (Simples ou Múltiplos) (FLYNN e RUDD, 1996). Deste modo, formaram-se quatro classes de computadores, as quais são apresentadas na Figura 3: SISD (*Single Instruction Single Data*), MISD (*Multiple Instruction Single Data*), SIMD (*Single Instruction Multiple Data*) e MIMD (*Multiple Instruction Multiple Data*).

Seqüências de Instruções	Seqüências de dados	Nome	Exemplos
1	1	SISD	Máquinas clássica de Von Neumann
1	1	SIMD	Supercomputador vetorial, processador matricial
Várias	Várias	MISD	Nenhum exemplo
Várias	Várias	MIMD	Multiprocessador, multicomputador

Figura 3 - Taxonomia de Flynn (TANENBAUM, 2001b).

Os computadores que pertencem à classe SISD têm como característica possuir somente um único fluxo de instruções operando sobre um único fluxo de dado. A

maioria das máquinas convencionais utilizadas atualmente é baseada no modelo tradicional de Von Neumann (JAQUIE, 1999). Também são classificadas como SISD, mquinas com processadores superescalares que permitem a realização de um conjunto de instruções limitadas em paralelo, por possuírem somente uma única unidade de controle (QUINN, 1994).

O que caracteriza a classe SIMD é a existência de um único fluxo de instrução atuando sobre múltiplos fluxos de dados distintos. Apenas uma única unidade controle que controla as múltiplas unidades de processamento. Enquadram-se nesta classe as máquinas vetoriais e matriciais (STALLINGS, 2003).

A classe de máquinas MISD consiste em ter múltiplos fluxos de instruções ao mesmo tempo atuando sobre um único fluxo de dado. É considerado apenas um modelo teórico, não existindo nenhuma implementação na prática de computadores com este tipo de arquitetura (STALLINGS, 2003).

A classe MIMD se refere ao modelo de execução paralela onde se tem processamento simultâneo de múltiplos fluxos de instruções diferentes sobre múltiplos fluxos de dados diferentes.

A taxonomia de Flynn, como apresentado, é muito difundida e em uma primeira instância conveniente para classificação das arquiteturas, entretanto se limita às quatro categorias citadas, as quais não são suficientes para acomodar de forma apropriada as variedades de arquitetura existente e que virão a existir.

Duncan (DUNCAN, 1990) propôs uma classificação mais refinada e abrangente, a fim de permitir a acomodação mais adequada de novas arquiteturas. A taxonomia de Duncan consiste em dividir as arquiteturas em síncronas e assíncronas (DUNCAN, 1990).

Nas máquinas com arquiteturas síncronas as operações concorrentes baseiam-se em sinais de único relógio global e unidade de controle centralizada. Máquinas com processadores vetoriais, matriciais e de arranjos sistólicos se enquadram nesta categoria (DUNCAN, 1990).

Nas máquinas com arquiteturas assíncronas, os processadores podem operar de maneira autônoma, já que o controle não é centralizado pelo *hardware*. Esta categoria corresponde à classe MIMD da taxonomia de Flynn. Fazem parte desta categoria máquinas convencionais (MIMD com memória compartilhada ou distribuída) e as máquinas não convencionais (máquinas híbridas, fluxo de dados, redução, e de frente de onda) (DUNCAN, 1990).

A taxonomia apresentada por Tanenbaum (TANENBAUM, 2001b) subdivide a categoria MIMD em multiprocessadores (máquinas com memória compartilhada) e multicomputadores (máquinas com memória distribuída) (TANENBAUM, 1999; STALLINGS, 2003). Para essa subdivisão foi adotado um critério de organização física da memória central e o tipo de acesso que cada processador tem a essa memória.

Um conceito importante é diferenciar memória compartilhada de memória distribuída. A memória compartilhada (*shared memory*) é caracterizada por possuir múltiplos processadores operando de forma independente, compartilhando um único espaço de endereçamento virtual que será usado de forma implícita para comunicação entre todos os processadores que se comunicam por meio de *load* e *store* (carrega e armazena) nos endereços de memória (CULLER, GUPTA e SINGH, 1999; STALLINGS, 2003; TANENBAUM e VAN STEEN, 2002).

A memória distribuída ou privada (*distributed memory*) é caracterizada por possuir múltiplos processadores, na qual cada um possui e usa sua própria memória privada apresentando um espaço de endereçamento distinto para cada processador. A comunicação entre os processadores ocorre através de troca/passagem de mensagens feita com as operações *send* e *receive* (envia e recebe) por meio de uma rede de comunicação (CULLER, GUPTA e SINGH, 1999; STALLINGS, 2003; TANENBAUM e VAN STEEN, 2002).

Os multiprocessadores são as arquiteturas que utilizam memória compartilhada, e sua estrutura é semelhante à colocação de múltiplos processadores em uma máquina de von Neumann tradicional. Os múltiplos processadores são conectados à memória por meio de uma rede de interconexão como apresentado na Figura 4.

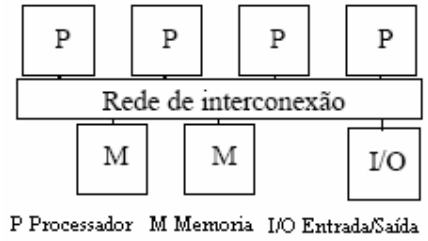

Figura 4 - Arquitetura multiprocessador (TANENBAUM, 2001a)

Existem três tipos de multiprocessadores e estes se diferem na maneira como a memória compartilhada é implementada. São conhecidos como UMA (*Uniform*

Memory Access), COMA (*Cache-Only Memory Architecture*) e NUMA (*NonUniform Memory Access*) (TANENBAUM, 2001b).

A razão da existência dessas categorias, conforme Tanenbaum (TANENBAUM, 2001b), consiste no fato dos sistemas multiprocessadores de grande porte possuírem diversos módulos de memória diferentes.

UMA – São máquinas operando com um bloco de memória centralizada onde todos os processadores enxergam os módulos de memória como tendo o mesmo tempo de acesso. Essas arquiteturas também são chamadas de SMP (*Symmetric MultiProcessor*).

NUMA – O acesso aos módulos próximos ao processador é muito mais rápido do que o acesso aos módulos mais distantes. Isto ocorre dado o fato que a memória é distribuída nos processadores e implementada com vários módulos. Cada processador está associado a um módulo, contudo o espaço de endereçamento é único podendo assim cada processador acessar toda a memória disponível, mas não necessariamente a mesma memória física.

Os multicomputadores caracterizam-se por utilizar memória distribuída onde cada processador tem acesso somente a sua própria memória, fazendo com que não existam variáveis globais, o que obriga, conseqüentemente, os processadores a se comunicarem por meio de trocas de mensagens pela rede de interconexão como apresentado na Figura 5.

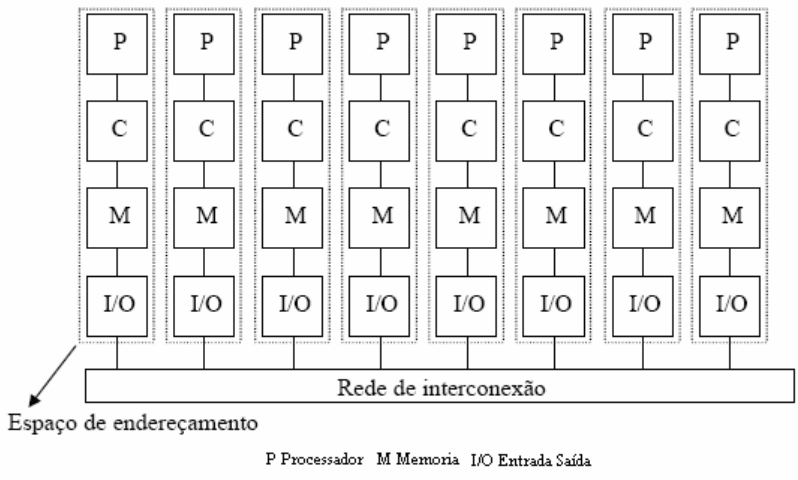

Figura 5 - Arquitetura Multicomputador (TANENBAUM, 2001a).

Multicomputadores podem ser divididos em máquinas simples conectadas em redes, COWs (Cluster of Workstations) e MPPs (Massively Parallel Processor) (TANENBAUM, 2001b).

Um COW é composto de algumas centenas de PCs ou estações de trabalho interligadas por redes de comunicações tradicionais. Portanto, tais máquinas paralelas são compostas por um conjunto de máquinas autônomas fazendo o uso de uma biblioteca de passagem de mensagem para se comunicarem.

Os MPPs são supercomputadores imensos, compostos por um grande número de processadores comerciais como o IBM RS/6000, a família Dec Alpha ou a linha Sun UltraSPARC. Umas das principais características dos MPPs é uso de redes de interconexão proprietárias de alto desempenho, baixa latência e banda passante alta, projetadas para a troca de mensagens (TANENBAUM, 2001b; STALLINGS, 2003).

Como pode ser observada na Figura 6, as definições citadas permitem que várias topologias de máquinas paralelas e de redes de computadores sejam enquadradas como arquitetura MIMD.

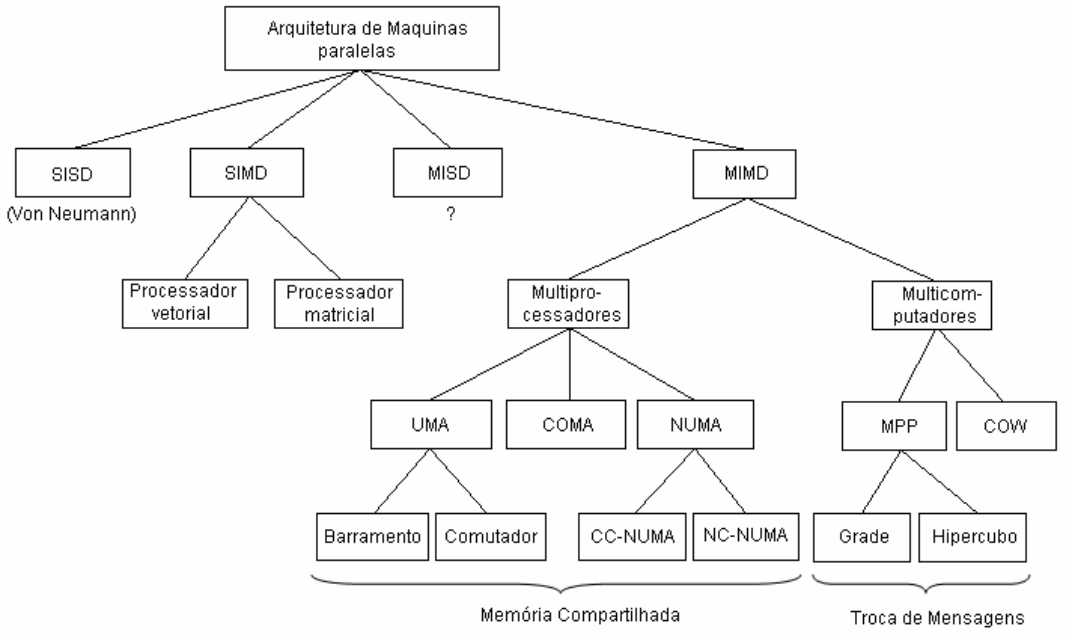

Figura 6 - Taxonomia para máquinas de arquitetura paralela (TANENBAUM, 2001b)

1.2. Sistemas Distribuídos

É inegável o crescimento e a importância dos sistemas computacionais distribuídos tanto no meio acadêmico como no comercial. Isto foi possível devido ao fato de se obter computadores mais rápidos e baratos, e ao surgimento das redes de computadores de alta-velocidade a partir da década 80.

Muitas definições de sistemas distribuídos são encontradas na literatura. De acordo com Tanenbaum (TANENBAUM, 1995), sistemas distribuídos são como uma coleção de computadores independentes que aparentam ser um único computador aos seus usuários. Uma definição proposta por Coulouris *et. al* (COULOURIS *et al.,* 2007) define os sistemas distribuídos como um sistema em que componentes de *hardware* e *software* localizados em computadores em rede se comunicam e coordenam suas ações por meio de passagem de mensagens.

Essas definições implicam em conceitos relacionados com o *hardware* formado por máquinas autônomas e *software* que fornecem a abstração de uma única máquina.

Os sistemas distribuídos apresentam inúmeras vantagens sobre os sistemas centralizados, o que permitiu que se tornasse ainda mais difundido. Dentre estas vantagens é possível citar: econômica (oferece uma melhor relação preço/desempenho do que os *mainframes*), distribuição inerente, confiabilidade e crescimento incremental (o poder computacional pode ser aumentado com a inclusão de novos equipamentos) (TANENBAUM, 1999).

Entretanto, os sistemas distribuídos apresentam desvantagens sendo estas: complexidade de *software,* falta de segurança (dados são compartilhados) e saturação da rede de comunicação (TANENBAUM, 1999):

Segundo alguns autores (TANENBAUM, 1995; COULOURIS *et al*., 2007) os sistemas distribuídos devem apresentar as seguintes características:

- transparência: o objetivo final é passar a impressão de um sistema único aos usuários, ou seja, o uso de múltiplos processadores deve ser invisível ao usuário, conceito chave dos sistemas distribuídos;
- flexibilidade: representa a facilidade de se fazer reconfiguração (inter-operabilidade);

- confiabilidade: que caracteriza a disponibilidade, tolerância à falhas, e a segurança, a idéia é que se uma máquina falha, outra máquina do sistema pode assumir suas funções;
- escalabilidade: refere-se à capacidade do sistema expandir-se com o mínimo de degradação de desempenho.

Outro fator importante é a comunicação em sistemas distribuídos. Devido à ausência de memória compartilhada, a comunicação entre processos IPC (*InterProcess Communication)* é realizada por meio de troca de mensagens (primitivas *send/receive* bloqueantes ou não bloqueantes), RPC (*Remote Procedure Call*), RMI (*Remote Method Invocation*) e CORBA (*Common Objetc Request Broker Architecture*) (TANENBAUM e VAN STEEN, 2002).

1.2.1. Modelos de Arquitetura

Em um Sistema Distribuído, os processos possuem responsabilidades bem definidas e interagem para realizar uma atividade útil, sendo que um aspecto evidente de um projeto de sistema distribuídos é a divisão destas responsabilidades entre os componentes de um sistema. Os principais modelos de arquitetura para realizar a distribuição de responsabilidade são basicamente divididos em Modelo Cliente/Servidor, *Peer-to-Peer* (COULOURIS *et al.,* 2007).

O modelo Cliente/Servidor, historicamente mais importante, consiste em estruturar o sistema operacional como um grupo de processos cooperantes (servidores) que oferecem serviços aos usuários (clientes). Os servidores podem, por sua vez, serem clientes de outros servidores (COULOURIS *et al.,* 2007 ; TANENBAUM e VAN STEEN, 2002).

Este modelo utiliza o protocolo requisição/resposta que é projetado para associar repostas as suas correspondentes requisições. Possui como vantagem a simplicidade e a eficiência, e como desvantagem a falha do servidor, a qual pode deixar o sistema inoperante, porém é possível evitar quando utilizadas técnicas de replicação de servidores (TANENBAUM e VAN STEEN, 2002).

O modelo *Peer-to-Peer* fornece uma alternativa à arquitetura cliente/servidor tradicional, a qual implica na comunicação direta entre *hosts*, deste modo elimina

qualquer exigência de servidores. Neste contexto o par (*peer*) age tanto como um cliente quanto como um servidor (COULOURIS *et al.,* 2007 ; TANENBAUM e VAN STEEN, 2002).

Os aplicativos *Peer-to-Peer* fornecem compartilhamento de arquivos, uso de *cache web*, distribuição de informação entre outros, apresentado maior eficácia quando usados para armazenar conjuntos muito grandes de dados imutáveis (COULOURIS *et al.,* 2007).

Um problema que se destaca é referente às sobrecargas indevidas que determinados *hosts* possam vir a sofrer, visto que além do gerenciamento local, cada *host* deve também tratar os acessos remotos. A vantagem é a capacidade de explorar recursos ociosos, a estabilidade para suportar grandes números de clientes e *host*, entre outros.

1.3. Computação Paralela Distribuída

O custo elevado e pouca flexibilidade das máquinas SMP e MPP tornavam a computação paralela menos acessível, entretanto com o surgimento e principalmente seu aperfeiçoamento (meios de comunicação, computadores mais potentes computacionalmente), os sistemas distribuídos apresentaram-se como um alternativa economicamente viável para a aplicação da computação paralela.

Mesmo surgindo por razões diferentes, observou-se uma rápida convergência ao longo dos últimos anos dos sistemas distribuídos com a computação paralela, oferecendo a possibilidade de compartilhar recursos, suportar um conjunto de serviços de forma transparente à sua localização e um alto desempenho.

Segundo Cáreces, Mongelli e Song (CÁCERES et al., 2001) a computação paralela distribuída consiste em uma coleção de elementos de computação, interconectados de acordo com uma determinada topologia para permitir a coordenação de suas atividades e a troca de mensagens.

A computação paralela e distribuída explora a possibilidade da utilização de *clusters*, motivado pela sua semelhança com as máquinas MPPs.

1.4. Considerações Finais

Neste capítulo foi feita uma revisão bibliográfica de computação paralela, sistemas distribuídos e uma síntese da convergência de ambos, a chamada de computação paralela distribuída.

A computação paralela surgiu devido à necessidade de obter um desempenho computacional maior do que o que se conseguia com arquiteturas seqüenciais. As arquiteturas foram aprimoradas e desenvolvidas a fim de se ter um aumento computacional. Inúmeras classificações foram propostas, mas as mais difundidas foram as de Flynn e de Duncan.

Com o grande avanço dos computadores e das redes de comunicação foram criados os sistemas distribuídos que tinham por objetivo realizar o compartilhamento de recursos. Mesmo tendo sido originadas por razões distintas, computação paralela e sistemas distribuídos se convergiram. Várias soluções, vantagens e melhorias foram criadas após a união dessas duas áreas, sendo o aumento do poder computacional, a economia quando comparadas ás MPPs, as melhorias no desempenho, entre outras.

A utilização da computação paralela sobre sistemas distribuídos é feita através de ambientes de troca de mensagens. A troca de mensagens pode ser feita de forma manual com a utilização de *sockets* ou por meio de uma biblioteca (APIs - *Application Program Interface*) que provê rotinas para passagem de mensagens. Os ambientes de troca de mensagem mais populares são PVM e MPI.

Estes ambientes de passagem de mensagens foram desenvolvidos inicialmente para serem usados em máquinas MPPs onde, devido à ausência de um padrão, cada fabricante desenvolveu seu próprio ambiente sem se preocupar com portabilidade. Com o passar dos anos essas bibliotecas passaram a ser utilizadas com o propósito de utilizar computadores pessoais para a composição da máquina paralela virtual (MCBRYAN, 1994). Desta maneira, maiores detalhes sobre esses ambientes de passagem de mensagem serão abordados no próximo capítulo.

CAPÍTULO 2. AMBIENTES DE PASSAGEM DE MENSAGENS

A computação paralela sobre sistemas distribuídos tornou-se possível devido à utilização de ambientes de passagem de mensagem, que são, na maioria das vezes, bibliotecas que estendem as linguagens de programação seqüenciais, adicionando mecanismos para a criação, a comunicação e a sincronização de tarefas, permitindo o desenvolvimento de aplicações paralelas.

No modelo de passagem de mensagens os processadores se comunicam por meio do envio de mensagem de uns para os outros. Este modelo apresenta-se como uma alternativa viável devido a inúmeras vantagens como (GROPP *et al.,* 1994):

- baixo custo: os ambientes mais comuns como PVM (*Parallel Virtual Machine*) ou MPI (*Message Passing Interface*) são programas de domínio publico, além de não exigirem o aprendizado de uma nova linguagem de programação;
- generalidade: pode-se construir um ambiente de troca de mensagens para qualquer linguagem, sendo executado em ambiente com máquinas tanto homogêneas quanto heterogêneas, interconectadas por qualquer tipo de rede;
- desempenho: é possível obter ganhos consideráveis de desempenho.

Entretanto o modelo de passagem de mensagem apresenta limitações como necessidade de programação explícita, sendo o programador responsável pela paralelização, e o custo de comunicação em certos ambientes, o que pode inviabilizar a troca de mensagens em alguns casos (GROPP *et al.,* 1994).

Uma troca de mensagem envolve no mínimo dois processos. Para comunicação entre os processos são utilizadas a rotina *send* (envio) e a rotina *receive* (recebimento). Essas rotinas podem ser bloqueantes, o que garante que tanto o processo emissor como receptor fiquem bloqueados até que o envio ou recebimento da mensagem seja completado, ou não bloqueantes, quando os processos continuam suas execuções mesmo que a mensagem não tenha sido recebida.

Os ambientes de passagem de mensagem como PVM e o MPI têm tido grande destaque na literatura, não só pela flexibilidade, mas também pelo fato de constituírem

um tipo de solução para o problema da portabilidade de programas paralelos entre sistemas diferentes (JAQUIE, 1999).

O ambiente de passagem de mensagem alvo desse trabalho é o JPVM, o qual possui como base o PVM (*Parallel Virtual Machine*). Ambos serão detalhados nas seções que seguem.

2.1. *Parallel Virtual Machine* (PVM)

O PVM é um conjunto integrado de bibliotecas e de ferramentas de *software* cuja finalidade é emular um sistema computacional concorrente heterogêneo, permitindo que um conjunto de máquinas heterogêneas seja enxergado como uma única máquina paralela virtual. Ainda, o PVM permite que a máquina paralela virtual possa ser usada de maneira cooperativa para computação paralela por meio de troca de mensagem (BEGUELIN, 1994).

O projeto PVM iniciou em 1989 no ORNL (*Oak Ridge National Laboratory*), com a versão PVM 1.0. Em 1991 foi desenvolvida a versão PVM 2.0 que contou com a participação de instituições como a *University of Tennissee, Carnegie Mellon University* entre outras.

A partir desta versão deu-se início à distribuição gratuita do PVM, o que permitiu que novas versões surgissem como resultado das críticas e sugestões de equipes de várias outras instituições. Passou a ser um ambiente paralelo virtual disponibilizado em domínio público e alcançou grande aceitação em diversos setores, sendo utilizado nos mais variados tipos de projetos ao redor do mundo. Por esses motivos o PVM passou a ser considerado um padrão de direito.

O PVM é formado por dois componentes básicos (GEIST *et al.,* 1994): PVM *daemon* (*pvmd*) e a biblioteca libpvm.

O *pvmd* é o processo que deve estar em execução em cada *host* que fará parte da máquina virtual. O *pvmd* não faz processamento, este é responsável pelo roteamento das mensagens e trabalha como um gerenciador de processos e da máquina virtual.

A biblioteca libpvm é escrita em linguagem C e foi desenvolvida com o intuito de torná-la tão pequena quanto possível, a fim de aumentar sua eficiência. A biblioteca

libpvm contém um conjunto de rotinas do PVM para efetuar passagem de mensagens, solicitar geração de processos, coordenar tarefas e modificar a máquina virtual.

Foi projetado para, sempre que possível, não impor nenhum tipo de limitação de acesso aos seus recursos. Normalmente os limites são impostos pelo *hardware* ou pelo sistema operacional utilizado.

O PVM apresenta como aspectos básicos a comunicação, tolerância à falhas e tratamento de mensagem.

A comunicação pode ser realizada com base: no protocolo TCP *(Transmission Control Protocol),* protocolo orientado a conexões confiáveis e com entrega de um fluxo de *bytes* sem erros ao destino; no protocolo UDP *(User Datagrama Protocol)*, protocolo não orientado a conexão e não confiável dirigido a aplicações que não solicitam controle de fluxo e nem manutenção da seqüência das mensagens enviadas; em *sockets*, que conceitualmente pode ser definido como uma porta sobre a qual um processo pode enviar ou receber mensagem através da rede utilizando como protocolo tanto UDP quanto TCP.

Há esquemas básicos de notificação de falha como re-configuração automática da máquina virtual em caso de falha ou perda, acessibilidade em qualquer *host* escravo, e caso ocorra perda ou falha no *host* mestre a máquina virtual é finalizada. Em relação à aplicação não existe nenhuma forma de recuperação automática após um erro, porém, são disponibilizados ao programador recursos necessários para que se construa aplicações tolerantes a falhas (JAQUIE, 1999).

As mensagens podem assumir tamanhos arbitrários contendo diferentes tipos de dados, sendo possível identificada por um número inteiro escolhido pelo usuário. Em caso de envio para *host* com formato de dados incompatíveis é feita uma conversão dos dados pelo formato *XDR (eXternal Data Representation)* automaticamente .

Para usuário PVM a comunicação empregada utiliza emissão bloqueante assíncrona, a recepção bloqueante assíncrona e a recepção não bloqueante. A transferência de dados entre os *hosts* pode ser ponto-a-ponto (entre dois *host*), *broadcasting* (para um grupo de *host*) e *multicasting* (para um conjunto de *host*) (JAQUIE, 1999).

2.2. *Message Passing Interface* (MPI)

Segundo Gropp *et al.* (GROPP *et al.*, 1994) e Foster (FOSTER, 1995), o MPI é um padrão de interface para troca de mensagens em máquinas paralelas com memória distribuída. No padrão MPI uma aplicação é constituída por um ou mais processos que se comunicam pode meio de funções para o envio e recebimento de mensagens entre os processos. Dessa forma, a comunicação e a sincronização dos processos são garantidas.

Por não prover gerenciamento dinâmico de processos, ao contrário do PVM, o MPI pode deixar de ser atraente em aplicações onde cada processador é responsável por uma tarefa específica. Apesar disso, programas escritos em MPI tendem a ser mais eficientes pelo fato de não haver sobrecarga de processos em tempo de execução. Há a necessidade de se explicitar as funções de criação das tarefas, suas comunicação e destruição, por este motivo a programação com o MPI é mais simples e mais legível que a do PVM.

Existem muitas implementações de MPI para aplicações desenvolvidas em linguagens como Fortran, C e C++. Com o surgimento de Java, inúmeras propostas foram apresentadas para a utilização de MPI nessa linguagem.

2.3. Java e Ferramentas para Troca de Mensagem

Java é uma linguagem de programação orientada a objetos que possibilita a fácil abstração de um problema, tendo sido adotada rapidamente como uma das linguagens preferidas para aplicações distribuídas. Em particular, Java provê coleta de lixo automática, o que alivia os programadores da administração de memória, provê um excelente apoio para programação concorrente (*threads)*, permite salvar o estado de um objeto e recriar este objeto em outras máquinas, além de apoiar objetos persistentes e migração de objetos.

Entretanto, a linguagem não oferece nenhum recurso de alto nível para a comunicação via troca de mensagens voltada para a computação paralela distribuída. Existem diversas propostas como adoção de recursos disponibilizados por bibliotecas como PVM e MPI. Neste contexto inúmeras propostas foram apresentadas para a

utilização dessas bibliotecas nessa linguagem, entre as quais é possível citar o mpiJava (BAKER *et al.*, 1998) e o JPVM (FERRARI, 1998).

2.3.1. *Message Passing Interface Java* (mpiJava)

O 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 *et al.,* 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 *et al.,* 1999). O uso de mpiJava já foi validado em diversas implementações e avaliações de desempenho (TABOADA *et al.,* 2003).

2.3.2. *Java Parallel Virtual Machine* (JPVM)

O JPVM foi desenvolvido em 1997 pelo Prof. Dr. Adam Ferrari do departamento de ciência da computação da Universidade da Virgínia, em 1999. Foi disponibilizada uma nova versão com auxílio do Prof. Thomas R. James do departamento de ciências matemáticas da faculdade de Otterbein. Esta nova versão denominada JPVM v.0.2.1, a qual reparou erros encontrados na versão anterior.

Trata-se de uma pequena API implementada totalmente em Java que permite a troca de mensagens explícitas baseadas em memória distribuída MIMD. Essa API combina as vantagens da linguagem Java como portabilidade e interoperabilidade, com as técnicas de troca de mensagem entre processos paralelos em ambientes distribuídos (FERRARI, 1998).

O pacote JPVM roda em praticamente todas as plataformas que tem suporte a máquina virtual Java com *Unix* e seus derivados, *Windows* e *Macintosh*. E desde que a rede de comunicação esteja devidamente instalada, não apresenta dificuldade em integrar as máquinas de plataformas diferentes na máquina virtual.

O JPVM apresenta uma interface de programação, intencionalmente, muito próxima ao sistema PVM, com a finalidade de que programadores acostumados com o PVM migrem para o JPVM com maior facilidade.

Contudo novas características foram adicionadas ao JPVM (FERRARI, 1998):

- segurança de *threads*;
- múltiplos pontos de comunicação por tarefa;
- roteamento padrão para mensagens diretas.

É possível considerar o JPVM como uma ferramenta de fácil disponibilidade para o ensino da computação paralela distribuída, tendo em vista sua simplicidade em instalação e utilização. É um *software* de domínio público e não é necessária a instalação do PVM.

O JPVM apresenta um fraco desempenho comparado ao PVM e MPI especialmente em termos de latência de comunicação. Segundo Lee, (1999) a responsável por não permitir um desempenho superior do JPVM é a linguagem Java, porém, melhorias significantes no desempenho foram alcançadas com o uso Java JIT *Compiler*.

Segundo Ferrari (FERRARI, 1998), o JPVM é composto por duas partes:

- J*pvmdaemon*: um processo que é executado em todos os nós, formando uma máquina virtual paralela, é responsável por realizar a comunicação entre os processos criados e coordenar as tarefas em execução;
- *JpvmEnvironment*: biblioteca que trata funções básicas, como troca de mensagens, criação e eliminação de processos, sincronização de tarefas e modificação da máquina virtual, envio e recebimento de mensagens.

2.3.2.1. Modelo computacional

O modelo de computação do JPVM é baseado na idéia de que várias tarefas compõem uma aplicação. O usuário escreve sua aplicação como uma coleção de tarefas

cooperativas a fim de resolver um problema específico. Basicamente, uma aplicação pode ser paralelizável pelas seguintes decomposições (CÁCERES *et al.,* 2001):

- decomposição de domínio ou paralelismo de dados: as tarefas são iguais. Eeste paradigma implica que o código fonte a ser executado é o mesmo em cada processador. Os dados referentes à aplicação são divididos em tamanhos aproximadamente iguais e então distribuídos aos processadores, seguindo o modelo denominado como SPMD;
- decomposição funcional ou paralelismo funcional: as tarefas têm funcionalidades diferentes. São distribuídos códigos fontes distintos para cada processador para execução simultânea;
- híbrida: o JPVM permite também um modelo híbrido (uma mistura dos dois modelos anteriormente citados).

2.3.2.2. Comunicação no JPVM

A comunicação é feita de forma direta, tarefa-para-tarefa, implementada sobre TCP *sockets* utilizando serialização de objetos como JAVA *objeto serialization interface*. A cada instância do *jpvmEnvironment* cria-se um *server socket* que atribui o nome do *host* e um número de porta de conexão internamente para realizar troca de mensagens e para gerenciar as as *threads*. Quando uma tarefa X deseja se comunicar com uma tarefa Y, ela simplesmente se conecta com Y usando o nome do *host* e a porta que estão contidos no identificador da tarefa Y. Caso a conexão seja aceita cria-se uma *thread* dedicada à gerência da mesma (FERRARI, 1998).

Para identificação é usado número inteiro e único como identificador das tarefas executadas nas trocas de mensagens denominado *jpvmTaskId*.

Como na biblioteca PVM, a comunicação utiliza emissão bloqueante assíncrona, a recepção bloqueante assíncrona e recepção não bloqueante, a transferência dos dados como apresentado anteriormente pode ser tarefa-para-tarefa ou *multicasting* (para um conjunto de tarefas).

2.3.2.3. Interface de Programação

O JPVM apresenta métodos, quase na sua totalidade, idênticos aos do PVM, entretanto em um número reduzido, tendo somente a implementação das funções básicas. Para aplicação ter acesso aos métodos é necessário importar tais recursos da biblioteca JPVM por meio da linha de código.

```
import jpvm.*;
```

jpvmEnvironment – principal classe da biblioteca JVPM, é por meio dela que uma aplicação tem acesso a recursos como controle e criação de processo, métodos que retornam a configuração atual da máquina virtual, e os métodos para troca de mensagens, ao ser instanciada é criado um processo que é, por sua vez, registrado junto à máquina virtual local.

São apresentadas, na Tabela 1, as funções básicas para controle e gerencialmente de processos no JPVM.

Tabela 1. Funções de controle do PVM

Métodos de Controle	Descrição
pvm_mytid	Essa função retorna a identificação do processo na máquina virtual além de registrar o processo na máquina virtual
pvm_parent	Retorna a identificação da tarefa pai ou o valor *PvmNoParent* caso a tarefa não tenha sido criada pela função *pvm_spawn*.
pvm_spawn	Criação de novos processos.
pvm_exit	Processo informa ao *daemon* local que está se desassociando da máquina virtual

Métodos de Troca de Mensagens:

- O envio de mensagens consiste de três passos básicos:

- criação de um *buffer* temporário para guardar o conteúdo da mensagem;
- empacotamento da mensagem dentro do *buffer*;
- envio do *buffer* para outra tarefa ou para um grupo de tarefas.

- Para o recebimento de mensagem:
  - recebimento da mensagem por uma função bloqueante ou não bloqueante;
  - desempacotamento da mensagem retirando do buffer os dados.

Na Tabela 2, são apresentadas funções básicas para troca de mensagem no JPVM.

Tabela 2. Funções de passagem de mensagem do PVM

Métodos de passagem de mensagem	Descrição
pvm_send	Envia de mensagem a um outro processo
pvm_mcast	Envia de mensagem a um grupo de processos
pvm_recv	Recebe as mensagens enviadas pelas funções *send* ou mcast, sendo uma função bloqueante.

jpvmBuffer – a classe *jpvmBuffer*() fornece métodos que são utilizados para armazenamento do conteúdo da mensagem.

public *jpvmBuffer*()

Existem dois tipos de métodos:

- *pack* (empacotamento) utilizado na fase de envio da mensagem;
- *unpack* (desempacotamento) utilizado no recebimento da mensagem.

Em ambos os métodos os principais tipos de dados aceitos são: *Byte, Char Short, Integer, Long, Float, Double, String.*

2.4. Considerações Finais

A computação paralela distribuída apresentou-se nos últimos anos como um modelo integrador de componentes de baixo custo e de elevada capacidade processamento.

A formação de sua estrutura se tornou possível por meio da utilização de ambientes de troca de mensagens. Estes ambientes, em sua maioria, são bibliotecas que estendem as linguagens de programação tradicionais, fornecendo recurso para criação, envio, recebimento e sincronização de tarefas.

O ambiente de passagem de mensagem PVM tornou-se uma biblioteca popular, e sua utilização é significativa nas arquiteturas existentes. Como uma alternativa ao PVM surgiu o MPI que se tornou um padrão em comunicação por troca de mensagens.

Para prover maior portabilidade e interoperabilidade foi criado o JPVM. O JPVM forma uma “máquina virtual" composta por vários elementos de processamento (máquinas independentes) e envia tarefas para que estes elementos para que executem. O resultado, na maioria dos casos, é um desempenho superior a uma aplicação executada em seqüencial, isto é, sem a colaboração de vários elementos de processamento.

É possível utilizar o ambiente paralelo distribuído para executar diversos tipos de aplicações paralelas de modo a aumentar seu desempenho referente ao tempo de execução. Um exemplo de área que é possível construir aplicações que fazem uso deste ambiente é o processamento de imagem. Desta maneira, o próximo capítulo apresenta conceitos inerentes de processamento de imagens em ambientes paralelos distribuídos.

CAPÍTULO 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, 2006).

Imagens monocromáticas podem ser representadas matematicamente por uma função de valores discretos de intensidade da luz f(x,y), onde x e y denotam as coordenadas espaciais e o valor f em qualquer ponto (x, y) fornece o brilho (ou níveis de cinza) da imagem naquele ponto (GONZALES e WOODS, 2002).

A imagem digital é geralmente representada no computador como uma matriz bidimensional cujos índices de linhas e de colunas identificam um ponto na imagem, e o correspondente valor do elemento da matriz identifica o nível de cinza naquele ponto. Cada ponto ou elemento que constitui essa matriz digital é chamado *pixel*, sendo este a menor unidade sobre a qual é possível realizar operações (GONZALES e WOODS, 2002). Quanto mais *pixels* uma imagem tiver melhor é a sua resolução espacial.

A seguir são apresentadas algumas relações fundamentais entre *pixels* (GONZALES e WOODS, 2002).

Vizinhança

Sendo um *pixel p* nas coordenadas (*x,y*), a vizinhança é o conjunto de *pixels* que está a uma unidade de distância do *pixel p* (GONZALES e WOODS, 2002).

A Vizinhança-de-4 de *p* é representada por $N_4(p)$, ilustrada na Figura 7, é composta por vizinhos horizontais e verticais, cujas coordenadas são dadas por (x + 1, y), (x - 1, y), (x, y + 1), (x, y -1).

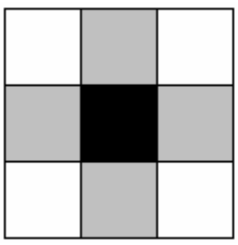

Figura 7 - Exemplo de Vizinhança N4(P)

A Vizinhança diagonal do *pixel p* é representada por $N_D(P)$, cujas coordenadas são dadas por (x + 1,y + 1), (x + 1,y - 1), (x - 1,y + 1), (x - 1,y - 1) é representada na Figura 8.

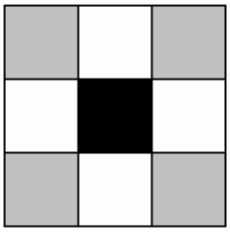

Figura 8 - Exemplo de Vizinhança ND(P)

A união destes dois conjuntos de elementos forma a "vizinhança-de-8", ou $N_8(P)$, representada na Figura 9.

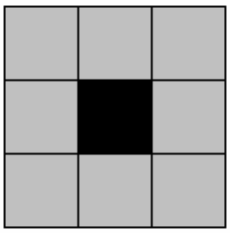

Figura 9 - Exemplo de Vizinhança N8(P)

Adjacência

Gonzalez e Woods (2002) afirmam que adjacência é a característica de um par de *pixels* vizinhos que compartilham uma borda ou um vértice, sendo que:

a) "adjacente por borda" ou "4-adjacente" é um par de *pixels* de uma imagem que compartilham uma borda;

b) "adjacente por vértice" ou "8-adjacente" é um par de *pixels* de uma imagem que compartilham um vértice.

De forma geral, as operações com imagens podem ser classificadas em baixo nível (pré-processamento), nível médio (segmentação) e nível alto (reconhecimento de padrões) GONZALES e WOODS, 2002:

- processamento de baixo nível: caracterizado por efetuar operações na imagem completa, fornece funções que variam desde a aquisição da imagem, como a redução de ruídos e o melhoramento de contraste entre outros.
- processamento de nível médio: utiliza como entrada os resultados do pré-processamento, trata da extração e caracterização de partes de uma imagem, como por exemplo, regiões e utiliza os processos de segmentação, representação e descrição.
- processamento de nível alto: trabalha com a interação da imagem com um banco de conhecimento, executando tarefas como o reconhecimento e a interpretação das informações extraídas da imagem.

As operações nesses níveis podem ser aplicadas tanto no domínio da freqüência quanto no domínio espacial (GONZALES e WOODS, 2002; NUNES, 2001).

O domínio da freqüência tem como princípio básico o teorema da convolução, efetuando-se alterações na transformada de Fourier da imagem.

O domínio espacial refere-se ao conjunto de *pixels* que compõem a imagem. As técnicas de processamento de imagem que trabalham neste domínio são aplicadas diretamente nos *pixels* da imagem.

Segundo Nunes (2006) as técnicas que trabalham com o domínio espacial não exigem grande complexidade matemática comparada às técnicas relaciona ao domínio da freqüência, entretanto seu processamento é lento uma vez que são necessárias várias operações a fim de obter um novo valor a determinado *pixel*.

Existem inúmeros métodos de processamento de imagens. A escolha de procedimentos a serem aplicados depende do objetivo que se deseja em relação a uma determinada categoria de imagem, definida pela modalidade, por meio da qual, está sendo obtida a imagem e o objeto nela representado.

3.1. Filtragem Espacial

Segundo Gonzales e Woods (2002) em processamento de imagens o uso de máscaras espacial é usualmente chamado filtragem espacial, e as máscaras são chamadas filtros espaciais. As técnicas baseadas na vizinhança usam uma máscara com objetivo de substituir o valor do nível de cinza de um *pixel* com o valor obtido em função de si próprio e de seus vizinhos.

Na prática, é uma matriz usualmente de dimensão pequena cujo elemento central é posicionado no *pixel* de interesse. É possível atribuir "pesos" aos vizinhos e ao *pixels* de interesse. Os elementos da vizinhança, incluindo o *pixel* em questão, são multiplicados pelos valores indicados nas posições correspondentes da matriz (NUNES, 2001).

É ilustrado na Figura 10, a título de exemplificação, uma máscara 3x3 e os valores dos pesos são representados por w1,w2,...,w9.

w1	w2	w3
w4	w5	w6
w7	w8	w9

Figura 10 - Exemplo de uma máscara 3x3 genérica

3.1.1. Suavização

Filtros de suavização são utilizados em uma etapa de pré-processamento para a redução de ruídos e uniformização dos níveis de cinza dos *pixels*. Entre as técnicas de suavização conhecidas existem as de suavização conservativa e técnicas de redução de ruídos (NUNES, 2001; GONZALES e WOODS, 2002).

Em decorrência do processamento, efeitos colaterais podem ser observados nas imagens processadas. As técnicas de suavização não conservativas, muitas vezes, geram borramento podendo eliminar detalhes como linhas finas e curvas agudas. Entre as técnicas mais comuns de suavização estão os filtros de média e mediana.

A filtragem mediana 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. Considerando a máscara 3x3 da Figura 11, o *pixel* central, de interesse, tem seu valor de cinza em 50 e será substituído pelo valor de cinza mediano 48, o quinto elemento na seqüência, em ordem crescente.

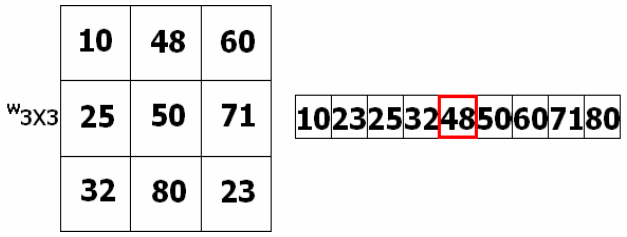

Figura 11 - Exemplo filtro mediana máscara 3x3

Com essa técnica obtém-se a redução do ruído e diminui-se o borramento quando comparado aos obtidos por outros métodos de suavização (NUNES, 2001). O problema desse filtro é o seu custo relativamente complexo e caro. Como citado, para encontrar a mediana é necessária efetuar a ordenação de todos os valores da vizinhança, o que é relativamente lento e dependente principalmente do método de ordenação utilizado.

É apresentado na Figura 12 o algoritmo para implementação do filtro da mediana.

```
defina tamanho_template
para lin = 1 ate quant_linhas
para col = 1 ate quant_colunas
defina vet_elem_med com tam_template² elem
 para   ind_lin   =   lin   -tam_template/2   ate   lin   +
tam_template/2
       para  ind_col  =  col  -  tam_template/2  ate  col  +
tam_template/2
vet_elem_med ← pixels da vizinhança
fim para fim para
ordena vet_elem_med
fim para
fim para
```

Figura 12 - Algoritmo para implementação do filtro da mediana (NUNES, 2006).

3.2. Segmentação

A fase de segmentação tem a responsabilidade de identificar as formas significativas de uma imagem a fim de fornecer informações para a sua interpretação. Segundo Gonzales e Woods (2002), a segmentação pode ser definida como "o processo que subdivide uma imagem em suas partes ou objetos constituintes".

Os algoritmos de segmentação são geralmente baseados em propriedades básicas de valores de nível de cinza, como a descontinuidade (mudanças bruscas nos níveis de cinza, nessa categoria incluem técnicas como detecção de pontos isolados e detecção de linhas e borda) e a similaridade (baseiam-se em limiarização, crescimento de região entre outros) (GONZALES e WOODS, 2002).

3.2.1. Detecção de Bordas

A detecção de bordas é utilizada quando se deseja conhecer informações a respeito de tamanho e forma dos objetos representados na imagem.

Assim como os demais filtros no domínio espacial, esses algoritmos exigem uma série de operações aritméticas sobre o *pixel* e sua vizinhança.

Os operadores de Sobel executam operações na vizinhança do *pixel* fazendo com que o valor do *pixel* resultante seja diretamente proporcional à diferença existente entre o *pixel* e sua vizinhança. Assim, se o valor de um *pixel* for exatamente igual aos valores de seus vizinhos, o valor resultante no *pixel* processado será zero. Para realizar estas operações são utilizadas máscaras de coeficientes (GONZALES e WOODS, 2002).

No exemplo da Figura 13, considera-se uma vizinhança de 3x3 *pixels* em torno de um ponto central, representado por X5 na Figura 13 (a). Para se obter o valor do *pixel* central, efetua-se a soma das multiplicações de cada *pixel* da vizinhança pelo coeficiente correspondente na máscara em questão.

As máscaras horizontal, vertical, diagonal superior e diagonal inferior são apresentadas, respectivamente, nas Figura 13 (b), (c), (d) e (e)..

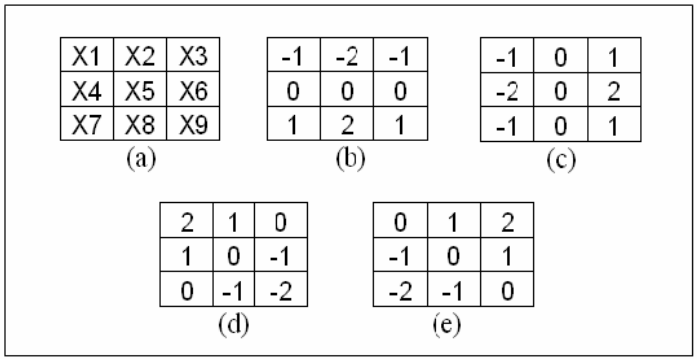

Figura 13 - Região e máscaras para detecção de bordas: (a) máscara para detecção de bordas horizontais; (b) máscara para detecção de bordas verticais; (c) máscara para detecção de bordas 45 graus; (c) máscara para detecção de bordas 45 graus negativos.

E as equações de 1 a 4 definem os operadores de Sobel na direção horizontal, vertical, diagonal superior e diagonal inferior.

(1) Sh = (x7 + 2x8 + x9) - (x1 + 2x2 + x3)

(2) Sv = (x3 + 2x6 + x9) - (x1 + 2x4 + x7)

(3) Sds = (x2 + 2x1 + x4) - (x6 + 2x9 + x8)

(4) Sdi = (x2 + 2x3 + x6) - (x4 + 2x7 + x8)

Onde:

Sh é o valor resultante do operador de detecção de bordas horizontal que será atribuído ao *pixel* X5;

Sv é o valor resultante do operador de detecção de bordas vertical que será atribuído ao *pixel* X5;

Sds é o valor resultante do operador de detecção de bordas 45 graus;

Sdi é o valor resultante do operador de detecção de bordas 45 graus negativos que será atribuído ao *pixel* X5 e;

Os demais X_is são os *pixels* pertencentes à vizinhança-de-8 de X5.

3.3. JAI (Java Advanced Imaging)

A JAI (*Java Advanced Imaging*) é uma API que fornece um conjunto de classes e métodos implementados para a manipulação de imagens.

Permite a rápida criação de aplicações de processamento de imagens de forma sofisticada e com alto desempenho. Está disponível gratuitamente e sem restrições de

distribuição. Oferece vantagens como independência de plataforma, orientação objeto flexibilidade e facilidade de extensão e eficácia, além de conseguir ler e escrever em diversos formatos (BMP, GIF, FPX (FlashPix), JPEG, PNG, PNM, TIFF e WBMP) (SANTOS, 2004).

Na JAI a classe básica para representar uma imagem é a classe `PlanarImage, (`somente para leitura, não possui métodos capazes de modificar os valores dos *pixels* que ela contém) que armazena os elementos da imagem os *pixels* em outra classe chamada `Raster`. A Raster contém uma instância de uma outra classe, a `DataBuffer` (SANTOS, 2004).

Por sua vez a calsse `DataBuffer` é moldada segundo as regras da classe `SampleModel` que define, entre outras coisas, se a imagem será colorida ou em níveis cinza `ColorModel.` Estas classes são usadas para a JAI representar uma imagem na memória, para a alteração dos *pixels* deve ser usada a classe `TiledImage` (SANTOS, 2004).

3.4. Um Modelo de Paralelismo para o Processamento de Imagens

Os requisitos básicos de um sistema de processamento paralelo de imagem consistem em uma infra-estrutura que permite executar de forma eficiente quaisquer algoritmos de nível baixo, médio ou alto.

Essa infra-estrutura é composta essencialmente de funções de comunicação e distribuição de dados adequados ao processamento de imagem (BARBOSA, 2000).

Como mencionado anteriormente, os filtros executados no domínio espacial manipulam diretamente os *pixels* que compõem a imagem. São os mais utilizados devido à facilidade de implementação, mas exigem alto poder de processamento, visto que as imagens constituem, na maioria das vezes, matrizes enormes de pontos a serem processados. É neste contexto, que as técnicas de processamento de imagens e, em especial aquelas desenvolvidas especificamente 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 mais realçada, visto que esta classe de imagens não pode permitir armazenamento com perdas de dados se a imagem for utilizada para diagnóstico. Além disso, muitas vezes, exigem precisão na sua aquisição, gerando um conjunto ainda maior de dados.

Segunda Barbosa (BARBOSA, 2000) para uma arquitetura de memória distribuída como a utilizada o modelo de programação mais adequado e eficiente é a divisão da imagem em blocos (sub-imagens) distribuídos pelos processadores.

Para imagens médicas, a divisão da imagem em blocos e a posterior junção desses blocos sem perdas de processamento. Na Figura 14 são apresentadas as formas mais comuns de divisão de uma imagem em blocos.

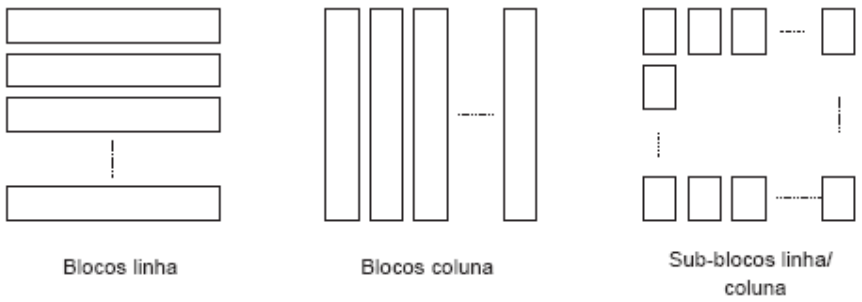

Figura 14 - Organização dos dados Blocos (BARBOSA, 2000)

Para algoritmos que fazem uso de máscaras coeficientes que operam sobre uma "vizinhança" de pontos da imagem, é necessário garantir que os processadores tenham as linhas de fronteira do bloco adjacente necessárias para calcular os elementos de fronteira do seu bloco (Figura 15), sendo assim há necessidade de redundância nos blocos para o processamento (NICOLESCU e JONKER 2002; BARBOSA, 2000). Segundo Barbosa (2000) existe duas possibilidades para redundância: o mestre divide os dados entre os escravos que ficam responsáveis por identificar os dados extra que são necessários para o processamento solicitado os mesmo ao mestre, ou o mestre distribui de forma redundante os dados garantindo que todos recebem os dados que precisa, sendo está ultima a mais eficiente.

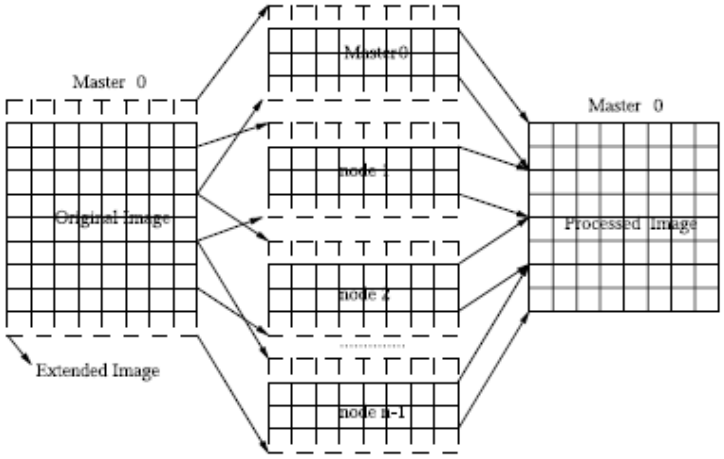

Figura 15 - Estratégia de paralelização para algoritmos baseado em máscara (NICOLESCU e JONKER 2002)

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.

Outro ponto importante é a definição do modelo de algoritmo paralelo que segundo Navaux (NAVAUX *et al.,* 2001), são possíveis de serem resumidos em cinco paradigmas principais: Divisão e Conquista, *Pipeline*, Mestre-Escravo, *Pool* de Trabalho e Fases Paralelas. É possível considerar o paradigma mestre/escravo o que melhor se adapte ao projeto.

No paradigma Mestre-Escravo, o processo denominado como mestre executa as tarefas essenciais do programa paralelo além atribuir tarefas os processos escravos e atuar como um supervisor dos mesmos. Este paradigma é bastante simples, visto que o controle das ações está centralizado num processo mestre; desvantagem é a possibilidade de gargalo no mestre, além da sobrecarga de comunicação (NAVAUX *et al.,* 2001).

3.4.1. Trabalhos Correlatos

A seguir são apresentadas sínteses de alguns trabalhos de importância relevante para as áreas de programação paralela e processamento de imagens e para o desenvolvimento deste trabalho.

O Paralelismo em processamento de Imagens Médicas foi proposto em (BARBOSA, 2000). Neste trabalho, o objetivo principal constituiu na redução do tempo de processamento a fim de permitir principalmente que algoritmos mais sofisticados pudessem ser utilizados para aplicações médicas sem gerar um desconforto ao usuário, aproveitando a capacidade de processamento disponível em uma rede de computadores.

As imagens que foram paralelizadas utilizaram técnicas que se enquadram nos seguintes níveis: baixo (histograma, *thresholding* e detecção de arestas por filtros digitais IIR), alto (fatoração de LU, redução de tridiagonal, interação QR e correlação de matriz).

Foi apresentado um sistema de processamento paralelo de imagem auto-configurável, designado por NetPro, que utiliza o WPVM para troca de mensagem e é executado sobre o sistema operacional Windows NT. O NetPro admite tarefas tanto seqüenciais como paralelas, sendo que a tolerância a falhas é feita de forma parcial. Antes de iniciar a execução paralela dos algoritmos, o NetPro determina a melhor configuração atendendo as máquinas que estão disponíveis, a fim de ter equilíbrio na distribuição de carga sendo ela distribuída para um conjunto de máquinas com característica semelhantes para obter uma sincronização no processamento.

Para o teste foram utilizados três tipos de *cluster*: um homogêneo composto por 6 máquinas de 141Mflop/s e outros dois heterogêneos compostos por máquinas que vão desde 244Mflop/s ate 49Mflop/s. Como conclusão foi possível obter uma redução no tempo de processamento e permitir o uso de tais algoritmos com mais comodidade ao usuário.

O problema do balanceamento de carga quando a utilização do processamento de imagens em ambientes distribuídos foi alvo do artigo *Distributed Image Processing On A Network Of Workstations* (LI et al., 2003), este trabalho comparou duas estratégias para divisão da imagem: EQS (*Equal-Partitioning Strategy*), onde a imagem é dividida em tamanhos iguais não levando em conta a capacidade de processamento dos processadores envolvidos; outra estratégia abordada pelo artigo é a PSSD (*Divisible*

Load-Scheduling Theory), estratégia baseada no paradigma DLT (*Divisible Load Theory*). Como parâmetro para distribuição de carga, o DLT utiliza, velocidade do processador e comunicação, sua preocupação primária é determinar frações da carga proporcionais aos processadores a fim de reduzir o tempo de processamento.

O processamento de imagem através do algoritmo de Sobel foi utilizado como base para comparação entre EQS e PSSD, sua utilização vem em decorrência principalmente da necessidade de intenso poder computacional e viabilização da exploração pelo paralelismo, para troca de mensagem foi utilizado PVM. A divisão da imagem é feita pelo mestre que envia os *pixels* de limite respectivos a cada processador, cada processador processa sua parte envia ao mestre para posterior junção, compondo o paradigma mestre/escravo. Para o processamento foi utilizado um cluster heterogêneo formado por 15 máquinas sub-divididas em 4 grupos, sendo seis máquinas HP-C200, quatro máquinas SGI O2 e cinco DEC AlphaStation500/500. Como conclusão foi possível observar que o DLT teve um impacto significante para aplicações que necessitam de intenso uso da UCP.

Seinstra e Koelma (2004) apresentaram o trabalho *A Fully Sequential Programming Model for Efficient Data Parallel Image Processing* (SEINSTRA e KOELMA, 2004), onde buscaram promover uma ferramenta a fim de permitir o uso transparente de aplicações de processamento de imagem em paralelo para execução em multicomputadores baseados na arquitetura MIMD homogênea utilizando um *cluster* do tipo *Beowulf*. Os autores justificam o trabalho em decorrência da uma grande gama de algoritmos ou operações em processamento de imagem que podem ser beneficiados pelo uso da computação paralela. Foi proposta a implementação de um grande número de operações de processamento de imagem a fim de constituir uma biblioteca de processamento de imagens em paralelo. Mais especificamente, em vez de uma biblioteca completamente nova, foi proposto integrar as funcionalidades em paralelo à implementação da biblioteca Horus, de tal maneira que os códigos seqüenciais existentes permanecessem intactos. Os algoritmos de processamento de imagem definidos para implementação foram: *template matching, multi-baseline stereo vision e line detection*, sendo que execução é feita basicamente na transferência de dados entre os nós, que processam seqüencialmente sua parte estabelecida e ao final são juntados a uma única unidade de processamento.

Vale ressaltar que cada algoritmo tem a sua característica, além disso, os autores apresentaram mais de uma técnica para a paralelização de determinados

algoritmos. As aplicações foram desenvolvidas em linguagem C, sendo utilizado o MPI-LFC (implementação do MPI aperfeiçoado para cluster utilizado nos testes) para troca de mensagem. Resultados obtidos comprovaram um significante ganho de desempenho nos três tipos de algoritmos propostos.

3.5. Considerações Finais

Neste trabalho o objetivo é verificar o desempenho de técnicas implementadas de forma seqüencial e paralela. Foram escolhidas técnicas no domínio espacial (suavização e detecção de bordas) a fim de verificar a relação custo/benefício decorrente do uso da computação paralela distribuída.

CAPÍTULO 4. IMPLEMENTAÇÃO DE ALGORITMO PARALELO PARA APOIAR O PROCESSAMENTO DE IMAGENS UTILIZANDO JPVM

Estratégia de paralelização em que uma imagem médica é dividida em blocos pelo mestre, e subseqüentemente é transmitida por meio da biblioteca JPVM aos escravos. Os escravos têm a incumbência de processar a imagem e retransmitir os blocos já processados ao mestre, o qual os une reconstituindo a imagem (NICOLESCU e JONKER 2002; BARBOSA, 2000).

Na Figura 16 é demonstrada uma estratégia voltada para imagens médicas. A imagem é dividida em blocos de linha, é possível observar que há uma redundância em cada parte da imagem (destacado em vermelho) necessária para processamento. No bloco 1 a redundância só ocorre na parte final, já no bloco 2 ocorre tanto no inicio como no final e por fim, no bloco 3 a redundância ocorre no inicio do bloco, sendo assim, a redundância varia dependendo da parte da imagem que esta sendo dividida.

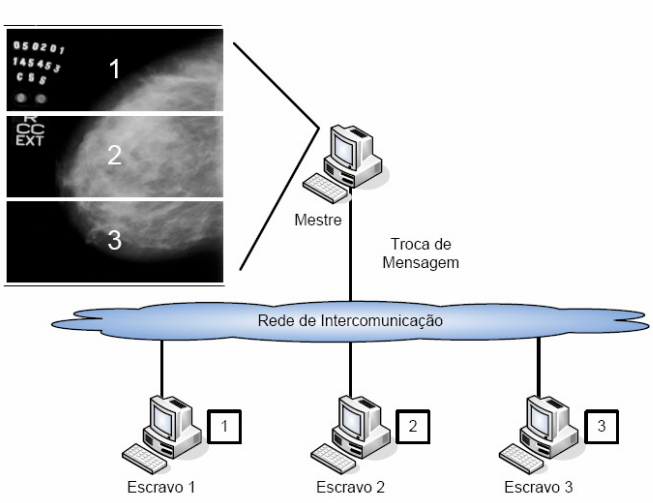

Figura 16 - Estratégia de paralelização de imagens médica.

Para desenvolvimento de algoritmo paralelo para processamento de imagem é necessário enfatizar alguns pontos importantes, como responsabilidade que cada *host* tem no processamento, redundância que deve conter em cada bloco, entre outros. Na Figura 17 é demonstrado uma implementação genérica do algoritmo paralelo utilizado

para o processamento de imagens, note que o tamanho da redundância (linha 5) é definida de acordo com o tamanho da máscara que está sendo utilizada.

```
1 abre imagem
2 define num_processos
3 caso seja mestre
4    para i = 0 até num_processos
5    tam_bloco  =  (altura_imagem  div  num_processos)  +
máscara
6    define bloco com tam_bloco
7    para j = 0 até tam_bloco
8        copia pixels da imagem no bloco
9    envia(bloco)
10   para k = 0 até num_processos
11   recebe(bloco)
12     para j = 0 até tam_bloco
13        copia pixels do bloco na imagem
14 caso seja escravo
15    recebe(bloco)
16    processa(bloco)
17    envia(bloco)
```

Figura 17 - Implementação genérica do algoritmo paralelo utilizado para o processamento de imagens

4.1. Desenvolvimento do Algoritmo Paralelo

Com base nos estudos efetuados para este trabalho foi desenvolvida uma aplicação utilizando a linguagem de programação Java. Como especificado na seção 4.3, a aplicação baseia-se no paradigma mestre-escravo com paralelismo de dados, isso é, o código fonte a ser executado é o mesmo em cada *host*.

O desenvolvimento do algoritmo é divido em duas etapas: a primeira trata do desenvolvimento do programa mestre e a segunda trata da implementação do programa escravo. É importante relatar que a API JPVM não exige que os códigos referentes ao mestre e ao escravo estejam em uma mesma classe.

4.1.1. Implementação do Mestre

Pode-se considerar o mestre o elemento mais importante da aplicação como um todo. Atua como gerenciador de todo processamento paralelo, é atribuído à tarefa de obtenção das informações referente à imagem de interesse a ser processada, divisão da imagem em blocos para posterior envio aos *hosts* pertencente à máquina virtual paralela, reconstrução da imagem final a partir dos blocos processados pelos *hosts* e por fim geração em arquivo da imagem final processada.

O programa mestre é constituído de duas classes: Image.java e a MasterProcess.java, as quais são ilustradas na Figura 18. As classes implementadas foram criadas utilizando a modelagem orientada a objetos UML (*Unifed Modeling Language*), conforme (BOOCH *et al.,* 2000).

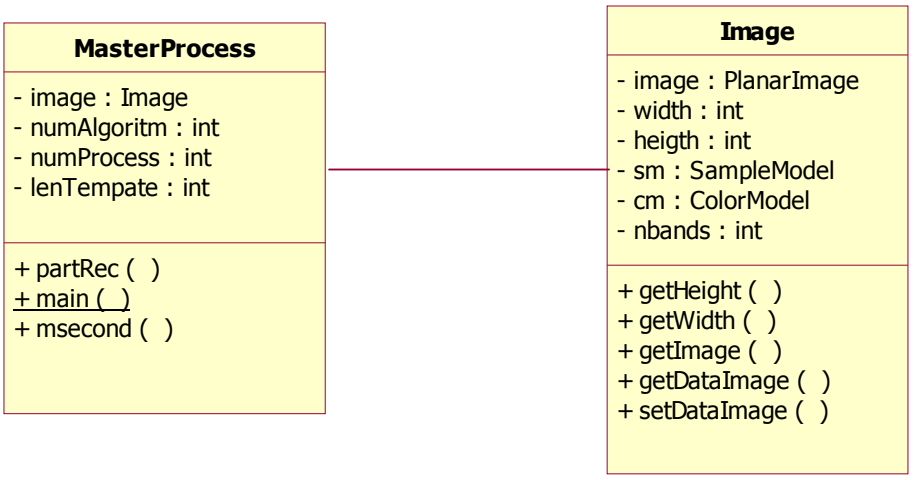

Figura 18 - Diagrama de classe referente ao programa mestre

4.1.1.1. Classe Image

Na classe *Image* (APÊNDICE A) são definidos os métodos para obtenção de informações referentes à imagem como altura, largura, número de canais, vetor de *pixels,* além de fornecer método para salvar a imagem.

Na Figura 19 é demonstrada a estrutura interna do construtor da classe *Image*. É utilizada a função básica JAI.create fornecida pela API JAI para a abertura de um

arquivo (imagem). Para tal o arquivo pode estar armazenado em qualquer tipo de mídia, porém deve estar nos formatos suportados pela JAI. A palavra-chave *this* é um atalho conveniente disponibilizado pela linguagem Java para acessar o objeto atual por inteiro, não uma variável de instância em particular.

```
public Image(String ima){
          this.image  = JAI.create("fileload",ima);
          this.width  = image.getWidth();
          this.height = image.getHeight();
          this.sm     = image.getSampleModel();
          this.cm     = image.getColorModel();
          this.nbands = sm.getNumBands();
     }
```

Figura 19 - Construtor da classe Image

É importante destacar que a API JAI diferentes de outras APIs Java e linguagens de programação como Delphi que dão suporte ao processamento imagem, trata a imagem digital não como uma matriz de pixels, mas como um vetor de pixels, assim métodos devidos na classe Image tem como retorno ou parâmetro um vetor de pixels e não uma matriz. Os métodos da classe Image são listados na Tabela 3.

Tabela 3. Métodos da classe Image

Método	Descrição
`public int[] getDataImage ()`	Retorna vetor de pixels referente à imagem de interesse.
`public int getWidth()`	Retorna largura da imagem.
`Public int getHeight()`	Retorna altura da imagem.
`Public int getBands()`	Retorna número de faixas da imagem.
`public void setDataImage (int[] pixels)`	Salva imagem no formato *tiff*, referente ao vetor de pixels passado por parâmetro.

Os métodos *get* e *set,* muito utilizados na linguagem Java, têm como função essencial tornar publico as variáveis de instancias geralmente definidas como privadas.

4.1.1.2. Classe MasterProcess

A classe *MasterProcess* (APÊNDICE B) é a principal referente ao mestre e aplicação como todo, atuando como gerenciador de todo o processo.

As variáveis de instância da classe *MaterProcess* foram ilustrada na Figura 18, a variável ***nameImage*** guarda o nome e o caminho da imagem a ser processada, ***numAlgorit*** define qual algoritmo de processamento de imagem, para mediana fica específico o número 0 e detecção de borda qualquer outro valor diferente de 0, ***numProcess*** o número de processo que será gerado para o processamento ou de maneira mais simples refere-se ao número de blocos na qual a imagem de interesse será dividida, e por fim ***lenTemplate*** define o tamanho da máscara que será utilizado pelos algoritmo de processamento de imagem (Mediana ou Detecção de Borda).

Na Figura 20 é demonstrada a estrutura interna do construtor da classe *MasterProcess*. O método construtor é usado para iniciar objetos de classe, dando as variáveis de instancia ou atributos da classe *MasterProcess* o estado inicial que é desejável. Por fim é invocado o método *setDataImage* (definido na seção anterior) pela variável *image* instanciada como um objeto da classe *Image*, recebendo como parâmetro o retorno do método *PartRec()*.

```
public MasterProcess(String nomeImage, int numAlgoritm, int
process, int template) {
     this.image        = new Image(nomeImage);
     this.numAlgoritm  = numAlgoritm;
     this.numProcess   = process;
     this.lenTemplate  = template;

       image.setDataImage(PartRec());
}
```

Figura 20 - Constutor classe MasterProcess

Método msecond

O utilizado para capturar o horário da máquina a fim de medir o tempo de execução.

Método *ParRec()*

O método *PartRec* é o principal método definido na classe *MasterProcess*, tendo como principais funções, definir dimensões do bloco, inicializar o ambiente JPVM, divisão da imagem e posterior envio para outros *hosts* da máquina virtual, reconstrução imagem final a partir dos blocos processados.

Definição dimensões dos Blocos

Para definir as dimensões (altura e largura) que cada bloco vai possuir, foi utilizado o bloco de linha para a implementação, neste tipo de bloco a largura será igual a da imagem original, já altura padrão para os blocos é possível de ser definida de acordo com o número de processos definido, como demonstrado na Figura 21.

```
int heigthSubimage = (heigth/numProcess);
```

Figura 21 - Altura do bloco

Início do Ambiente Paralelo

Iniciar o ambiente JPVM a fim de permitir que a aplicação tenha acesso a os recursos como criação de processo, métodos para troca de mensagens, necessários para viabilizar o processamento paralelo. É demonstrado na Figura 22 o trecho de código que define o início do ambiente, a fim de evitar exceção, um problema durante a execução do programa é definido um bloco *try/catch*, dentro deste bloco é instanciada objeto da classe *jpvmEnvironment*, o mesmo cria um processo que é por sua vez registrado junto à máquina virtual local, a identificação deste processo é guarda na variável *mytid*, na identificação do processo consta o nome da máquina e a porta associada ao processo.

É invocado o método pvm_spawn, usado para criação da tarefa.

```
try {
     jpvmEnvironment jpvm = new jpvmEnvironment();
     jpvmTaskId mytid = jpvm.pvm_mytid();
      jpvmTaskId tids[] = new jpvmTaskId[numProcess];
      jpvm.pvm_spawn("Slave",numProcess,tids);
     ...
     ...
       jpvm.pvm_exit();
    }
     catch (jpvmException jpe) {
             System.out.println("Error - jpvm exception");
     }
```

Figura 22 – Inicializando Ambiente Paralelo

Divisão da Imagem

Para divisão da imagem em blocos é necessário um comando de repetição que varia 0 até o número de processos definidos.

O primeiro ponto tratado é a redundância que deve conter em cada bloco, como já abordado esta varia dependendo da parte da imagem que esta sendo dividida, sempre

no primeiro bloco a redundância só ocorre na parte final, já nos blocos intermediários ocorre tanto no inicio como no final, e no último bloco a redundância ocorre somente no início do bloco.

Para tratar a questão da redundância foi definida uma variável ***redundancy*** que quando o valor é igual a 1, a redundância ocorre somente na parte final do bloco (primeiro bloco), caso o valor seja igual a 2, a redundância ocorre em dois pontos, sendo tanto no início como no final do bloco (blocos intermediários), o último bloco foi incluído neste caso.

Para o tamanho do bloco foram levados em consideração à altura definida, a quantidade de ponto que haverá de redundância e o tamanho da redundância que esta relacionada como o tamanho da máscara. Para copiar as informações (*pixels*) da imagem original para o bloco foi utilizado a método *arraycopy* fornecido pela classe *System*, este método requer como parâmetro vetor a qual será copiado os dados, posição de início da cópia dentro deste vetor, vetor que receberá a cópia e a quantidade de dados a serem copiados, como demonstrado na Figura 23.

```
int[] subimage =new int[(width*heigthSubimage*nbands)+

(redundancy*(lenTemplate/2)*width*nbands)];

System.arraycopy(pixelsOriginal,(beginSubimage-
((redundancy-
1)*(lenTemplate/2)*width*nbands)),subimage,0,subimage.lengt
h);
```

Figura 23 – Geração do bloco

Após a criação do bloco é iniciado o processo de envio do mesmo para o processamento. Na Figura 24 é demonstrada a criação de um *buffer* sendo armazenado no mesmo as informações necessárias para o programa escravo possa efetuar o processamento, lembrando que a ordem de empacotamento de ser a mesma ordem de desempacotamento.

Após o empacotamento é invocado método *send* tendo como parâmetros o *buffer* com a mensagem, a identificação do processo escravo que vai receber a mensagem, e o número de identificação da mensagem que será fundamental para posterior reconstrução da imagem na ordem correta.

```
      jpvmBuffer buf = new jpvmBuffer();
           buf.pack((numAlgoritm));
           buf.pack((lenTemplate));
           buf.pack(width);
           buf.pack(nbands);
           buf.pack((subimage.length)/(width*nbands));
           buf.pack(subimage.length);
           buf.pack(subimage,subimage.length,1);
           jpvm.pvm_send(buf,tids[j],j);
```

Figura 24 – Empacotamento e Envio para Escravos.

O último bloco tem uma particularidade em relação ao demais: tamanho do mesmo pode variar, exemplo uma imagem com dimensões de 100X100 (linha X coluna) dividida em quatro blocos, não considerando a redundância cada bloco vai apresentar dimensões de 25X100, caso se divida em três processos a dimensão dos dois primeiros será 33X100 e o último bloco 34X100, para controle da redundância é rotulado como bloco intermediário, neste caso deve-se ocorrer uma redundância não somente na parte inicial da imagem, mas agora também na parte final do bloco, para isso é reduzida à altura da imagem original de acordo com o tamanho da redundância e as linhas que sobrarem no final são consideradas como redundância na parte final do bloco. Como é necessário em certos casos alterar a altura deste bloco, o trecho de código correspondente é demonstrado na Figura 25.

```
if((heigth- ((beginSubimage+((lenTemplate/2)*width*nbands))
              /(width*nbands)))< heigthSubimage)

heigthSubimage = (heigth-((beginSubimage+(redundancy*
(lenTemplate/2)*width*nbands))/(width*nbands)));
```

Figura 25 – Redimensiona altura do último bloco

Para proporcionar uma melhor análise é demonstrado na Figura 26 toda implementação que diz respeito à divisão da imagem.

```
  redundancy=1;
  for(int j=0;j<numProcess;j++){

  int[] subimage =new int[(width*heigthSubimage*nbands)+

(redundancy*(lenTemplate/2)*width*nbands)];

System.arraycopy(pixelsOriginal,(beginSubimage-
            ((redundancy-1)*(lenTemplate/2)*width*nbands))
                 ,subimage,0,subimage.length);
       jpvmBuffer buf = new jpvmBuffer();
       buf.pack((numAlgoritm));
       buf.pack((lenTemplate));
       buf.pack(width);
       buf.pack(nbands);
       buf.pack((subimage.length)/(width*nbands));
       buf.pack(subimage.length);
       buf.pack(subimage,subimage.length,1);
       jpvm.pvm_send(buf,tids[j],j);

       beginSubimage += (heigthSubimage*width*nbands);
       redundancy=2;

  if((heigth-((beginSubimage+((lenTemplate/2)*
           width*nbands))/(width*nbands)))< heigthSubimage)
    heigthSubimage = (heigth-((beginSubimage+(redundancy*
           (lenTemplate/2)*width*nbands))/(width*nbands)));
 }
```

Figura 26 – Trecho de código referente a divisão da imagem

Reconstrução da Imagem Final

Para fase de reconstrução da imagem é fundamental identificar a qual parte pertence na imagem original o bloco processado.

Para reconstrução da imagem é necessário um comando de repetição que varia de 0 até o número de processos definidos.

Para o recebimento da mensagem enviada pelos escravos é utilizado o método *pvm_recv*, é bom ressaltar que este é um método bloqueante. No conteúdo da mensagem tem-se o tamanho vetor e o vetor de *pixels* processado, demonstrado na Figura 27.

```
jpvmMessage message = jpvm.pvm_recv();
       int lenSubimage = message.buffer.upkint();
       int[] subimage = new int[lenSubimage];
       message.buffer.unpack(subimage,lenSubimage,1);
```

Figura 27 – Recebimento das Mensagens Enviadas pelos Escravos

Após o recebimento da mensagem é necessário identificar qual é a posição correta do bloco dentro da imagem, além de eliminar a redundância gerada na fase de envio. A identificação da mensagem atribuída na fase de envio é utilizada como parâmetro para a identificação do bloco.

A Figura 28 demonstra o trecho de código que calcula a posição correspondente do bloco enviado pelo escravo na imagem final. Como é possível observar são considerados dois casos:

- Caso a mensagem tenha como identificação o valor 0, trata-se então do primeiro bloco, sendo assim tanto o índice da imagem final como o bloco deve iniciar em zero.
- Caso a mensagem tenha como identificação um valor diferente de 0, para identificar a posição correspondente do bloco na imagem é necessário aliar o valor do identificador com altura padrão definido para os blocos, já para o índice do bloco é necessário eliminar a redundância.

```
if( message.messageTag == 0){
 indexBeginSubImage = 0;
      indexImageOriginal = 0;
   }
else{
 indexBeginSubImage =(lenTemplate/2)*width*nbands;
 indexImageOriginal =(heigthSubimage*message.messageTag)
                        *width*nbands;
}
```

Figura 28 - Define posição inicial correspondente do bloco na imagem final

Para eliminar a redundância no final de cada bloco é utilizado o trecho de código apresentado na Figura 29, a eliminação ocorre para todos os blocos, entretanto caso seja o último não deve ocorrer a eliminação da redundância, sendo assim é reposto no mesmo a redundância a fim de evitar perdas de dados.

```
indexFinalSubimage=               lenSubimage                -
((lenTemplate/2)*width*nbands);

  if( message.messageTag+1 == numProcess ){
  if((heigth%(heigth/numProcess)) <= 1)
  indexFinalSubimage = indexFinalSubimage
                        +(width*(lenTemplate/2)*nbands);
}
```

Figura 29 – Eliminação da redundância nos blocos

Na Figura 30 é ilustrado o trecho de código responsável pela cópia do bloco para a imagem final.

```
for(int k=indexBeginSubImage;
     k<indexFinalSubimage;
     k++,indexImageOriginal++)
     pixelsResult[indexImageOriginal]=subimage[k];
```

Figura 30 – Cópia dos dados do bloco para imagem original

4.1.2. Implementação do Escravo

O programa escravo é disparado pelo JPVM em resposta à solicitação feita pelo programa mestre, ao escravo é atribuída à tarefa de aplicar os algoritmos de processamento de imagem (Mediana e Sobel) ao bloco a ele destinado.

O programa escravo pode ser executado em qualquer *host* que faça parte da máquina virtual paralela, incluindo o *host* mestre. Outra importante observação é que como a aplicação foi dividida em dois programas mestre e escravo, é necessário ter uma cópia do programa escravo em todos os *hosts,* ao contrário do programa mestre que é necessário estar presente somente no *host* mestre.

Para a implementação programa escravo foram desenvolvidas duas classes: *Slave.java* e a *ImageProcesing*, as quais são ilustradas na Figura 31. As classes implementadas foram criadas utilizando a modelagem orientada a objetos UML, conforme (BOOCH *et al.,* 2000).

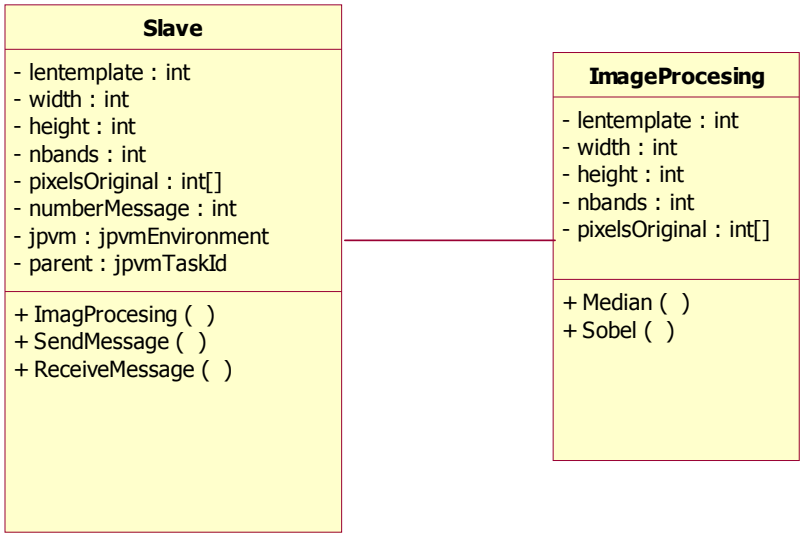

Figura 31 - Diagrama de classe referente ao programa mestre

4.1.2.1. Classe Slave

A classe *Slave* (APÊNDICE C) é a principal referente ao programa escravo, sendo inicializada pelo JPVM em resposta à solicitação feita pelo programa mestre.

Na Figura 32 é demonstrada a estrutura interna do construtor da classe *Slave*, a fim de evitar exceção é definido um bloco *try/catch*, o ambiente paralelo com instância da classe *jpvmEnvironment.* Como já detalhado, é criado um processo que é por sua vez registrado junto à máquina virtual local, por meio do método *pvm_parent()* é possível identificar o pai que gerou este processo escravo no nosso é processo mestre. Esta identificação é importante para escravo, saber para quem ele deve retornar o bloco processado.

```
public Slave(){
  try{
      jpvm = new jpvmEnvironment();
      parent = jpvm.pvm_parent();
      ReceiveMessage();
      ImagProcesing();
      SendMessage(ImagProcesing());
      jpvm.pvm_exit();
      }
     catch (jpvmException e) {
          System.out.print(e.getMessage());
     }
   }
```

Figura 32 – Estrutura interna do construtor da classe Slave

.

O funcionamento do programa escravo foi dividido em três fases.

- Recebimento da Mensagem contendo o bloco a ser processados;
- Aplicação de algoritmo de processamento de imagem;
- Envio do bloco processado ao mestre.

Recebimento da Mensagem

A primeira fase consiste no recebimento por parte do programa escravo de uma mensagem enviada pelo mestre, sendo que o conteúdo da mensagem é composto por somente informações necessárias ao programa escravo para realizar o processamento.

Na Figura 33 é apresentado o trecho de código responsável pelo recebimento das mensagens por parte dos escravos, é utilizado o método bloqueante *pvm_recv*, já a

seqüência de desempacotamento do pacote deve estar de acordo com a seqüência de empacotamento realizado pelo programa mestre. Além de todas as informações necessárias para o processamento, número da mensagem é uma informação fundamental para reconstrução da imagem.

```
public void ReceiveMessage(){

     try{
        jpvmMessage message = jpvm.pvm_recv();

         optionAlgor          = message.buffer.upkint();
        lenTemplate          = message.buffer.upkint();
        width                = message.buffer.upkint();
        nbands               = message.buffer.upkint();
        height               = message.buffer.upkint();
        lenSubVector         = message.buffer.upkint();
        pixelsOriginal       = new int[lenSubVector];

message.buffer.unpack(pixelsOriginal,lenSubVector,1);
         numberMessage       = message.messageTag;

     }
     catch (jpvmException e) {
        System.out.print(e.getMessage());
         }
    }
```

Figura 33 – Método *ReceiveMessage* classe *Slave*

Algoritmo de processamento de imagem

Para iniciar a passagem de alguns dos filtros definidos neste trabalho é chamado o método *ImagProcesing*. Com todas as informações necessárias este método instancia a classe *ImageProcesing* (APÊNDICE D) e dependendo da escolha do usuário é aplicado ou filtro de mediana ou é feita a detecção das bordas, ambos de forma seqüencial no bloco, como demonstrado na Figura 34.

```
public int[] ImagProcesing(){
     ImageProcess ima = new ImageProcess(lenTemplate,width,
height,nbands,pixelsOriginal);
          if(optionAlgor == 0){return ima.Mediana();}
          else{ return ima.calcula();}
     }
```

Figura 34 – Método *ImagProcesing* classe *Slave*

Envio do bloco processado ao mestre

A última fase realizada pelo escravo é a geração de uma mensagem destina ao mestre contendo com o bloco processado.

O método responsável é apresentado na Figura 35. É criado um *buffer*, os dados necessários para o mestre reconstruir a imagem são empacotados, é possível observar, o destinatário neste caso o mestre e a identificação da mensagem é a mesma que foi passada pelo mestre ao escravo.

```
public void SendMessage(int[] pixelsResult){
      try{
      jpvmBuffer buf = new jpvmBuffer();
            buf.pack(pixelsResult.length);
            buf.pack(pixelsResult,pixelsResultado.length,1);
            jpvm.pvm_send(buf,parent,numberMessage);
      }
      catch (jpvmException e) {
                  System.out.print(e.getMessage());
            }
    }
```

Figura 35 – Método SendMessage classe Slave

4.2. Considerações Finais

Os conceitos e operações de processamento de imagens apresentadas no início deste capítulo foram usadas na implementação da aplicação. Para a implementação das mesmas foi necessário o auxílio da API JAI específica para a manipulação de imagens

A implementação foi bem sucedida principalmente pelas facilidades providas pelas bibliotecas Java, entre elas especialmente a JAI e o JPVM, que com suas classes e métodos ofereceram suporte à maioria dos procedimentos que foram usados.

A implementação tem como objetivo o desenvolvimento de uma aplicação orientada à objeto, explorando as característica fornecida pela linguagem Java. A divisão da aplicação em duas partes Mestre e Escravo vêm ao encontro com da necessidade em se ter uma aplicação de fácil manutenção, simples e principalmente eficiente e extensível possibilitando a inclusão de outros algoritmos de processamento de imagem com rapidez.

CAPÍTULO 5. TESTES E RESULTADOS

As execuções foram realizadas no Laboratório sete do Centro Universitário "Eurípides Soares da Rocha" de Marília - UNIVEM. Os experimentos foram realizados em uma rede de computadores pessoais conectados por uma rede padrão *ethernet* 100 Mbits interligada por um *switch*, em que todos utilizavam o sistema operacional Linux (*kernel* 2.6). A rede foi composta por 10 máquinas homogêneas (Pentium IV de 2.7GHz com 512Mbytes de RAM), a qual possibilitou a formação de uma arquitetura MIMD com memória distribuída por meio da API JPVM, tornando-se factível a execução de aplicações paralelas desenvolvidas.

A análise de desempenho dos algoritmos de filtro de mediana e de detecção de bordas foi realizada por meio de diferentes testes reais em um ambiente paralelo distribuído controlado. Foram utilizadas imagens mamográficas, no formato TIFF, com resolução de contraste de 16 bits, 2048x2751(linha X Coluna) e 2855x3835 pixels, respectivamente. Tais imagens fazem parte de um banco de imagens desenvolvido pelo LAPIMO (Laboratório de Processamento de Imagens Médicas e Odontológicas, da EESC/USP)..

O filtro de mediana foi avaliado com máscara de tamanhos diferentes (3x3, 5x5 e 7x7) utilizando o algoritmo de ordenação *shellsort*. O filtro de detecção de bordas foi executado com máscara de tamanhos 9x9 e 11x11. Observa-se que o tamanho da máscara é o que define o tamanho da vizinhança a ser considerada. Dependendo do tipo da imagem, uma maior vizinhança indica um melhor resultado de processamento.

Após a realização dos testes obteve-se uma média dos tempos de processamento tanto para aplicação seqüencial quanto para a paralela para os diferentes tipos de máscaras, o que permitiu efetuar uma análise estatística dos resultados obtidos.

5.1. Análise Estatística Considerada

O objetivo de realizar uma análise estatística neste trabalho é verificar se as diferenças de desempenho da aplicação, considerando o modo de execução das mesmas (seqüencial e paralelo). Desse modo, utiliza-se a técnica estatística de Teste de Hipóteses (ACHCAR, J.A., RODRIGUES, J. 1995; FRANCISCO, W. 1995).

O primeiro passo para a utilização do Teste de Hipóteses consiste na definição dessas duas hipóteses que, após realizar o cálculo dos testes, uma delas será aceita e a outra rejeitada. A hipótese H_0 ou hipótese de nulidade é geralmente formulada procurando-se "discordar" dos valores obtidos com o experimento. A hipótese H_1 ou hipótese alternativa é aquela que geralmente "concorda" com os valores obtidos no experimento quando comparados "in-loco". Assim, a hipótese H_0 é a negação da hipótese H_1.

A hipótese H_0 é utilizada quando os valores obtidos com a aplicação paralela são menores que os obtidos com a aplicação seqüencial, isso é, o desempenho da aplicação paralelizada (AP) é melhor que o desempenho da mesma aplicação seqüencial (AS).

Para se realizar a análise estatística dos tempos coletados faz-se, portanto, os seguintes testes de hipóteses:

- para amostras onde o tempo do AP < tempo do AS:

H0: $\mu AP \geq \mu AS$

H1: $\mu AP < \mu AS$

- para amostras onde o tempo do AP > tempo do AS:

H0: $\mu AP \leq \mu AS$

H1: $\mu AP > \mu AS$

onde: μAP e μAS representam respectivamente as médias dos tempos de execução de uma aplicação de processamento de imagens médicas em paralelo e seqüencial.

Considerando-se que os 30 tempos obtidos para cada média comparada possuem uma distribuição normal, a estatística dos testes de hipóteses acima é dada por:

$$Z = \frac{\overline{X}_{AP} - \overline{X}_{AS}}{\sqrt{\frac{S_{AP}^2}{n_{AP}} + \frac{S_{AS}^2}{n_{AS}}}} \qquad \text{Equação 4.1}$$

onde: $\overline{X}_{AP}$ e $\overline{X}_{AS}$ são as médias amostrais dos tempos obtidos; S_{AP}^2 e S_{AS}^2 representam o desvio padrão amostral; n_{AP} e n_{AS} representam o tamanho das amostras (neste caso 30).

Para um nível de significância (α) igual a 0.01 (probabilidade de estar correto 99% das vezes que a análise estatística for feita), rejeita-se a hipótese nulidade quando Z ultrapassar o limite fornecido por $Z_{0,01}$, o qual é 2.57. O valor de $Z_{0,01}$ = 2.57 é fornecido pela Tabela de Distribuição Normalizada (ACHCAR, J.A., RODRIGUES, J. 1995). Rejeita-se a hipótese nulidade H0 caso $Z \leq -2.57$, ou então, $Z \geq 2.57$.

Speedup

Fator de redução do tempo de execução de um programa paralelizado em P processadores.

$$S = \frac{T_S}{T_P}$$ Equação 4.2

onde: T_P = tempo de execução do programa paralelo em P processadores

Speedup relativo: T_S = tempo de execução do programa paralelo em 1 processador

Speedup absoluto: T_S = tempo de execução do "melhor" programa seqüencial

Eficiência

Eficiência é a fração do tempo em que os processadores estão ativos. É usada para medir a qualidade de um algoritmo paralelo, é caracteriza como um algoritmo utiliza os recursos de um computador paralelo independentemente do tamanho do problema.

A definição de eficiência é representada pela Equação 4.3 abaixo:

$$E = \frac{S}{P} = \frac{T_S}{PT_P}$$ Equação 4.3

onde: T_P = tempo de execução do programa paralelo em P processadores

Eficiência Relativa: T_S = tempo de execução do programa paralelo em 1 processador

Eficiência absoluta: T_S = tempo de execução do "melhor" programa seqüencial

5.2. Resultados do Filtro de Mediana

Os resultados a seguir são referentes ao algoritmo da mediana. Com a realização dos testes obteve-se uma média dos tempos de processamento tanto para aplicação seqüencial quanto para a paralela.

Os resultados são apresentados em ambiente paralelo formado por 3, 4, 5, 6, 7, 8 e 9 máquinas a fim de verificar o custo benefícios em cada um destes ambientes, os testes de hipótese também são apresentados em cada um destes ambientes.

5.2.1. Imagem de Tamanho 2048x2751

Os resultados apresentados nesta seção são referentes à imagem de 2048x2751 (linha X coluna), como aproximadamente 11MB.

No Gráfico 1 é apresentada a comparação com o algoritmo seqüencial e o paralelo.

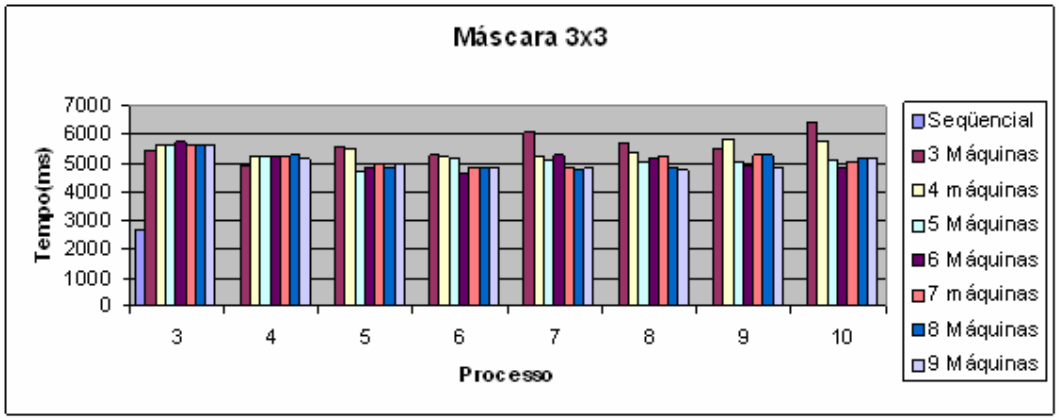

Gráfico 1 - Média do tempo de execução em paralelo do filtro de mediana com máscaras 3X3, imagem de 11MB

Uma vez que a quantidade de cálculos efetuados 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 bloco é maior que o

tempo gasto pelos escravos para realizar o processamento (formação e ordenação do vetor), tornando uma aplicação mais voltada para comunicação do que para processamento.

É possível observar nos resultados demonstrados entre as Tabela 4 a Tabela 9 o tempo de execução do algoritmo seqüencial, quando considerada a vizinhança de tamanho 3x3, é significativamente melhor que o tempo do mesmo em paralelo, independentemente do número de máquinas e da quantidade de processos iniciados em paralelo.

O *speedup* médio obtido é de aproximadamente 0,5 em quase todos os casos, já eficiência que caracteriza como um algoritmo utiliza os recursos do ambiente paralelo teve um resultado abaixo do esperado, sendo quanto mais aumento o número de máquinas menor é eficiência.

Tabela 4. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 3x3 em imagens de 11MB com 3 máquinas

		3 Máquinas							
3x3	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**2678,43**	**5446,50**	**4925,50**	**5545,80**	**5277,80**	**6080,70**	**5679,20**	**5515,70**	**6425,70**
Desvio Padrão	**13,64**	**109,09**	**82,77**	**254,25**	**200,21**	**278,46**	**285,23**	**229,55**	**314,72**
Hipótese a= 0,01		**1368,54**	**1253,45**	**959,54**	**973,58**	**1090,34**	**950,71**	**996,51**	**1132,65**
Speedup		**0,49**	**0,54**	**0,48**	**0,51**	**0,44**	**0,47**	**0,49**	**0,42**
Eficiência		**0,16**	**0,18**	**0,16**	**0,17**	**0,15**	**0,16**	**0,16**	**0,14**

Tabela 5. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 3x3 em imagens de 11MB com 4 máquinas

		4 máquinas							
3x3	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**2678,43**	**5625,60**	**5195,40**	**5504,30**	**5224,30**	**5200,30**	**5353,70**	**5824,00**	**5789,80**
Desvio Padrão	**13,64**	**48,97**	**96,20**	**265,82**	**130,49**	**366,66**	**203,47**	**212,72**	**279,29**
Hipótese a= 0,01		**2040,00**	**1315,39**	**925,88**	**1161,50**	**708,31**	**994,46**	**1145,14**	**995,71**
Speedup		**0,48**	**0,52**	**0,49**	**0,51**	**0,52**	**0,50**	**0,46**	**0,46**
Eficiência		**0,12**	**0,13**	**0,12**	**0,13**	**0,13**	**0,13**	**0,11**	**0,12**

Tabela 6. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 3x3 em imagens de 11MB com 5 máquinas

		5 Máquinas							
3x3	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**2678,43**	**5630,30**	**5215,50**	**4669,30**	**5156,10**	**5070,00**	**5032,90**	**5011,90**	**5073,90**
Desvio Padrão	**13,64**	**164,97**	**91,53**	**73,03**	**94,71**	**96,99**	**221,84**	**178,80**	**197,41**
Hipótese a= 0,01		**1209,78**	**1354,99**	**1171,30**	**1303,74**	**1245,41**	**840,37**	**921,31**	**903,15**
Speedup		**0,48**	**0,51**	**0,57**	**0,52**	**0,53**	**0,53**	**0,53**	**0,53**
Eficiência		**0,08**	**0,09**	**0,10**	**0,09**	**0,09**	**0,09**	**0,09**	**0,09**

Tabela 7. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 3x3 em imagens de 11MB com 6 máquinas

		6 Máquinas							
3x3	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**2678,43**	**5732,80**	**5183,10**	**4853,50**	**4650,00**	**5264,70**	**5158,60**	**4943,90**	**4785,50**
Desvio Padrão	**13,64**	**253,26**	**56,46**	**21,22**	**88,92**	**108,83**	**141,54**	**97,04**	**109,33**
Hipótese a= 0,01		**1024,01**	**1638,54**	**2017,58**	**1066,32**	**1280,01**	**1090,50**	**1179,45**	**1040,73**
Speedup		**0,47**	**0,52**	**0,55**	**0,58**	**0,51**	**0,52**	**0,54**	**0,56**
Eficiência		**0,08**	**0,09**	**0,09**	**0,10**	**0,08**	**0,09**	**0,09**	**0,09**

Tabela 8. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 3x3 em imagens de 11MB com 7 máquinas

		7 Máquinas							
3x3	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**2678,43**	**5633,20**	**5186,30**	**4955,20**	**4835,60**	**4861,70**	**5206,10**	**5253,20**	**5009,50**
Desvio Padrão	**13,64**	**90,67**	**34,16**	**20,84**	**23,79**	**77,97**	**186,25**	**115,01**	**110,99**
Hipótese a= 0,01		**1584,57**	**1986,65**	**2123,75**	**1931,13**	**1249,38**	**979,23**	**1243,36**	**1143,67**
Speedup		**0,48**	**0,52**	**0,54**	**0,55**	**0,55**	**0,51**	**0,51**	**0,53**
Eficiência		**0,07**	**0,07**	**0,08**	**0,08**	**0,08**	**0,07**	**0,07**	**0,08**

Tabela 9. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 3x3 em imagens de 11MB com 8 máquinas

		8 Máquinas							
3x3	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**2678,43**	**5616,10**	**5236,30**	**4871,00**	**4843,60**	**4768,00**	**4826,10**	**5266,20**	**5171,30**
Desvio Padrão	**13,64**	**73,42**	**73,82**	**41,07**	**32,81**	**15,21**	**38,08**	**177,12**	**163,38**
Hipótese a= 0,01		**1724,43**	**1498,02**	**1623,63**	**1740,07**	**2130,90**	**1635,68**	**1026,23**	**1026,22**
Speedup		**0,48**	**0,51**	**0,55**	**0,55**	**0,56**	**0,55**	**0,51**	**0,52**
Eficiência		**0,06**	**0,06**	**0,07**	**0,07**	**0,07**	**0,07**	**0,06**	**0,06**

No Gráfico 2 são apresentados os resultados levando em consideração a máscara 5x5 em imagem de 11Mb. É possível observar que o tempo de execução do algoritmo seqüencial para ambiente paralelo formado por 3 máquinas ainda é superior ao paralelo, mas é possível observar que a partir do uso dessas máscaras o tempo seqüencial é pior que o tempo de execução em paralelo, na maioria dos casos de teste.

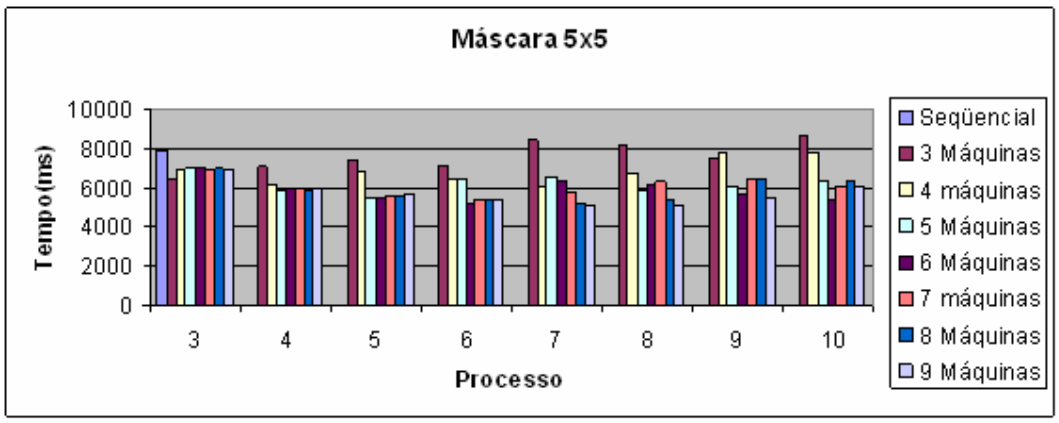

Gráfico 2 - Média do tempo de execução em paralelo do filtro de mediana com máscaras 5X5, imagem de 11MB

São demonstrados nas Tabela 10 a Tabela 15, com maiores detalhes, o desempenho do algoritmo paralelo. É possível observar que o *speedup* obtido é de aproximadamente 1.5 para os melhores casos e o melhor índice de eficiência do algoritmo paralelo é obtido em um ambiente formado por cinco máquinas.

Tabela 10. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 5x5 em imagens de 11MB com 3 máquinas

		3 Máquinas							
5x5	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**7871,03**	**7096,30**	**7356,60**	**7356,60**	**7135,60**	**8438,00**	**8116,80**	**7509,00**	**8667,50**
Desvio Padrão	**16,07**	**95,71**	**585,17**	**585,17**	**579,26**	**566,14**	**631,35**	**450,93**	**524,90**
Hipótese a= 0,01		**-832,60**	**-401,37**	**-114,91**	**-165,09**	**128,70**	**52,90**	**-91,76**	**187,56**
Speedup		**1,11**	**1,07**	**1,07**	**1,10**	**0,93**	**0,97**	**1,05**	**0,91**
Eficiência		**0,37**	**0,36**	**0,36**	**0,37**	**0,31**	**0,32**	**0,35**	**0,30**

Tabela 11. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 5x5 em imagens de 11MB com 4 máquinas

		4 máquinas							
5x5	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**7871,03**	**6130,70**	**6800,50**	**6800,50**	**6396,30**	**6072,10**	**6683,40**	**7723,80**	**7834,60**
Desvio Padrão	**16,07**	**186,50**	**67,76**	**67,76**	**145,05**	**273,10**	**535,19**	**632,99**	**769,66**
Hipótese a= 0,01		**-550,83**	**-669,74**	**-640,44**	**-636,37**	**-579,43**	**-277,06**	**-31,65**	**-7,12**
Speedup		**1,28**	**1,16**	**1,16**	**1,23**	**1,30**	**1,18**	**1,02**	**1,00**
Eficiência		**0,32**	**0,29**	**0,29**	**0,31**	**0,32**	**0,29**	**0,25**	**0,25**

Tabela 12. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 5x5 em imagens de 11MB com 5 máquinas

		5 Máquinas							
5x5	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**7871,03**	**5900,00**	**5478,10**	**5478,10**	**6458,10**	**6596,40**	**5852,40**	**6078,50**	**6321,50**
Desvio Padrão	**16,07**	**32,12**	**90,98**	**90,98**	**271,58**	**135,43**	**322,96**	**451,71**	**282,35**
Hipótese a= 0,01		**-556,60**	**-1555,26**	**-1266,80**	**-456,30**	**-567,21**	**-600,49**	**-453,95**	**-491,31**
Speedup		**1,33**	**1,44**	**1,44**	**1,22**	**1,19**	**1,34**	**1,29**	**1,25**
Eficiência		**0,27**	**0,48**	**0,48**	**0,41**	**0,40**	**0,45**	**0,43**	**0,42**

Tabela 13. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 5x5 em imagens de 11MB com 6 máquinas

		6 Máquinas							
5x5	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**7871,03**	**5943,60**	**5490,67**	**5490,67**	**5171,90**	**6268,33**	**6117,00**	**5650,20**	**5355,90**
Desvio Padrão	**16,07**	**19,34**	**27,57**	**27,57**	**112,78**	**196,85**	**276,50**	**135,66**	**231,53**
Hipótese a= 0,01		**-584,72**	**-1774,21**	**-1973,67**	**-1302,44**	**-601,61**	**-561,68**	**-987,53**	**-875,49**
Speedup		**1,32**	**1,43**	**1,43**	**1,52**	**1,26**	**1,29**	**1,39**	**1,47**
Eficiência		**0,22**	**0,24**	**0,24**	**0,25**	**0,21**	**0,21**	**0,23**	**0,24**

Tabela 14. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 5x5 em imagens de 11MB com 7 máquinas

		7 Máquinas							
5x5	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**7871,03**	**5948,20**	**5587,87**	**5587,87**	**5408,70**	**5674,93**	**6297,73**	**6435,80**	**6033,43**
Desvio Padrão	**16,07**	**14,37**	**29,87**	**29,87**	**26,81**	**104,27**	**71,18**	**166,16**	**158,12**
Hipótese a= 0,01		**-596,70**	**-1908,96**	**-1845,06**	**-2059,68**	**-1096,54**	**-922,57**	**-582,34**	**-762,61**
Speedup		**1,32**	**1,41**	**1,41**	**1,46**	**1,39**	**1,25**	**1,22**	**1,30**
Eficiência		**0,19**	**0,20**	**0,20**	**0,21**	**0,20**	**0,18**	**0,17**	**0,19**

Tabela 15. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 5x5 em imagens de 11MB com 8 máquinas

		8 Máquinas							
5x5	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**7871,03**	**5903,00**	**5507,20**	**5507,20**	**5412,50**	**5221,90**	**5393,07**	**6466,10**	**6305,00**
Desvio Padrão	**16,07**	**65,09**	**42,65**	**42,65**	**15,92**	**34,19**	**80,37**	**216,73**	**156,89**
Hipótese a= 0,01		**-596,75**	**-1196,59**	**-1689,68**	**-2381,00**	**-2046,87**	**-1382,12**	**-504,35**	**-652,22**
Speedup		**1,33**	**1,43**	**1,43**	**1,45**	**1,51**	**1,46**	**1,22**	**1,25**
Eficiência		**0,17**	**0,18**	**0,18**	**0,18**	**0,19**	**0,18**	**0,15**	**0,16**

No Gráfico 3 são apresentados os resultados levando em consideração a máscara 7x7 em imagem de 11Mb. Pelas figuras é possível observar que o tempo de execução do algoritmo seqüencial a partir do uso dessas máscaras é significativamente pior que o tempo de execução em paralelo em todos os casos de teste.

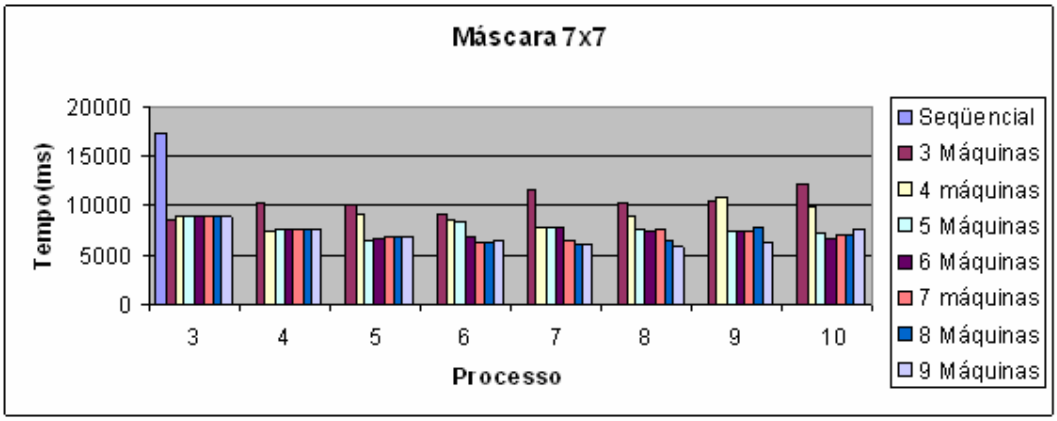

Gráfico 3 - Média do tempo de execução em paralelo do filtro de mediana com máscaras 7x7, imagem de 11MB

São apresentados nas Tabela 16 a Tabela 21, com maiores detalhes, o desempenho do algoritmo paralelo. É possível observar que o *speedup* obtido é de aproximadamente 2.8 para os melhores casos e o melhor índice de eficiência do algoritmo paralelo é obtido em um ambiente formado por três máquinas.

Tabela 16. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 7x7 em imagens de 11MB com 3 máquinas

		3 Máquinas							
7x7	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**17361,50**	**8551,90**	**10273,60**	**10040,70**	**9127,30**	**11626,00**	**10344,90**	**10503,00**	**12194,80**
Desvio Padrão	**23,13**	**99,75**	**78,24**	**942,68**	**1017,70**	**873,85**	**1364,12**	**860,50**	**898,33**
Hipótese a= 0,01		**-4352,98**	**-3855,92**	**-1290,25**	**-1397,95**	**-1048,92**	**-1031,84**	**-1263,73**	**-932,26**
Speedup		**2,03**	**1,69**	**1,73**	**1,90**	**1,49**	**1,68**	**1,65**	**1,42**
Eficiência		**0,68**	**0,56**	**0,58**	**0,63**	**0,50**	**0,56**	**0,55**	**0,47**

Tabela 17. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 7x7 em imagens de 11MB com 4 máquinas

		4 máquinas							
7x7	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**17361,50**	**9001,50**	**7365,00**	**9213,80**	**8436,70**	**7857,43**	**8998,70**	**10794,50**	**9890,60**
Desvio Padrão	**23,13**	**120,19**	**221,72**	**131,41**	**487,70**	**269,60**	**627,71**	**754,35**	**716,66**
Hipótese a= 0,01		**-3824,84**	**-3499,14**	**-3589,83**	**-2162,84**	**-3042,54**	**-1795,47**	**-1289,99**	**-1504,45**
Speedup		**1,93**	**2,36**	**1,88**	**2,06**	**2,21**	**1,93**	**1,61**	**1,76**
Eficiência		**0,48**	**0,59**	**0,47**	**0,51**	**0,55**	**0,48**	**0,40**	**0,44**

Tabela 18. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 7x7 em imagens de 11MB com 5 máquinas

		5 Máquinas							
7x7	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**17361,50**	**9025,80**	**7447,90**	**6528,83**	**8359,50**	**7857,50**	**7487,07**	**7351,20**	**7203,70**
Desvio Padrão	**23,13**	**270,34**	**45,62**	**87,22**	**88,70**	**120,43**	**141,04**	**165,36**	**241,61**
Hipótese a= 0,01		**-2665,17**	**-6549,01**	**-5648,28**	**-4662,56**	**-4344,64**	**-4221,11**	**-3993,61**	**-3419,45**
Speedup		**1,92**	**2,33**	**2,66**	**2,08**	**2,21**	**2,32**	**2,36**	**2,41**
Eficiência		**0,38**	**0,47**	**0,53**	**0,42**	**0,44**	**0,46**	**0,47**	**0,48**

Tabela 19. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 7x7 em imagens de 11MB com 6 máquinas

		6 Máquinas							
7x7	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**17361,50**	**8915,60**	**7499,10**	**6732,70**	**6756,23**	**7906,30**	**7412,80**	**7262,70**	**6705,30**
Desvio Padrão	**23,13**	**57,62**	**22,12**	**5,42**	**73,35**	**148,01**	**77,85**	**218,46**	**166,30**
Hipótese a= 0,01		**-5148,01**	**-8030,23**	**-10895,38**	**-5913,91**	**-3958,73**	**-5422,61**	**-3558,74**	**-4240,74**
Speedup		**1,95**	**2,32**	**2,58**	**2,57**	**2,20**	**2,34**	**2,39**	**2,59**
Eficiência		**0,32**	**0,39**	**0,43**	**0,43**	**0,37**	**0,39**	**0,40**	**0,43**

Tabela 20. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 7x7 em imagens de 11MB com 7 máquinas

		7 Máquinas							
7x7	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**17361,50**	**8925,00**	**7525,90**	**6824,30**	**6371,47**	**6459,20**	**7590,20**	**7251,70**	**6929,80**
Desvio Padrão	**23,13**	**50,01**	**43,90**	**24,13**	**15,54**	**98,90**	**77,09**	**177,15**	**154,10**
Hipótese a= 0,01		**-5403,40**	**-6580,01**	**-8395,35**	**-9680,78**	**-5405,73**	**-5346,26**	**-3912,83**	**-4291,93**
Speedup		**1,95**	**2,31**	**2,54**	**2,72**	**2,69**	**2,29**	**2,39**	**2,51**
Eficiência		**0,28**	**0,33**	**0,36**	**0,39**	**0,38**	**0,33**	**0,34**	**0,36**

Tabela 21. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 7x7 em imagens de 11MB com 8 máquinas

		8 Máquinas							
7x7	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**17361,50**	**8977,00**	**7437,30**	**6764,40**	**6395,30**	**6065,70**	**6463,17**	**7897,00**	**7061,50**
Desvio Padrão	**23,13**	**53,13**	**8,19**	**40,75**	**49,39**	**11,87**	**92,66**	**256,72**	**398,30**
Hipótese a= 0,01		**-5259,01**	**-9713,94**	**-7262,42**	**-7053,17**	**-10457,64**	**-5547,32**	**-3098,82**	**-2748,14**
Speedup		**1,93**	**2,33**	**2,57**	**2,71**	**2,86**	**2,69**	**2,20**	**2,46**
Eficiência		**0,24**	**0,29**	**0,32**	**0,34**	**0,36**	**0,34**	**0,27**	**0,31**

5.2.2. Imagem de Tamanho 2855X3835

Os resultados apresentados nesta seção são referentes à imagem de 2855x3835, como aproximadamente 21MB.

Como ocorreu em imagem de 11MB, pelo Gráfico 4 é possível observar que o tempo de execução do algoritmo seqüencial é significativamente melhor que o tempo do mesmo em paralelo, quando considerada a vizinhança de tamanho 3x3. A complexidade imposta pelo filtro com máscara 3x3 não cobre os gastos com a paralelização e comunicação.

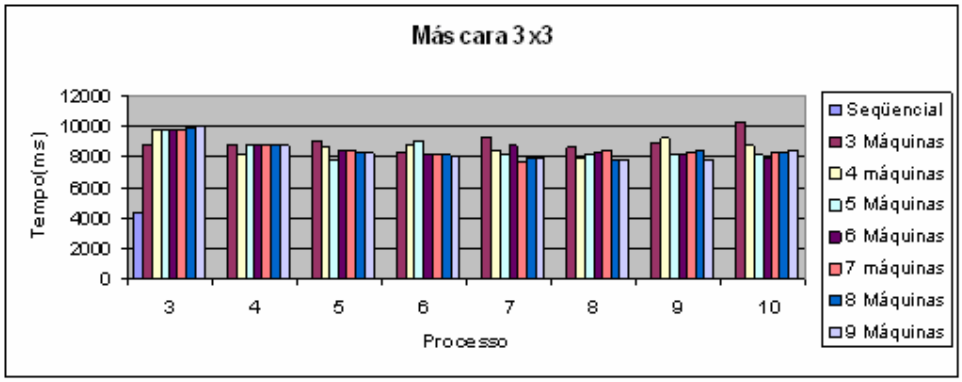

Gráfico 4 - Média do tempo de execução em paralelo do filtro de mediana com máscaras 3X3, imagem de 21MB

São apresentados nas Tabela 19 a Tabela 24, com maiores detalhes, o desempenho do algoritmo paralelo. É possível observar que o speedup obtido é de aproximadamente de 0.55 para os melhores casos e o melhor, sendo este muito abaixo do esperado e o melhor índice de eficiência do algoritmo paralelo é obtido em um ambiente formado por três máquinas.

Tabela 22. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 3x3 em imagens de 21MB com 3 máquinas

		3 Máquinas							
3x3	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**4332,40**	**8866,80**	**8791,80**	**9053,50**	**8239,60**	**9299,40**	**8667,60**	**8939,90**	**10276,40**
Desvio Padrão	**35,24**	**208,20**	**365,05**	**1116,24**	**466,75**	**261,50**	**495,23**	**449,48**	**213,42**
Hipótese a= 0,01		**1591,77**	**1220,81**	**762,03**	**955,16**	**1579,29**	**1030,95**	**1146,25**	**2064,60**
Speedup		**0,49**	**0,49**	**0,48**	**0,53**	**0,47**	**0,50**	**0,48**	**0,42**
Eficiência		**0,16**	**0,16**	**0,16**	**0,18**	**0,16**	**0,17**	**0,16**	**0,14**

Tabela 23. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 3x3 em imagens de 21MB com 4 máquinas

		4 Máquinas							
3x3	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**4332,40**	**9770,90**	**8092,40**	**8636,00**	**8684,30**	**8385,90**	**7900,80**	**9267,20**	**8734,00**
Desvio Padrão	**35,24**	**128,37**	**295,77**	**164,30**	**515,39**	**451,10**	**460,39**	**326,95**	**347,66**
Hipótese a= 0,01		**2328,79**	**1131,95**	**1668,69**	**1015,80**	**1006,74**	**877,92**	**1420,23**	**1232,04**
Speedup		**0,44**	**0,54**	**0,50**	**0,50**	**0,52**	**0,55**	**0,47**	**0,50**
Eficiência		**0,11**	**0,13**	**0,13**	**0,12**	**0,13**	**0,14**	**0,12**	**0,12**

Tabela 24. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 3x3 em imagens de 21MB com 5 máquinas

		5 Máquinas							
3x3	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**4332,40**	**9779,00**	**8713,60**	**7872,80**	**9056,50**	**8163,70**	**8115,90**	**8156,30**	**8109,50**
Desvio Padrão	**35,24**	**195,10**	**143,80**	**354,07**	**276,82**	**332,47**	**305,73**	**488,45**	**360,96**
Hipótese a= 0,01		**1965,60**	**1793,41**	**982,79**	**1464,72**	**1094,35**	**1122,27**	**915,23**	**1039,34**
Speedup		**0,44**	**0,50**	**0,55**	**0,48**	**0,53**	**0,53**	**0,53**	**0,53**
Eficiência		**0,09**	**0,10**	**0,11**	**0,10**	**0,11**	**0,11**	**0,11**	**0,11**

Tabela 25. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 3x3 em imagens de 21MB com 6 máquinas

		6 Máquinas							
3x3	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**4332,40**	**9818,00**	**8769,20**	**8389,40**	**8160,90**	**8690,20**	**8247,20**	**8101,60**	**7906,50**
Desvio Padrão	**35,24**	**365,02**	**97,45**	**89,08**	**91,38**	**427,85**	**293,03**	**252,51**	**250,00**
Hipótese a= 0,01		**1501,79**	**2109,61**	**1992,89**	**1863,49**	**1109,16**	**1183,46**	**1217,02**	**1159,09**
Speedup		**0,44**	**0,49**	**0,52**	**0,53**	**0,50**	**0,53**	**0,53**	**0,55**
Eficiência		**0,07**	**0,08**	**0,09**	**0,09**	**0,08**	**0,09**	**0,09**	**0,09**

Tabela 26. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 3x3 em imagens de 21MB com 7 máquinas

		7 Máquinas							
3x3	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**4332,40**	**9788,20**	**8775,30**	**8417,70**	**8134,00**	**7674,50**	**8439,10**	**8268,70**	**8275,70**
Desvio Padrão	**35,24**	**316,96**	**122,04**	**43,07**	**98,70**	**221,38**	**237,77**	**338,32**	**192,24**
Hipótese a= 0,01		**1592,30**	**1940,39**	**2528,48**	**1799,12**	**1142,70**	**1361,32**	**1115,50**	**1432,00**
Speedup		**0,44**	**0,49**	**0,51**	**0,53**	**0,56**	**0,51**	**0,52**	**0,52**
Eficiência		**0,06**	**0,07**	**0,07**	**0,08**	**0,08**	**0,07**	**0,07**	**0,07**

Tabela 27. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 3x3 em imagens de 21MB com 8 máquinas

		8 Máquinas							
3x3	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**4332,40**	**9923,90**	**8839,70**	**8342,60**	**8142,60**	**7956,80**	**7863,40**	**8431,00**	**8338,40**
Desvio Padrão	**35,24**	**278,25**	**131,84**	**122,43**	**118,21**	**88,78**	**159,82**	**440,21**	**239,49**
Hipótese a= 0,01		**1729,71**	**1909,87**	**1749,24**	**1684,70**	**1782,53**	**1384,75**	**1029,53**	**1323,78**
Speedup		**0,44**	**0,49**	**0,52**	**0,53**	**0,54**	**0,55**	**0,51**	**0,52**
Eficiência		**0,05**	**0,06**	**0,06**	**0,07**	**0,07**	**0,07**	**0,06**	**0,06**

No Gráfico 5 são apresentados os resultados levando em consideração a máscara 5x5 em imagem de 21Mb. Pelas figuras é possível observar que há uma significativa melhora de desempenho, quando se faz uso de uma arquitetura paralela distribuída, é possível observar também que para o ambiente paralelo formado por três máquinas que uma perde de desempenho quando se aumenta o numero de processos.

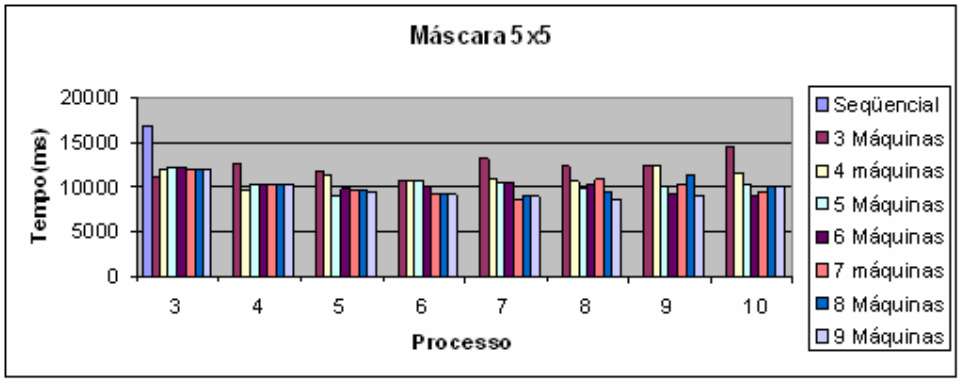

Gráfico 5 - Média do tempo de execução em paralelo do filtro de mediana com máscaras 5X5, imagem de 21MB

São apresentados nas Tabela 28 a Tabela 33, com maiores detalhes, o desempenho do algoritmo paralelo. É possível observar que os *speedup* mais significativos são de aproximadamente 1.95 para os melhores casos, e o melhor índice de eficiência do algoritmo paralelo é obtido em um ambiente formado por seis máquinas.

Tabela 28. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 5x5 em imagens de 21MB com 3 máquinas

		3 Máquinas							
5x5	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**16842,00**	**12545,40**	**11830,30**	**11830,30**	**10599,10**	**13034,70**	**12186,20**	**12397,20**	**14407,90**
Desvio Padrão	**51,42**	**141,67**	**861,55**	**861,55**	**859,84**	**805,72**	**657,56**	**1384,04**	**1342,88**
Hipótese a= 0,01		**-2569,25**	**-1693,58**	**-908,48**	**-1132,73**	**-712,28**	**-957,72**	**-642,57**	**-357,04**
Speedup		**1,34**	**1,42**	**1,42**	**1,59**	**1,29**	**1,38**	**1,36**	**1,17**
Eficiência		**0,45**	**0,47**	**0,47**	**0,53**	**0,43**	**0,46**	**0,45**	**0,39**

Tabela 29. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 5x5 em imagens de 21MB com 4 máquinas

		4 Máquinas							
5x5	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**16842,00**	**9547,70**	**11221,10**	**11221,10**	**10561,50**	**10861,50**	**10590,40**	**12412,20**	**11597,60**
Desvio Padrão	**51,42**	**110,85**	**111,27**	**111,27**	**486,34**	**904,53**	**744,86**	**709,75**	**909,92**
Hipótese a= 0,01		**-3040,40**	**-3136,37**	**-2413,76**	**-1483,42**	**-1059,45**	**-1213,44**	**-879,44**	**-926,44**
Speedup		**1,76**	**1,50**	**1,50**	**1,59**	**1,55**	**1,59**	**1,36**	**1,45**
Eficiência		**0,44**	**0,38**	**0,38**	**0,40**	**0,39**	**0,40**	**0,34**	**0,36**

Tabela 30. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 5x5 em imagens de 21MB com 5 máquinas

		5 Máquinas							
5x5	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**16842,00**	**10180,60**	**8884,70**	**8884,70**	**10520,50**	**10459,10**	**9825,50**	**10156,20**	**10282,10**
Desvio Padrão	**51,42**	**82,70**	**154,00**	**154,00**	**115,81**	**631,68**	**433,58**	**848,09**	**846,79**
Hipótese a= 0,01		**-2269,42**	**-3150,59**	**-3040,97**	**-2677,51**	**-1337,63**	**-1745,07**	**-1220,99**	**-1198,86**
Speedup		**1,65**	**1,90**	**1,90**	**1,60**	**1,61**	**1,71**	**1,66**	**1,64**
Eficiência		**0,33**	**0,38**	**0,38**	**0,32**	**0,32**	**0,34**	**0,33**	**0,33**

Tabela 31. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 5x5 em imagens de 21MB com 6 máquinas

		6 Máquinas							
5x5	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**16842,00**	**10293,50**	**9703,30**	**9703,30**	**9959,60**	**10476,20**	**10318,70**	**9287,80**	**8879,70**
Desvio Padrão	**51,42**	**75,04**	**256,93**	**256,93**	**415,70**	**553,76**	**288,58**	**287,45**	**223,98**
Hipótese a= 0,01		**-1470,20**	**-3189,56**	**-2226,68**	**-1744,17**	**-1417,34**	**-1937,71**	**-2247,68**	**-2627,95**
Speedup		**1,64**	**1,74**	**1,74**	**1,69**	**1,61**	**1,63**	**1,81**	**1,90**
Eficiência		**0,27**	**0,58**	**0,58**	**0,56**	**0,54**	**0,54**	**0,60**	**0,63**

Tabela 32. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 5x5 em imagens de 21MB com 7 máquinas

		7 Máquinas							
5x5	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**16842,00**	**10312,20**	**9592,90**	**9592,90**	**9151,40**	**8624,00**	**10768,40**	**10211,50**	**9383,50**
Desvio Padrão	**51,42**	**47,13**	**79,93**	**79,93**	**76,95**	**472,49**	**703,45**	**462,16**	**704,32**
Hipótese a= 0,01		**-3037,58**	**-3602,73**	**-3464,52**	**-3717,86**	**-1966,52**	**-1210,80**	**-1602,53**	**-1486,03**
Speedup		**1,63**	**1,76**	**1,76**	**1,84**	**1,95**	**1,56**	**1,65**	**1,79**
Eficiência		**0,23**	**0,25**	**0,25**	**0,26**	**0,28**	**0,22**	**0,24**	**0,26**

Tabela 33. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 5x5 em imagens de 21MB com 8 máquinas

		8 Máquinas							
5x5	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**16842,00**	**10311,00**	**9622,50**	**9622,50**	**9124,10**	**8876,40**	**9429,70**	**11191,90**	**10130,40**
Desvio Padrão	**51,42**	**78,45**	**98,79**	**98,79**	**66,03**	**89,19**	**438,05**	**441,66**	**440,35**
Hipótese a= 0,01		**-2385,76**	**-3139,02**	**-3226,47**	**-3900,63**	**-3679,34**	**-1835,07**	**-1393,67**	**-1657,70**
Speedup		**1,63**	**1,75**	**1,75**	**1,85**	**1,90**	**1,79**	**1,50**	**1,66**
Eficiência		**0,20**	**0,22**	**0,22**	**0,23**	**0,24**	**0,22**	**0,19**	**0,21**

No Gráfico 6 são apresentados os resultados levando em consideração a máscara 7x7 em imagens de 21Mb. É possível observar que a aplicação paralelo demonstrou um desempenho superior e estável, independente do numero de processos e maquinas.

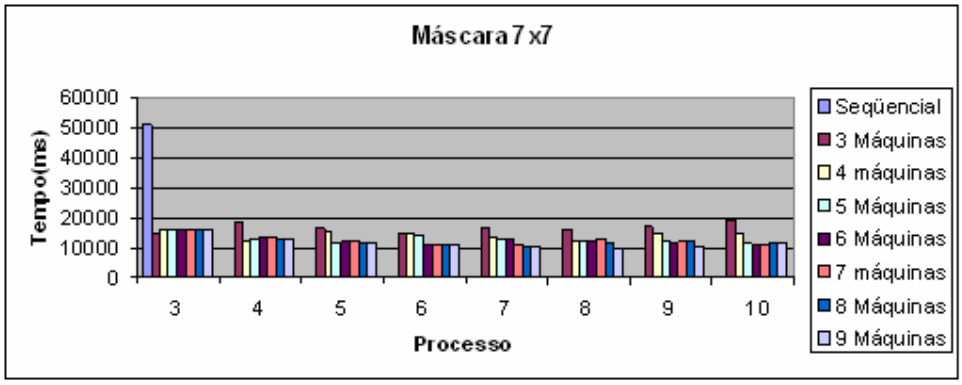

Gráfico 6 - Média do tempo de execução em paralelo do filtro de mediana com máscaras 7x7, imagem de 21MB

São apresentados nas Tabela 34 a Tabela 39, com maiores detalhes, o desempenho do algoritmo paralelo. É possível observar que foi possível obter *speedup* superiores 4.0 próximos a 5.0 para os melhores casos, e o melhor índice de eficiência do algoritmo paralelo é obtido em um ambiente formado por seis máquinas.

Tabela 34. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 7x7 em imagens de 21MB com 3 máquinas

		3 Máquinas							
7x7	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**51107,00**	**14838,70**	**18271,80**	**16563,60**	**14398,50**	**16618,80**	**16192,70**	**17013,50**	**18807,10**
Desvio Padrão	**100,09**	**109,60**	**177,71**	**711,86**	**1188,70**	**937,81**	**1308,40**	**1509,47**	**2477,44**
Hipótese a= 0,01		**-13718,17**	**-10790,24**	**-6639,89**	**-5600,62**	**-5863,45**	**-5095,49**	**-4654,55**	**-3484,65**
Speedup		**3,44**	**2,80**	**3,09**	**3,55**	**3,08**	**3,16**	**3,00**	**2,72**
Eficiência		**1,15**	**0,93**	**1,03**	**1,18**	**1,03**	**1,05**	**1,00**	**0,91**

Tabela 35. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 7x7 em imagens de 21MB com 4 máquinas

		4 Máquinas							
7x7	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**51107,00**	**15769,70**	**12362,10**	**15705,70**	**14272,10**	**13534,80**	**12263,30**	**15223,20**	**14868,10**
Desvio Padrão	**100,09**	**38,06**	**106,23**	**185,26**	**587,61**	**1065,06**	**685,58**	**1091,97**	**1225,58**
Hipótese a= 0,01		**-16466,92**	**-14774,19**	**-11478,50**	**-7693,40**	**-6028,86**	**-7590,34**	**-5692,58**	**-5451,51**
Speedup		**3,24**	**4,13**	**3,25**	**3,58**	**3,78**	**4,17**	**3,36**	**3,44**
Eficiência		**0,81**	**1,03**	**0,81**	**0,90**	**0,94**	**1,04**	**0,84**	**0,86**

Tabela 36. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 7x7 em imagens de 21MB com 5 máquinas

		5 Máquinas							
7x7	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**51107,00**	**15823,30**	**13080,00**	**11596,90**	**14206,70**	**13231,00**	**12538,00**	**11972,50**	**11461,10**
Desvio Padrão	**100,09**	**121,94**	**93,90**	**227,39**	**135,41**	**565,58**	**583,51**	**525,22**	**1323,95**
Hipótese a= 0,01		**-12969,59**	**-14954,15**	**-11958,49**	**-13170,28**	**-8040,68**	**-8079,76**	**-8571,81**	**-5754,35**
Speedup		**3,23**	**3,91**	**4,41**	**3,60**	**3,86**	**4,08**	**4,27**	**4,46**
Eficiência		**0,65**	**0,78**	**0,88**	**0,72**	**0,77**	**0,82**	**0,85**	**0,89**

Tabela 37. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 7x7 em imagens de 21MB com 6 máquinas

		6 Máquinas							
7x7	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**51107,00**	**15786,30**	**13279,90**	**11785,30**	**11172,00**	**13209,90**	**12207,60**	**11506,10**	**11077,70**
Desvio Padrão	**100,09**	**75,71**	**246,57**	**180,90**	**89,39**	**84,47**	**111,06**	**169,86**	**404,32**
Hipótese a= 0,01		**-14590,87**	**-11127,83**	**-12848,37**	**-15890,13**	**-15279,09**	**-14662,34**	**-13201,51**	**-9762,13**
Speedup		**3,24**	**3,85**	**4,34**	**4,57**	**3,87**	**4,19**	**4,44**	**4,61**
Eficiência		**0,54**	**0,64**	**0,72**	**0,76**	**0,64**	**0,70**	**0,74**	**0,77**

Tabela 38. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 7x7 em imagens de 21MB com 7 máquinas

		7 Máquinas							
7x7	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**51107,00**	**15953,10**	**13268,20**	**11767,50**	**10862,60**	**10910,10**	**12777,60**	**11914,60**	**11033,20**
Desvio Padrão	**100,09**	**526,61**	**144,08**	**104,09**	**9,96**	**263,75**	**551,32**	**620,25**	**190,33**
Hipótese a= 0,01		**-7691,34**	**-13263,24**	**-15079,29**	**-21011,68**	**-11542,43**	**-8225,52**	**-7998,18**	**-12879,73**
Speedup		**3,20**	**3,85**	**4,34**	**4,70**	**4,68**	**4,00**	**4,29**	**4,63**
Eficiência		**0,46**	**0,55**	**0,62**	**0,67**	**0,67**	**0,57**	**0,61**	**0,66**

Tabela 39. Teste hipótese, *Speedup* e Eficiência filtro da mediana para máscara 7x7 em imagens de 21MB com 8 máquinas

		8 Máquinas							
7x7	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**51107,00**	**15785,40**	**13202,10**	**11556,00**	**10876,70**	**10357,60**	**11644,60**	**12360,60**	**11661,80**
Desvio Padrão	**100,09**	**78,80**	**75,87**	**286,43**	**52,38**	**86,57**	**605,48**	**804,09**	**820,93**
Hipótese a= 0,01		**-14464,57**	**-15651,09**	**-11018,76**	**-17845,05**	**-16336,28**	**-8137,15**	**-7057,71**	**-7118,99**
Speedup		**3,24**	**3,87**	**4,42**	**4,70**	**4,93**	**4,39**	**4,13**	**4,38**
Eficiência		**0,40**	**0,48**	**0,55**	**0,59**	**0,62**	**0,55**	**0,52**	**0,55**

5.3. Resultados Filtro de Detecção de Borda

Os resultados a seguir são referentes ao algoritmo de detecção de borda utilizando o operador de Sobel, com a realização dos testes obteve-se uma média dos tempos de processamento tanto para aplicação seqüencial quanto para a paralela.

Os resultados são apresentados em ambiente paralelo formado por 3, 4, 5, 6, 7, 8 e 9 máquinas a fim de verificar o custo benefícios em cada um destes ambientes, os testes de hipótese também são apresentados em cada um destes ambientes.

5.3.1. Imagem de Tamanho 2751x2021

Os resultados apresentados nesta seção são referentes à imagem de 2751x2001, com aproximadamente 11MB.

No Gráfico 7 são apresentados os resultados levando em consideração a máscara 9x9 em imagens de 11Mb. É possível observar que em ambiente paralelo formado por três máquinas utilizando máscara de tamanho 9x9, os resultados do algoritmo paralelo comparado ao seqüencial para maioria dos processos são superiores, em um ambiente formado por quatro máquinas o algoritmo paralelo apresenta para alguns processos um tempo muito próximos ao tempo seqüencial. A partir de cinco processos o tempo do algoritmo paralelo é sempre melhor ao seqüencial.

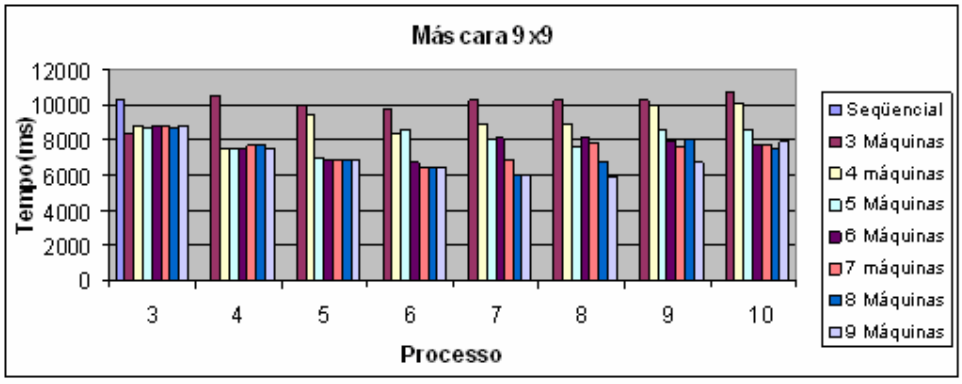

Gráfico 7 - Média do tempo de execução em paralelo do filtro de detecção de borda com máscaras 9x9, imagem de 11MB

São apresentados nas Tabela 40 a Tabela 45, com maiores detalhes, o desempenho do algoritmo paralelo. É possível observar que os *speedups* mais significativo são de aproximadamente 1.60 para os melhores casos, e o melhor índice de eficiência do algoritmo paralelo é obtido em um ambiente formado por três máquinas.

Tabela 40. Teste hipótese, *Speedup* e Eficiência filtro de detecção de borda para máscara 9x9 em imagens de 11MB com 3 máquinas

		3 Máquinas							
9x9	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**10259,10**	**8332,27**	**10547,67**	**10005,53**	**9725,20**	**10310,67**	**10301,13**	**10313,60**	**10821,53**
Desvio Padrão	**38,42**	**56,12**	**148,57**	**491,32**	**529,74**	**328,74**	**673,35**	**870,58**	**663,23**
Hipótese a= 0,01		**-1085,44**	**115,59**	**-60,34**	**-122,68**	**14,74**	**8,63**	**9,90**	**116,30**
Speedup		**1,23**	**0,97**	**1,03**	**1,05**	**0,99**	**1,00**	**0,99**	**0,95**
Eficiência		**0,41**	**0,32**	**0,34**	**0,35**	**0,33**	**0,33**	**0,33**	**0,32**

Tabela 41. Teste hipótese, *Speedup* e Eficiência filtro de detecção de borda para máscara 9x9 em imagens de 11MB com 4 máquinas

		4 Máquinas							
9x9	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**10259,10**	**8790,20**	**7499,33**	**9425,00**	**8407,40**	**8897,27**	**8872,93**	**9979,53**	**10107,67**
Desvio Padrão	**38,42**	**171,14**	**110,97**	**94,44**	**152,91**	**415,07**	**510,60**	**726,81**	**859,63**
Hipótese a= 0,01		**-555,77**	**-1236,74**	**-396,35**	**-733,23**	**-350,27**	**-324,03**	**-55,35**	**-27,68**
Speedup		**1,17**	**1,37**	**1,09**	**1,22**	**1,15**	**1,16**	**1,03**	**1,01**
Eficiência		**0,29**	**0,34**	**0,27**	**0,31**	**0,29**	**0,29**	**0,26**	**0,25**

Tabela 42. Teste hipótese, *Speedup* e Eficiência filtro de detecção de borda para máscara 9x9 em imagens de 11MB com 5 máquinas

		5 Máquinas							
9x9	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**10259,10**	**8737,07**	**7520,80**	**7002,80**	**8619,07**	**8071,53**	**7580,73**	**8515,80**	**8550,33**
Desvio Padrão	**38,42**	**32,51**	**88,05**	**105,84**	**137,83**	**173,42**	**97,22**	**465,92**	**791,39**
Hipótese a= 0,01		**-989,89**	**-1333,66**	**-1484,96**	**-676,62**	**-823,23**	**-1259,65**	**-425,18**	**-324,90**
Speedup		**1,17**	**1,36**	**1,46**	**1,19**	**1,27**	**1,35**	**1,20**	**1,20**
Eficiência		**0,20**	**0,23**	**0,24**	**0,20**	**0,21**	**0,23**	**0,20**	**0,20**

Tabela 43. Teste hipótese, *Speedup* e Eficiência filtro de detecção de borda para máscara 9x9 em imagens de 11MB com 6 máquinas

		6 Máquinas							
9x9	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**10259,10**	**8803,07**	**7526,07**	**6856,93**	**6756,00**	**8088,47**	**8133,47**	**7926,00**	**7788,47**
Desvio Padrão	**38,42**	**143,63**	**72,00**	**63,42**	**208,94**	**111,45**	**649,26**	**720,49**	**744,72**
Hipótese a= 0,01		**-591,08**	**-1424,57**	**-1846,54**	**-1219,97**	**-971,16**	**-443,97**	**-463,87**	**-483,56**
Speedup		**1,17**	**1,36**	**1,50**	**1,52**	**1,27**	**1,26**	**1,29**	**1,32**
Eficiência		**0,19**	**0,23**	**0,25**	**0,25**	**0,21**	**0,21**	**0,22**	**0,22**

Tabela 44. Teste hipótese, *Speedup* e Eficiência filtro de detecção de borda para máscara 9x9 em imagens de 11MB com 7 máquinas

		7 Máquinas							
9x9	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**10259,10**	**8842,53**	**7716,33**	**6904,13**	**6371,40**	**6907,33**	**7861,20**	**7638,33**	**7730,07**
Desvio Padrão	**38,42**	**120,81**	**368,65**	**109,31**	**108,65**	**467,26**	**328,31**	**681,83**	**779,89**
Hipótese a= 0,01		**-614,89**	**-690,30**	**-1511,91**	**-1755,89**	**-816,39**	**-685,84**	**-534,87**	**-484,24**
Speedup		**1,16**	**1,33**	**1,49**	**1,61**	**1,49**	**1,31**	**1,34**	**1,33**
Eficiência		**0,17**	**0,19**	**0,21**	**0,23**	**0,21**	**0,19**	**0,19**	**0,19**

Tabela 45. Teste hipótese, *Speedup* e Eficiência filtro de detecção de borda para máscara 9x9 em imagens de 11MB com 8 máquinas

		8 Máquinas							
9x9	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**10259,10**	**8725,20**	**7688,40**	**6860,60**	**6355,53**	**6003,40**	**6774,87**	**8070,73**	**7477,27**
Desvio Padrão	**38,42**	**58,07**	**775,25**	**67,73**	**80,04**	**34,55**	**217,12**	**129,08**	**414,21**
Hipótese a= 0,01		**-855,33**	**-493,62**	**-1806,73**	**-1964,49**	**-2728,84**	**-1193,82**	**-926,14**	**-716,18**
Speedup		**1,18**	**1,33**	**1,50**	**1,61**	**1,71**	**1,51**	**1,27**	**1,37**
Eficiência		**0,15**	**0,17**	**0,19**	**0,20**	**0,21**	**0,19**	**0,16**	**0,17**

No Gráfico 8 são apresentados os resultados levando em consideração a máscara 11x11 em imagens de 11Mb. É possível observar que o tempo de execução do algoritmo paralelo é superior ao seqüencial. É possível observar que a partir do aumento no numero de processos os resultados variam de forma mais expressiva.

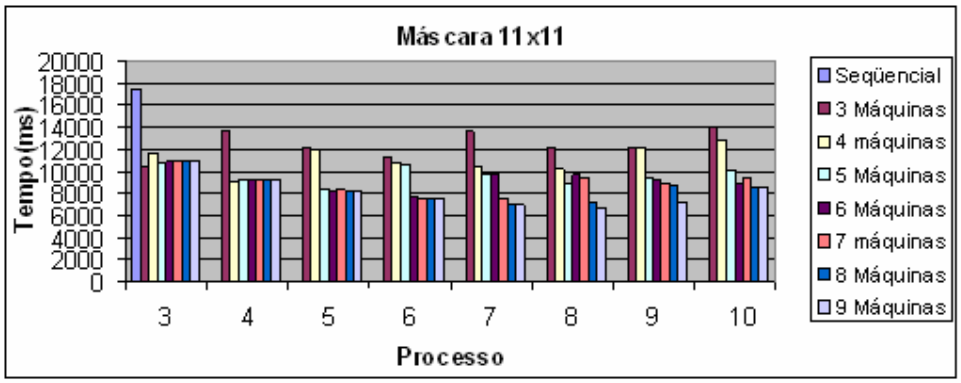

Gráfico 8 - Média do tempo de execução em paralelo do filtro de detecção de borda com máscaras 11x11, imagem de 11MB

São demonstrados nas Tabela 46 a Tabela 51 com maiores detalhes o desempenho do algoritmo paralelo. É possível observar que os melhores *speedups* obtidos ficam aproximadamente entre 1.35 a 1.50 para os melhores casos, e o melhor índice de eficiência do algoritmo paralelo é obtido em um ambiente formado por cinco máquinas.

Tabela 46. Teste hipótese, *Speedup* e Eficiência filtro de detecção de borda para máscara 11x11 em imagens de 11MB com 3 máquinas

		3 Máquinas							
11x11	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**17449,67**	**10406,53**	**13577,87**	**12172,20**	**11341,87**	**13582,93**	**12118,40**	**12124,73**	**13981,93**
Desvio Padrão	**27,60**	**120,47**	**136,80**	**469,83**	**911,61**	**741,77**	**718,02**	**744,26**	**888,57**
Hipótese a= 0,01		**-3170,32**	**-1653,96**	**-1296,05**	**-1091,60**	**-763,55**	**-1069,38**	**-1049,80**	**-627,51**
Speedup		**1,68**	**1,29**	**0,84**	**0,90**	**0,76**	**0,85**	**0,85**	**0,73**
Eficiência		**0,56**	**0,43**	**0,28**	**0,30**	**0,25**	**0,28**	**0,28**	**0,24**

Tabela 47. Teste hipótese, *Speedup* e Eficiência filtro de detecção de borda para máscara 11x11 em imagens de 11MB com 4 máquinas

		4 Máquinas							
11x11	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**17449,67**	**9037,13**	**12004,00**	**12004,00**	**10725,53**	**10394,13**	**10261,60**	**12176,53**	**12840,33**
Desvio Padrão	**27,60**	**112,47**	**193,50**	**193,50**	**816,33**	**712,32**	**969,51**	**693,01**	**884,97**
Hipótese a= 0,01		**-744,59**	**-3893,37**	**-2005,95**	**-1267,79**	**-1420,69**	**-1246,82**	**-1075,92**	**-835,73**
Speedup		**1,14**	**0,85**	**0,85**	**0,96**	**0,99**	**1,00**	**0,84**	**0,80**
Eficiência		**0,28**	**0,21**	**0,21**	**0,24**	**0,25**	**0,25**	**0,21**	**0,20**

Tabela 48. Teste hipótese, *Speedup* e Eficiência filtro de detecção de borda para máscara 11x11 em imagens de 11MB com 5 máquinas

		5 Máquinas							
11x11	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**17449,67**	**9102,20**	**8300,40**	**8300,40**	**10577,73**	**9627,87**	**8799,80**	**9337,20**	**10006,67**
Desvio Padrão	**27,60**	**127,11**	**93,65**	**93,65**	**65,67**	**129,28**	**250,62**	**589,62**	**915,02**
Hipótese a= 0,01		**-3899,92**	**-3675,89**	**-4551,01**	**-3897,37**	**-3420,53**	**-2840,42**	**-1788,52**	**-1327,82**
Speedup		**1,13**	**1,24**	**1,24**	**0,97**	**1,07**	**1,17**	**1,10**	**1,03**
Eficiência		**0,23**	**0,41**	**0,41**	**0,32**	**0,36**	**0,39**	**0,37**	**0,34**

Tabela 49. Teste hipótese, *Speedup* e Eficiência filtro de detecção de borda para máscara 11x11 em imagens de 11MB com 6 máquinas

		6 Máquinas							
11x11	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**17449,67**	**9065,00**	**8143,40**	**8143,40**	**7645,13**	**9725,73**	**9671,67**	**9197,47**	**8859,93**
Desvio Padrão	**27,60**	**75,22**	**122,85**	**122,85**	**59,91**	**68,64**	**812,39**	**703,00**	**805,64**
Hipótese a= 0,01		**-3010,79**	**-4529,22**	**-4155,78**	**-5740,89**	**-4312,58**	**-1469,92**	**-1672,22**	**-1629,89**
Speedup		**1,13**	**1,26**	**1,26**	**1,34**	**1,05**	**1,06**	**1,12**	**1,16**
Eficiência		**0,19**	**0,21**	**0,21**	**0,22**	**0,18**	**0,18**	**0,19**	**0,19**

Tabela 50. Teste hipótese, *Speedup* e Eficiência filtro de detecção de borda para máscara 11x11 em imagens de 11MB com 7 máquinas

		7 Máquinas							
11x11	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**17449,67**	**9235,47**	**8285,93**	**8285,93**	**7496,73**	**7515,60**	**9373,40**	**8901,93**	**9355,20**
Desvio Padrão	**27,60**	**245,60**	**412,01**	**412,01**	**108,83**	**242,07**	**200,16**	**767,42**	**782,41**
Hipótese a= 0,01		**-3031,06**	**-2722,01**	**-2393,87**	**-4667,26**	**-3313,41**	**-2931,11**	**-1660,44**	**-1557,77**
Speedup		**1,11**	**1,24**	**1,24**	**1,37**	**1,37**	**1,09**	**1,15**	**1,10**
Eficiência		**0,16**	**0,18**	**0,18**	**0,20**	**0,20**	**0,16**	**0,16**	**0,16**

Tabela 51. Teste hipótese, *Speedup* e Eficiência filtro de detecção de borda para máscara 11x11 em imagens de 11MB com 8 máquinas

		8 Máquinas							
11x11	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**17449,67**	**9128,33**	**8137,33**	**8137,33**	**7423,33**	**6949,53**	**7034,73**	**8690,93**	**8402,73**
Desvio Padrão	**27,60**	**204,19**	**93,42**	**93,42**	**92,71**	**85,49**	**309,91**	**127,89**	**454,36**
Hipótese a= 0,01		**-2466,26**	**-2993,69**	**-4636,48**	**-5006,82**	**-5408,27**	**-3105,12**	**-3847,30**	**-2257,13**
Speedup		**1,12**	**1,26**	**1,26**	**1,38**	**1,48**	**1,46**	**1,18**	**1,22**
Eficiência		**0,14**	**0,16**	**0,16**	**0,17**	**0,18**	**0,18**	**0,15**	**0,15**

5.3.2. Imagem de Tamanho 2751x2321

Os resultados apresentando nesta seção são referente a imagem de 2751x2001, como aproximadamente 21MB.

No Gráfico 9 são apresentados os resultados levando em consideração a máscara 9x9 em imagem de 21Mb. É possível observar que em ambiente paralelo formado por três máquinas os resultados do algoritmo paralelo são muito próximos aos resultados do algoritmo seqüencial, a partir de um ambiente paralelo formado por quatro máquinas o algoritmo paralelo apresenta um ganho considerável de desempenho.

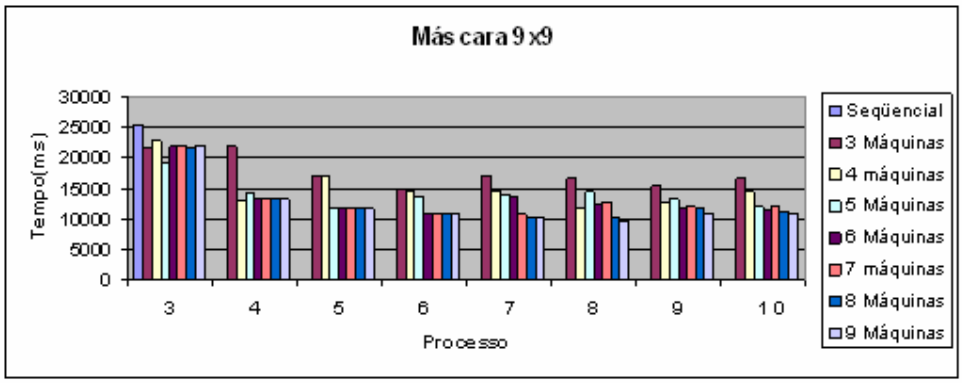

Gráfico 9 - Média do tempo de execução em paralelo do filtro de detecção de borda com máscaras 9x9, imagem de 21MB

São apresentados nas Tabela 52 a Tabela 57, com maiores detalhes, o desempenho do algoritmo paralelo. É possível observar que o maior *speedup* obtido é de 2.50, e o melhor índice de eficiência do algoritmo paralelo é obtido em um ambiente formado por três máquinas.

Tabela 52. Teste hipótese, *Speedup* e Eficiência filtro de detecção de borda para máscara 9x9 em imagens de 21MB com 3 máquinas

		3 Máquinas							
9x9	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**25270,40**	**21450,55**	**21892,25**	**17108,55**	**14807,08**	**17122,87**	**16581,62**	**15311,80**	**16612,40**
Desvio Padrão	**10,95**	**148,38**	**1380,97**	**449,54**	**325,62**	**351,02**	**339,95**	**190,99**	**596,81**
Hipótese a= 0,01		**-1657,52**	**-495,94**	**-2083,23**	**-3123,88**	**-2345,58**	**-2540,55**	**-3838,38**	**-1923,58**
Speedup		**1,18**	**1,15**	**1,48**	**1,71**	**1,48**	**1,52**	**1,65**	**1,52**
Eficiência		**0,39**	**0,38**	**0,49**	**0,57**	**0,49**	**0,51**	**0,55**	**0,51**

Tabela 53. Teste hipótese, *Speedup* e Eficiência filtro de detecção de borda para máscara 9x9 em imagens de 21MB com 4 máquinas

		4 Máquinas							
9x9	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**25270,40**	**22692,75**	**12977,75**	**16818,00**	**14525,80**	**14408,20**	**11614,75**	**12577,33**	**14393,17**
Desvio Padrão	**10,95**	**287,69**	**106,03**	**582,85**	**458,36**	**347,02**	**255,43**	**426,53**	**388,25**
Hipótese a= 0,01		**-816,98**	**-6225,05**	**-1899,85**	**-2716,57**	**-3144,53**	**-4582,70**	**-3323,89**	**-2981,82**
Speedup		**1,11**	**1,95**	**1,50**	**1,74**	**1,75**	**2,18**	**2,01**	**1,76**
Eficiência		**0,28**	**0,49**	**0,38**	**0,43**	**0,44**	**0,54**	**0,50**	**0,44**

Tabela 54. Teste hipótese, *Speedup* e Eficiência filtro de detecção de borda para máscara 9x9 em imagens de 21MB com 5 máquinas

		5 Máquinas							
9x9	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**25270,40**	**19038,90**	**14147,00**	**11710,75**	**13628,65**	**13983,10**	**14282,33**	**13017,00**	**12008,67**
Desvio Padrão	**10,95**	**1711,57**	**608,09**	**179,52**	**637,39**	**332,56**	**1039,00**	**103,12**	**143,75**
Hipótese a= 0,01		**-822,38**	**-2448,72**	**-5381,43**	**-2504,24**	**-3335,64**	**-1857,36**	**-6283,89**	**-5839,98**
Speedup		**1,33**	**1,79**	**2,16**	**1,85**	**1,81**	**1,77**	**1,94**	**2,10**
Eficiência		**0,27**	**0,36**	**0,43**	**0,37**	**0,36**	**0,35**	**0,39**	**0,42**

Tabela 55. Teste hipótese, *Speedup* e Eficiência filtro de detecção de borda para máscara 9x9 em imagens de 21MB com 6 máquinas

		6 Máquinas							
9x9	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**25270,40**	**21936,80**	**13275,20**	**11761,60**	**10952,50**	**13518,71**	**12339,33**	**11606,29**	**11434,25**
Desvio Padrão	**10,95**	**483,90**	**273,02**	**103,58**	**210,39**	**384,45**	**333,55**	**237,84**	**142,79**
Hipótese a= 0,01		**-820,80**	**-3898,83**	**-6913,84**	**-5271,25**	**-3236,98**	**-3815,94**	**-4744,90**	**-6111,90**
Speedup		**1,15**	**1,90**	**2,15**	**2,31**	**1,87**	**2,05**	**2,18**	**2,21**
Eficiência		**0,19**	**0,32**	**0,36**	**0,38**	**0,31**	**0,34**	**0,36**	**0,37**

Tabela 56. Teste hipótese, *Speedup* e Eficiência filtro de detecção de borda para máscara 9x9 em imagens de 21MB com 7 máquinas

		7 Máquinas							
9x9	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**25270,40**	**22010,80**	**13027,33**	**11583,17**	**10761,00**	**10663,50**	**12573,63**	**12010,75**	**12119,75**
Desvio Padrão	**10,95**	**393,40**	**291,28**	**238,86**	**122,03**	**228,14**	**417,40**	**320,32**	**304,69**
Hipótese a= 0,01		**-887,86**	**-3857,31**	**-4743,20**	**-6891,53**	**-5174,13**	**-3360,11**	**-3990,26**	**-4054,24**
Speedup		**1,15**	**1,94**	**2,18**	**2,35**	**2,37**	**2,01**	**2,10**	**2,09**
Eficiência		**0,16**	**0,28**	**0,31**	**0,34**	**0,34**	**0,29**	**0,30**	**0,30**

Tabela 57. Teste hipótese, *Speedup* e Eficiência filtro de detecção de borda para máscara 9x9 em imagens de 21MB com 8 máquinas

		8 Máquinas							
9x9	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**25270,40**	**21610,40**	**13079,40**	**11620,57**	**10709,80**	**10231,50**	**10117,50**	**11763,67**	**11149,00**
Desvio Padrão	**10,95**	**307,34**	**157,13**	**241,02**	**181,38**	**154,76**	**171,31**	**434,14**	**271,23**
Hipótese a= 0,01		**-1123,65**	**-5150,45**	**-4709,93**	**-5750,60**	**-6398,88**	**-6147,69**	**-3506,59**	**-4604,44**
Speedup		**1,17**	**1,93**	**2,17**	**2,36**	**2,47**	**2,50**	**2,15**	**2,27**
Eficiência		**0,15**	**0,24**	**0,27**	**0,29**	**0,31**	**0,31**	**0,27**	**0,28**

No Gráfico 10 são apresentados os resultados levando em consideração a máscara 11x11 em imagem de 21Mb. É possível observar que como já ocorrido em imagem de 21MB utilizando máscara 9x9 o ambiente paralelo formado por três máquinas apresenta resultados muito distinto comparado ao demais ambientes paralelo.

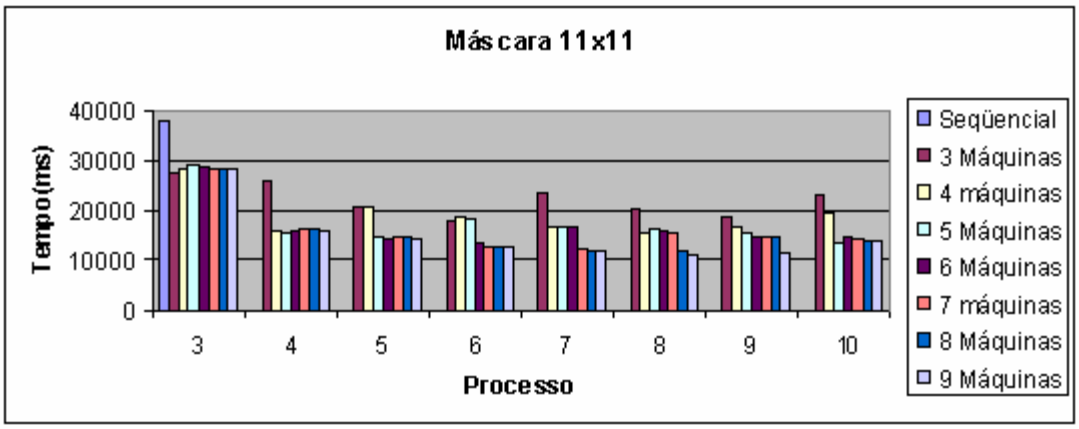

Gráfico 10 - Média do tempo de execução em paralelo do filtro de detecção de borda com máscaras 11x11, imagem de 21MB

São apresentados nas Tabela 58 a Tabela 63, com maiores detalhes, o desempenho do algoritmo paralelo. É possível observar que foi possível obter *speedup* superior 3.0 para os melhores casos, e o melhor índice de eficiência do algoritmo paralelo é obtido em um ambiente formado por seis máquinas.

Tabela 58. Teste hipótese, *Speedup* e Eficiência filtro de detecção de borda para máscara 11x11 em imagens de 21MB com 3 máquinas

		3 Máquinas							
11x11	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**38029,47**	**26003,35**	**20490,00**	**20490,00**	**17953,43**	**23429,10**	**20308,25**	**18339,00**	**22778,45**
Desvio Padrão	**12,52**	**641,72**	**138,89**	**138,89**	**341,12**	**482,06**	**204,07**	**133,79**	**536,55**
Hipótese a= 0,01		**-6370,16**	**-2575,25**	**-7807,32**	**-5847,37**	**-3595,88**	**-6595,39**	**-8916,22**	**-3564,90**
Speedup		**1,46**	**1,86**	**1,86**	**2,12**	**1,62**	**1,87**	**2,07**	**1,67**
Eficiência		**0,49**	**0,62**	**0,62**	**0,71**	**0,54**	**0,62**	**0,69**	**0,56**

Tabela 59. Teste hipótese, *Speedup* e Eficiência filtro de detecção de borda para máscara 11x11 em imagens de 21MB com 4 máquinas

		4 Máquinas							
11x11	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**38029,47**	**15717,75**	**20621,80**	**20621,80**	**18419,20**	**16929,60**	**15364,75**	**16315,67**	**19522,67**
Desvio Padrão	**12,52**	**266,75**	**413,28**	**413,28**	**468,49**	**375,93**	**193,61**	**406,16**	**305,04**
Hipótese a= 0,01		**-3500,04**	**-7312,77**	**-4620,60**	**-4897,41**	**-5863,73**	**-8646,62**	**-5812,43**	**-5688,29**
Speedup		**2,42**	**1,84**	**1,84**	**2,06**	**2,25**	**2,48**	**2,33**	**1,95**
Eficiência		**0,60**	**0,46**	**0,46**	**0,52**	**0,56**	**0,62**	**0,58**	**0,49**

Tabela 60. Teste hipótese, *Speedup* e Eficiência filtro de detecção de borda para máscara 11x11 em imagens de 21MB com 5 máquinas

		5 Máquinas							
11x11	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**38029,47**	**15520,85**	**14385,00**	**14385,00**	**18186,31**	**16539,68**	**16001,95**	**15510,75**	**13583,00**
Desvio Padrão	**12,52**	**387,24**	**197,30**	**197,30**	**328,13**	**308,63**	**485,78**	**715,12**	**286,57**
Hipótese a= 0,01		**-2460,46**	**-6166,11**	**-8940,62**	**-5888,68**	**-6568,14**	**-5404,82**	**-4572,44**	**-7742,49**
Speedup		**2,45**	**2,64**	**2,64**	**2,09**	**2,30**	**2,38**	**2,45**	**2,80**
Eficiência		**0,49**	**0,53**	**0,53**	**0,42**	**0,46**	**0,48**	**0,49**	**0,56**

Tabela 61. Teste hipótese, *Speedup* e Eficiência filtro de detecção de borda para máscara 11x11 em imagens de 21MB com 6 máquinas

		6 Máquinas							
11x11	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**38029,47**	**15851,67**	**14175,40**	**14175,40**	**13355,25**	**16765,88**	**15771,75**	**14458,00**	**14346,25**
Desvio Padrão	**12,52**	**228,67**	**153,79**	**153,79**	**221,75**	**358,24**	**510,85**	**299,53**	**163,97**
Hipótese a= 0,01		**-2783,65**	**-7821,66**	**-10131,30**	**-8829,78**	**-6048,58**	**-5328,89**	**-7308,63**	**-9764,26**
Speedup		**2,40**	**2,68**	**2,68**	**2,85**	**2,27**	**2,41**	**2,63**	**2,65**
Eficiência		**0,40**	**0,89**	**0,89**	**0,95**	**0,76**	**0,80**	**0,88**	**0,88**

Tabela 62. Teste hipótese, *Speedup* e Eficiência filtro de detecção de borda para máscara 11x11 em imagens de 21MB com 7 máquinas

		7 Máquinas							
11x11	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**38029,47**	**16217,00**	**14555,38**	**14555,38**	**12688,50**	**11966,25**	**15613,50**	**14907,25**	**14134,75**
Desvio Padrão	**12,52**	**255,36**	**393,02**	**393,02**	**115,52**	**275,42**	**429,27**	**184,43**	**953,24**
Hipótese a= 0,01		**-2828,34**	**-7299,53**	**-6384,62**	**-12266,21**	**-8412,71**	**-5841,34**	**-9024,38**	**-4211,43**
Speedup		**2,35**	**2,61**	**2,61**	**3,00**	**3,18**	**2,44**	**2,55**	**2,69**
Eficiência		**0,34**	**0,37**	**0,37**	**0,43**	**0,45**	**0,35**	**0,36**	**0,38**

Tabela 63. Teste hipótese, *Speedup* e Eficiência filtro de detecção de borda para máscara 11x11 em imagens de 21MB com 8 máquinas

		8 Máquinas							
11x11	**Seqüencial**	**3 processos**	**4 processos**	**5 processos**	**6 processos**	**7 processos**	**8 processos**	**9 processos**	**10 processos**
Média(ms)	**38029,47**	**16147,75**	**14300,50**	**14300,50**	**12640,33**	**11702,00**	**11729,75**	**14534,33**	**13695,00**
Desvio Padrão	**12,52**	**222,01**	**264,94**	**264,94**	**132,59**	**116,56**	**171,50**	**294,04**	**139,10**
Hipótese a= 0,01		**-3567,40**	**-7826,01**	**-7802,68**	**-11544,23**	**-12692,20**	**-10619,06**	**-7349,92**	**-10824,43**
Speedup		**2,36**	**2,66**	**2,66**	**3,01**	**3,25**	**3,24**	**2,62**	**2,78**
Eficiência		**0,29**	**0,33**	**0,33**	**0,38**	**0,41**	**0,41**	**0,33**	**0,35**

5.4. Considerações Finais

Para a realização dos testes foi utilizado um *cluster* homogêneo, ou seja, uma plataforma com nove máquinas com o mesmo poder computacional. Para a comunicação das mesmas utilizou-se a biblioteca de troca de mensagens JPVM e a partir dos resultados obtidos foi efetuada a análise comparativa entre o processamento seqüencial e o processamento paralelo.

Foi possível observar nos resultados do filtro da mediana de imagens de 11MB e de 21MB que ao utilizar a máscara 3x3 o *speedup* foi menor que 1 em todos os casos, o que demonstra que o programa paralelo para este tipo de máscara é ineficiente. Ao utilizar a máscara 5x5 para ambos tamanho de imagem o *speedup* tem o valor de 1,55, e como máscara 7x7 aproxima-se de 3 para imagem de 11MB e de 4 para imagem de 21MB.

Para detecção de borda os resultados obtidos foram expressivos sendo possível obter ganho para a maioria dos casos de teste, sendo possível obter *speedup* superior a 3.

Pode-se observar também que o uso computação paralela distribuída em imagens de tamanho mais elevado agrega um ganho ainda maior de desempenho.

CAPÍTULO 6. CONCLUSÃO

Partindo-se dos resultados obtidos pode-se verificar que existe na maioria dos testes um ganho em se fazer uso do processamento paralelo distribuído quando se tratar de processamento de imagens médicas.

O filtro de mediana quando aplicada em imagens de até 11MB com máscara 3x3 e em alguns casos com a máscara 5x5 não apresenta resultado favorável, isso se deve a alguns fatores que podem gerar gargalo na aplicação, como por exemplo, o JPVM utilizar o protocolo TCP e utilizar rotinas bloqueantes necessárias para o processamento da aplicação. Apesar do resultado obtido não ser favorável à execução paralela para estes casos, foi possível observar que para processamentos intensos o uso do processamento paralelo é bastante vantajoso.

Acredita-se que a utilização de outros filtros no domínio espacial também possa prover melhora significativa de desempenho para a versão paralela quando comparada à execução seqüencial. Além de que estes algoritmos podem perfeitamente ser combinados tendo assim um ganho imensurável comparado ao tempo seqüencial.

6.1. Trabalhos futuros

Com base nos resultados apresentados tem-se como trabalho futuro o desenvolvimento de novos algoritmos de processamento de imagens. Além disso, testes adicionais devem ser executados, o que permitirá a obtenção de um conjunto maior de dados a partir do qual tais informações poderão ser extraídas com maior fidelidade a fim de construir uma base de comparação efetiva.

Sugere-se ainda, como trabalhos futuros, a implementação utilizando *threads* a fim de amenizar o gargalo; o desenvolvimento de novas técnicas para quebra da imagem em blocos, a fim de permitir novos algoritmos faz uso do processamento paralelo; combinação de técnicas de processamento de imagem a fim de verificar qual o real ganho.

Análise de desempenho deste algoritmo em um ambiente heterogêneo e possivelmente o desenvolvimento de políticas de balanceamento cargas voltadas para aplicação de processamento de imagens medica.

6.2. Produção Bibliográfica

Trabalhos com os resultados da paralelização da filtragem mediana e a comparação de dois ambientes de passagem de mensagens, mpiJava e JPVM, resultaram em artigos e foram publicados em alguns eventos científicos.

6.2.1. Artigos Completos Publicados em Periódicos

- SAITO, Priscila Tiemi Maeda, SABATINE, R. J., NUNES, F. L. S., BRANCO, K. R. L. J. C. Otimização de Algoritmos de Processamento de Imagens Médicas Utilizando a Computação Paralela. Revista Diciplinarum Scientia. , v.1, p.1 - 8, 2007.

6.2.2. Trabalhos Completos Publicados em Anais de Congressos

- SAITO, Priscila Tiemi Maeda, SABATINE, R. J., NUNES, F. L. S., BRANCO, K. R. L. J. C. Otimização do Processamento de Imagens Médicas Utilizando a Computação Paralela In: CISCI 2007: 6a. Conferencia Iberoamericana en Sistemas, Cibernética e Informática, 2007, Orlando, Flórida. **6a. Conferencia Iberoamericana en Sistemas, Cibernética e Informática: CISCI 2007**. , 2007. v.1. p.1 – 1

- SAITO, Priscila Tiemi Maeda, SABATINE, R. J., NUNES, F. L. S., BRANCO, K. R. L. J. C. Otimização do Processamento de Imagens Médicas: Uma Abordagem Utilizando Java In: III Workshop de Visão Computacional – WVC' 2007, 2007, São José do Rio Preto. **Anais do III Workshop de Visão Computacional – WVC' 2007**. , 2007.

- SAITO, Priscila Tiemi Maeda, SABATINE, R. J., NUNES, F. L. S., BRANCO, K. R. L. J. C. Processamento de Imagens Médicas: Otimização no Tempo de Execução Usando Computação Paralela Distribuída In: III Simpósio de Instrumentação e Imagens Médicas, 2007, São Carlos. **Anais do III Simpósio de Instrumentação e Imagens Médicas**. , 2007.

- SAITO, Priscila Tiemi Maeda, SABATINE, R. J., NUNES, F. L. S., BRANCO, K. R. L. J. C. Uma Abordagem Utilizando mpiJava para a Paralelização de Algoritmos de Processamento de Imagens In: VIII WSCAD - VIII Workshop em Sistemas Computacionais de Alto Desempenho, 2007, Gramado - Rio Grande do Sul. **Anais do VIII WSCAD - VIII Workshop em Sistemas Computacionais de Alto Desempenho**. , 2007.

- SAITO, Priscila Tiemi Maeda, SABATINE, R. J., NUNES, F. L. S., BRANCO, K. R. L. J. C. Otimização de Algoritmos de Processamento de Imagens Médicas Utilizando a Computação Paralela In: VI Simpósio de Informática da Região Centro – SIRC 2007, 2007, Santa Maria. **Anais do VI SIRC/RS 2007 - VI Simpósio de Informática da Região Centro - Centro Universitário Franciscano (UNIFRA)**. , 2007.

- SAITO, Priscila Tiemi Maeda, SABATINE, R. J., NUNES, F. L. S., BRANCO, K. R. L. J. C. Uso da Computação Paralela Distribuída para Melhoria no Tempo de Processamento de Imagens Médicas In: XIV ERI/PR - XIV Escola Regional de Informática da SBC, 2007, Guarapuava. **Anais do XIV ERI/PR - XIV Escola Regional de Informática da SBC**. , 2007. v.1. p.36 - 47

- SABATINE, R. J., SAITO, Priscila Tiemi Maeda, NUNES, F. L. S., BRANCO, K. R. L. J. C. Utilização da Computação Paralela Distribuída na Otimização do Processamento de Imagens Médicas Fazendo Uso do JPVM In: VIII WSCAD - VIII Workshop em Sistemas Computacionais de Alto Desempenho, 2007, Gramado - Rio Grande do Sul. **Anais do VIII WSCAD - VIII Workshop em Sistemas Computacionais de Alto Desempenho**. , 2007.

6.2.3. Resumos Publicados em Anais de Congressos

- SAITO, P. T. M.; SABATINE, R. J.; NUNES, F. L. S.; BRANCO, K. R. L. J. C. *Paralelização de Algoritmos de Processamento de Imagens Médicas Fazendo uso de mpiJava.* In: 15º SIICUSP – 15º Simpósio Internacional de Iniciação Científica da Universidade de São Paulo., 2007, Marília. Anais do Simpósio Internacional de Iniciação Científica da USP, 2007b.

- SAITO, Priscila Tiemi Maeda, BRANCO, K. R. L. J. C., NUNES, F. L. S., SABATINE, R. J. Paralelização de Algoritmos de Processamento de Imagens

Médicas no Domínio Espacial - Uma Abordagem Usando Java In: XV CIC UFSCar - XV Congresso de Iniciação Científica da UFSCar - 7ª Jornada Científica da UFSCar, 2007, São Carlos. **Anais de Eventos da UFSCar - XV Congresso de Iniciação Científica da UFSCar - 7ª Jornada Científica da UFSCar**. , 2007. v.3. p.52 - 52

- SAITO, Priscila Tiemi Maeda, BRANCO, K. R. L. J. C. Otimização do Processamento de Imagens Médicas Utilizando a Computação Paralela In: VIII SIC UNIVEM – VIII Seminário de Iniciação Científica, 2007, Marília. **Anais do Seminário de Iniciação Científica do UNIVEM**, 2007.

- SABATINE, R. J., BRANCO, K. R. L. J. C., NUNES, F. L. S., SAITO, Priscila Tiemi Maeda. Utilização da Computação Paralela Distribuída na Otimização do Processamento de Imagens Médicas Utilizando JPVM In: XIV CIC UFSCar - XIV Congresso de Inciação Cientifica da UFSCar, 2007, São Carlos. **Anais de Eventos da UFSCar - XIV Congresso de Inciação Cientifica da UFSCar**. , 2007. v.3. p.49 - 49

REFERÊNCIAS BIBLIOGRÁFICAS

ACHCAR, J.A., RODRIGUES, J. 1995. ACHCAR, J.A., RODRIGUES, J. Introdução à Estatística para Ciências e Tecnologia. ICMSC-USP, São Carlos - Apostila de Consulta, 1995.

ALMASI e GOTTLIEB, 1994 ALMASI, G. S.; GOTTLIEB, A. Highly Parallel Computing. 2ed. Redwood City: The Benjamin/Cummings Publishing Company, Inc, 1994.

BARBOSA, 2000 BARBOSA, J. M. G. Paralelismo em Processamento e Analise de Imagem Médica; Faculdade de Engenharia da Universidade do Porto, 2000.

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

BAKER *et al.*, 1999 BAKER, M. *et al.* mpiJava: An Object-Oriented Java Interface to MPI, 1999.

BEGUELIN, 1994 BEGUELIN, A. *PVM: Parallel Virtual Machine. A User's Guide and Tutorial for Networked Parallel Computing*, The MIT Press, 1994.

BOOCH *et a*l., 2000 BOOCH, G; RUMBAUGH, J; JACOBSON, I. UML: Guia do Usuário,1ed. Rio de Janeiro: Campus, 2000.

BRANCO, 2004 BRANCO, K. R. L. J. C. Índice de Carga E Desempenho em Ambientes Paralelos/ Distribuídos – Modelagem e Métricas. Tese de mestrado. ICMC-USP. 2004.

CÁCERES *et al.*, 2001 CÁCERES, E. N.; MONGELLI, H.; SONG, S. W. Algorítmos Paralelos usando CGM/PVM/MPI: Uma Introdução, In: As Teconologias da Informação e a Questão Social. Ed. Porto Alegre: Sociedade Brasileira de Computação, 2001.

CAVALHEIRO, 2004 CAVALHEIRO G. G. H., Principio da Programação Concorrente, IV Escola Regional de Alto Desempenho, SBC/UFPel/UCPel/UFSM, pg 3 – 37, Janeiro, 2004.

COULOURIS *et al*., 2007 COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T. Sistemas Distribuídos: Conceitos e Projetos , 4. ed, Bookman, Porto Alegre, 2007.

CULLER, GUPTA e SINGH, 1999 CULLER, David E.; GUPTA, Anoop; SINGH, Jaswinder Pal. Parallel computer architecture: a *hardware/ software* approach. San Francisco, Caliornia: Morgan Kaufmann Publishers, 1999.

DOWNTON e CROOKES, 1998 DOWNTON A.; Crookes D.; Parallel architectures for image processing, Elect. & Comm. Eng. Jour., 139-151, 1998.

DUNCAN, 1990 DUNCAN, R. A Survey of Parallel Computer Architectures. IEEE Computer, 1990.

FERRARI, 1998 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.

FLYNN, 1972 FLYNN, M. J. Some Computer Organizations and Their Effectiveness, IEEE Transactions on Computers, 1972.

FLYNN e RUDD, 1996 FLYNN, M. J.; RUDD, K. W. Parallel Architectures. ACM Computing Surveys. 1996.

FOSTER, 1995 FOSTER, I.; Designing and building parallel programs: Concepts and tools for parallel software engineering. Addison-Wesley, 1995.

FRANCISCO, W. 1995 FRANCISCO, W.. Estatística Básica, Unimep. Piracicaba, 2 ª Edição, 1995.

GEIST *et al.*, 1994 GEIST, A.; BEGUELIN, A.; DONGARRA, J.; JIANG, W.; MANCHECK, B.; SUNDERAM, V.; PVM: Parallel Virtual Machine – A User´s Guide and Tutorial for Network Parallel Computing, The MIT Press, 1994.

GONZALES e WOODS, 2002 GONZALEZ, R.C.; WOODS, R.E. Processamento de Imagens Digitais. Ed Edgard Blücher LTDA. São Paulo, 2002

GROPP *et al.*, 1994 GROPP, W.; LUSK, E.; SKJELLUM, A. Using MPI: Portable Parallel Programming with the Message Passing-Interface. MIT Press, 1994.

HAMDI e LEE, 1997 HANDI M.; LEE C. K., Adaptive load-balancing of image processing applications on clusters of workstations, IEEE Trans. on Computers, vol. 39, No. 10, pg. 1477-1232, 1997.

JAQUIE, 1999 JAQUIE, K. R. L. Extensão da Ferramenta de Apoio à Programação Paralela (F.A.P.P.) para Ambientes Paralelos Virtuais. Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação – USP, São Carlos, São Paulo. 1999.

KIRNER, 1991 KIRNER C., Arquitetura de Sistemas Avançados de Computação, Anais da Jornada EPUSP/IEEE em Sistemas de Computação de Alto de Desempenho, pg. 307-353, 1991.

LEE *et al.*, 1999 LEE, B.; GU, Y.; CAI, W.; HENG, A. Parallel Processing Letters, Vol. 9, No. 3, pg 401-410, 1999

LI *et al.*, 2003 LI, X.L.; BHARADWAJ, V.; KO, C.C. Distributed image processing on a network of workstations. Int. J. Computers and Applications (ACTA Press), 25(2):1–10, 2003.

NAVAUX, 1989 NAVAUX P. O., Introdução ao processamento paralelo, RBC – Revista Brasileira de Computação, v 5, n 2, pg. 31-43, Outubro, 1989

NAVAUX *et al*., 2001 NAVAUX, P. O. A., BARRETO, M. E., ÁVILA, R. B., F.OLIVEIRA, Execução de aplicações em ambientes concorrentes In: Escola Regional de Alto Desempenho ed.Porto Alegre : SBC, v.1, pg. 179-193, 2001.

NICOLESCU e JONKER 2002, NICOLESCU, Cristina.; JONKER, Pieter, "A Data and Task Parallel Image Processing Environment," Parallel Computing, vol. 28, nos. 7-8, pg. 945-965, Aug. 2002.

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

NUNES, 2006 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, v. 1, p. 73-126, 2006

MORIN *et al.*, 2002 MORIN, S.; KOREN, I.; KRISHNA, C. M.; JMPI: Implementing the Message Passing Standard in Java. In thIPDPS 02: Proceedings of the 16 International Parallel and Distributed Processing Symposium, pg. 191, IEEE Computer Society, 2002.

PEZZI, 2005 PEZZI, G. P. Aplicação Paralela de um Filtro Gráfico, Trabalho do Curso de Pós - Graduação em Ciência da Computação apresentado à Universidade Federal do Rio Grande do Sul – UFRGS, 2005.

QUINN, 1994 QUINN, M. J. Parallel Computing: Theory and practice. 2. ed. New York: McGraw Hill, 1994.

SANTOS, 2004 SANTOS, R. D. C. Java Advanced Imaging API: A Tutorial. Revista de Informática Teórica e Aplicada, Rio Grande do Sul, v. 11, pg. 93-123, 2004.

SCHNORR, 2003 SCHNORR, L. M. Paralelização do Filtro de Mediana, Trabalho do Curso de Pós – Graduação em Ciência da Computação apresentado à Universidade Federal do Rio Grande do Sul – UFRGS, 2003.

SEINSTRA e KOELMA, 2004 SEINSTRA, F.J.; KOELMA, D. User Transparency: A Fully Sequential Programming Model for Efficient Data Parallel Image Processing, Concurrency and Computation: Practice and Experience, vol. 16, no. 6, pg. 611-644, May 2004.

SNIR et al, 1996 SNIR, M. OTTO, S. M., HUSS-LEDERMAN, S., DONGARRA, J. MPI: The Complete Reference, The MIT Press. 1996.

SUN, 2006 SUN Microsystems. JAI (Java Advanced Imaging) Application Programming Interface) document home page. Disponível em: <http://java.sun.com/products/javamedia/jai/forDevelopers/jai-apidocs/index.html> Acesso em: dezembro de 2006.

STALLINGS, 2003 STALLINGS, W. Arquitetura e Organização de Computadores: Projeto para o desempenho. 5. ed. São Paulo: Prentice Hall, 2003. Tradução: Carlos Camarão de Figueiredo e Lucília Camarão de Figueiredo.

TABOADA *et al.*, 2003 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, pg. 29-36, 2003.

TANENBAUM e VAN STEEN, 2002 TANENBAUM, Andrew S.; VAN STEEN, Maarten. Distributed systems: principles and paradigms. Upper Saddle River, New Jersey: Prentice Hall, 2002.

TANENBAUM, 2001a TANENBAUM, A. S. Modern Operating Systems. 2. ed. Upper Saddle River, New Jersey: Prentice-Hall, Inc., 2001.

TANENBAUM, 2001b TANENBAUM, A. S. Organização Estruturada de Computadores. 4. ed. Rio de Janeiro: Livros Técnicos e Científicos, 2001.

TANENBAUM, 1992 TANENBAUM, Andrew S. Modern Operating Systems. New Jersey, Prentice Hall International, Inc. 1992.

TANENBAUM, 1995 TANENBAUM, Andrew S. Distributed operating systems. New Jersey: Prentice Hall, 1995.

TANENBAUM, 1999 TANENBAUM, Andrew S. Sistemas operacionais modernos. Rio de Janeiro: Livros Técnicos e Científicos, 1999.

MCBRYAN, 1994 MCBRYAN, O. A. “An overview of message passing environments”, Parallel Computing, v. 20.1994.

PITANGA, 2003 PITANGA M. Computação em Cluster: o estado da arte da computação. Brasport, 2003.

SNIR *et al.*, 1996 SNIR, M.; OTTO, S. M.; HUSS-LEDERMAN, S.; DONGARRA, J. MPI: The Complete Reference, The MIT Press, 1996.

Apêndice A

```
import java.awt.image.*;
import javax.media.jai.*;

public class Image {

private PlanarImage image;
private int width;
private int height;
private int nbands;
private SampleModel sm;
private ColorModel  cm;

   public Image(){}
  public Image(String ima){
                this.image  = JAI.create("fileload",ima);
                this.width   = image.getWidth();
                this.height  = image.getHeight();
                this.sm      = image.getSampleModel();
                this.cm      = image.getColorModel();
                this.nbands = sm.getNumBands();
        }

        public int[] getDataImage () {
                Raster inputRaster = image.getData();
                int[] pixels = new int[nbands*width*height];
                inputRaster.getPixels(0,0,width,height,pixels);
           return pixels;
        }

        public int getWidth(){
                return this.width;
        }

       public Image getImage(){
                return this.image;
        }

        public int getHeight(){
                return this.height;
        }

        public int getBands(){
                return this.nbands;
        }

        public void setDataImage (int[] pixels) {
                Raster inputRaster = image.getData();
                WritableRaster outputRaster = inputRaster.createCompatibleWritableRaster();
                outputRaster.setPixels(0,0,width,height,pixels);
                TiledImage tiledImage = new TiledImage(0,0,width,height,0,0,sm,cm);
                tiledImage.setData(outputRaster);
                JAI.create("filestore",tiledImage,"r.TIF","TIFF");
        }

}
```

Apêndice B

```
import javax.media.jai.*;
import java.util.*;
import jpvm.*;
public class MasterProcess {

private Image               image;
private int                 numAlgoritm;
private int                 numProcess=0;
private int                 lenTemplate;

        public MasterProcess() { }

        public MasterProcess(String nomeImage, int numAlgoritm, int process, int template) {
                this.image      = new Image(nomeImage);
                this.numAlgoritm   =    numAlgoritm;
                this.numProcess = process;
                this.lenTemplate = template;
                image.setDataImage(PartRec());
        }

        public static double msecond() {
                    Date d = new Date();
                    double msec = (double) d.getTime();
                    return msec;
        }

        public int[] PartRec(){

                     double startTime = msecond();
                     int heigth = image.getHeight();
                    int width  = image.getWidth();
                    int nbands = image.getBands();
                    int[] pixelsOriginal = image.getDataImage();
                    int[] pixelsResult = new int[heigth*width*nbands];
                    int redundancy = 1,beginSubimage=0;
                    int heigthSubimage = (heigth/numProcess);

                    try {

                    jpvmEnvironment jpvm = new jpvmEnvironment();
                    jpvmTaskId mytid = jpvm.pvm_mytid();
                     jpvmTaskId tids[] = new jpvmTaskId[numProcess];
                    jpvm.pvm_spawn("Slave",numProcess,tids);

                                for(int j=0;j<numProcess;j++){

                                 int[] subimage =new
int[(width*heigthSubimage*nbands)+(redundancy*(lenTemplate/2)*width*nbands)];
                                 System.arraycopy(pixelsOriginal,(beginSubimage-((redundancy-
1)*(lenTemplate/2)*width*nbands)),subimage,0,subimage.length);

                                jpvmBuffer buf = new jpvmBuffer();
                                buf.pack((numAlgoritm));
                                buf.pack((lenTemplate));
```

```
                                buf.pack(width);
                                buf.pack(nbands);
                                buf.pack((subimage.length)/(width*nbands));
                                buf.pack(subimage.length);
                                buf.pack(subimage,subimage.length,1);
                                jpvm.pvm_send(buf,tids[j],j);

                                beginSubimage += (heigthSubimage*width*nbands);
                                redundancy=2;

                                if((heigth-
((beginSubimage+((lenTemplate/2)*width*nbands))/(width*nbands)))< heigthSubimage)
                                heigthSubimage = (heigth-
((beginSubimage+(redundancy*(lenTemplate/2)*width*nbands))/(width*nbands)));
}

                        for (int i=0;i<numProcess; i++) {
                                int indexBeginSubImage, indexFinalSubimage, indexImageOriginal;

                                jpvmMessage message = jpvm.pvm_recv();
                                int lenSubimage = message.buffer.upkint();
                                int[] subimage = new int[lenSubimage];
                                message.buffer.unpack(subimage,lenSubimage,1);

                                //calcula inicio da image
                                if( message.messageTag == 0){
                                indexBeginSubImage = 0;
                                indexImageOriginal = 0;
                                }

                                else{
                                indexBeginSubImage =  (lenTemplate/2)*width*nbands;
                                indexImageOriginal = (heigthSubimage*message.messageTag)*width*nbands;
                                }

                                //calcula final da subimage
                                indexFinalSubimage= lenSubimage -((lenTemplate/2)*width*nbands);

                                if( message.messageTag+1 == numProcess ){
                                if((heigth%(heigth/numProcess)) <= 1)
                        indexFinalSubimage = indexFinalSubimage + (width*(lenTemplate/2)*nbands);
                        }
                                for(int
k=indexBeginSubImage;k<indexFinalSubimage;k++,indexImageOriginal++)
                                pixelsResult[indexImageOriginal]=subimage[k];
            }
            jpvm.pvm_exit();
        }
            catch (jpvmException jpe) {
            System.out.println("Error - jpvm exception");
}
       System.out.println(msecond() - startTime);
        return(pixelsResult);
}

public static void main(String[] args){
    newMasterProcess(args[0],Integer.parseInt(args[1]),Integer.parseInt(args[2]), Integer.parseInt(args[3]));
}
}
```

Apêndice C

```
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

import jpvm.*;

public class Slave{

        private int   optionAlgor;
        private int   lenTemplate;
        private int   width;
        private int   height;
        private int   nbands;
        private int[] pixelsOriginal;
        private int           numberMessage;
        private jpvmEnvironment jpvm;
       private jpvmTaskId parent;

   public Slave(){
          try{
               jpvm = new jpvmEnvironment();
                parent = jpvm.pvm_parent();
                ReceiveMessage();
                ImagProcesing();
                SendMessage(ImagProcesing());
                jpvm.pvm_exit();
           }
                catch (jpvmException e) {
                        System.out.print(e.getMessage());
                }
    }

   public int[] ImagProcesing(){
                ImageProcess ima = new ImageProcess(lenTemplate,width,height,nbands,pixelsOriginal);
                 if(optionAlgor == 0){
                        return ima.Median();
                 }
                 else{
                        return ima.Sobel();
          }
       }

   public void ReceiveMessage(){

       try{
                jpvmMessage message = jpvm.pvm_recv();
                optionAlgor         = message.buffer.upkint();
                lenTemplate          = message.buffer.upkint();
                width               = message.buffer.upkint();
                nbands              = message.buffer.upkint();
                height              = message.buffer.upkint();
                int lenSubVector    = message.buffer.upkint();
                pixelsOriginal      = new int[lenSubVector];
                message.buffer.unpack(pixelsOriginal,lenSubVector,1);
```

```
                numberMessage           = message.messageTag;
        }
        catch (jpvmException e) {
                        System.out.print(e.getMessage());
        }
    }

    public void SendMessage(int[] pixelsResultado){
            try{
                    jpvmBuffer buf = new jpvmBuffer();
                    buf.pack(pixelsResultado.length);
                    buf.pack(pixelsResultado,pixelsResultado.length,1);
                    jpvm.pvm_send(buf,parent,numberMessage);
             }
             catch (jpvmException e) {
                        System.out.print(e.getMessage());
             }
    }

     public static void main(String args[]) {
            new Escravo();
        }
}
```

Apêndice D

```
public class ImageProcess {

        private int   lenTemplate;
        private int   width;
        private int   height;
        private int   nbands;
        private int[] pixelsOriginal;

        public ImageProcess(){}

        public ImageProcess(int lenTemplate, int width, int height, int nbands, int[] pixelsOriginal){
                this.lenTemplate    = lenTemplate;
                this.width                  = width;
                this.height             = height;
                this.nbands             = nbands;
                this.pixelsOriginal  = pixelsOriginal;
        }

        public int[] Sort(int[] vector){
               int i , j , hi = 1, value ;

               do { hi = 3 * hi + 1; } while ( hi < vector.length );
                   do {
                       hi /= 3;
                       for ( i = hi; i < vector.length; i++) {
                            value = vector[ i ];
                            j = i - hi;
                           while (j >= 0 && value < vector[ j ])
                           {
                           vector[ j + hi ] = vector[ j ];
                           j -= hi;
                        }
                    vector[ j + hi ] = value;
                  }
              } while ( hi > 1 );

           return vector;
        }

        public int[] Median(){

                int[] pixelsResult = pixelsOriginal;
                int tamVectorSort = (lenTemplate)*(lenTemplate);
                int halfTemplate = lenTemplate/2;
                int offset;int offset2;
                 for (int heightP = halfTemplate; heightP<height-halfTemplate; heightP++)
                    for (int widthP= halfTemplate; widthP<width-halfTemplate; widthP++) {
                       offset = heightP*width* nbands+widthP*nbands;
                        int[] vectorNeighborhood = new int[tamVectorSort];
                        int indexVectorNeighborhood = 0;

                            for (int heightPNeighborhood = heightP-halfTemplate; heightPNeighborhood <=
heightP+halfTemplate; heightPNeighborhood++)
                                for (int widthPNeighborhood= widthP-halfTemplate; widthPNeighborhood <=
widthP+halfTemplate;widthPNeighborhood++){
```

```
                                    offset2                                                                    =
(heightPNeighborhood*width*nbands)+(widthPNeighborhood*nbands);

                                vectorNeighborhood [indiceVectorSort] = pixelsOriginal[offset2+0];
                                    indexVectorNeighborhood++;
                                }

                            int[] vectorSort = Sort(vectorNeighborhood);
                            for (int band=0; band< nbands; band++)
                                pixelsResult[offset+band] = vectorSort[(tamVectorSort/2)];
                            }
        return pixelsResult;
  }

                public int[] getTemplateH11(){

                        return (new int[]{5, 6, 7, 8, 9,10, 9, 8, 7, 6, 5,
                                    4, 5, 6, 7, 8, 9, 8, 7, 5, 6, 4,
                                    3, 4, 5, 6, 7, 8, 7, 6, 5, 4, 3,
                                    2, 3, 4, 5, 6, 7, 6, 5, 4, 3, 2,
                                    1, 2, 3, 4, 5, 6, 5, 4, 3, 2, 1,
                                    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                   -1,-2,-3,-4,-5,-6,-5,-4,-3,-2,-1,
                                   -2,-3,-4,-5,-6,-7,-6,-5,-4,-3,-2,
                                   -3,-4,-5,-6,-7,-8,-7,-6,-5,-4,-3,
                                   -4,-5,-6,-7,-8,-8,-9,-7,-5,-6,-4,
                                   -5,-6,-7,-8,-9,-10,-9,-8,-7,-6,-5,});

                }

                public int[] getTemplateV11(){

                        return (new int[]{5,4,3,2,1,0,-1,-2,-3,-4,-5,
                                        6,5,4,3,2,0,-2,-3,-4,-5,-6,
                                        7,6,5,4,3,0,-3,-4,-5,-6,-7,
                                        8,7,6,5,4,0,-4,-5,-6,-7,-8,
                                        9,8,7,6,5,0,-5,-6,-7,-8,-9,
                                       10,9,8,7,6,0,-6,-7,-8,-9,-10,
                                        9,8,7,6,5,0,-5,-6,-7,-8,-9,
                                        8,7,6,5,4,0,-4,-5,-6,-7,-8,
                                        7,6,5,4,3,0,-3,-4,-5,-6,-7,
                                        6,5,4,3,2,0,-2,-3,-4,-5,-6,
                                        5,4,3,2,1,0,-1,-2,-3,-4,-5});
                }

                public int[] getTemplateD11(){
                        return (new int[]{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
                                        -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
                                        -2,-1, 0, 1, 2, 3, 4, 5, 6, 7, 8,
                                        -3,-2,-1, 0, 1, 2, 3, 4, 5, 6, 7,
                                        -4,-3,-2,-1, 0, 1, 2, 3, 4, 5, 6,
                                        -5,-4,-3,-2,-1, 0, 1, 2, 3, 4, 5,
                                        -6,-5,-4,-3,-2,-1, 0, 1, 2, 3, 4,
                                        -7,-6,-5,-4,-3,-2,-1, 0, 1, 2, 3,
                                        -8,-7,-6,-5,-4,-3,-2,-1, 0, 1, 2,
                                        -9,-8,-7,-6,-5,-4,-3,-2,-1, 0, 1,
                                       -10,-9,-8,-7,-6,-5,-4,-3,-2,-1, 0});
                }
```

```
public int[] getTemplateS11(){

        return (new int[]{10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0,
                          9, 8, 7, 6, 5, 4, 3, 2, 1, 0,-1,
                          8, 7, 6, 5, 4, 3, 2, 1, 0, 1,-2,
                          7, 6, 5, 4, 3, 2, 1, 0,-1,-2,-3,
                          6, 5, 4, 3, 2, 1, 0,-1,-2,-3,-4,
                          5, 4, 3, 2, 1, 0,-1,-2,-3,-4,-5,
                          4, 3, 2, 1, 0,-1,-2,-3,-4,-5,-6,
                          3, 2, 1, 0,-1,-2,-3,-4,-5,-6,-7,
                          2, 1, 0,-1,-2,-3,-4,-5,-6,-7,-8,
                          1, 0,-1,-2,-3,-4,-5,-6,-7,-8,-9,
                          0,-1,-2,-3,-4,-5,-6,-7,-8,-9,-10});
}

public int[] getTemplateH9(){

        return (new int[]{ 4, 5, 6, 7, 8, 7, 5, 6, 4,
                           3, 4, 5, 6, 7, 6, 5, 4, 3,
                           2, 3, 4, 5, 6, 5, 4, 3, 2,
                           1, 2, 3, 4, 5, 4, 3, 2, 1,
                           0, 0, 0, 0, 0, 0, 0, 0, 0,
                          -1,-2,-3,-4,-5,-4,-3,-2,-1,
                          -2,-3,-4,-5,-6,-5,-4,-3,-2,
                          -3,-4,-5,-6,-7,-6,-5,-4,-3,
                          -4,-5,-6,-7,-8,-7,-5,-6,-4});

}

public int[] getTemplateV9(){

        return (new int[]{4,3,2,1,0,-1,-2,-3,-4,
                          5,4,3,2,0,-2,-3,-4,-5,
                          6,5,4,3,0,-3,-4,-5,-6,
                          7,6,5,4,0,-4,-5,-6,-7,
                          8,7,6,5,0,-5,-6,-7,-8,
                          7,6,5,4,0,-4,-5,-6,-7,
                          6,5,4,3,0,-3,-4,-5,-6,
                          5,4,3,2,0,-2,-3,-4,-5,
                          4,3,2,1,0,-1,-2,-3,-4});
}

public int[] getTemplateD9(){
        return (new int[]{0, 1, 2, 3, 4, 5, 6, 7, 8,
                         -1, 0, 1, 2, 3, 4, 5, 6, 7,
                         -2,-1, 0, 1, 2, 3, 4, 5, 6,
                         -3,-2,-1, 0, 1, 2, 3, 4, 5,
                         -4,-3,-2,-1, 0, 1, 2, 3, 4,
                         -5,-4,-3,-2,-1, 0, 1, 2, 3,
                         -6,-5,-4,-3,-2,-1, 0, 1, 2,
                         -7,-6,-5,-4,-3,-2,-1, 0, 1,
                         -8,-7,-6,-5,-4,-3,-2,-1, 0});
}

public int[] getTemplateS9(){

        return (new int[]{8, 7, 6, 5, 4, 3, 2, 1, 0,
                          7, 6, 5, 4, 3, 2, 1, 0,-1,
                          6, 5, 4, 3, 2, 1, 0, 1,-2,
```

```
                                5, 4, 3, 2, 1, 0,-1,-2,-3,
                                4, 3, 2, 1, 0,-1,-2,-3,-4,
                                3, 2, 1, 0,-1,-2,-3,-4,-5,
                                2, 1, 0,-1,-2,-3,-4,-5,-6,
                                1, 0,-1,-2,-3,-4,-5,-6,-7,
                                0,-1,-2,-3,-4,-5,-6,-7,-8});
}


public int[]  Sobel(){

    int[] pixelsResult = new int[pixelsOriginal.length];
    int somah = 0;
    int somav = 0;
    int somads = 0;
    int somadi = 0;
    int size_vector = lenTemplate*lenTemplate;
    int halfTemplate = lenTemplate/2;
            int offset, offset2;

            int[] veth=null,vetv=null,vetdi=null,vetds=null;
            veth = getTemplateH11();
            vetv = getTemplateV11();
            vetdi = getTemplateD11();
            vetds = getTemplateS11();

            if(lenTemplate == 9){
                    veth = getTemplateH9();
                    vetv = getTemplateV9();
                    vetdi = getTemplateD9();
                    vetds = getTemplateS9();
            }

            if(lenTemplate == 11){
                    veth = getTemplateH11();
                    vetv = getTemplateV11();
                    vetdi = getTemplateD11();
                    vetds = getTemplateS11();
            }


            for(int h = halfTemplate; h < height - halfTemplate; h++)
                    for (int w = halfTemplate; w < width - halfTemplate; w++) {
                    offset = h * width * nbands + w * nbands;
                    int [] vector = new int [size_vector];
                    int ele = 0;
                    for (int a = h - halfTemplate; a <= h + halfTemplate; a++)
                    for (int l = w - halfTemplate; l <= w + halfTemplate; l++) {
                    offset2 = a * width * nbands + l * nbands;
                    vector[ele] = pixelsOriginal[offset2 + 0];
                    ele++;

                    }

                    somah = 0;
                    somav = 0;
                    somads = 0;
                    somadi = 0;
```

```
                    for (int i=0; i<vector.length; i++) {
                        somah += vector[i] * veth[i];
                        somav += vector[i] * vetv[i];
                        somads += vector[i] * vetds[i];
                somadi += vector[i] * vetdi[i];
            }

                    for (int band = 0; band < nbands; band++) {
                    int rn=0;
                    int rd=0;
                    if (somah < somav) {
                        rn = somav;
                    }
                    else {
                        rn = somah;
                    }
                    if (somads < somadi) {
                        rd = somadi;
                    }
                    else {
                        rd = somads;
                    }
                    if (rn < rd) {
                        if (rd < 0)
                            pixelsResult[offset + band] = 0;
                            else
                             if (rd > 65535)
                                pixelsResult[offset + band] = 65535;
                            else
                                pixelsResult[offset + band] = rd;
                          }
                          else {
                                if (rn < 0)
                                   pixelsResult[offset + band] = 0;
                                else
                                  if (rn > 65535)
                                 pixelsResult[offset + band] = 65535;
                                    else
                                pixelsResult[offset + band] = rn;
                             }
                    }
            }
        return(pixelsResult);
    }
}
```

ANEXO

INSTALANDO JPVM

A instalação do pacote JPVM muito simples, não requer privilégio de administrador, podem ser realizado como usuário comum, por ser escrito totalmente em Java o JPVM requer um Kit de desenvolvimento Java JDK versão 1.1 ou superior.

O pacote JPVM com os códigos fontes e a documentação esta disponível em:

http://www.cs.virginia.edu/jpvm/src/jpvm.zip

Desempacote no diretório de sua escolha, entretanto para facilitar o acesso escolha diretório como home ou root no linux ou C:\ no Windows, será criada uma pasta jpvm contendo todos os códigos fontes já pré-compilados, o pacote JPVM deve ser instalado em todas as máquinas que fará parte da máquina virtual ou ser instalada apenas no mestre e compartilhar a pasta entre os nós restantes através do sistema de arquivo NFS ou SAMBA.

Adicionar o caminho do diretório do pacote JPVM na variável de ambiente CLASSPATH, vale lembrar que não é obrigatório, entretanto esta variável é importante tanto para compilar quanto para executar o programa, já que especifica onde estão armazenados os arquivos e bibliotecas necessárias, tanto para a compilação, quanto para a execução, caso não queria utilizar variável CLASSPATH todos os comandos deveram ser executados dentro do diretório corrente /jpvm.

Tabela 64 – Comando de Configuração CLASSPATH JPVM

Sistema Operacional	Comando
Windows	$ set CLASSPATH=%CLASSPATH%;C:\diretório_do_pacote
Linux	$ export CLASSPATH= diretório do pacote JPVM

Para não perder a configuração todo vez que a máquina for desligada, gravar este comando no Linux no arquivo .bash_profile e no Windows ir em propriedades do sistema, avançado, variáveis de ambiente, adicionar em variáveis do usuário.

Caso esteja utilizando uma plataforma Windows superior a versão Windows NT será necessária atualizar o pacote.

No diretório do pacote entre na pasta /jpvm/jpvm, abra o arquivo com editor de texto jpvmEnvironment.java altere o método pvm_*daemon*_file_name(), para que possa ser reconhecida a versão do Windows que estará sendo utilizada, como sugere o exemplo.

```
1     */JPVM PARA WINDOWS XP./*
2
3      public static String pvm_daemon_file_name() {
4          String osName   = System.getProperty("os.name");
5          String userName = System.getProperty("user.name");
6          String fileName = null;
7          if(osName.equals("Windows 95") ||
8             osName.equals("Windows NT") ||
9             osName.equals("Windows 3.1")) {
10            fileName = "c:\\temp\\jpvmd-"+userName+".txt";
11         }
12         else if (osName.equals("Windows XP")){
13         fileName="c:\\Windows\\Temp\\jpvmd-"+userName+".txt";
14         }
15         else {
16               fileName = "/tmp/jpvmd."+userName;
17         }
18         return fileName;
19    }
```

Figura 36 – Método pvm_daemon_file_name() da *class* jpvmEnvironment

Neste caso esta sendo adicionada à versão Windows XP, salve o arquivo, abra terminal entre na pasta do código-fonte /jpvm/jpvm, compile com o comando

```
$ javac jpvmEnvironment.java
```

caso não tenha setado a localização do diretório na variável CLASSPATH a compilação retornara diversos erros, para compilar será necessário setar a variável CLASSPATH ou copiar o código-fonte para o diretório corrente /jpvm sendo que todo os arquivo gerados com a compilação devem ser copiados novamente para pasta /jpvm/jpvm.

A partir deste momento a biblioteca de troca de mensagem JPVM esta devidamente instalada, em seguida devemos iniciar o *Daemon* manualmente em todos nós que foram parte da máquina virtual paralela.

No Prompt de comando do Dos ou em terminal no Linux.

```
$ java jpvm.jpvmDaemon
```

Será retornado na tela o nome da máquina e o número da porta associado ao *Daemon*.

CONSOLE DO JPVM

Console é interface entre o sistema e o usuário, a partir dele é possível executar o gerenciamento da máquina virtual paralela, o console deve ser executado na máquina que tenha acesso a todas os *host* que formam a máquina virtual, geralmente é utilizado o mestre. Uma observação importante é com relação à configuração correta do arquivo *host* para estabelecer comunicação entre os *host*s tanto no sistema Linux quanto no Windows.

O console do JPVM pode ser ativado com o comando

```
$ java jpvm.jpvmConsole
```

Será exibido na tela "jpvm >", significando que o Console esta em execução pronta para executar as operações como.

Tabela 65 – Comando do JPVM CONSOLE

Comandos	Descrição
add	Adicionar um *host* a Máquina Virtual, solicita o nome da máquina e porta na qual o *daemon* esta executando.

conf	Informa o número total de máquina disponível na máquina virtual.
ps	Listas todos as tarefas em execução atualmente na máquina virtual, as tarefas são listado por máquina.
halt	Finaliza a execução de todos os *daemon* executados na máquina virtual, inclusive o console.
help	Imprime na tela informação sobre os comandos do Console.
quit	Sai do console sem finalizar a máquina virtual.

EXECUTANDO PROGRAMA

Para compilar e rodar alguma aplicação Java utilizando a biblioteca JPVM, tal deve estar dentro do diretório /jpvm aonde os *daemon* foram inicializados, ou adicionar o caminho da aplicação na variável de ambiente CLASSPATH, aplicação deve estar visível aos *daemon*s que fazem parte da máquina virtual.

Compilando aplicação.

```
$ javac nome_do_programa.java
```

Executar aplicação.

```
$ java nome_do_programa
```