UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
DEPARTAMENTO DE CIÊNCIAS EXATAS E NATURAIS
CURSO DE CIÊNCIA DA COMPUTAÇÃO

LUIZ FELIPE SAMPAIO DE OLIVEIRA

UM SOFTWARE DE ACESSO REMOTO PARA PROCESSAMENTO DIGITAL DE IMAGENS EM PARALELO

Mossoró/RN
2015

LUIZ FELIPE SAMPAIO DE OLIVEIRA

UM SOFTWARE DE ACESSO REMOTO PARA PROCESSAMENTO DIGITAL DE IMAGENS EM PARALELO

Monografia apresentada à Universidade Federal Rural do Semi-Árido - UFERSA para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Paulo Henrique Lopes Silva - UFERSA

Mossoró/RN

2015

© Todos os direitos estão reservados a Universidade Federal Rural do Semi-Árido. O conteúdo desta obra é de inteira responsabilidade do (a) autor (a), sendo o mesmo, passível de sanções administrativas ou penais, caso sejam infringidas as leis que regulamentam a Propriedade Intelectual, respectivamente, Patentes: Lei n° 9.279/1996 e Direitos Autorais: Lei n° 9.610/1998. O conteúdo desta obra tomar-se-á de domínio público após a data de defesa e homologação da sua respectiva ata. A mesma poderá servir de base literária para novas pesquisas, desde que a obra e seu (a) respectivo (a) autor (a) sejam devidamente citados e mencionados os seus créditos bibliográficos.

O48s Oliveira, Luiz Felipe Sampaio de.
Um software de acesso remoto para processamento digital de imagens em paralelo / Luiz Felipe Sampaio de Oliveira. - 2015.
52 f. : il.

Orientador: Paulo Henrique Lopes Silva.
Monografia (graduação) - Universidade Federal Rural do Semi-árido, Curso de Ciência da Computação, 2015.

1. Programação paralela. 2. Acesso remoto. 3. Processamento de imagens. I. Silva, Paulo Henrique Lopes, orient. II. Título.

Ficha catalográfica elaborada por sistema gerador automático em conformidade com AACR2 e os dados fornecidos pelo) autor(a).
Biblioteca Campus Mossoró / Setor de Informação e Referência
Bibliotecária: Keina Cristina Santos Sousa e Silva
CRB: 15/120

O serviço de Geração Automática de Ficha Catalográfica para Trabalhos de Conclusão de Curso (TCC´s) foi desenvolvido pelo Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo (USP) e gentilmente cedido para o Sistema de Bibliotecas da Universidade Federal Rural do Semi-Árido (SISBI-UFERSA), sendo customizado pela Superintendência de Tecnologia da Informação e Comunicação (SUTIC) sob orientação dos bibliotecários da instituição para ser adaptado às necessidades dos alunos dos Cursos de Graduação e Programas de Pós-Graduação da Universidade.

LUIZ FELIPE SAMPAIO DE OLIVEIRA

UM SOFTWARE DE ACESSO REMOTO PARA PROCESSAMENTO DIGITAL DE IMAGENS EM PARALELO

Monografia apresentada à Universidade Federal Rural do Semi-Árido - UFERSA para obtenção do título de Bacharel em Ciência da Computação.

Mossoró-RN, 30 de Janeiro de 2015.

Prof. M.Sc. Paulo Henrique Lopes Silva – UFERSA
Presidente

Prof. Dr. Helcio Wagner da Silva – UFERSA
Primeiro Membro

Prof. Dr. Judson Santos Santiago – UFERSA
Segundo Membro

Prof. M.Sc. Leandro Carlos de Souza – UFERSA
Terceiro Membro

Mossoró/RN

2015

AGRADECIMENTOS

Por incrível que pareça esta foi a parte mais difícil da monografia, pois tenho tantas pessoas a agradecer que seria difícil citar todas. Então eu gostaria de agradecer a todos que contribuíram para minha formação, sejam as pessoas do meu passado que me deram a base pra ser quem eu sou, ou as pessoas do presente que me ensinam a cada dia.

A graduação foi uma tarefa difícil e de dedicação, mas graças a vocês se tornou mais fácil. Uma boa turma me ajudou durante o curso, uma parte deles conhecidos como os tesouras: Dácio tesoura master, Weliton preguiça, Ulysses negativo, Ramiro tesoura Jr e David Naj turista e sem contar as turmas dos outros períodos que tive oportunidade de passar um bom tempo. Sejam nos piores ou melhores momentos sempre tínhamos com o que rir e vocês podem ter certeza vou guardar essas lembranças comigo.

O caminho foi interessante, infelizmente tive uma perda no final, minha avó. Eu gostaria bastante que você ainda tivesse viva para ver minha formatura. Sei que é egoísmo, pois sei que a senhora estava sofrendo, mas eu agradeço por tudo que fez por mim.

Não seria justo se também não agradecesse aos meus amigos dos jogos online, os quais me deram um bom suporte ao longo desses anos me aturando vários dias. Vocês me ajudaram bastante durante as madrugadas, portanto agradeço a todas guildas, pelas quais passei: Priorado, Take it Easy, Faith, Pandaemia e Bussiness class. Vocês me ajudaram bastante, muito obrigado!

Por fim, gostaria de agradecer a uma pessoa do meu passado, fênix, sei que é estranho, afinal faz cinco anos desde de que te vi pela última vez. Mesmo sem você saber, me ajudou bastante, me guiou pra ser uma pessoa melhor e só tenho a agradecer o tempo que pude passar com você.

RESUMO

Este trabalho tem como objetivo desenvolver e analisar o comportamento de um sistema que implementa filtros de imagens que são implementados usando técnicas de programação paralela. Além disso, também é propósito deste trabalho disponibilizar esses filtros por meio de um servidor remoto. Para tanto, é adotado o modelo clássico cliente/servidor. Esse sistema, possui uma implementação híbrida. Ele combina características importantes como o processamento paralelo para os filtros de imagens, bem como a flexibilização do acesso de clientes a estes filtros por meio de um servidor remoto.

Palavras-chaves: Programação Paralela, Acesso Remoto, Processamento de Imagens

ABSTRACT

This report aims to develop and analyze the behavior of a system for images that is implemented using parallel programming techniques filters. Furthermore, it is also purpose of this work to provide such filters by means of a remote server. Therefore, the classic client / server model is adopted. This system therefore has a hybrid implementation and tries to combine important features such as parallel processing for the image filters as well as the flexibility of the clients to access these filters by a remote server.

Key-Words: Parallel Programming, Remote Access, Image Processing

Sumário

1 INTRODUÇÃO 10

1.1 OBJETIVOS 11

1.2 ORGANIZAÇÃO DO TRABALHO 11

2 PROCESSAMENTO DIGITAL DE IMAGENS 13

2.1 MÉTODOS DE FILTRAGEM 15

2.1.1 Transformações de Intensidade 17

2.1.1.1 Inversa 18

2.1.1.2 Logarítmica 18

2.1.1.3 Gamma 19

2.1.1.4 Escala de cinza 20

2.1.1.5 Divisor de canais 20

2.1.2 Detecção de bordas 21

2.1.2.1 Passa-alta 22

2.1.2.2 Prewitt 23

2.1.2.3 Sobel 24

2.1.2.4 Laplaciano 25

2.1.2.5 Gradiente de Roberts 26

2.1.2.6 Média 26

2.1.2.7 Mediana 27

3 SISTEMAS DISTRIBUÍDOS E PARALELOS 29

3.1 ARQUITETURAS PARALELAS 29

3.2 MEDIDAS DE DESEMPENHO 33

3.2.1 *Speed-up* e Eficiência 33

3.2.2 Utilização de CPU 34

3.2.3 Granularidade 34

4 DESENVOLVIMENTO DO SOFTWARE 35

4.1 MODELAGEM DO SOFTWARE 35

4.1.1 Modelagem do Servidor 35

4.1.1.1 Pacote Server.Filter.Local 35

4.1.1.2 Pacote Server.Filter.Thread 36

4.1.1.3 Pacote Server.Filter.OpenMP 36

4.1.1.4 Pacote Server.Core 37
4.1.2 Modelagem do Cliente 38
4.2 TECNOLOGIAS UTILIZADAS NA IMPLEMENTAÇÃO DO SERVIDOR 40
4.2.1 RMI 41
4.2.2 Java Native Interface(JNI) 42
4.2.3 OpenMP 43
4.2.4 Thread 43
4.3 FUNCIONAMENTO DO SOFTWARE 44
4.4 RESULTADOS OBTIDOS 45
4.4.1 Granularidade 45
4.4.2 Utilização da CPU 46
4.4.3 Speed-Up 46
5 CONCLUSÃO E TRABALHOS FUTUROS 50
Referências 52

1 INTRODUÇÃO

Com a popularização dos computadores e o crescimento da ciência da computação, sugiram diversos métodos para resolução de problemas do mundo real. Com esse crescimento, muitas novas áreas de conhecimento foram criadas, tais como computação gráfica, processamento digital de imagens, realidade virtual, sistemas multimídia, entre outras. Mesmo com o avanço da tecnologia ainda temos dificuldades em lidar com problemas de otimização ou de alta quantidade de dados para processar, tais como, por exemplo, operações em matrizes. O foco deste trabalho está no processamento de alto desempenho, ou seja, desejamos processar a maior quantidade de dados possível minimizando o tempo gasto para o processamento total. Para tanto é proposto o desenvolvimento de uma ferramenta para processamento digital de imagens, pois através dela será possível observar as melhorias que a programação paralela pode resultar. Além disto, o software contará a possibilidade de usar acesso remoto.

A ferramenta proposta tem como principal objetivo possibilitar que computadores sem alta capacidade de processamento possam realizar tais operações no menor tempo possível. Para isso, essas operações são divididas em tarefas que são executadas em um servidor remoto. Entretanto, esta melhoria não será apenas para um cliente sem grande poder computacional, mas também para o servidor visto que serão utilizadas técnicas de programação paralela com o objetivo de maximizar a utilização de CPU no servidor.

O processamento paralelo é recomendado em regiões que demandam elevado esforço computacional (tempo, memória ou processamento) e consiste em dividir o fluxo do programa entre as centrais de processamento disponíveis, podendo ser núcleos (*cores*) ou processadores. Essa técnica pode ser aplicada em ambientes de memória compartilhada, distribuída e ambientes que combinam essas arquiteturas, denominados híbridos (TANENBAUM, 2012).

Em ambientes de memória distribuída, o fluxo do programa é dividido entre vários processadores que possuem recursos computacionais individuais. Nesse caso, o trabalho computacional é dividido para cada computador na forma de processos e a comunicação entre os computadores é feita através de alguma biblioteca de comunicação, como Message Passing Interface (Mpi,

2015). Estratégias paralelas baseadas na criação de múltiplos processos são, na maioria das vezes, utilizadas em *clusters* de computadores. Em ambientes de memória compartilhada, o fluxo do programa é dividido entre vários núcleos de processamento/processadores que compartilham uma única memória. Nesse modelo, o trabalho computacional é dividido em *threads*. Como alternativa para essa estratégia de paralelização destacam-se os padrões como: Java Threads, POSIX Threads (POSIX Threads, 2015) e o Open Multi-Processing (OpenMP, 2015).

O processamento paralelo em ambientes de memória compartilhada pode ser baseado em *threading* explícito e diretivas de compilação. Em *threading* explícito, o programador é o responsável por criar, definir a quantidade e destruir as *threads*. No uso de diretivas de compilação, o programador é responsável por indicar onde as *threads* serão criadas e de que forma elas devem ser executadas, mas o compilador é o responsável por fazer todo o gerenciamento. A ferramenta mais comum para essa abordagem e a utilizada neste trabalho é a OpenMP, pois deseja-se verificar as vantagens da utilização da programação paralela em memória compartilhada. Em ambientes híbridos, é possível dividir o fluxo do programa em duas camadas, a primeira relativa ao processamento distribuído e a segunda ao processamento compartilhado em cada processo.

1.1 OBJETIVOS

Este trabalho tem como propósito desenvolver um sistema para filtros de imagens executados em paralelo em uma arquitetura cliente/servidor, utilizando as tecnologias Método de Invocação Remota (Remote Method Invocation, RMI, 2015), *Threads* e OpenMP. Será contemplado um estudo de caso sobre a utilização de memória compartilhada, propondo duas soluções distintas: a primeira utilizando processamento paralelo com OpenMP, e a segunda, utilizando *threads.* Ambas serão comparadas e analisadas. Os resultados obtidos são comparados entre si e com o processamento sequencial.

1.2 ORGANIZAÇÃO DO TRABALHO

No Capítulo 2, é realizada uma revisão bibliográfica sobre processamento digital de imagens. No Capítulo 3, é realizada uma revisão bibliográfica sobre sistemas distribuídos e paralelos, e suas aplicações. No Capítulo 4, são descritos os resultados do estudo de caso sobre utilização de memória compartilhada com os filtros implementados. Por fim, no Capítulo 5, são listadas as conclusões sobre os desempenhos obtidos em geral, bem com sugestões de trabalhos futuros.

2 PROCESSAMENTO DIGITAL DE IMAGENS

Para realizar um processamento digital de imagens costuma-se seguir algumas etapas de processamento, que são: aquisição, realce, segmentação e classificação (GONZALES; WOODS ,1992). A aquisição trata da forma como as imagens são captadas de uma cena especifica, Para obtenção de imagens digitais são necessários dois elementos: dispositivos físicos captadores de imagem e digitalizador. Dispositivos físicos são sensíveis a espectros de energia eletromagnética e o digitalizador converte o sinal elétrico desses dispositivos para o formato digital. Estes elementos são chamados de sistemas de imageamento. O exemplo mais conhecido deste sistema são as câmeras, que por exemplo podemos encontrar em celulares. Outros são scanners e sensores presentes em satélites. Detalhes sobre aquisição de imagem podem ser vistos em (GONZALES; WOODS ,1992).

O realce, é a etapa responsável pelo melhoramento visual da imagem ou extração de características de interesse, onde aplicamos máscaras para revelar pontos de interesse que podem ter sofrido de algum tipo de degradação ou ruído, seja obtido pelo equipamento ou meio de transmissão.

A etapa de segmentação, é responsável, por encontrar objetos de interesse. Diferentemente do realce, aqui busca-se apenas a localização dos objetos, com significado semântico relevante para a aplicação em questão, para utilização na etapa de classificação. Esta, por sua vez, é responsável pelo uso de técnicas de reconhecimento de padrão.

Um conhecimento necessário para realizar o processamento de uma imagem é o relacionamento entre os pixels. A vizinhança de um pixel p na coordenada(x, y) possui quatro vizinhos, cujas coordenadas são dadas pela equação 1:

$$\mathrm{N}_4(\mathrm{x},\mathrm{y}) = \{(\mathrm{x}+1,\mathrm{y}),(\mathrm{x}-1,\mathrm{y}),(\mathrm{x},\mathrm{y}+1),(\mathrm{x},\mathrm{y}1)\} \quad (1)$$

Este conjunto é conhecido por vizinhança-4 de p, o qual comumente utiliza-se a expressão $N_4(p)$. Analogamente são adicionados e expressa os quatros vizinhos diagonais, chamada por vizinhança-8, com coordenadas e expresso por $N_8(p)$é dado pela Equação 2.

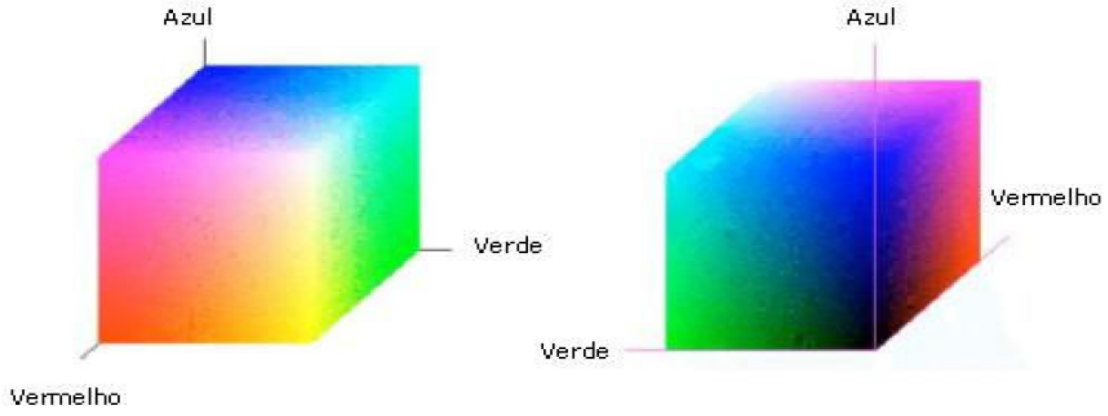
$$N_8(x,y) = N_4 \cup \{(x+1,y+1),(x-1,y-1),(x-1,y+1),(x+1,y-1)\} \quad (2)$$

Imagens digitais podem ser representadas de duas maneiras: em tons de cinza ou colorida. A primeira é obtida usando a intensidade da luz, normalmente, usando o preto para o menor brilho e branco para o brilho máximo, tendo os demais valores distribuídos de acordo com alguma escala determinada. Para imagens coloridas, além de observar a intensidade, utiliza-se a crominância da luz para identificar a medida da cor do pixel. Cada cor é definida como uma combinação de outras cores, entre os sistemas de representação mais conhecidos temos o RGB, cuja separação é dada por canais de cor, que são: *Red* (vermelho), *Green* (verde), *Blue* (azul); o HSV, que utiliza a matriz (hue), a saturação e o valor (HSV, 2015)

Visto que a aplicação também trabalha com imagens digitais coloridas, abordaremos o padrão RGB dado que este é o mais utilizado.

A representação das cores baseia-se em um sistema de coordenada cartesiana 3-D, tendo por portanto o subespaço formado pela combinação das três cores utilizadas pelo modelo RGB, como no cubo demonstrado na Figura 2.1.

Figura 2.1 - Modelo RGB em representação cúbica



Fonte: (GONZALES; WOODS ,1992)

Uma cena possui uma infinita quantidade de cores. Entretanto quando ela é digitalizada perde-se muito dessas cores, pois tem-se que utilizar uma escala de representação para os bits, e para isso utiliza-se alguns atributos do ambiente como: tonalidade, saturação e luminosidade ou brilho.Com isto, podemos realizar a combinação de cores necessária para gerar o mais

fielmente possível a imagem captada. Mais detalhes podem ser encontrados em (GONZALES; WOODS ,1992) e (PEDRINI; SCHWARTZ, 2008).

2.1 MÉTODOS DE FILTRAGEM

Os métodos de filtragem de imagem permitem melhor visualizar certas características de uma imagem, transformando-a em uma forma adequada à aplicação (PEDRINI; SCHWARTZ, 2008). Tais métodos estão estruturados em dois domínios: espacial e de frequência. O domínio espacial está relacionado ao processamento da vizinhança de um pixel sobre ele. No domínio espacial, o processamento de um ponto $p(x, y)$ depende do nível de cinza original do próprio ponto e de seus pixels vizinhos.

A convolução é uma técnica que atua sobre o domínio espacial por meio de uma matriz, denominada de janela ou máscara, que é aplicada sobre todos os pixels da imagem original afim de produzir outra imagem de saída. Esta técnica pode fazer uso de diferentes algoritmos: soma ponderada dos pixels vizinhos, mediana dos pixels, número de vizinhos distintos, entre outros. A cada elemento da janela está associado um valor numérico denominado de coeficiente. A aplicação da janela com centro na coordenada (x, y), sendo x uma linha e y uma coluna, consiste na substituição do valor do pixel central por um novo valor que depende de operações com os coeficientes vizinhos.

No domínio da frequência é aplicada na imagem uma função de transformação, por exemplo, a transformada de Fourier, nesta abordagem trabalhamos com as frequências da imagem utilizando filtros específicos como o *notch*, passa/rejeita banda, entre outros. A principal vantagem do domínio da frequência é que o mesmo permite tratar problemas de ruídos periódicos que não são possíveis de serem tratados eficientemente no domínio espacial.

O interesse nos métodos de processamento digital de imagens provém de duas áreas principais de aplicação: melhora das informações visuais para interpretação humana e processamento de dados para armazenamento, transmissão e representação. Uma imagem pode ser definida como uma função bidimensional, $f(x, y)$, em que x e y são coordenadas espaciais, e a amplitude de f em qualquer par de coordenadas é chamada de intensidade ou nível de cinza da imagem. Quando x,y e os valores de intensidade de f são

quantidade finitas e discretas, chamamos de imagem digital.

Os filtros podem ser divididos em dois domínios, o primeiro, o da frequência, onde uma função pode ser convertida do domínio do tempo para o da frequência através de um operador matemático chamado genericamente de transformada integral. Um exemplo é a transformada de Fourier, que decompõe uma função na soma de um (potencialmente infinito) número de componentes senoidais, produzindo um espectro de frequências. A transformada inversa correspondente converte esse espectro de volta para o domínio do tempo, ou seja, para a função original. Existem ainda transformadas que permitem a conversão para um domínio misto do tempo e da frequência ao mesmo tempo, como é o caso da transformada de *wavelet*. No domínio espacial, pode-se realizar algumas operações. São elas: ponto-a-ponto, operações de vizinhança e transformação geométrica.

A operação ponto-a-ponto consiste em alterar o pixel da imagem com base na sua intensidade, este tipo de processo pode ser descrito em termos de uma transformada, como por exemplo a Equação 5; onde zé a intensidade de um pixel na imagem original e s é a intensidade mapeada do pixel na imagem que desejamos alterar.

$$s = T(z) \quad (5)$$

Na operação por vizinhança, teremos um conjunto de coordenadas centradas em um ponto(x, y), no qual teremos o processamento por vizinhança. Este processamento gera um pixel na mesma coordenadas em uma imagem de saída, de forma que o valor desse novo pixel é determinado por uma operação específica envolvendo os pixels da imagem original. Por exemplo, pode-se citar um filtro de média:

$$g(x,y) = \frac{1}{N}\sum_{x=0}^{x=n}\sum_{y=0}^{y=n} f(x-k, y-1), (k,1) \in IMG \quad (6)$$

A imagem g é criada variando-se as coordenadas (x, y) de forma que o centro da vizinhança mova-se sobre cada pixel aplicando a máscara pretendida, no caso, a média.

As operações de transformações geométricas, modificam a relação espacial entre os pixels de uma imagem. Essas transformações são do tipo *rubber sheet* (superfície de borracha), porque podem ser vistas de forma

análoga à "impressão" de uma imagem numa superfície de borracha que pode ser esticada de acordo com um conjunto de regras predefinidas. Estas operações não são alvos deste trabalho, pois estas tendem a distorcer imagens devido à ampliação, redução, rotação, entre outras técnicas utilizadas.

Para uma melhor compreensão, será utilizada uma abordagem matemática para os filtros utilizados no projeto e os mesmos serão agrupados por área de atuação (por exemplo, melhoramento de contraste). Desta forma, pode-se verificar a equação matemática que deu origem a máscara. Também será apresentado um algoritmo base para processamento digital de imagens. Neste algoritmo será demonstrado o procedimento padrão para aplicação da função a ser utilizada.

O Algoritmo 1 descreve como realizar uma convolução em uma imagem, dado uma máscara representada pela função $g(x)$. Como podemos observar, no passo 4 do algoritmo apresentado, tem-se a convolução da máscara na imagem original para obtermos o resultado pretendido. As funções utilizadas para máscaras foram divididas em áreas de aplicação, agrupadas em máscaras para contraste, para detecção de bordas e ruído.

Algoritmo 1 -Algoritmo para processamento digital de imagens

ALGORITMO Processamento de Imagens	
ENTRADA:	Imagem img,
	Função de Máscara g(x)
SAÍDA:	Imagem resultante da convolução.
1.	Criar matriz Aux de imagem
2.	**Para cada** $x \in img(x.y)$ **faça**
3.	**Para cada** $y \in img(x.y)$**faça**
4.	$\mathrm{Aux}(x, y) =$ Aplicar g(x) em $\mathrm{Img}(x, y)$
5.	**Retornar** Aux

Fonte: autoria própria

Para auxiliar nos resultados produzidos pelos filtros será utilizada a Figura 2.2. Esta imagem é bastante utilizada para demonstrações dos filtros, conhecida como Lena.

2.1.1 Transformações de Intensidade

As transformações de intensidade são mais utilizadas para apresentação da imagem ao olho humano. Através de transformações é possível visualizar

algumas características da imagem que antes não podiam ser vistas mas estavam na imagem.

Figura 2.2 Lena

Fonte: (GONZALES; WOODS ,1992)

As máscaras escolhidas para a aplicação foram: inversa, gamma, logaritmo, divisor de canais e escala de cinza. L-1, irá corresponde ao valor máximo do pixel, normalmente, 255. As funções f_{red}, f_{green} e f_{blue} correspondem aos *pixels* de cada camada de frequência do padrão RGB.

2.1.1.1 Inversa

A função inversa (Equação 7) inverte os valores dos *pixels* da imagem de entrada Os *pixels* de valor 0 são mapeados para L-1 e vice-versa. Os valores intermediários são mapeados para outros valores intermediários, seguindo o comportamento da reta da função inversa.

$$g(x,y) = L - 1 - T[(f(x,y)] \quad (7)$$

2.1.1.2 Logarítmica

A função logarítmica (Equação 8) possui como objetivo mapear intervalos estreitos de baixa intensidade em intervalos mais largos de valores de alta intensidade. Ocorrendo o oposto para valores mais altos.

Figura 2.3 Aplicação da equação 7 na Figura 2.2

Fonte: Autoria Própria

Sua principal utilização é expandir os valores mais escuros da imagens ao mesmo tempo em que comprimimos os valores de níveis mais altos. Por fim, c, corresponde ao fator de ajustes, normalmente variando entre 0 e 255. O resultado pode ser visto na Figura 2.4.

$$g(x,y) = \ c * \log\left(1 + \left(\frac{f(x,y)}{L-1}\right)\right) \quad (8)$$

Figura 2.4 Aplicação da equação 8 na Figura 2.2

Fonte: autoria própria

2.1.1.3 Gamma

Esta função (Equação 9) é utilizada para realçar as áreas claras da imagem, observa-se que a inclinação da função é maior próximo a áreas claras da imagem. Os multiplicadores c e γ, correspondem aos fatores de ajustes,

normalmente variando entre 0 e 255. O resultado da utilização desta transformação pode ser visualizado na Figura 2.5.

$$g(x,y) = \mathrm{c} * \left(\frac{f(x,y)}{L-1}\right)^{\gamma} \quad (9)$$

Figura 2.5 Aplicação da equação 9 na Figura 2.2

Fonte: autoria própria

2.1.1.4 Escala de cinza

Esta função (Equação 10) é utilizada para transformar uma imagem colorida em tons de cinza, utilizando como base o padrão RBG já apresentado.

Os valores das constantes em cada uma das frequência são atribuídos devido à como olho humano visualiza essas cores. Mais detalhes pode ser visto em (GONZALES; WOODS ,1992).

$$g(x,y) = 0.299 * f_{Red}(x,y) + 0.587 * f_{Green}(x,y) + 0.114 f_{Blue}(x,y) \quad (10)$$

2.1.1.5 Divisor de canais

Para esta função são criadas três novas funções, separando os canais de cores da imagem. Desta forma tem-se que $g(x,y)$ pode ser uma das três equações (Equação 11, Equação 12 ou Equação 13). O resultado produzido por cada equação pode ser visualizado na Figura

$$g(x,y) = f_{Red}(x,y) \quad (11)$$

$$g(x,y) = f_{Green}(x,y) \quad (12)$$

$$g(x,y) = f_{Blue}(x,y) \quad (13)$$

Figura 2.6 Aplicação da equação 11 na Figura 2.2

Fonte: autoria própria

Figura 2.7 Aplicação da equação 12 na Figura 2.2

Fonte: autoria própria

2.1.2 Detecção de bordas

A detecção de bordas é uma técnica bastante usada para processamento digital de imagens, com a finalidade de detectar pontos em uma imagem digital em que a intensidade de luz muda. Essas mudanças repentinas em imagens demonstram que algo importante está acontecendo na cena, como por exemplo, a descontinuação da profundidade, orientação da superfície, mudanças na propriedade dos materiais, entre outros.

É bastante utilizada na área de visão computacional para extração de

características pertencentes à cena, pois sua utilização reduz significativamente a quantidade de dados a serem processados, visto que descarta informações que são consideradas menos relevantes para o escopo do problema. A detecção pode ter alguns problemas quando imagem possui ruído, sejam provenientes da digitalização, compressão ou através do mecanismo de captura. Para amenizar estes problemas, utiliza-se alguma técnica de redução de ruído.

Devido a isto, as máscaras selecionadas foram: passa-alta, Prewitt, Sobel, Laplaciano, gradiente de Roberts, média, mediana, para facilitar a representação de algumas destas máscaras será utilizado uma figura para representar as máscaras a serem aplicadas podendo conter também sua equação matemática.

Figura 2.8 Aplicação da equação 13 na Figura 2.2

Fonte: autoria própria

2.1.2.1 Passa-alta

Filtros do tipo passa-alta produzem uma imagem em que os componentes de baixa frequência são atenuados. A frequência limite a que as baixas frequências são atenuadas varia com a definição dos coeficientes utilizados. Este tipo de filtragem é utilizado para realce de bordos ou contornos uma vez que os contornos de uma imagem estão relacionados com as altas frequências espaciais da imagem. A Figura 2.9 exemplifica uma possível

máscara passa-alta. A Figura 2.10 demonstra a utilização da Figura 2.9 como máscara de convolução

Figura 2.9 Exemplo de uma máscara de convolução passa-alta

-1	-1	-1
-1	8	-1
-1	-1	-1

Fonte: autoria própria

Figura 2.10 Aplicação da máscara 2.9 na Figura 2.2

Fonte: autoria própria

2.1.2.2 Prewitt

Este operador utiliza duas matrizes 3x3(Figura 2.11). É usado particularmente para detecção de arestas, em temos simples, o operador calcula o gradiente da intensidade da imagem em cada ponto, nas direções horizontal e vertical. Os módulos dos valores obtidos são somados para gerar a intensidade final. O resultado (Figura 2.12) mostra como abruptamente ou suavemente a imagem muda naquele ponto.

Figura 2.11 Operadores de Prewitt

-1	0	1
-1	0	1
-1	0	1

-1	-1	-1
0	0	0
1	1	1

Fonte: autoria própria

Figura 2.12 Aplicação das máscaras 2.11 na Figura 2.2

Fonte: autoria própria

2.1.2.3 Sobel

O operador de Sobel calcula o gradiente da intensidade da imagem em cada ponto, dando a direção da maior variação de claro para escuro e a quantidade de variação nessa direção. Assim, obtém-se uma noção de como varia a luminosidade em cada ponto, de forma mais suave ou abrupta.

Com isto consegue-se estimar a presença de uma transição claro-escuro e de qual a orientação desta. Como as variações claro-escuro intensas correspondem a fronteiras bem definidas entre objetos, consegue-se fazer a detecção de contornos.

Este utiliza duas matrizes 3x3(Figura 2.13), uma na direção horizontal, -G_x, e outra na vertical, G_y, obtendo o seu resultado pela magnitude das matrizes, representados pela Equação 14. A Figura 2.14 demonstra o resultado obtido pela utilização do operador de Sobel.

$$g(x,y) = \sqrt{g(x)^2 + g(y)^2} \quad (14)$$

Figura 2.6 Matrizes de Sobel

1	2	1
0	0	0
-1	-2	-1

Gx

-1	0	1
-2	0	2
-1	0	1

Gy

Fonte: autoria própria

Figura 2.14 Aplicação da equação 14 na Figura 2.2

Fonte: autoria própria

2.1.2.4 Laplaciano

O laplaciano é o operador da segunda derivada (Figura 2.15), utilizado como filtro isotrópico para aguçamento de imagens. Ou seja, estamos interessados em filtros que dependam da direção das continuidades da imagem.

Figura 2.15 Matriz Laplaciano

0	-1	0
-1	4	-1
0	-1	0

Fonte: autoria própria

Figura 2.16 Aplicação da matriz 2.15 na Figura 2.2

Fonte: autoria própria

2.1.2.5 Gradiente de Roberts

O gradiente de Roberts utiliza duas matrizes 2x2(Figura 2.16), uma na direção horizontal, G_x, e outra na vertical, G_y, tendo o seu resultado dado pela magnitude do componentes através da Equação 14. Devido às máscaras serem 2X2, ao invés de 3X3, o gradiente de Roberts é muito mais sensível ao ruído do que as demais.

Figura 2.16 Gradientes X e Y de Roberts

0	1
-1	0

Gx

1	0
0	-1

Gy

Fonte: autoria própria

2.1.2.6 Média

Um dos mais simples filtros lineares é implementado através de uma operação local de média onde o valor de cada pixel é substituído pela média de todos os valores na sua vizinhança local. Em geral, num filtro de média, as ponderações utilizadas são valores iguais, utiliza-se a Equação 15.

Figura 4.16 Aplicação das matrizes da Figura 2.16 na Figura 2.2

Fonte: autoria própria

$$g(x,y) = \frac{1}{N}\sum_{x=0}^{x=n}\sum_{y=0}^{y=n} f(x-k, y-1), (k,1) \in IMG \tag{15}$$

Figura 4.17 Aplicação da equação 15 na Figura 2.2

Fonte: autoria própria

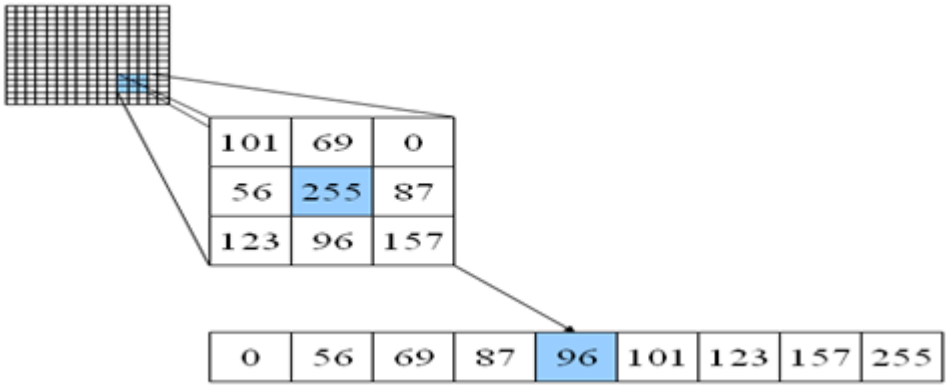
2.1.2.7 Mediana

Para calcular a filtragem por mediana numa vizinhança de um pixel P, deve-se selecionar o valor do pixel e dos seus vizinhos, após isso determinar a mediana, (a mediana é calculada ordenando os valores dos *pixels* vizinhos e alvo por ordem numérica e substituindo o valor do pixel alvo pelo valor que se encontra no meio da ordenação) e finalmente atribuir o valor da mediana ao

equivalente de *P* na imagem resultante, este processo pode ser observado na Figura 2.9. Este tipo de filtro é não linear e é útil na remoção de *pixels* ou linhas isolados, preservando a resolução espacial da imagem. Apresentando bons resultados quando o ruído é do tipo binário, este filtro tem dificuldades quando o ruído é Gaussiano. Quando o número de *pixels* com ruído é maior ou igual a metade dos números de *pixels* na vizinhança, o seu desempenho está comprometido, esta máscara está definida na Equação 16.

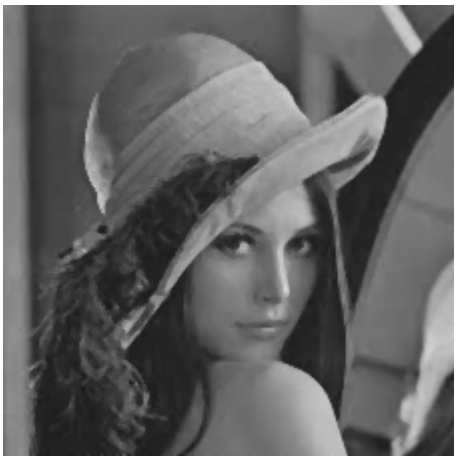
$$g(\mathrm{x},\mathrm{y})mediana\{f(x-\ k,y-1),(\mathrm{x},1) \in W\ \} \quad (16)$$

Figura 2.9 Mediana



Fonte: autoria própria

Figura 4.17 Aplicação da equação 16 na Figura 2.2



Fonte: autoria própria

3 SISTEMAS DISTRIBUÍDOS E PARALELOS

Os avanços da ciência e tecnologia aumentaram a demanda por plataformas de hardware e software que possam oferecer maior capacidade de processamento para a execução de aplicações associadas a problemas complexos do mundo real. Nesse contexto, a programação paralela é uma técnica que tem sido utilizada para tratar tais aplicações nos casos em que um único processador requeira muito tempo para processá-las sequencialmente (Ivanov, 2006).

Tradicionalmente, a programação paralela foi motivada pela resolução/simulação de problemas fundamentais da ciência/engenharia de grande relevância científica e econômica, denominados como Grande Desafios de Problemas (*Grand Challenge Problems*- GCPs). Estas aplicações requerem ou um grande poder de computação ou o processamento de grandes quantidades de informação. Alguns exemplos são os serviços associados a tecnologias multimídia e telecomunicações, a computação gráfica, realidade virtual e diagnóstico médico assistido por computador. Neste trabalho, uma aplicação de processamento digital de imagens é apresentada para demonstrar as melhorias que a computação de alto desempenho pode oferecer, visto que os dois principais motivos para utilizar programação paralela são:reduzir o tempo necessário para solucionar um problema e resolver problemas mais complexos e de maior dimensão.

Operações de processamento digital de imagens são consideradas como exemplos significativos de aplicações que demandam uma grande quantidade de recursos computacionais para serem executadas, pois são compostos por uma grande quantidade de dados, normalmente, dispostos em forma matricial. As operações sobre matrizes requerem um grande poder computacional para serem executadas. Por esta razão é um ótimo foco para visualizar os resultados da aplicação da computação de alto desempenho.

3.1 ARQUITETURAS PARALELAS

As arquiteturas paralelas permitem a execução das aplicações em menor tempo, através da execução em paralelo de diversas tarefas. Este

paralelismo por sua vez, pode ser obtido de diferentes formas, seja através de linguagens de programação. Ou, através de hardware, em que temos uma máquina física com várias CPUs acopladas (*on-board*) ou um conjunto de máquinas monoprocessadas interconectadas. Os níveis de paralelismo são, segundo (FLYNN, 1992): nível de tarefa, nível de programa, nível de instruçãoe a nível de bit. Este último por sua vez, está divido em bit-serial *e* bit-paralelo (Tabela 1).

Tipos	Características
Nível de tarefa	Várias tarefas independentes são executadas simultaneamente no mesmo computador.
Nível de programa	Vários programas são executados simultaneamente para resolver um único problema comum
Nível de instrução	O processamento de uma operação pode ser dividida em operações que podem ser sobrepostas usando uma técnica de *pipelining*
Bit-serial	Quando os bits de uma palavra são tratados um após o outro
Bit-paralelo	Quando os bits são tratados ao mesmo tempo

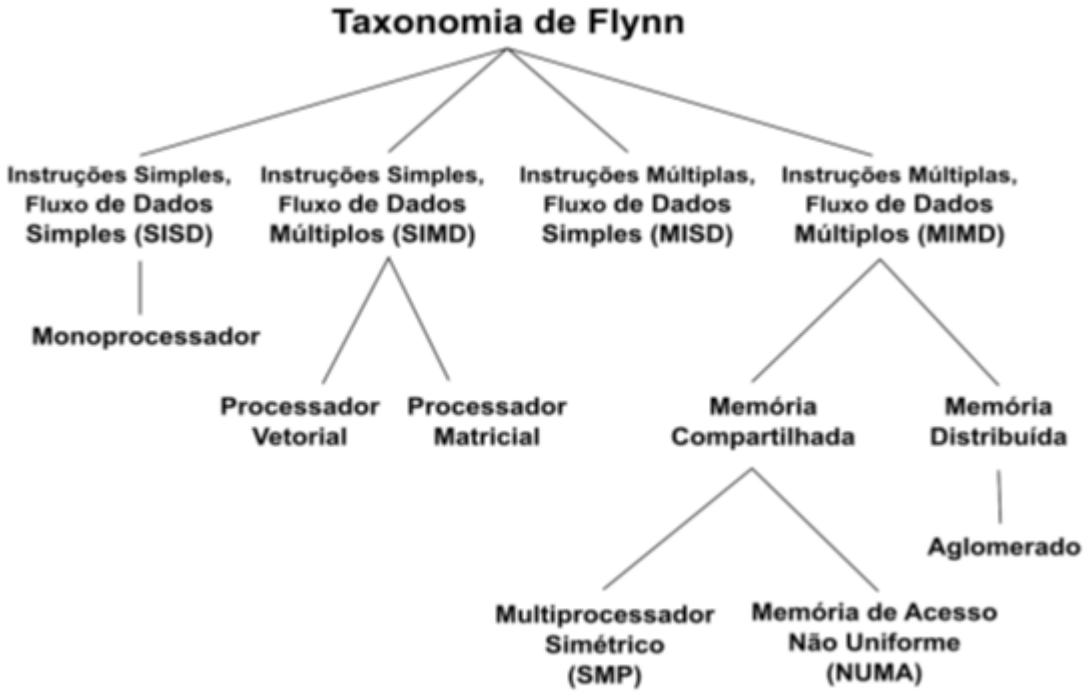
Tabela 1 - Tipos de Paralelismo

Para classificar a arquiteturas paralelas foi utilizado a taxonomia de Flynn (FLYNN, 1992), que possui como critérios o fluxo de instruções de processamento e o fluxo de dados processados. Portanto, pode-se ter quatro classificações possíveis, Única Instrução Único Dado (SISD - *Single Instruction, Single Data*), Múltiplas Instruções Único Dado (MISD – *Multiple Instruction, Single Data*), Única Instrução Múltiplos Dados (SIMD - *Single Instruction, Multiple Data*) e Múltiplas Instruções Múltiplos Dados(MIMD - *MultipleInstruction, Multiple Data*), estes modelos estão demonstrados na Figura 3.1.

No modelo UIUD (Figura 3.2), temos apenas um processador e uma entrada para o fluxo de memória. Este modelo utiliza apenas uma unidade de controle. Em computadores ou *mainframes* antigos, apesar de alguns possuírem múltiplas unidades de processamento sequencial, o fluxo de dados

não tinha qualquer relação ou dependência, em outras palavras é um modelo baseado na arquitetura de von Neumman (TANENBAUM, 2012).

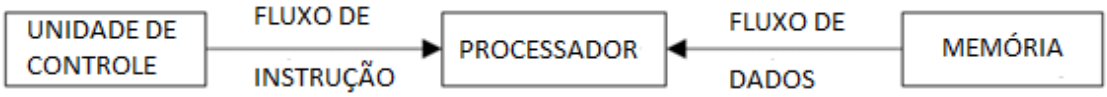
Figura 3.1 Taxonomia de Flynn



Fonte: Flynn, 1992

No modelo MIUD (Figura 3.3), observamos a presença de N processadores, portanto podemos processar N dados. Entretanto, como só possuímos um fluxo de memória, todas as instruções são executadas em cima do mesmo dado. Este tipo pode ser aplicado para resoluções de problemas específicos, tais como cálculos matemáticos sólidos e processamento de vários algoritmos de quebra de criptografia (FLYNN, 1992).

Figura 3.2 Modelo UIUD

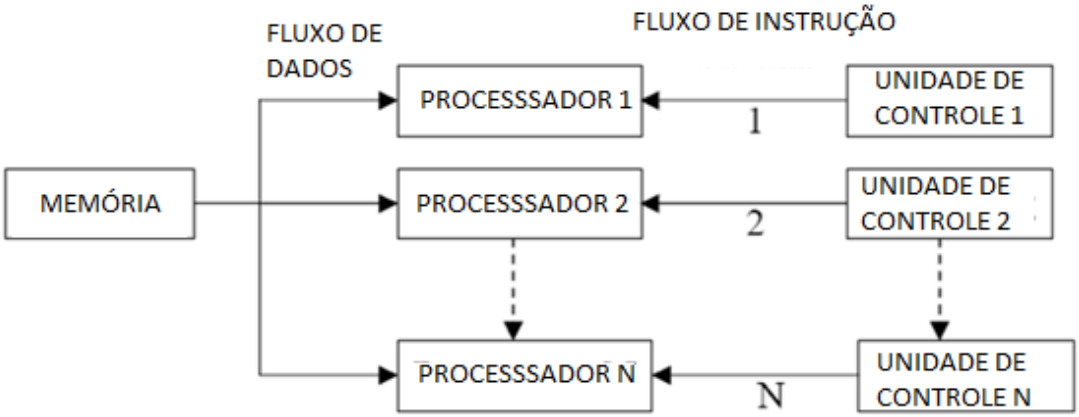


Fonte: Autoria Própria

O modelo UIMD (Figura 3.4), trata de uma memória compartilhada ou interconexão de rede que servirá de alimentação para os processadores, são utilizados para a resolução de problemas. Computadores SIMD computacionalmente intensivos da área científica e de engenharia, em que existem estruturas de dados como vetores e matrizes. Essas máquinas são

caracterizadas por possuírem apenas uma unidade de controle que executa uma instrução de cada vez, mas cada instrução opera sobre vários dados (FLYNN, 1992).

Figura 3.3 Modelo MIUD

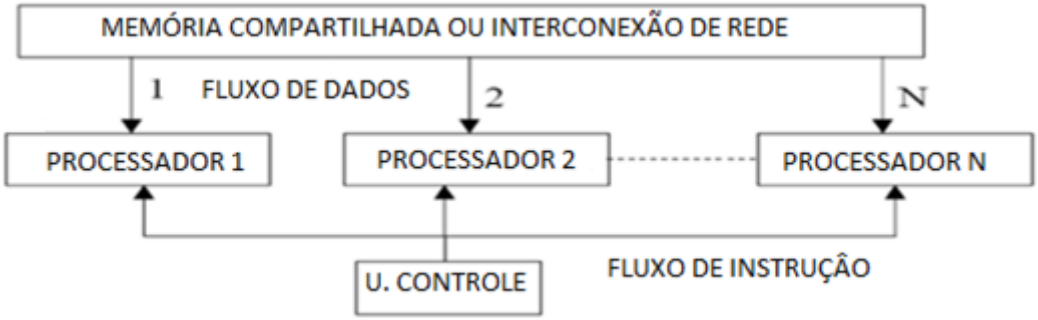


Fonte: Autoria Própria

É uma classe importante de processadores, devido a fatores como: simplicidade de conceitos e programação, regularidade da estrutura, facilidade de escalabilidade em tamanho e desempenho, aplicação direta em uma série de aplicações que requerem paralelismo para obter o desempenho necessário, os processadores executam em sincronia a mesma instrução sobre dados diferentes. Utilizam-se vários processadores especiais muito mais simples, organizados em geral de forma matricial (FLYNN, 1992) Muito eficiente em aplicações onde cada processador pode ser associado a uma sub-matriz independente de dados (processamento de imagens, algoritmos matemáticos, etc.). Temos como exemplo as placas de vídeos que trabalham com GPU (FLYNN, 1992).

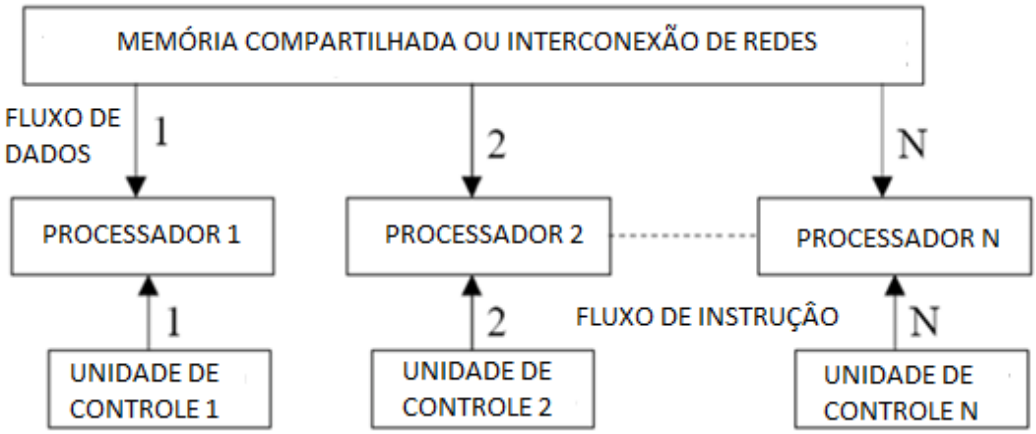
O modelo MIMD (Figura 3.5), é atualmente utilizado na maioria dos processadores atuais, visto que é possível realizar N tarefas diferentes em N dados paralelamente. Desta forma é possível realizar o paralelismo de instruções independentes para cada processador, com a vantagem de executar várias tarefas ao mesmo tempo, contudo existe o problema da necessidade de sincronização em tempo real ao final de uma aplicação simples.

Figura 3.4 Modelo UIMD



Fonte: Autoria Própria

Figura 3.5 Modelo MIMD



Fonte: Autoria Própria

3.2 MEDIDAS DE DESEMPENHO

Este trabalho se propõe a investigar a viabilidade da disponibilização de um serviço remoto em que filtros de imagens são executados em paralelo, portanto, um dos assuntos a serem tratados são as medidas de desempenho, as quais podem ser: vazão, utilização de CPU, tempo de resposta, confiabilidade, disponibilidade, granularidade, *speed-up,* etc. Nos tópicos a seguir daremos ênfase aos quais são utilizados como métrica deste trabalho.

3.2.1 *Speed-up* e Eficiência

O *Speed-up* mede a razão entre o tempo gasto para execução de um

algoritmo ou aplicação em um único processador e o tempo gasto na execução em ***n*** processadores (Equação 17).

$$S(n) = \frac{T(1)}{T(n)} \tag{17}$$

Os valores da eficiência normalmente variam entre 0 e 1.Por exemplo, se tiver uma tarefa que leve 40 minutos para ser realizada com um processador, ao paralelizamos com 4 processadores é esperado que o tempo de computação baixe para 10 minutos. E intrinsecamente, é possível acontecerem dois casos especiais. O primeiro, ter um programa que seja impossível de paralelizar, pois são puramente sequenciais. E o segundo, a super-paralelização, quando o algoritmo for altamente paralelizável pode acontecer que este valor supere a 100% de eficiência, desta forma podemos calcular a eficiência utilizando a Equação,

$$E(n) = \frac{S(n)}{n} \tag{18}$$

3.2.2 Utilização de CPU

É a fração do tempo em que o recurso permanece ocupado atendendo aos pedidos dos usuários. Desta forma, pretendemos observar o quanto o servidor poderá ficar sobrecarregado de acordo com a utilização de *threads*.

3.2.3 Granularidade

A granularidade trata do problema da decomposição da computação de modo a obter a máxima performance. O número e o tamanho das tarefas em que um problema é decomposto determina a granularidade da decomposição. A granularidade é dividida em três tipos: fina, média, grossa. Na granularidade fina, a decomposição é realizada para um grande número de pequenas tarefas. Na granularidade grossa a decomposição é realizada para um pequeno número de grandes tarefas. Esta é uma medida importante, pois o tempo de execução de uma tarefa tem que compensar os custos de criação, comunicação e sincronização, o que pode ser um problema caso a granularidade seja fina. O tempo de ocupação dos processadores disponíveis

deve ser o máximo possível, o que pode ser um problema quando é utilizada uma granularidade grossa.

4 DESENVOLVIMENTO DO SOFTWARE

Este capítulo apresenta as discussões relativas ao projeto, apresentando a modelagem do software, a implementação dos algoritmos e os resultados obtidos da programação sequencial e da programação paralela, no qual serão observados os fatores granularidade, utilização de CPU e *speed-up*.

4.1 MODELAGEM DO SOFTWARE

Esta seção descreve o modelo sobre o qual foi desenvolvido o software, divido em duas subseções, cliente e servidor. Desta forma, pode-se detalhar melhor cada parte da modelagem.

4.1.1 Modelagem do Servidor

O servidor desenvolvido pode ser descrito pelo diagrama de pacotes da UML, feito com o auxílio do *software* Astah. Na Figura 4.1, pode-se observar os subpacotes *Core* e *Filter*. Este último, por sua vez é subdividido em *Thread*, *Local* e OpenMP. Para explicar os pacotes utiliza-se uma abordagem *bottom-up.*

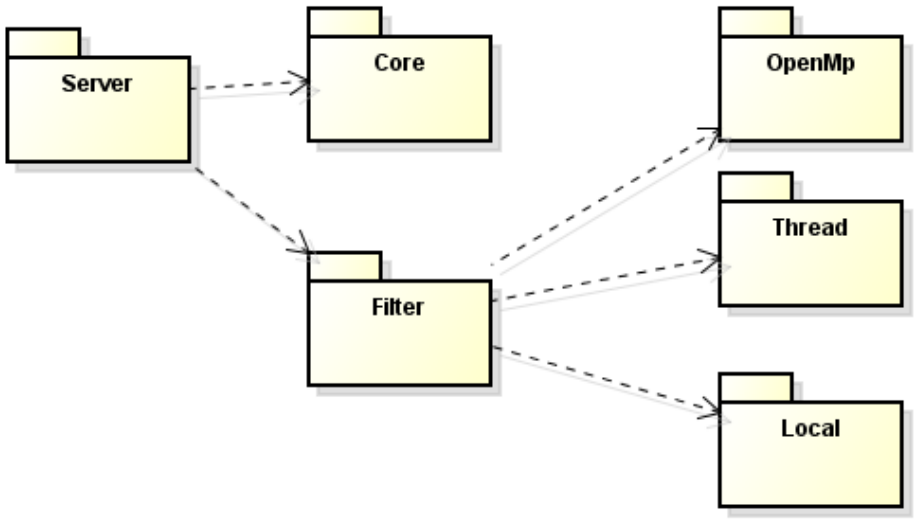
4.1.1.1 Pacote Server.Filter.Local

Neste pacote, as classes referentes aos filtros foram agrupadas em dois grupos, o de bordas, fronteiras e vizinhança e os de contraste. Como é uma abordagem local, todos seguiram o algoritmo demonstrado no capitulo de processamento digital imagens.

4.1.1.2 Pacote Server.Filter.Thread

Neste pacote, as classes referentes aos filtros foram agrupadas da mesma forma que no pacote *Server.Filter.Local*. Entretanto é utilizada uma abordagem diferente para o tratamento da imagem, pois para utilizar as *threads* foi implementado um vetor *threads*. Este, por sua vez, recebe um código para o filtro desejado e, então, através de polimorfismo o vetor de threads torna-se um vetor do tipo do filtro que deseja-se utilizar, visto que agora o processamento dos filtros é realizado com uma classe filha da classe *Thread*.

Figura 4.1 - Diagrama de Pacotes Servidor



Fonte: autoria própria

4.1.1.3 Pacote Server.Filter.OpenMP

Neste pacote, as classes referentes aos filtros foram agrupadas da mesma forma que no pacote *Server.Filter.Local.* A abordagem utilizada foi criar uma classe para carregamento da DLL através de um bloco *static* e com as declarações dos métodos nativos. Para cada classe que implementa algum filtro é criado vetores de componentes para serem passados ao Java NativeInterface, que é uma tecnologia utilizada para execução de códigos

nativos escritos em C/C++ pela plataforma Java, mais detalhes serão abordados na seção 4.3.2.

4.1.1.4 Pacote Server.Core

Neste pacote, ilustrado pela Figura 4.2, estão contidas as classes e as interfaces essenciais para o funcionamento do processamento das imagens no servidor. O projeto conta com três classes e duas interfaces.

A primeira interface a ser abordada remoteFilters, essa interface é necessária devido à utilização do RMI, pois devemos definir previamente quais serão os métodos de processamento remoto a serem implementados no servidor. Nesta interface, temos três tipos de chamadas: a chamada de filtro, na qual é invocado o filtro remoto que o cliente deseja utilizar, o modelo de processamento, onde o cliente determina qual tipo de processamento deseja (que podem ser *Local*, *Thread* ou Open MP). E por fim (caso seja *Thread)*,a quantidade do *pool* de *threads* desejadas.

A segunda interface, trata dos métodos necessários caso deseje-se implementar um novo filtro. Desta forma é necessário implementar duas funções: a *Apply* e *getBoundaries*, o segundo sendo um método abstrato. Desta forma, caso deseja-se implementar um novo filtro, deve-se implementar a interface proposta.

No método A*pply* tem-se como parâmetro uma imagem e como retorno um vetor de imagens. Enquanto o método, *getBoundaries,* é utilizado caso seja necessário limitar algum processamento na imagem, por exemplo em filtros gamma, quando desejamos limitar o fator multiplicativo do filtro.

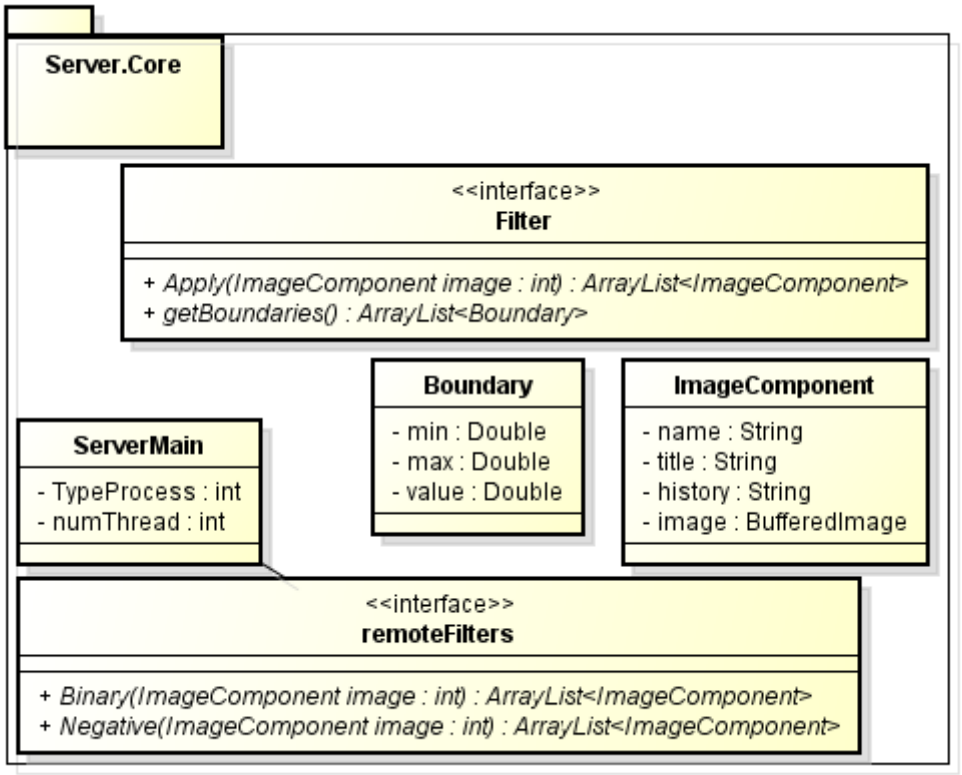
Dentre as classes deste pacote, a primeira a ser abordada é a *imageComponent*, nela tem-se o nome e o título da imagem, que serão utilizadas para demonstrar qual filtro e em qual imagem foi processado, um dado para manter o histórico das operações realizadas naquela imagem até o momento, e por fim, uma bufferedImage a imagem a qual pretende-se processar no servidor, por este motivo recebe o atributo *transient*, para esta classe também é necessário utilizar a interface *serializable*, devido à operação

de *marshling* e *unmarshling,* requisitadas para a utilização do RMI, pois esta irá realizar as trocas de informações de dados não primitivos,

A segunda classe a ser abordada é a classe *Boundary*, que é utilizada para as restrições na imagem, como já comentado previamente.

Por fim, a terceira classe é a do servidor em que se utilizam todos os filtros desenvolvidos para a aplicação através da implementação da interface *remoteFilter*, já previamente comentada, e utilizando a herança do objeto *UnicastRemoteObject.* A principal vantagem desse método é poder atualizar os métodos no servidor enquanto o cliente não precisará de *update* para manutenções de erro, visto que a interface não será modificada.

Figura 4.2 - Diagrama de classes do pacote *Server.Core*



Fonte: autoria própria

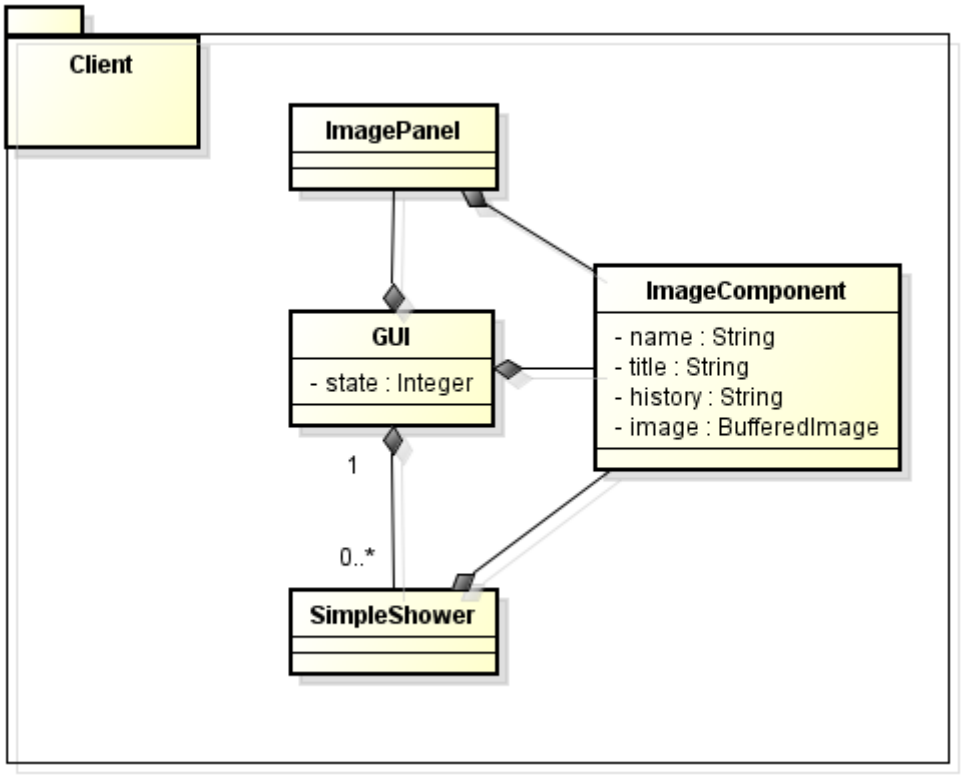
4.1.2 Modelagem do Cliente

A modelagem do cliente é dividida apenas em classes de *UserInterface,* como pode ser observado na Figura 4.3. Este pacote consiste de cinco classes: *GUI, ImageComponent, ImagePanel* e *SimpleShower*.

A classe *ImageComponent* serve para interface de transmissão do cliente para o servidor. Portanto, não existem diferenças entre as classes implementadas no servidor e no cliente.

Já as demais classes referem-se à interface de usuário será apresentada além do funcionamento da classe a sua *User Interface* proposta. A primeira classe a ser abordada é a classe *ImagePanel.* Esta classe é utilizada para fazer um container para imagens definindo, portanto, a região ocupada pela Imagem no *JFrame* utilizando a classe *Graphics* nativa do Java.

Figura 4.3 Diagrama de classes do pacote *Client*



Fonte: autoria própria

A segunda classe a ser abordada é a classe *SimpleShower.* Esta é o visualizador dos filtros selecionados, como pode ser observado na Figura 4.4.

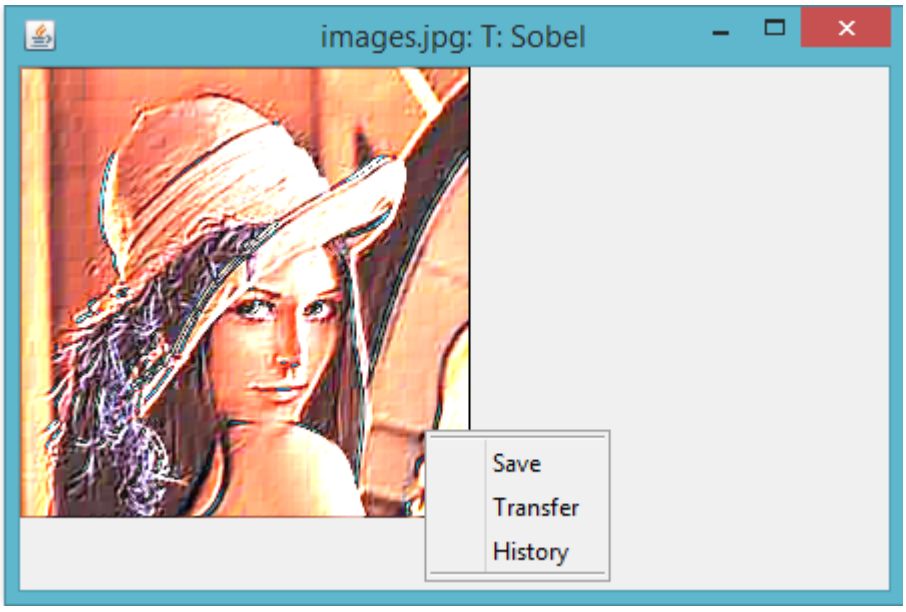
O *SimpleShower*, além de demonstrar o resultado, utilizada no título da imagem a operação solicitada, e caso seja processada em modo *Thread,* possuirá o prefixo "T:" antes do nome do filtro aplicado. Além disto, possui um menu interno, acessível através do clique do botão direito do mouse, disponibilizando três opções: *Save, Transfer* e*History.* Na primeira opção, *Save,* o usuário pode salvar a imagem obtida pela utilização do filtro. Na

segunda opção, *Transfer,* o usuário pode transferir a imagem do *SimpleShower* para a GUI, desta forma poderá utilizar um outro filtro que também deseje. Por fim, na terceira opção, *History,* o cliente pode visualizar os filtros que foram aplicados na imagem até o momento.

Por fim, a última classe é a GUI, demonstrada na Figura 4.4, nela possuímos 4 menus, File, Filters, Window e Help.

O menu help possui apenas um *about* em referência ao projeto. O menu window permite ao usuário fechar todos os *SimpleShower* ou utilizar a tecla F4, para realizar o mesmo processamento, tal como a prioridade de visualização da GUI sobre os *Simpleshower*. No menu*Filter,* estão localizados os filtros implementados divididos por categoria, por fim, no menu *File,* é possível abrir e salvar imagens e alterar o modo de operação para o processamento das imagens, tal como escolher a quantidade de *Threads* desejadas para o processamento das imagens.

Figura 4.4 SimpleShower



Fonte: autoria própria

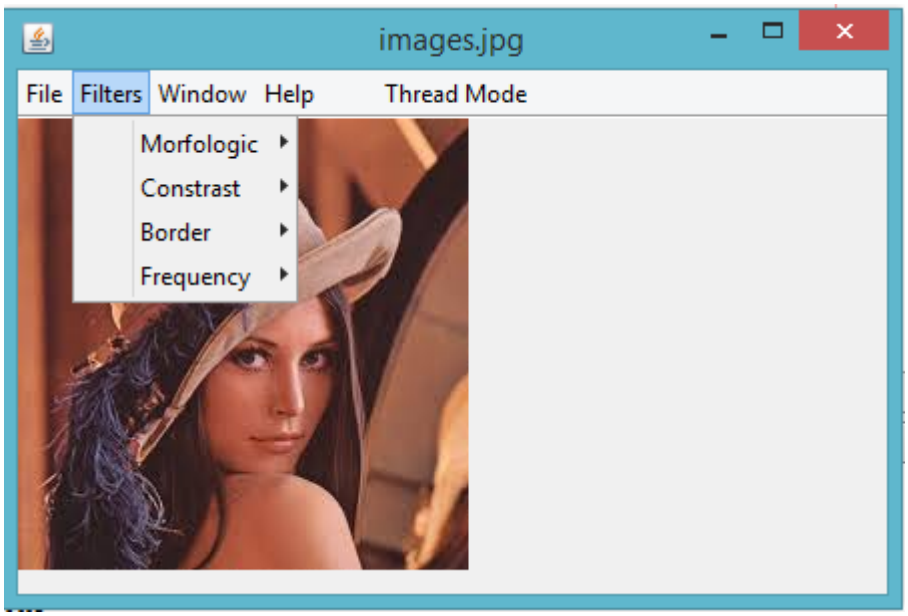
4.2 TECNOLOGIAS UTILIZADAS NA IMPLEMENTAÇÃO DO SERVIDOR

Nesta seção serão abordadas as tecnologias RMI, JNI, OpenMP e Threads e o porquê de sua utilização no projeto.

4.2.1 RMI

Uma das características da ferramenta proposta é ser acessada via Internet, e por esta razão iremos usar uma arquitetura cliente/servidor. Para realizar este segmento do trabalho, o RMI da plataforma Java é utilizado. O RMI é um mecanismo Java que permite realizar chamadas a métodos remotos, tendo suas invocações a métodos de outros objetos feita de forma transparente. Diferentemente dos *sockets,* o RMI retira do programador a responsabilidade pelo tratamento de endereçamento, codificação/decodificação de mensagens. Além disto, outra das principais vantagens do RMI é sua capacidade de baixar o código de um objeto, caso a classe desse objeto não seja definida na máquina virtual do receptor. Os tipos e o comportamento de um objeto, previamente disponíveis apenas em uma máquina virtual, agora podem ser transmitidos para outra máquina virtual, possivelmente remota. Essa funcionalidade do RMI permite que o código da aplicação seja atualizado dinamicamente, sem a necessidade de recompilar o código.

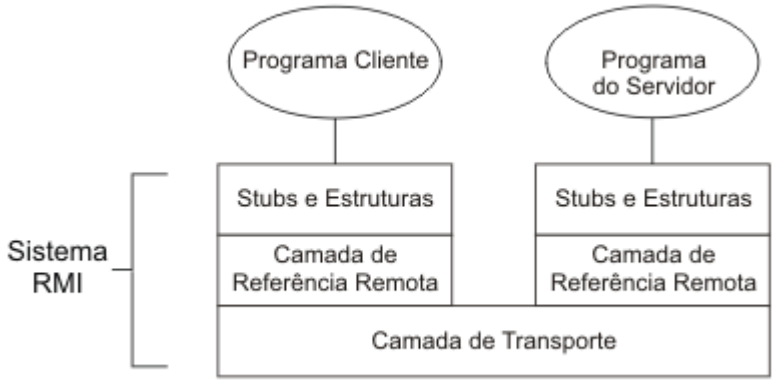
Figura 4.5 GUI - RPDI



Fonte: autoria própria

A escolha da utilização do RMI, frente às tecnologias como CORBA e RPC, foi devido à portabilidade da linguagem Java. A principal vantagem percebida sobre o RPC, foi devido ao mesmo requisitar que a programação do servidor/cliente fosse em ANSI C, o que impossibilitou a portabilidade pretendida para a aplicação. A vantagem sobre o CORBA ocorreu que mesmo com a vantagem de desenvolver o cliente em Java e o servidor em C/C++, poderia haver conflitos, visto que seriam executados em código de máquina, enquanto o Java, através da JVM é interpretada desta forma não havendo conflito entre plataformas. A Figura 4.6 apresenta o funcionamento do RMI.

Figura 4.6 Esquema básico de funcionamento RMI



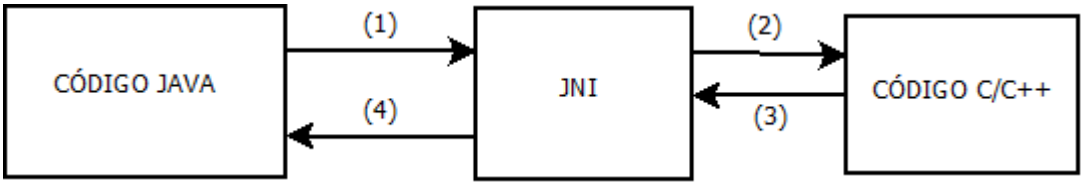
Fonte: autoria própria

4.2.2 Java Native Interface(JNI)

É um mecanismo que permite que métodos escritos em linguagens nativas sejam invocados em Java. Ele habilita aos programadores tratar situações na qual uma aplicação não pode ser inteiramente escrita em Java, por exemplo, quando o Java não suporta características especificas da plataforma. A Figura 4.7 demonstra o esquema da utilização do JNI. No passo um, o código Java solicita ao JNI a sua utilização. Em sequência, no passo dois, o JNI envia a solicitação ao sistema operacional para este realizada uma chamada à uma de suas bibliotecas. O passo três acontece quando a solicitação é finalizada retornando a JNI. Por fim, no passo quatro, o JNI retorna para o Java para continuação do fluxo do programa.

Existe uma versão mais recente da JNI chamada, Java Nativa Access(JNA). Enquanto o JNI necessita que o código nativo siga algumas regras de declaração de métodos, o JNA é capaz de abstrair isto. Entretanto, devido ao JNA necessitar mais uma camada para realizar tal abstração, portanto mais uma camada de indireção foi escolhida a tecnologia JNI.

Figura 4.7 - Esquema básico de funcionamento do JNI

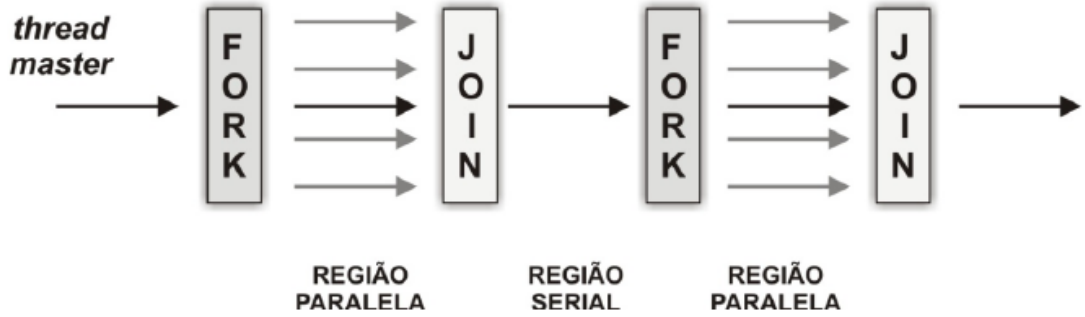


Fonte: autoria própria

4.2.3 OpenMP

O OpenMP é uma API para programação paralela explícita para memória compartilhada, sobretudo para paralelismo nos dados. Ela é constituída através de diretivas de programação, biblioteca de funções e variáveis de ambiente. A biblioteca funciona utilizando o modelo de execução *fork-join* como observa-se na Figura 4.22, que inicia sua execução com processo (*threadmaster*). Em seguida no início do construtor paralelo cria um grupo de threads e ao terminá-las as sincroniza utilizando uma barreira implícita, e então apenas a *thread master* continua sua execução.

Figura 4.22 - Modelo de execução OpenMP



Fonte: autoria própria

4.2.4 Thread

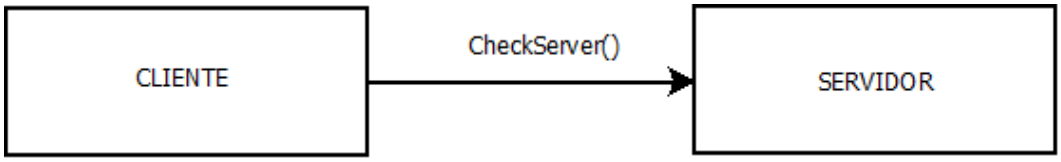
A plataforma Java nos oferece a implementação de *threads,* no seu pacote *java.lang.Thread* para quando for necessário utilizar-se da programação paralela. As *threads* são subdivisões de um processo. Na plataforma Java a utilização do paralelismo ocorre sem uso de pré-diretivas de compilação, tais como na biblioteca OpenMP, isto é, não é permitindo ao programador definir a divisão do fluxo do programa. Outro fato sobre as *threads* é que não se sabe qual será a será a ordem de execução, pois esta tarefa caberá ao escalonador, do qual o programador não possui nenhum controle.

4.3 FUNCIONAMENTO DO SOFTWARE

Nesta seção, iremos abordar o funcionamento e a integração do software com as tecnologias utilizadas. Para demonstrar o funciona do software irá utilizar-se de um cenário em que o cliente e o servidor encontram-se em redes distintas, para isto, esta parte da aplicação utiliza a tecnologia RMI. Desta forma, o primeiro passo é o cliente verificar se o servidor encontra-se disponível para a realização do processamento remoto (Figura 4.23). Uma vez verificado se o servidor está disponível, o cliente só voltará a comunicar-se com o servidor quando for necessário realizar algum método remoto.

Para o segundo cenário, imagina-se que o cliente requisitou algum filtro para o servidor. O servidor, irá verificar a implementação do filtro e a solicitação do cliente. Para todos os métodos remotos implementados no servidor é passado como parâmetros os dados referentes a imagem, o tipo processamento que o cliente deseja, algum parâmetros adicional que filtro venha a precisar, e caso seja do tipo *Thread* a quantidade de *threads* que o cliente deseja (Figura 4.24 (1)), vale lembrar que após chamada ao método remoto o cliente irá esperar a solicitação, podendo retornar de três formas (Figura 4.24 (2)). A primeira, ser atendida e mostrar o resultado do filtro desejado. A segunda, servidor irá comunicar alguma anomalia encontrado. E por fim, caso o servidor tenha sofrido uma queda ou não possa atender no momento será exibida uma mensagem avisando ao cliente.

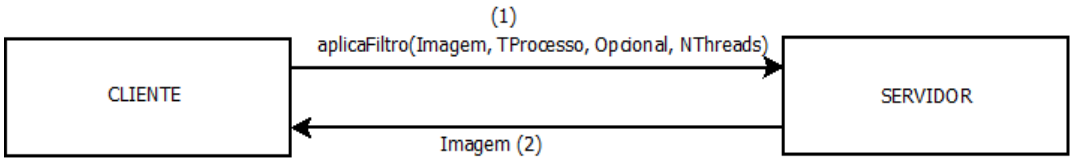
Figura 4.23 – Verificação de disponibilidade de serviço



Fonte: Autoria Própria

Para o terceiro cenário, o servidor irá realizar o processamento solicitado, para isto, após a detecção do tipo de processamento desejado, os quais podem ser: sequencial, utilizando OpenMP e JNI e utilizando Java *Treads*, o servidor irá alocar a memória necessária para o seu processamento e realizar a operação, está que por sua vez poderá ser feita de forma paralela ou sequencial de acordo com o que o cliente solicitou. Por fim, o servidor retornará ao cliente o resultado do filtro requisitado.

Figura 4.24 – Requisição de um filtro



Fonte: Autoria Própria

Para o processamento utilizando OpenMP e JNI, é necessário mais uma camada de indireção, desta forma, o JNI através JVM irá solicitar ao sistema o acesso a .DLL ou .SO gerado para utilização do OpenMP. Quando o sistema atender à solicitação da JVM o programa continuará seu fluxo normalmente.

4.4 RESULTADOS OBTIDOS

Nesta seção, iremos observar o comportamento da aplicação utilizando as medidas de desempenho comentadas no capitulo de sistema distribuídos e paralelos.

4.4.1 Granularidade

A granularidade do sistema funciona através da entrada do usuário, podendo o mesmo definir a granularidade das tarefas, quando estas forem

solicitadas para serem executadas em paralelo. Desta forma o servidor prepara um *pool* de threads para melhor a eficiência, pois o mesmo provavelmente não irá precisar alocar as *threads* em tempo de execução.

4.4.2 Utilização da CPU

A utilização da CPU foi uma medida difícil de se obter, pois trata-se de um serviço, portanto a conexão com servidor é aberta apenas para requisição da tarefa pretendida. Para melhor visualizar os impactos é necessário utilizar imagens de tamanho 3840x2160 e alguma máscara de borda ou fronteira, visto o alto tempo necessário para processar tamanha quantidade de dados. Para simular um pequeno teste de *stress* no servidor, foram utilizados 12 clientes, requisitando quase simultaneamente o mesmo filtro para uma imagem de tamanho 3840x2160, cada par requisitando um grupo de processamento. Portanto tivemos 12 clientes requisitando em sequencial.

4.4.3 Speed-Up

Para visualizar o *speed-up* foi decidido realizar os testes apenas com três filtros, um de contraste e dois de borda. Esta escolha deve-se ao fato que para operações de contraste utilizam-se operações ponto-a-ponto na imagem, desta forma os comportamentos dos filtros são bastantes semelhantes. Diferenciando nas operações matemáticas utilizadas, e por esta mesma razão, o filtro gamma foi utilizado nos testes, devido ao mesmo conter uma operação de exponenciação. Para os filtros de borda apesar de comportarem-se semelhantes foram escolhidas duas máscaras. A primeira, Prewitt, foi escolhida pois necessita após a aplicação de suas duas matrizes uma operação para normalização da intensidade, desta forma executará um pouco a mais que as demais, enquanto a segunda, a máscara media, foi escolhida por se tratar de um filtro estatístico, desta forma pode-se analisar a diferença entre esses tipos de máscaras.

O cenário no qual foram executados os testes utilizaram um computador *desktop* com as seguintes configurações: placa mãe GA61m-ds2h, 8 GB de Memória, Processador Intel Core I-3 terceira geração, HD Sata 1 Tera 7200

rpm. Os testes foram executados trinta e cinco vezes. Para as imagens foram escolhidos três tamanhos. 1280x720(720p), 1920x1080(1080p) e 3840x2160(Ultra HD 4k). No modelo Java Threads foram atribuídos 256, 384 e 768 à quantidade do *pool* de threads, a escolha destes valores foram através de testes empíricos no qual estes se mostraram o melhor valor de granularidade para os tamanhos 1280x720 e 1920x1080 e 3840x2160

A Tabela 2 apresenta os resultados de tempo de execução obtidos para uma imagem de tamanho 1280x720.

	Tempo médio de processamento em milissegundos(ms)		
	Gamma	**Prewitt**	**Media**
Sequencial	**739**	**803**	**630**
OpenMP	**407**	**541**	**417**
Thread	**482**	**699**	**624**

Tabela 2 - resultados de tempo de execução obtidos para uma imagem de tamanho 1280x720.

A Tabela 3 apresenta os resultados de tempo de execução obtidos para uma imagem de tamanho 1920x1080,

	Tempo médio de processamento em milissegundos(ms)		
	Gamma	**Prewitt**	**Media**
Sequencial	**1324**	**1770**	**1331**
OpenMP	**854**	**1137**	**881**
Thread	**1033**	**1557**	**1300**

Tabela 3 - resultados de tempo de execução obtidos para uma imagem de tamanho 1920x1080.

A Tabela 4 apresenta os resultados de tempo de execução obtidos para uma imagem de tamanho 3840x2160.

	Tempo médio de processamento em milissegundos(ms)		
	Gamma	**Prewitt**	**Media**
Sequencial	**5250**	**7793**	**5263**
OpenMP	**3329**	**5539**	**3667**
Thread	**3803**	**6485**	**4926**

Tabela 4 - resultados de tempo de execução obtidos para uma imagem de tamanho 3840x2160.

A Tabela 5 apresenta os resultados do *speed-up* da aplicação (Equação 19).

	Gamma	**Prewitt**	**Media**
Imagem 1280x720			
***Speed-up* OpenMP**	**1.81**	**1.48**	**1.51**
***Speed-up* Thread**	**1.53**	**1.15**	**1.00**
Imagem 1920x1080			
***Speed-up* OpenMP**	**1.55**	**1.56**	**1.54**
***Speed-up* Thread**	**1.28**	**1.14**	**1.05**
Imagem 3840x2160			
Speed-up	**1.58**	**1.40**	**1.44**

OpenMP			
***Speed-up* Thread**	**1.38**	**1.18**	**1.06**

Tabela 5 - *Speed-up* da aplicação

Para uma comparação mais direta entre os modelos em paralelos pode-se analisar a Tabela 6, a qual demonstra o ganho da utilização do OpenMP sobre o uso de Threads. Apesar dos resultados serem bastantes promissores para utilização de um código nativo escrito em C/C++, deve-se lembrar que o modelo de *Thread* em Java não foi exaustivamente testado para melhor utilização das *Threads*, entretanto algumas estratégias em ambos os modelos podem ser melhoradas as quais são dadas como sugestões no Capitulo 5.

	Gamma	**Prewitt**	**Media**
Imagem 1280x720	**0.85**	**0.77**	**0.67**
Imagem1920x1080	**0.83**	**0.73**	**0.68**
Imagem 3840x2160	**0.88**	**0.84**	**0.74**

Tabela 6 –Razão do modelo *OpenMp sobre Thread*

5 CONCLUSÃO E TRABALHOS FUTUROS

O software desenvolvido conseguiu realizar as propostas iniciais e foi possível observar o comportamento do paralelismo em memória compartilhada utilizando as tecnologias OpenMP e Thread. Os resultados obtidos sobre o processamento digital de imagens demonstraram o quão a área pode ser beneficiada de paralelismo visto que os filtros são operações matemáticas e estas podem ser executadas de forma independente, mediante uma boa estratégia de paralelização.

O estudo da tecnologia necessária para o desenvolvimento da aplicação, demonstrou que a portabilidade do Java pode ser integrada ao C/C++ ao utilizar uma DLL, realizando a comunicação através da JNI. Esta é uma característica que possibilita uma hibridização no desenvolvimento do software para maximização da portabilidade e desempenho.

Com o estudo de caso realizado sobre o comportamento do paralelismo foi possível constatar a viabilidade do desenvolvimento de software híbridos, possibilitando a equipes de desenvolvimento trabalhar com diferentes linguagens de programação, entretanto observou-se uma sobrecarga na rede durante o teste de *stress* do servidor, revelando um ponto fraco da aplicação.

Para trabalhos futuros fica como sugestão utilizar a API *Java Media Framework*, para permitir a utilização dos filtros em vídeos, visto que o processamento com imagens já foi realizado e testado.

Para a parte de sistemas paralelos, seria interessante estudar o comportamento da aplicação utilizando memória distribuída, usando bibliotecas como o MPI. Utilizar a tecnologia CUDA da Nvidia, para verificar as melhoras que o processamento da GPU pode trazer. Um estudo de caso para auto detecção da melhor granularidade de tarefa para desta forma determinar a melhor quantidade de *threads* necessárias para otimização do tempo de processamento.

Para a parte de processamento digital de imagens, torna-se interessante o estudo da viabilidade da utilização de operações utilizando deslocamento de bits, uma vez que esta poderá influenciar na quantidade de dados transmitidos.

Por fim, na parte comunicação, melhorar a forma de transmissão de dados do cliente para o servidor e vice-versa e propor um algoritmo para

balanceamento de cargas nos servidores para melhor utilização dos recursos computacionais, visto que o sistema pode funcionar em múltiplos servidores.

Referências

TANENBAUM, Andrew S.; STEEN, Maarten Van. **Sistemas Distribuídos: Princípios e Paradigmas**. São Paulo. Peason Education do Brasil, 2012.

PAGE, Daniel; **A Practical Introduction to Computer Architecture.** Eds. Springer, 2009.

GONZALEZ; WOODS, **Processamento digital de imagens**. Longman do Brasil

PEDRINI; SCHWARTZ, **Análise de imagens digitais: princípios, algoritmos e aplicações**. São Paulo: Thomson Learning, 2008

HWANG. **AdvancedComputer Architecture: Parallelism. Scalability, Programma- bility** McGraw-Hill, New York, 1993.

FLYNN; RUDD. **Parallel architectures**. ACM Comput. Surv. v. 28, n. 1, p. 6770, 1996.

BRAUNL. **Tutorial in data parallel image processing**. Australian Journal of Intelligent Information Processing Systems, 6(3):164–174.

BLELLOCH. **Programming parallel algorithms**. Communications of the ACM, 39(3):85–97.

STALLINGS, **Computer Organization and Architecture: Designing for Performance**, 6th ed. Pearson Education International, 2003

RMI, 2015 Disponível em: <http://www.gta.ufrj.br/grad/00_1/vicente/> 01 Jan. 2015.

Mpi, 2015. Disponível em http://www.mcs.anl.gov/research/projects/mpi. Acessado em Jan. de 2015

OpenMP, 2015. Disponível em http://openmp.org/wp/. Acessado em Jan. de 2015.

PosixThread, 2015. Disponível em https://computing.llnl.gov/tutorials/pthreads/. Acessado em Jan. de 2015.

HSV, 2015. Disponível em <http http://pt.wikipedia.org/wiki/HSV> Acessado em Jan. de 2015.