Notas em Matemática Aplicada e-ISSN 2236-5915

Volume 56, 2011

Editores

Célia A. Zorzo Barcelos
Universidade Federal de Uberlândia - UFU
Uberlândia, MG, Brasil
Eliana X.L. de Andrade
Universidade Estadual Paulista - UNESP
São José do Rio Preto, SP, Brasil
Maurílio Boaventura
Universidade Estadual Paulista - UNESP
São José do Rio Preto, SP, Brasil

A Sociedade Brasileira de Matemática Aplicada e Computacional - SBMAC publica, desde as primeiras edições do evento, monografias dos cursos que são ministrados nos CNMAC.

Para a comemoração dos 25 anos da SBMAC, que ocorreu durante o XXVI CNMAC, foi criada a série **Notas em Matemática Aplicada** para publicar as monografias dos minicursos ministrados nos CNMAC, o que permaneceu até o XXXIII CNMAC em 2010.

A partir de 2011, a série passa, também, a publicar livros nas áreas de interesse da SBMAC. Os autores que submeterem textos à série Notas em Matemática Aplicada devem estar cientes de que poderão ser convidados a ministrarem minicursos nos eventos patrocinados pela SBMAC, em especial nos CNMAC, sobre assunto a que se refere o texto.

O livro deve ser preparado em **Latex (compatível com o Miktex versão 2.7), as figuras em eps** e deve ter entre **80 e 150 páginas**. O texto deve ser redigido de forma clara, acompanhado de uma excelente revisão bibliográfica e de **exercícios de verificação de aprendizagem** ao final de cada capítulo.

Veja outros títulos publicados em formato e-book na página
http://www.sbmac.org.br/notas.php

Sociedade Brasileira de Matemática Aplicada e Computacional

2011

FUNDAMENTOS DE COMPUTAÇÃO PARALELA PARA A RESTAURAÇÃO DE IMAGENS DE MICROSCOPIA DE FORÇA ATÔMICA

Dalmo Stutz
stutz@iprj.uerj.br

Antônio J. Silva Neto
ajsneto@iprj.uerj.br

Universidade do Estado do Rio de Janeiro - UERJ
Instituto Politécnico - IPRJ
Nova Friburgo - RJ

Sociedade Brasileira de Matemática Aplicada e Computacional

São Carlos - SP, Brasil
2011

Coordenação Editorial: Elbert Einstein Nehrer Macau

Coordenação Editorial da Série: Eliana Xavier Linhares de Andrade

Capa: Matheus Botossi Trindade

Patrocínio: SBMAC

Catalogação elaborada pela Biblioteca do IBILCE/UNESP
Bibiotecária: Maria Luiza Fernandes Jardim Froner

Stutz, Dalmo
Fundamentos de Computação Paralela para a Restauração de Imagens de Microscopia de Força Atômica - São Carlos, SP : SBMAC, 2011, 144 p., 20.5 cm - (Notas em Matemática Aplicada; v. 56)

e-ISBN 978-85-86883-58-3

1. Processamento de imagens 2. Processamento paralelo (Computadores) 3. Algoritmos I. Stutz, Dalmo II. Silva Neto, Antônio José da. IV. Título. V. Série

CDD - 51

Conteúdo

Prefácio

A Microscopia de Força Atômica é uma técnica que permite a aquisição de imagens, em escala nanométrica, da superfície de quase todo tipo de material. Nessa escala, as imagens costumam apresentar uma relação sinal/ruído pobre, causados por efeitos degenerativos em sua qualidade, provenientes do processo de aquisição da imagem ou do próprio equipamento.

Para recuperar essas imagens, ou minimizar os efeitos da degradação, diversas técnicas têm sido desenvolvidas e vêm sendo aplicadas. Dentre elas, uma técnica de restauração, que será descrita neste livro, baseada na minimização do funcional de Tikhonov com uma família de termos de regularização a um parâmetro, tem sido usada já há alguns anos no tratamento de imagens obtidas com o Microscópio de Força Atômica, com resultados bastante satisfatórios.

O uso dessa técnica, porém, exige um esforço computacional grande, o que acaba resultando em um tempo de execução relativamente elevado, principalmente quando o programa que implementa o algoritmo de restauração é processado serialmente. Junte-se a isso, o fato de que à medida que os equipamentos eletrônicos aumentam as suas capacidades, as imagens obtidas por eles aumentam a resolução e, consequentemente, o esforço computacional e o tempo gasto para analisá-las e restaurá-las também aumentam.

Há algum tempo que o desempenho e a velocidade de processamento dos equipamentos vêm apresentando melhoras significativas, mas esse aumento, contudo, não tem ocorrido suficientemente rápido de modo a fazer frente à demanda por maior poder computacional que as aplicações relevantes em diversas áreas de conhecimento e pesquisa têm exigido crescentemente. Isso ocorre, em parte, devido ao surgimento de novas demandas e ao fato de que muitos problemas, que há alguns anos exigiriam demasiado tempo computacional para a execução nas máquinas disponíveis à época, hoje são factíveis de serem resolvidos e podem ser processados, empregando-se os equipamentos disponíveis no mercado.

Muitas vezes, o uso de equipamentos mais sofisticados e/ou dedicados é uma solução que pode ser empregada, para atender a necessidade cada vez maior de capacidade de processamento. Entretanto, a aquisição e a manutenção de tais equi-

pamentos implicam, normalmente, em custos elevados, reduzindo a acessibilidade e atratividade desta opção.

Como alternativa, técnicas de computação paralela são cada vez mais empregadas em equipamentos de menor custo e/ou não tão dedicados, com objetivo de buscar melhores desempenhos de hardware e/ou software a um custo menor.

Neste livro são apresentados, portanto, alguns fundamentos de computação paralela aplicados ao problema de restauração de imagens obtidas com Microscópios de Força Atômica. São apresentadas estratégias de implementação paralela do algoritmo de restauração, objetivando o aumento na velocidade de processamento e no desempenho computacional.

É apresentado, também, um novo método de busca do parâmetro de regularização ótimo que, apesar de introduzir um esforço computacional maior ao processo de restauração, com um tempo maior de processamento, traz ao algoritmo a busca automática do parâmetro de regularização que resulte em uma melhor solução para o problema de restauração de imagens empregando o funcional de regularização de Tikhonov.

Nova Friburgo, 30 de junho de 2011.

Dalmo Stutz Antônio J. Silva Neto

Capítulo 1

Aquisição de Imagens em Escala Nanométrica com Microscopia de Força Atômica

1.1 Microscopia de Força Atômica

O Microscópio de Força Atômica (*Atomic Force Microscope* - AFM), ou Microscópio de Varredura de Força (*Scanning Force Microscope* - SFM), inventado por Binnig, Quate e Gerber [5], é uma derivação da técnica da Microscopia de Varredura por Tunelamento (*Scanning Tunneling Microscopy* - STM) [22], capaz de produzir imagens em escala nanométrica ($1nm = 1 \times 10^{-9}m$) da superfície de quase todo o tipo de material, inclusive de amostras não-condutoras, tais como: polímeros, cerâmicas, amostras biológicas, entre outros, com grande utilidade e aplicação em nanociência e nanotecnologias.

O AFM faz parte de uma família de instrumentos conhecidos como Microscópios de Varredura por Sonda (*Scanning Probe Microscope* - SPM), que diferem entre si, basicamente, na forma como a sonda interage com a amostra. De modo geral, esses equipamentos são compostos por uma sonda sensora que varre a superfície de um objeto que se quer analisar, uma cerâmica piezoelétrica que é usada para fazer o posicionamento da amostra para varredura, circuitos de realimentação que são usados para controlar o posicionamento da sonda e um computador para controlar a varredura, armazenar dados e convertê-los em imagens.

1.2 Imagens obtidas pelo Microscópio de Força Atômica

1.2.1 O processo de aquisição

De forma simplificada, a Figura 1.1 ilustra, esquematicamente, o princípio de funcionamento de um AFM. Para obter uma imagem de uma amostra, o Microscópio de Força Atômica utiliza um braço de apoio (*cantilever*) que é uma sonda de dimensões reduzidas, com uma ponteira em formato piramidal presa em sua extremidade e que varre a superfície do material que está sendo analisado.

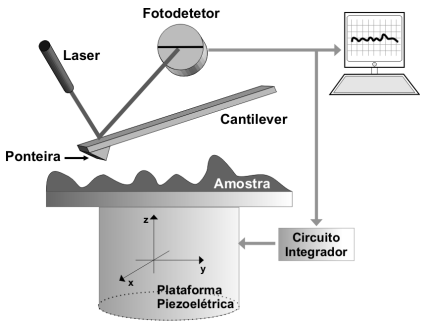


Figura 1.1: Esquema simplificado de funcionamento de um AFM.

Durante o processo de varredura, ocorre uma interação entre a ponta da sonda e a amostra com uma força de ordem interatômica e, ao interagir com a superfície da amostra, a ponteira gera, em função do relevo característico do objeto analisado, uma resposta mecânica que flexiona a sonda. Estas deflexões são, então, registradas através de um feixe de laser que incide sobre a sonda e é refletido sobre um sensor fotodetector que produz uma resposta elétrica, registrando as deflexões e movimentando a plataforma piezoelétrica ao longo do eixo z (vide Figura 1.1). Este sinal é, então, captado e processado por um computador que o interpreta, gerando uma representação gráfica (imagem) da superfície da amostra naquele ponto. Uma plataforma piezoelétrica, que realiza movimentos simultâneos nas direções x e y, permite a varredura em toda a superfície da amostra e, a cada registro, o sistema é reposicionado na direção z quase que imediatamente, protegendo o material analisado contra quaisquer riscos de ser atingido pela sonda. Finalmente, um circuito integrador amortiza a velocidade de reposicionamento da plataforma, a fim de evitar a ocorrência de oscilações indesejáveis que possam prejudicar a qualidade das imagens produzidas pelo AFM.

1.2.2 Interação ponteira-amostra

A interpretação do sinal resultante da interação ponteira-amostra, para a formação da imagem, entretanto, dependente muito das características da ponteira utilizada no microscópio. A escolha do tipo de ponteira influi diretamente na percepção dos detalhes do objeto, podendo até ocorrer o surgimento de artefatos na imagem. Conforme ilustrado na Figura 1.2, pode-se observar que diferentes geometrias de ponteira podem produzir alterações significativas nas imagens obtidas pelo AFM para uma mesma amostra. Ponteiras mais afiladas produzem imagens mais nítidas, ao passo que, ponteiras mais rombudas introduzem mais artefatos na imagem.

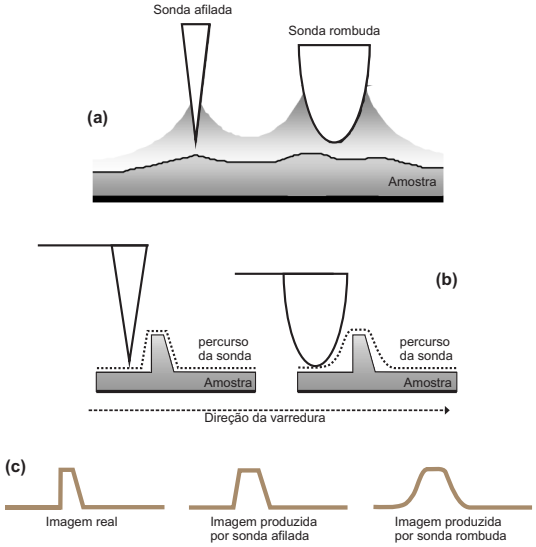


Figura 1.2: Influência da geometria da ponteira na formação de uma imagem de AFM. Ilustrações dos (a) efeitos da média de forças (equivalente à convolução de forças) de uma ponteira afilada e outra mais rombuda sobre a amostra; (b) interpretação da imagem, dependente da geometria da ponteira; e a (c) presença de artefatos nas imagens de uma mesma amostra, gerados por diferentes formatos de ponteira: uma afilada e a outra rombuda [12, 22].

1.2.3 O processo de formação das imagens

O processo de aquisição, descrito na Seção 1.2.1, é caracterizado como um sistema de formação de imagens (Figura 1.3), em que um objeto x (superfície da amostra) em um sistema de coordenadas (k, l), denominado *plano objeto*, é de alguma forma

captado pelo sistema que o transforma em uma imagem digital monocromática y, num sistema de coordenadas discretas (i, j), denominado *plano imagem*.

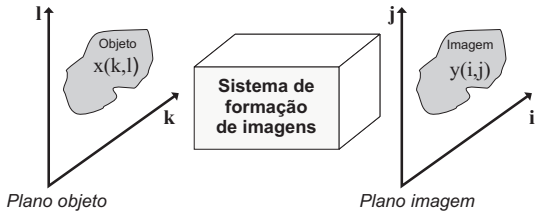


Figura 1.3: Diagrama esquemático do sistema de formação de imagens [2].

O valor ou amplitude $y(i, j)$ de uma imagem monocromática em cada ponto (i, j), também conhecido como *pixel* ou *pel* (do inglês, *picture element*), Figura 1.4, varia numa escala de inteiros 0 a 255 e representa uma dada altura da superfície da amostra acima do plano, sendo que o valor zero (ou cor preta) representa a menor altura (zero), 255 (cor branca) a maior altura, e os demais valores intermediários (tonalidades de cinza) as diversas alturas intermediárias.

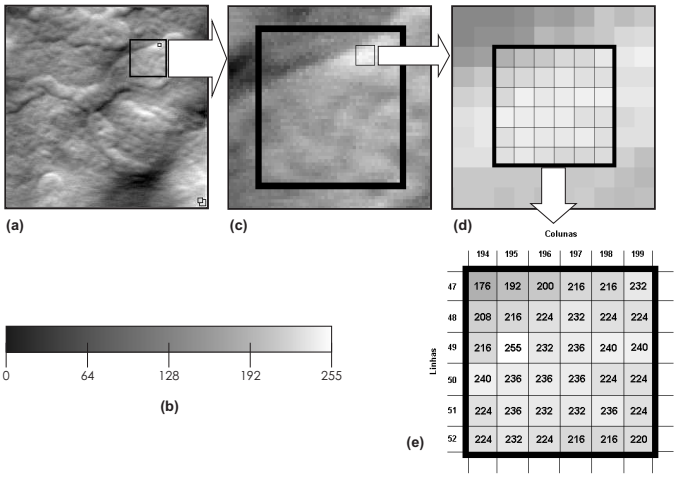


Figura 1.4: Estrutura de uma imagem digital monocromática. (a) Imagem biológica de AFM de 256x256 pixels de um eritroblasto em estado leucêmico de $1000nm \times 1000nm$ numa (b) escala de cor monocromática. Projeções (c) e (d) das áreas demarcadas na imagem, (e) a representação matricial dos pontos (pixels) de uma sessão da imagem representada em (d).

Fazendo um paralelo entre os pontos (k, l) no plano objeto e (i, j) no plano imagem, tem-se que o sistema de formação de imagens gera o valor $y(i, j)$ através das forças resultantes da iteração ponteira-amostra, captada da superfície do objeto amostrado x. No entanto, esse sistema recebe componentes de energia não só de $x(k, l)$ mas, também, de todos os pontos em uma vizinhança no plano objeto. Uma vez que a energia captada pelas forças de interação ponteira-amostra é acumulativa ou seja, é aditiva, isto significa que no processo de formação de imagens de AFM a composição do valor $y(i, j)$ sofre influências tanto do ponto (k, l), i.e. $x(k, l)$, como, também, dos demais pontos do objeto situados na vizinhança de (k, l).

A intensidade dessa influência, porém, varia de acordo com a distância entre o ponto (k, l) e os demais pontos no plano objeto. À medida que essa distância aumenta, a contribuição desses pontos do objeto para a formação de $y(i, j)$ diminui. Essa superposição de distribuições de energia pode ser representada por uma função h, conhecida como Função de Espalhamento Pontual (PSF - *Point Spread Function*), que descreve a transformação da energia do plano objeto para o plano imagem.

Além disso, tem-se que a função h é linear, uma vez que a resposta do sistema de formação de imagens do AFM a uma soma de duas entradas é igual à soma das duas respostas (Propriedade da aditividade) e a resposta a um múltiplo de qualquer entrada é igual àquela entrada multiplicada pela mesma constante (Propriedade da homogeneidade) [21].

Para efeitos práticos e computacionais, considerando que os pontos no objeto sejam discretos, e somando-se as devidas contribuições de todos os pontos do objeto para a formação dos pontos no plano imagem, podemos obter a equação geral de formação de imagens, dada por [2]

$$y(i, j) = \sum_{k,l=0}^{M} h_{i-k,j-l} x_{k,l} \tag{1.2.1}$$

ou, simplesmente,

$$y = h * x \tag{1.2.2}$$

onde o símbolo $*$ representa o operador de convolução discreta, x e y são respectivamente as imagens do plano objeto e do plano imagem, armazenados em vetores de dimensão $(M + 1)^2$, lexicograficamente ordenados.

Dado que o menor nível de energia é zero (não-negatividade), tem-se também que

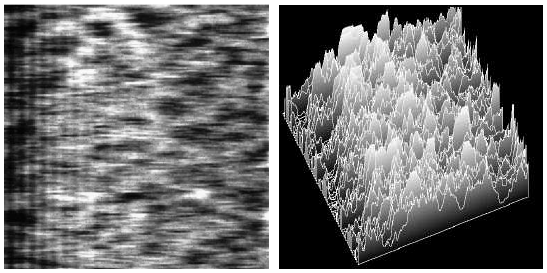
$$x(k, l) >= 0 \; e \; y(i, j) >= 0.$$

1.2.4 Características das imagens de AFM

As imagens características de um AFM são normalmente imagens digitais monocromáticas, variando em 256 gradações de cinza, compostas tipicamente por

256 linhas e 256 colunas, num total de 65536 pixels. Entretanto, dependendo do equipamento que está sendo utilizado, essas imagens podem ter resoluções ainda maiores, tais como, por exemplo: 512×512, 1024×1024, etc.

O valor da intensidade de cinza, em cada pixel da imagem, representa uma dada altura acima do plano, uma vez que as imagens de AFM representam a superfície do objeto amostrado no espaço euclidiano tridimensional (3D), projetadas sobre um plano bidimensional (2D) (Figura 1.5).



(a) (b)

Figura 1.5: (a) Imagem de AFM de filamentos de DNA em alta concentração, medindo $200nm \times 200nm$, visualizado em 2D e (b) a sua representação tridimensional (3D) [12].

Devido às características de construção do equipamento, as imagens obtidas pelo AFM, usualmente, apresentam uma relação sinal/ruído pobre [11], causada por efeitos degenerativos na qualidade da imagem, provocados seja por borramentos resultantes da interação ponteira-amostra (degradação espacial) (Figura 1.2) ou por ruídos aditivos originários do próprio equipamento (degradação pontual), causados por ruídos elétricos, eletrônicos, vibrações, etc.

1.3 Imagens utilizadas neste livro

O conjunto de imagens utilizadas neste livro é formado por nove imagens, a saber (Figura 1.6): (a) imagem-padrão criada por Cidade [11] e, propositadamente, estruturada com regiões de altas e baixas frequências espaciais para ser uma imagem de testes de restaurações; imagens modificadas da imagem-padrão, identificadas como (b) D3v20s20, (c) D3v20s40, (d) D5v20s20, (e) D5v20s40 e (f) D5v40s40, onde **D** representa a dimensão da matriz de borramento (associada à PSF), **v** é a variância e **s** a relação sinal-ruído, utilizadas nos testes de restaurações; imagens biológicas de AFM da superfície de um eritroblasto em estado leucêmico (g) de $1000nm \times 1000nm$ e (h) $600nm \times 600nm$; e imagem de AFM (k) da superfície de

uma amostra de ferro submetidas a dissolução em H_2SO_4 de $30\mu m \times 30\mu m$ [46]. Todas as imagens são monocromáticas e medem 256×256 pixels [12].

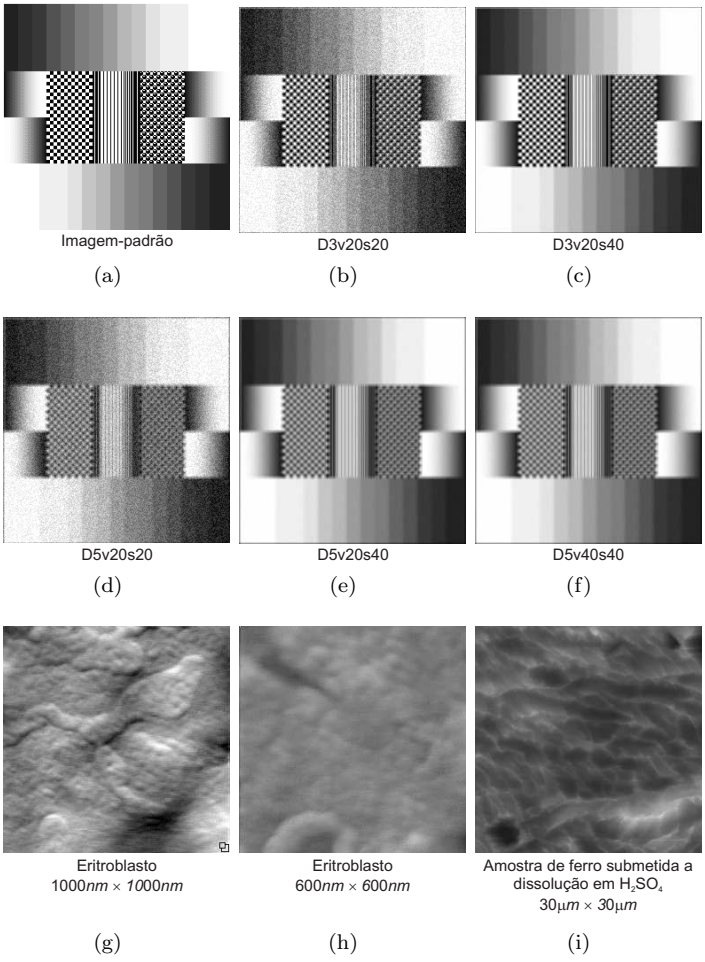


Figura 1.6: Relação de imagens utilizadas neste livro.

Capítulo 2

Restauração de Imagens de Microscopia de Força Atômica com o Funcional de Regularização de Tikhonov

2.1 O modelo do processo de degradação

Em restauração de imagens, o processo de degradação de uma imagem pode ser modelado da seguinte forma [2, 20, 21, 24]

$$y = Hx + \eta \tag{2.1.1}$$

onde x e y representam, respectivamente, as imagens original e degradada, lexicograficamente ordenadas, a matriz H um operador que descreve a função de borramento que, juntamente com um termo de ruído aditivo η operam, sobre a imagem x (plano objeto), produzindo uma imagem observada y (plano imagem) que é obtida através de um experimento.

Na Figura 2.1 é representado esquematicamente o modelo do processo de degradação de imagens. Nele, o operador degenerativo H (ou borramento) opera sobre uma imagem de entrada x (imagem real), que acrescida do ruído aditivo η, produz a imagem degradada y, observada experimentalmente.

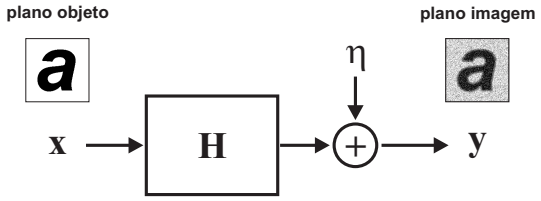


Figura 2.1: Modelo do processo de degradação de imagens.

2.2 O problema de restauração

A aplicação de técnicas de restauração de imagens surge da necessidade de melhorar algum aspecto e/ou recuperar uma informação qualquer de uma imagem, que tenha sofrido algum tipo de degradação: borramento, ruído aditivo, etc. Basicamente, o problema consiste em, usando um conhecimento *a priori* do fenômeno de degradação, tentar restaurar e/ou recuperar uma imagem degradada, buscando-se a imagem real do objeto x (ou uma estimativa $\hat{x}$ mais próxima dela), partindo de uma imagem y conhecida, obtida através de um experimento qualquer (p.ex. AFM) e que apresenta algum tipo de degradação.

Ao longo do tempo, diversas técnicas de restauração têm sido desenvolvidas e empregadas. Dentre elas, só para citar algumas, temos: morfologia matemática [21], deconvolução, FFT (*Fast Fourier Transform*), filtros (pseudo-inversa, Wiener, etc.), Wavelet, Pixon [43], dentre outras.

Técnicas no domínio da frequência

Tradicionalmente, a maioria das técnicas que opera no domínio da frequência utiliza, principalmente, a Transformada Rápida de Fourier (FFT) e a sua inversa (FFT^{-1}). Caso o ruído possa ser desprezado, a imagem real pode ser determinada e calculada eficazmente usando esse método que, apresenta uma boa velocidade de processamento. Entretanto, a técnica de deconvolução usando Fourier não é muito eficaz quando o ruído não puder ser desprezado [43]. Além disso, quando esta técnica é empregada em restaurações de imagens de AFM, ela penaliza tanto o ruído quanto o sinal de alta frequência, prejudicando o resultado da restauração e impedindo a recuperação de características importantes da imagem, tais como bordas e texturas, que estejam na mesma faixa de frequência do ruído aditivo [11, 61].

Técnicas no domínio espacial

No domínio espacial, a solução do problema de restauração resume-se em encontrar o operador inverso de H ou uma estimativa $\hat{x}$, que seja a mais próxima possível da imagem original x, sujeito a um critério de otimização.

Assumindo-se a Equação 2.1.1 sem o ruído aditivo ($\eta = 0$) e admitindo que H seja inversível, uma possível solução para o problema poderia ser obtida, fazendo-se

$$x = H^{-1}y \tag{2.2.2}$$

Entretanto, essa solução não é muito viável, já que na maioria dos problemas de interesse (p.ex. desfocamento) a matriz H é singular e, portanto, não possui inversa [36]. Além disso, soluções diretas em restaurações de imagem são notoriamente sensíveis a pequenas variações em y, significando que a busca de uma solução direta do problema através da Equação 2.1.1, além de ser ineficiente sob o ponto de vista computacional, devido ao custo operacional para se encontrar a matriz inversa H^{-1}, o mal-condicionamento da matriz de borramento H faz com que H^{-1} atue como um amplificador, fazendo com que uma pequena perturbação nos dados experimentais acabe por se traduzir em uma grande instabilidade na solução do problema, tornando essa solução pouco exequível [24, 26]. Tais problemas são conhecidos na literatura como problemas mal-postos [26, 40], pois não há garantia da existência, única e estável de uma solução para o problema com base na inversão direta.

No entretanto, considerando H fixo, uma estabilização poderia ser alcançada através do emprego de mínimos quadrados na determinação da solução, revertendo-se o problema para bem-posto, ou seja,

$$L(x) = \min_x \|y - Hx\|^2 \tag{2.2.3}$$

Porém, erros numéricos provenientes do processo computacional iterativo para determinação da solução do problema, fazem com que o operador de borramento H, implicitamente, comporte-se como se estivesse sujeito a variações, como que incorporasse tais perturbações, tornando o problema mal-posto novamente [10, 25].

Uma maneira para se solucionar este problema seria o uso de um termo de regularização [55]. Nesse caso, uma solução para o problema pode ser encontrada minimizando-se o funcional de regularização de Tikhonov [11, 12, 26, 40], dado por

$$Q(x) = \|y - Hx\|^2 + \alpha S(x), \alpha > 0 \tag{2.2.4}$$

onde α o parâmetro de regularização que determina a estabilidade do funcional Q, e S uma função de regularização ou funcional de estabilização do processo de restauração.

Fazendo-se $\alpha \to 0$, a solução da Equação 2.2.4 tenderia para $H^{-1}y$ (Equação 2.2.2), enquanto que para $\alpha \to \infty$, a solução tenderia para o funcional de estabilização S que, normalmente, representa uma possível solução suavizada do problema [26, 40]. A escolha entre a deconvolução e a suavização é determinada pelo valor de α, que estabelece um compromisso entre a acurácia e a estabilidade da solução do problema [12].

Jeng e Woods [23] afirmam que a restauração, usando técnicas do domínio espacial, possui uma grande vantagem sobre o domínio da frequência, pois o método utilizado para estimar a imagem tem um caráter recursivo, onde os pontos da imagem (pixels) são tratados em função de uma vizinhança local (tratamento em base local), sem penalizar significativamente os componentes do sinal (contraste e ruído).

2.3 O operador de borramento

Em processamento digital de sinais, o cálculo da convolução pode ser expresso de duas formas diferentes [50]:

(a) sob o ponto de vista do sinal de entrada; ou

(b) sob o ponto de vista do sinal de saída.

O cálculo apresentado na Equação 1.2.1 é dado sob o ponto de vista do sinal de entrada e é a forma mais geral usada para expressar a convolução. Dado dessa forma, a expressão analisa como cada ponto do plano objeto x (sinal de entrada) afeta (ou contribui para) a formação dos pontos no plano imagem y (sinal de saída).

Na segunda maneira, reverte-se o ponto de vista, analisando o processo de formação de cada um dos sinais de saída. Examinando-se, então, o processo de formação de imagens do AFM sob o ponto de vista do sinal de saída, observa-se que cada valor y_{ij} do plano imagem é formado pela convolução de elementos de x que estão ao redor do ponto $x(k,l)$ no plano objeto. Quanto mais distante esse ponto estiver de (k,l), menor é a sua influência e, portanto, menor a sua contribuição para a formação do sinal de saída y_{ij}. Dado sob essa ótica, reescreve-se a Equação 1.2.1 [11] como

$$y(i,j) = \sum_{k,l=-N}^{N} b_{k,l} x_{i+k,j+l} \tag{2.3.5}$$

sendo $B = [b_{kl}]_{2N+1 \times 2N+1}$ o operador de borramento, descrito sob o ponto de vista do sinal de saída, e N um valor arbitrário ($1 \leq N \leq \dfrac{M-1}{2}$), considerando

a imagem com $(M+1) \times (M+1)$) que define de forma discreta as dimensões de B ($dimB = 2N+1$) e limita uma área em torno do ponto (k,l) no plano objeto, descrevendo o grau de influência dos pontos dessa área para a formação de y_{ij} no plano imagem.

Sabendo-se que (a) quanto mais distante $x_{i+k,j+l}$ estiver de (k,l) no plano objeto, menor é a sua contribuição para a formação de y_{ij} no plano imagem e (b) dada a imprecisão e erros de arredondamento computacionais, podemos admitir que, a partir de uma determinada distância N de (k,l), o valor de contribuição do ponto para a formação do ponto imagem y_{ij} seja nulo.

Na Equação 2.3.5, a matriz B é um operador que determina a função de distribuição de energias observada na imagem y e que representa o modelo de interação de forças da ponteira do microscópio e a amostra analisada, descrevendo o efeito do borramento (função de espalhamento pontual ou PSF) que ocorre no processo de aquisição de imagens de AFM.

Nas abordagens clássicas de restauração, normalmente assume-se que o modelo de borramento e os seus parâmetros são conhecidos *a priori*. Entretanto, no mundo real, essa função geralmente não é conhecida, podendo até existir dificuldade em se obter uma representação de um modelo que expresse fielmente a interação ponteira-amostra. Assim, na falta de uma estimativa, normalmente, assume-se que a função segue um modelo de distribuição estatística conhecida.

Para representar o modelo de iteração ponteira-amostra do AFM, entretanto, Kokaram *et al.* [29] sugerem o emprego da distribuição gaussiana, enquanto Dongmo *et al.* [14] e Weisman *et al.* [61] sugerem um parabolóide de revolução. Neste livro, o operador B usado assumirá uma dependência espacial do tipo gaussiana, conforme ilustrado nas Figuras 2.2 e 2.3, dada por [11, 29]:

$$B \propto e^{-\frac{d^2}{\sigma^2}} \tag{2.3.6}$$

onde d é a distância euclidiana dos pontos em relação a um dado ponto P da amostra e σ^2 é a variância que simula os diferentes tipos de ponteiras, mais finas (afiladas) ou mais grossas (rombudas) [19].

$$B_{5x5} = \begin{bmatrix} 4{,}29\text{x}10^{-8} & 4{,}71\text{x}10^{-5} & 6{,}993\text{x}10^{-3} & 4{,}71\text{x}10^{-5} & 4{,}29\text{x}10^{-8} \\ 4{,}71\text{x}10^{-5} & 6{,}993\text{x}10^{-3} & 0{,}140464 & 6{,}993\text{x}10^{-3} & 4{,}71\text{x}10^{-5} \\ 6{,}993\text{x}10^{-3} & 0{,}140464 & 0{,}381821 & 0{,}140464 & 6{,}993\text{x}10^{-3} \\ 4{,}71\text{x}10^{-5} & 6{,}993\text{x}10^{-3} & 0{,}140464 & 6{,}993\text{x}10^{-3} & 4{,}71\text{x}10^{-5} \\ 4{,}29\text{x}10^{-8} & 4{,}71\text{x}10^{-5} & 6{,}993\text{x}10^{-3} & 4{,}71\text{x}10^{-5} & 4{,}29\text{x}10^{-8} \end{bmatrix}$$

Figura 2.2: Representação matricial de um operador de borramento normalizado, do tipo gaussiano, de dimensão 5×5, com $\sigma^2 = 20$.

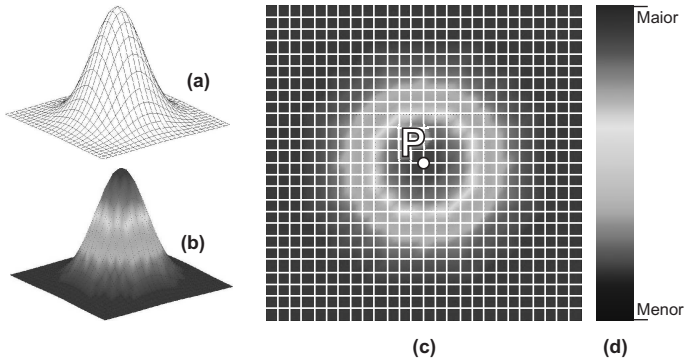


Figura 2.3: Visualização do operador de borramento do tipo gaussiano (a) e (b), mostrando em detalhes a distribuição das forças de iteração normalizadas (b), projetado sobre um plano (c), numa escala de valores (d) que varia do menor ao maior grau de contribuição dos pontos periféricos, para a formação da imagem no ponto central, na medida em que estes se aproximam de P.

Independente do modelo adotado, entretanto, uma vez que o sistema de formação de imagens normalmente não absorve e nem gera energia, o operador B deve satisfazer, então, a seguinte condição:

$$\sum_{k,l=-N}^{N} b_{kl} = 1 \tag{2.3.7}$$

Para garantir que a condição dada na Equação 2.3.7 seja satisfeita, divide-se todos os elementos de B pelo somatório desses elementos, normalizando-os. Além disso, como as intensidades (energias) envolvidas são não-negativas, tem-se também que

$$b_{kl} \geq 0;\ k, l = -N, ..., 0, ..., N$$

e, para os índices que estiverem fora desta faixa, tem-se que

$$b_{kl} = 0,\ \forall |k| > N\ e\ |l| > N.$$

2.4 Técnica de restauração baseada na minimização do Funcional de Regularização de Tikhonov

No sentido de encontrar uma possível adequação ao tratamento de imagens, Cidade *et al.* [11] propõem um método pouco tradicional na área de restauração de imagens, no qual se emprega um funcional geral de regularização S que corresponde a uma família de funções de regularização, construído com distâncias de Bregman [6], baseados no funcional q-discrepância [7]

$$S(x) = S_q(\widehat{x}, \overline{x}) = \frac{1}{1+q} \sum_{i=0}^{M} \sum_{j=0}^{M} \left\{ \widehat{x}_{ij} \left[\frac{\widehat{x}_{ij}^q - \overline{x}_{ij}^q}{q} \right] - \overline{x}_{ij}^q (\widehat{x}_{ij} - \overline{x}_{ij}) \right\} \tag{2.4.8}$$

onde q é um parâmetro ajustável a partir do qual outras normas podem ser obtidas, tais como:

(a) Mínima energia ($q \to 0$):

$$S(x) = -\sum_{i=0}^{M} \sum_{j=0}^{M} \left[\hat{x}_{ij} - \bar{x}_{ij} - \hat{x}_{ij} ln \frac{\hat{x}_{ij}}{\bar{x}_{ij}} \right]$$

(b) Entropia-cruzada ($q = 1$):

$$S(x) = \frac{1}{2} \sum_{i=0}^{M} \sum_{j=0}^{M} (\hat{x}_{ij} \bar{x}_{ij})^2$$

$\hat{x}$ representa o valor esperado da solução (ou a estimativa de x) e $\bar{x}$, um valor de referência, para tratar o ruído aditivo, que pode ser a própria imagem borrada, obtida experimentalmente com o AFM, por exemplo.

Aplicando-se as Equações 2.3.5 e 2.4.8 na Equação 2.2.4 e substituindo x (imagem real) por $\hat{x}$ (estimativa), obtém-se o funcional de regularização de Tikhonov [11] e, na busca de uma estimativa da imagem real, procura-se minimizar o funcional dado por

$$Q(\widehat{x}) = \sum_{i=0}^{M} \sum_{j=0}^{M} \left[y_{ij} - \sum_{k=-N}^{N} \sum_{l=-N}^{N} b_{kl} \widehat{x}_{i+k,j+l} \right]^2 + \alpha S_q(\widehat{x}, \overline{x}) \tag{2.4.9}$$

cujo valor residual é definido por

$$R(\widehat{x}) = \sum_{i=0}^{M}\sum_{j=0}^{M}\left[y_{ij} - \sum_{k=-N}^{N}\sum_{l=-N}^{N} b_{kl}\widehat{x}_{i+k,j+l}\right] \tag{2.4.10}$$

De modo a minimizar o funcional dado pela Equação 2.4.9, escreve-se a equação do ponto crítico, que leva a um sistema de $(M+1)^2$ equações não-lineares para ser resolvido, do tipo:

$$F_{rs}(\widehat{x}) = \frac{\partial Q(\widehat{x})}{\partial \widehat{x}_{rs}} = 0,\ r,s = 0,1,\ldots,M \tag{2.4.11}$$

Usando o método de Newton-Raphson multivariável [32, 42] para solucionar o sistema de Equações 2.4.11, novas estimativas podem ser obtidas, empregando-se

$$\widehat{x}^{t+1} = \widehat{x}^{t} + \gamma\Delta\widehat{x}^{t},\ t = 0,1,2,\ldots \tag{2.4.12}$$

onde t é o contador de iterações do método de Newton-Raphson, γ um fator de atenuação, de forma a auxiliar na convergência do algoritmo e, como estimativa inicial $\hat{x}^0$, pode-se usar a própria imagem obtida pelo microscópio.

Empregando-se uma expansão de Taylor e retendo os termos até a primeira ordem, tem-se

$$F_{rs}(\hat{x}^{t+1}) = F_{rs}(\hat{x}^{t} + \Delta\hat{x}^{t}) = F_{rs}(\hat{x}^{t}) + \sum_{m=0}^{M}\sum_{n=0}^{M}\left.\frac{\partial F_{rs}}{\partial \hat{x}_{mn}}\right|_{\hat{x}^t}\Delta\hat{x}^{t}_{mn} = 0 \tag{2.4.13}$$

Empregando-se o método iterativo de Gauss-Seidel [30, 42, 49] para a solução do sistema de equações lineares (2.4.13), escreve-se

$$\Delta\widehat{x}^{t,c+1}_{rs} = -\frac{1}{\left(\dfrac{\partial F_{rs}}{\partial \widehat{x}_{mn}}\right)\Big|^{t}_{\substack{m=r\\n=s}}}\left\{F_{rs}\Big|_{\hat{x}^{t,c}} + \sum_{\substack{m=0\\m\neq r}}^{M}\sum_{\substack{n=0\\n\neq s}}^{M}\left.\frac{\partial F_{rs}}{\partial \hat{x}_{mn}}\right|_{\hat{x}^{t,\tilde{c}}}\Delta\hat{x}^{t,\tilde{c}}_{mn}\right\} \tag{2.4.14}$$

onde $\Delta\widehat{x}^{t,0}_{rs} = 0$, c o contador de iterações do método iterativo de Gauss-Seidel, e

$$\tilde{c} = \begin{cases} c+1 & ,(m<r)\ ou\ (m=r\ e\ n<s)\\ c & ,caso\ contrário\end{cases}$$

$$\begin{aligned} F_{rs} = &-2\sum_{i=r-N}^{r+N}\sum_{j=s-N}^{s+N}\left[\left(y_{ij} - \sum_{k=-N}^{N}\sum_{l=-N}^{N} b_{kl}\widehat{x}_{i+k,j+l}\right)b_{r-i,s-j}\right] + \\ &+ \frac{\alpha}{q}(\widehat{x}^{q}_{rs} - \overline{x}^{q}_{rs}) \end{aligned} \tag{2.4.15}$$

e

$$F_{mn} = \frac{\partial F_{rs}}{\partial \widehat{x}_{mn}} = 2 \sum_{k=-N}^{N} \sum_{l=-N}^{N} b_{kl} b_{m-r-k,n-s-l} + \alpha \widehat{x}_{rs}^{q-1} \delta(r,m)\delta(s,n) \quad (2.4.16)$$

onde $\delta(.)$ representa o delta de Kronecker.

Critério de escolha da solução do problema de restauração

O critério adotado para determinar qual das imagens restauradas é a mais provável dentro do espaço de soluções baseia-se no menor valor do funcional Q, dado pela Equação 2.4.9. Como o funcional é de natureza convexa, sempre existirá um valor mínimo associado [12]. Isso implica dizer que, a cada iteração, é preciso comparar o valor do funcional Q da imagem recém-restaurada ($Q(\hat{x}^{t+1})$) com o valor obtido na iteração anterior ($Q(\hat{x}^{t})$). Se o novo valor de Q for menor que o anterior, aceita-se a imagem recém-restaurada como solução do problema. Caso contrário, mantém-se a última imagem restaurada com valor Q mínimo como solução.

Critério de parada do laço principal

O laço principal do algoritmo baseado na minimização do Funcional de Regularização de Tikhonov consiste em uma restauração completa da imagem e esse processo pode ser interrompido quando:

(a) para todos os pontos da imagem se tenha alcançado uma tolerância definida *a priori*:
$e(i,j) = \left| \frac{\hat{x}_{ij}^{t+1} - \hat{x}_{ij}^{t}}{\hat{x}_{ij}^{t}} \right| \leq TOL,\ i,j = 0, ..., Mx,$ com $\hat{x}_{ij}^{t} \neq 0,$
onde e é o erro relativo das imagens restauradas nos tempos $t+1$ e t e TOL é uma tolerância definida *a priori*;

(b) o processo iterativo alcançar um dado limite máximo de iterações $maxIter$, de modo a evitar ciclos (*loops*) infinitos; o que ocorrer primeiro.

2.4.1 Fluxograma do algoritmo de restauração serial

Na Figura 2.4, é apresentado o fluxograma simplificado do algoritmo de restauração dado na Seção 2.4, para implementação em um computador serial (com apenas um processador, sem vetorização). Um pseudo-código do algoritmo de restauração da Figura 2.4 é apresentado a seguir:

1. Faça $\widehat{x}_{ij}^{0} = y_{ij}$ e $\bar{x}_{ij}^{0} = y_{ij}$

2. Faça $\widehat{x}^0$ a solução inicial do problema
3. Faça $minQ = Q(\widehat{x}^0)$, o valor mínimo inicial do funcional Q
4. Iniciar maxIter com o número máximo de iterações
5. $t = 0$
6. Repetir
7. $t = t + 1$
8. Calcule as derivadas F_{rs} com Equação 2.4.15 e F_{mn} com Equação 2.4.16
9. Calcule as correções $\Delta\widehat{x}$ com Equação 2.4.14
10. Calcule as novas estimativas $\widehat{x}^{t+1}$ com Equação 2.4.12
11. Calcule o valor do funcional $Q(\widehat{x}^t)$ com Equação 2.4.9
12. Se $Q(\widehat{x}^t) < minQ$
13. Então Aceitar $\widehat{x}^t$ como solução do problema de restauração da imagem
14. Faça $minQ = Q(\widehat{x}^t)$
15. até $Q(\widehat{x}^t) > Q(\widehat{x}^{t-1})$ ou t=maxIter

2.5 Tratamento dos pontos da imagem próximos das bordas

Toda imagem digital é limitada espacialmente, ou seja, ela possui um tamanho fixo, com dimensões finitas. Como a matriz de borramento B define uma área de influência ao redor de um ponto na imagem (Figura 2.3c), percebe-se que os pontos próximos das fronteiras (bordas) da imagem, diferentemente dos demais pontos mais internos, sofrem influências de elementos que extrapolam essas extremidades, que existem no plano objeto, mas que não estão registradas na imagem digital (plano imagem) (Figura 2.5).

Em processamento de imagens, a palavra "borda" é usada, normalmente, para fazer referência às bordas de um ou mais objetos registrados em uma imagem, onde tecnicamente ocorre uma mudança brusca de intensidade. Esse termo também pode ser estendido para incluir as fronteiras do domínio de definição da imagem. Assim, para se fazer uma diferenciação entre essas definições, neste livro o termo "borda da imagem" (ou simplesmente "borda") é usado preferencialmente para fazer referência à fronteira do domínio de definição da imagem e/ou dos subdomínios

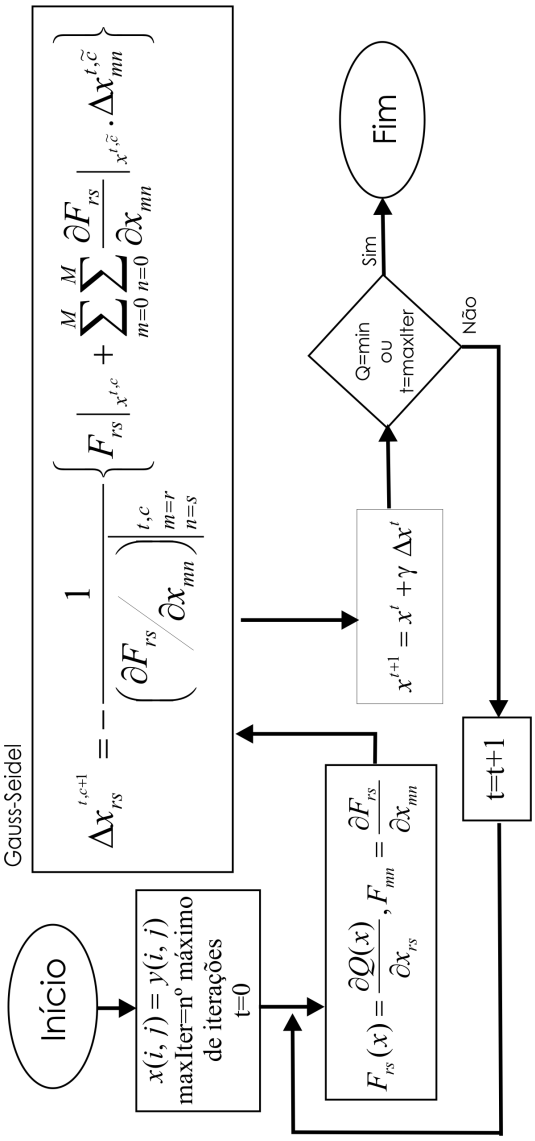


Figura 2.4: Fluxograma do algoritmo serial de restauração de imagens de AFM [11, 12].

usados nas partições, enquanto que o termo “magnitude de borda” é usado para

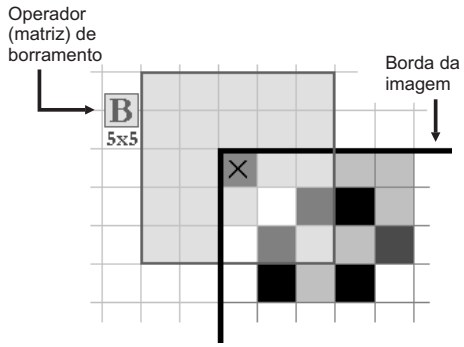


Figura 2.5: Visualização da área de influência do operador de borramento definida por uma matriz B de dimensão 5×5 ao redor do ponto da imagem, marcado com um "X".

fazer refência às bordas dos objetos em uma imagem, normalmente, obtidos por operador de detecção de bordas como, por exemplo, Sobel [20, 21].

Analisando-se a Equação 2.3.5, verifica-se que ela descreve como uma imagem experimental é composta ponto a ponto, mas ela não faz referência a nenhum tipo de tratamento de como o cálculo deve ser feito para os pontos próximos da borda. Para resolver esse tipo de problema, uma solução que pode ser adotada é o emprego da técnica de corte, onde o cálculo dos pontos próximos da borda é redefinido de modo a desconsiderar os elementos que não estão representados na imagem. Desta forma, os elementos da área de influência ao redor de um ponto na imagem, que extrapolam os limites da borda, são retirados do cálculo da Equação 2.3.5 e, para que a condição descrita na Equação 2.3.7 ainda seja satisfeita, a matriz de borramento B, nesses casos, precisa ser ajustada e os valores dos seus elementos renormalizados, de acordo com o número de pontos pertencentes à área de influência de B, que estão efetivamente registrados na imagem digital.

A título de exemplo, na Figura 2.6 é apresentado um ajuste de uma matriz de borramento B de dimensão 5×5, com variância $\sigma^2 = 20$, para ser usada no cálculo do ponto localizado no canto superior esquerdo da imagem, conforme ilustrado na Figura 2.5.

2.6 Técnica de realimentação

Na Equação 2.4.8, o termo $\bar{x}$ é um valor de referência, podendo este ser a própria imagem obtida experimentalmente pelo AFM, usado no cálculo da função de regularização S, permanecendo fixo durante todo o processo de restauração (Figura 2.7).

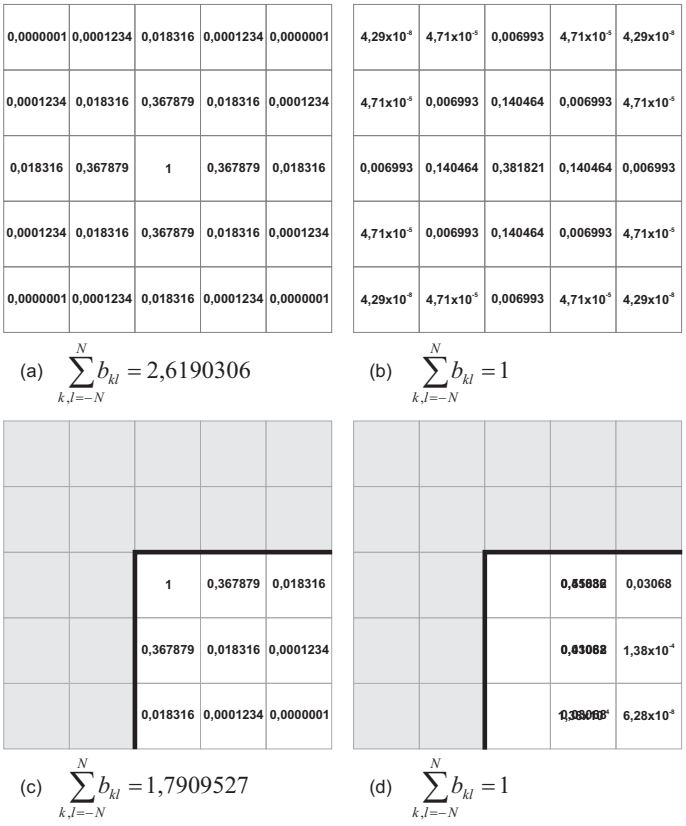


Figura 2.6: Exemplo de uma matriz de borramento B de dimensão 5×5 com (a) matriz completa com valores não normalizados, (b) matriz completa normalizada, (c) matriz parcial modificada e ajustada para o pixel do canto superior direito da imagem e (d) matriz parcial modificada normalizada.

Em 2002, entretanto, Furtado [19] propôs em sua dissertação de mestrado uma modificação nessa técnica, que ficou denominada como técnica de realimentação. Conforme ilustrado na Figura 2.8, esta técnica consiste, basicamente, em aplicar uma imagem inicial de referência qualquer em $\bar{x}^0$ (p.ex. a própria imagem que está sendo restaurada) e, a cada início de um novo ciclo de iteração, aplicar a imagem restaurada no ciclo anterior como referência para o novo ciclo de restauração ($\bar{x}^t = \hat{x}^t, t = 1, 2, ...$).

Furtado [19] descobriu que o emprego da técnica de realimentação, de algum modo, acelera o processo de restauração, fazendo com que a execução do programa

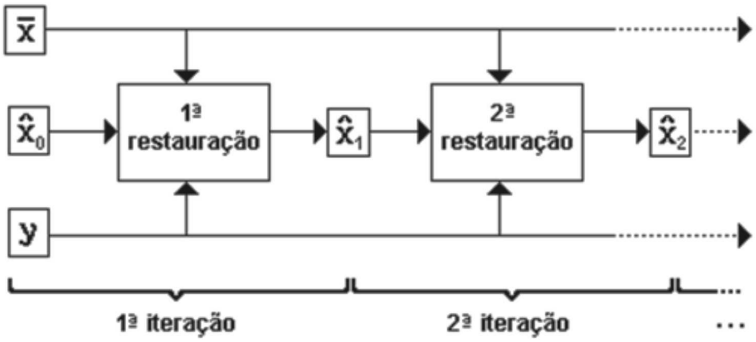


Figura 2.7: Ciclo de iterações do processo de restauração de imagens, usando uma referência fixa.

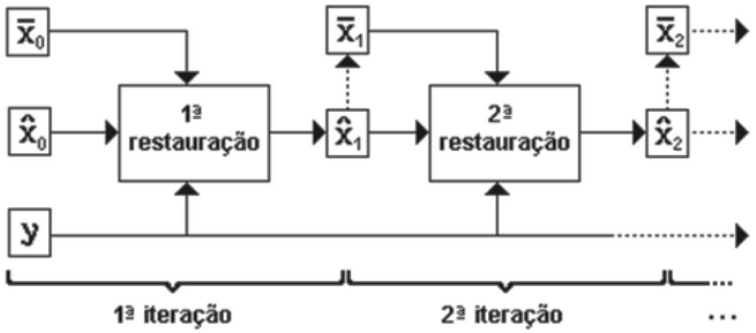


Figura 2.8: Ciclo de iterações do processo de restauração, usando realimentação.

alcance o valor mínimo Q em um número menor de iterações, resultando, consequentemente, em um tempo menor de processamento, empregando realimentação, quando comparado com os tempos obtidos com o mesmo programa de restauração, porém, empregando uma imagem de referência fixa (Figura 2.9).

2.7 Simulação dos efeitos degenerativos em uma imagem

Os efeitos de borramento nas imagens podem ser simulados, empregando-se a função de espalhamento (*PSF - Point Spread Function*) dado pelo operador de borramento do tipo gaussiano com variância (σ^2) ajustável, apresentado na Equação 2.3.6. Para introduzir o ruído aditivo η na imagem modificada (borrada),

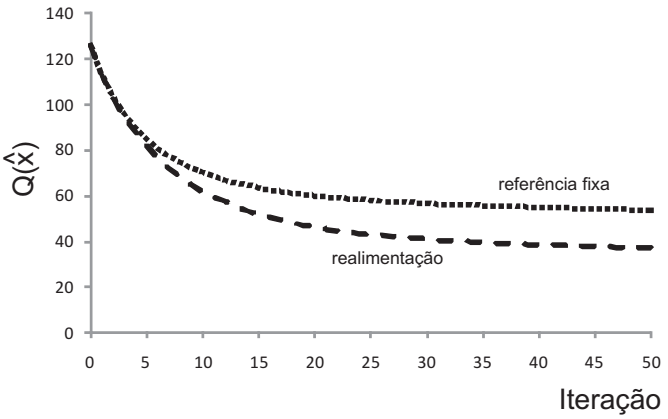


Figura 2.9: Gráfico da evolução dos mínimos quadrados obtidos, a cada iteração do programa, restaurando-se uma mesma imagem (a) usando como referência uma imagem fixa e (b) empregando-se realimentação [19].

pode-se utilizar um gerador de números pseudo-aleatórios [11, 21] que expressa uma relação sinal-ruído desejada em decibels (dB),

$$(SNR)_{dB} = 10\, log \left(\frac{\sum_{i,j=0}^{M} [g(i,j) + \eta(i,j)]^2}{\sum_{i,j=0}^{M} [k\bar{\eta}(i,j)]^2} \right) \tag{2.7.17}$$

onde g é a imagem borrada, (i, j) as coordenadas dos pontos dessa imagem e k é um fator de correção que determina a intensidade do ruído que será adicionado em cada ponto da imagem g, imposto sobre uma distribuição gaussiana normalizada $\bar{\eta}(i, j)$, para uma dada relação sinal-ruído SNR_{dB} expressa em decibels.

Para se identificar as modificações impostas numa imagem (p.ex. imagem-padrão), adotou-se a mesma notação empregada por Furtado [19]), cuja sintaxe é dada por:

$$D\beta v\delta s\xi$$

onde β é a dimensão da matriz de borramento B, δ a variância e ξ é a relação sinal-ruído (SNR - *Signal Noise Ratio*) em decibels (dB).

Por exemplo, para um borramento da imagem-padrão empregando uma matriz B de dimensão 5×5, variância $\sigma^2 = 20$ e ruídos aditivos de 20 dB e 40 dB, as imagens modificadas serão identificadas, respectivamente, como D5v20s20 e D5v20s40,

onde **D** representa a dimensão da matriz de borramento (associada à PSF), v é a variância e **s** a relação sinal-ruído em decibels (Figura 2.10).

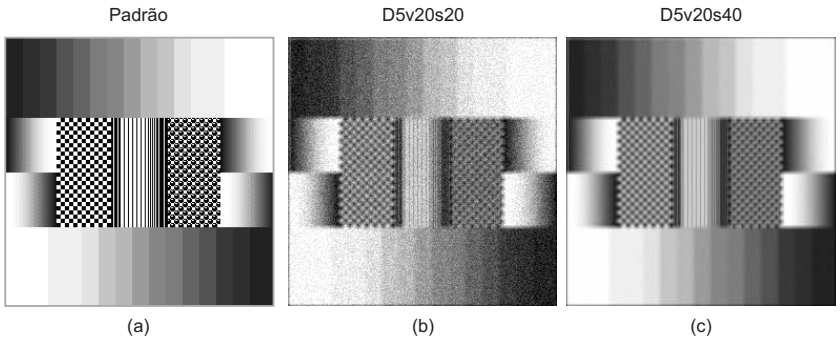


Figura 2.10: Imagem-padrão (a) e imagens modificadas usando uma matriz de borramento B de dimensão 5×5, e ruídos aditivos de 20 dB (b) e 40 dB (c).

2.8 Resultados da restauração com o algoritmo serial

Nas Figuras 2.11 e 2.12 são mostrados alguns exemplos de restauração, empregando a técnica aqui descrita. Na Figura 2.11 são mostradas restaurações de algumas imagens de AFM selecionadas da Figura 1.6 e na Figura 2.12 são mostradas restaurações de imagens artificialmente modificadas.

Uso de Realimentação × Referência fixa

Para efeito de exemplo e comparação, na Figura 2.13 são ilustradas algumas restaurações da imagem-padrão modificada (D5v20s40) (Figura 2.13a), uma usando a técnica de realimentação (Figura 2.13b) e a outra fixa (Figura 2.13c). Como referência $\bar{x}$, empregou-se nas restaurações a própria imagem-padrão modificada.

Como o processo de realimentação só é aplicado a partir da segunda restauração, os valores apresentados ao lado das duas restaurações (uma usando referência fixa e outra realimentação) na primeira iteração (primeira restauração) serão exatamente iguais. Só após a segunda restauração, é que se observa alguma variação no valor do funcional Q, Equação 2.4.9. Junto às imagens restauradas, são mostradas também algumas medidas de qualidade, que serão apresentadas no Capítulo 4, apontando uma pequena melhora, a cada iteração, na qualidade das imagens restauradas usando a realimentação, em relação às mesmas restaurações feitas na

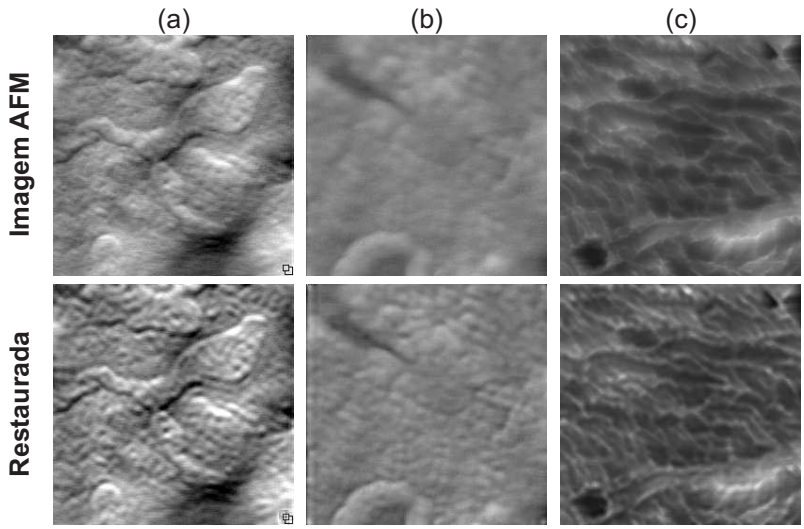


Figura 2.11: Exemplo de restaurações de imagens de AFM.

imagem modificada, sem aplicar essa técnica (referência fixa). Tomando-se apenas o valor do MSE, por exemplo, observa-se que apesar de ambas restaurações iniciarem com o mesmo valor ($MSE = 3143,45$), após 50 iterações (ciclos completos de restauração), o processo de restauração usando realimentação apresenta um resultado melhor ($MSE = 1340,01$) do que aquele obtido com o processo de restauração usando referência fixa ($MSE = 1475,34$).

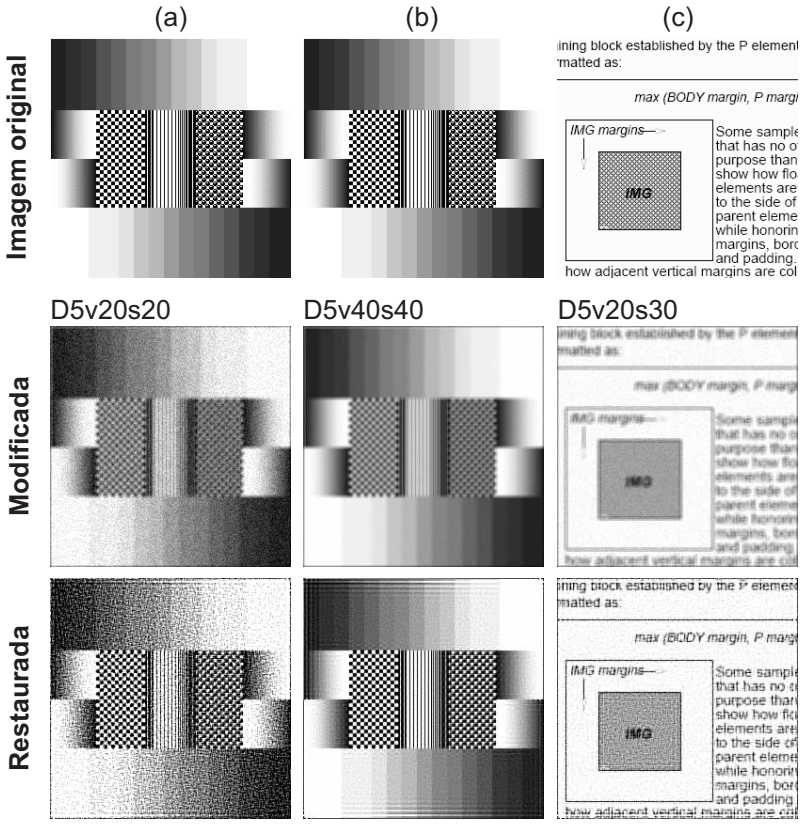


Figura 2.12: Exemplo de restaurações: imagens-padrão modificadas (a) D5v20s40 e (b) D5v40s40 e (c) imagem-texto modificada

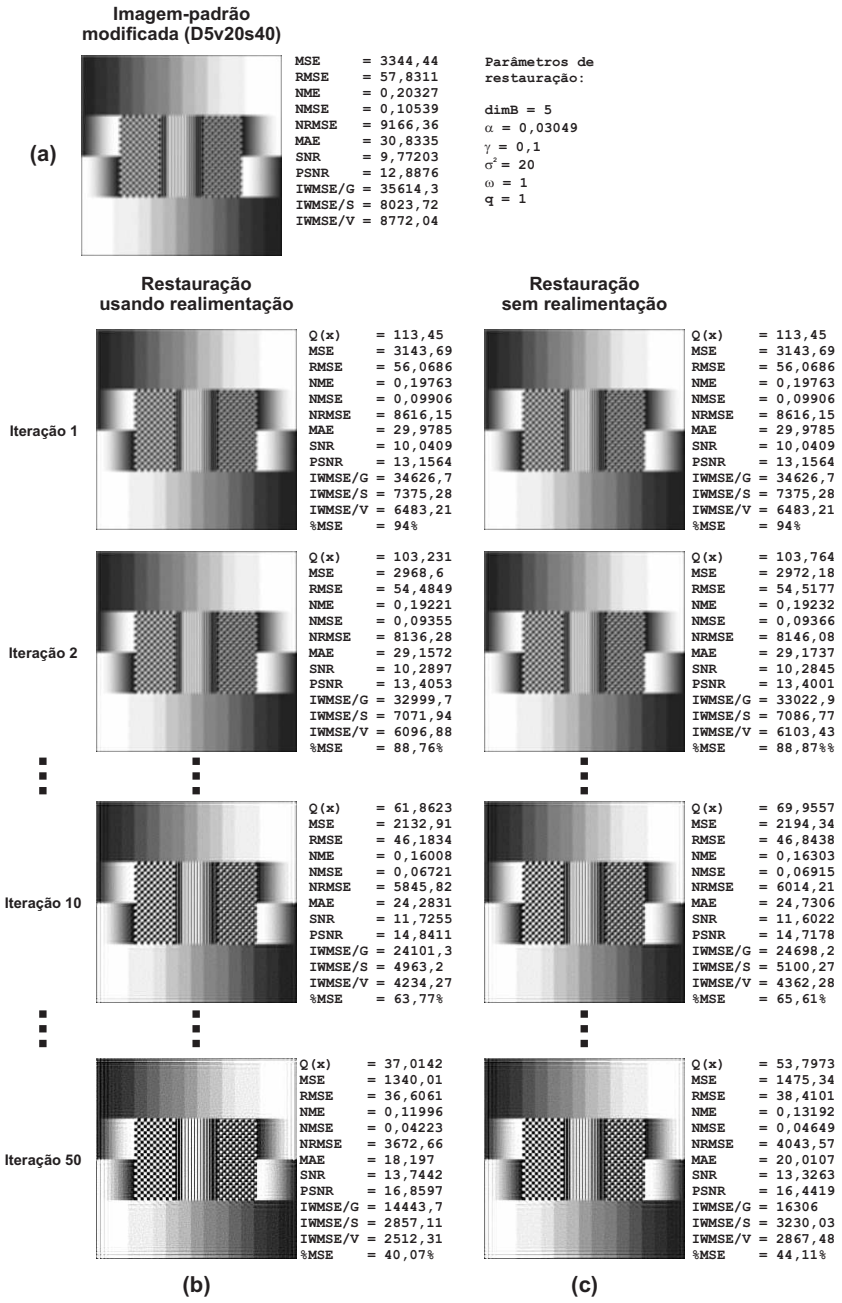


Figura 2.13: Restaurações da imagem-padrão modificada - D5v20s40 (a), usando a técnica de realimentação (b) e sem aplicá-la (c).

Capítulo 3

Implementações Computacionais Paralelas

3.1 Desempenho computacional

Há muito tempo se sabe que a tecnologia de fabricação de microprocessadores vem se aproximando do seu limite físico e de velocidade. Apesar disso, entretanto, para contornar este problema, a indústria de microprocessadores vem adotando novas técnicas e tecnologias mais modernas e, em função disto, ela tem conseguido alcançar desempenhos e velocidades de processamento cada vez maiores.

Esse aumento de processamento, entretanto, não tem ocorrido suficientemente rápido para fazer frente à demanda, cada vez maior, por mais poder computacional que se têm sido exigido no passar dos anos. Boa parte parte disto ocorre em virtude do surgimento de novas demandas (p.ex. restauração de vídeos ao invés de imagens) ou, simplesmente, ao fato de que muitos problemas, que há alguns anos exigiriam demasiado tempo computacional para serem processados nas máquinas disponíveis à época, hoje são factíveis de serem resolvidos, empregando-se os equipamentos atualmente disponíveis no mercado.

Para atender a necessidade cada vez maior de poder de processamento, uma solução comumente adotada é fazer uso de equipamentos mais sofisticados e/ou dedicados. Mas tal decisão, entretanto, nem sempre é a melhor saída para o problema, pois a aquisição e a manutenção desses equipamentos, normalmente, têm custos bem mais altos.

Como alternativa, pode-se empregar técnicas de computação paralela na busca de melhores desempenhos computacionais, a um custo bem menor, usando hardwares e softwares que estejam a nossa disposição ou que possam ser facilmente adquiridos a um custo mais baixo no mercado. Porém, essa decisão também leva a uma

dificuldade, mas de outra natureza: para que seja efetiva e alcance o máximo de desempenho, a programação em um ambiente paralelo deve ser específica e otimizada para a arquitetura em que está sendo utilizada e fica a cargo do programador toda a responsabilidade pela geração de códigos paralelos eficientes e que explorem da melhor forma a estrutura e a formulação do problema proposto e a arquitetura utilizada.

3.1.1 Desempenho das versões seriais do programa de restauração

Em 2000, Cidade [11] desenvolveu um programa serial do programa de restauração de imagens com o funcional de regularização de Tikhonov, conforme descrito do Capítulo 2. Apesar do programa realizar as restaurações eficazmente, ele não apresentava um bom desempenho computacional, pois a sua codificação não foi feita de maneira eficiente, aproveitando da melhor forma os recursos avançados de programação disponibilizados pela linguagem C++/ *Builder*®, usada na época para implementá-lo.

Em 2004, uma nova versão otimizada do programa serial de restauração foi desenvolvida por Stutz [52]. Nos testes realizados nos computadores daquela época, a nova versão otimizada chegou a apresentar um desempenho computacional superior, com relação à codificação anterior, na ordem de 70% aproximadamente. Mas apesar disso, entretanto, o esforço computacional exigido pelo método de restauração faz com que o programa de restauração, processado serialmente, alcance um tempo de execução relativamente elevado e apresente baixos desempenhos computacionais.

Com o passar do tempo, a necessidade de se aumentar a velocidade de processamento e do desempenho do programa de restauração têm se tornado um problema cada vez mais crítico. Mesmo que executado em equipamentos mais modernos, o programa de restauração serial otimizado acaba atendendo de forma precária e/ou restrita as novas demandas que vêm surgindo, tais como, por exemplo: a restauração de imagens de maior resolução (restauração de um maior número de pontos); a restauração de imagens de vídeos, produzidos a partir de imagens capturadas pelo AFM (restauração de múltiplos quadros); múltiplas restaurações em paralelo, empregando diferentes parâmetros de controle (possibilidade de escolha entre várias restaurações); ajustes interativos de parâmetros de controle, com respostas rápidas (restaurações *real-time*), etc. Assim, em virtude dos novos problemas e das novas demandas, buscar soluções que reduzam os tempos computacionais e o desenvolvimento de uma nova estratégia de implementação, usando computação paralela, torna-se o caminho necessário e natural para resolvê-los.

3.1.2 MPI - *Message Passing Interface*

O MPI (*Message Passing Interface*) foi desenvolvido por um fórum internacional aberto [35], composto por grandes empresas, centros acadêmicos e governos, para ser um padrão de desenvolvimento de sistemas paralelos, disponível para uma grande variedade de arquiteturas e sistemas (multiplataforma), acessível a vários estudantes e profissionais dos mais variados ramos da ciência e engenharia [51].

O MPI constitui uma biblioteca de definições e funções, para ser utilizada em programas C ou Fortran, e que torna possível o desenvolvimento de sistemas paralelos, usando memória distribuída. Por memória distribuída, aqui se entende que não existe uma memória global compartilhada que possa ser acessada por todos os processadores, que cada unidade de processamento possui a sua própria memória local e que a troca de dados entre as unidades é feita por meio de troca de mensagens (*message passing*), usando uma rede de comunicação como, por exemplo, uma rede local.

Os programas desenvolvidos com o MPI são implementados, normalmente, usando o modelo de programação SPMD (*Single Program Multiple Data*). Nessa abordagem, diversas unidades de processadores são reunidas através de um canal de comunicação físico qualquer (barramento, cabo de rede, etc.) e cada uma delas recebe uma cópia de um mesmo programa e os dados necessários para o seu processamento. Tão logo podem, iniciam a execução do programa paralelo independente dos demais processadores. Ao longo da execução, as unidades podem processar diferentes partes do programa, tomando como base um número de processo (ou *rank*) que identifica cada uma das unidades de processamento. Sempre que necessário, param a execução normal do programa para fazerem trocas de dados/controles com os demais processadores. Na maioria dos casos, o processo de *rank*=0 (também conhecido como *root*) é normalmente escolhido pelo programador MPI para fazer a distribuição e coleta dos resultados em paralelo.

3.1.3 Fatores críticos para a queda de desempenho de programas paralelos

O desejo de implementar programas em paralelo surge, normalmente, da necessidade em se obter um poder computacional maior, de modo a alcançar resultados de processamento cada vez mais rápidos. Ao se incluir mais de uma unidade de processamento na execução de uma aplicação, naturalmente, espera-se com isso que a velocidade de execução e o desempenho computacional cresçam na mesma proporção. Na realidade, entretanto, isso não é observado, pois alguns fatores limitam o uso dos sistemas paralelos, fazendo com que a velocidade e o desempenho sejam menores do que os desejados. Isso se deve a alguns obstáculos existentes na paralelização, que sobrecarregam os programas paralelos e que reduzem a sua eficiência, aumentando os tempos de execução das aplicações em paralelo. Essa

sobrecarga é conhecida na literatura como ***overhead*** e pode ter diversas origens [31, 39].

Quantidade de trabalho ou computação extra

A quantidade de trabalho executada por um programa paralelo é muitas vezes diferente daquela que é executada por um programa serial. Para se implementar um algoritmo em paralelo, são necessárias instruções extras na programação que não existem quando o mesmo algoritmo é implementado de modo serial. Por conta disso, a carga total de trabalho executada pelos processadores rodando uma aplicação em paralelo pode ser maior que a carga de um único processador rodando serialmente a mesma aplicação.

Complexidade dos programas paralelos

Nem sempre, a solução de um problema em paralelo é resolvida de forma simples e fácil. Na maioria das vezes, a solução em paralelo envolve uma complexidade muito maior do que aquele que é exigida quando o mesmo problema é resolvido serialmente. Por exemplo: (a) execução de tarefas em paralelo podem precisar que os processadores se comuniquem, (b) necessidade de sincronização entre os processos, (c) dificuldades na conversão do algoritmo serial para paralelo, (d) nem tudo é paralelizável, etc. Concluindo, o aumento na complexidade dos programas paralelos acaba resultando em codificações extras (*overheads*) no programa em paralelo que não existem no programa serial.

Comunicação entre processadores

Qualquer sistema paralelo não-trivial necessitará que os processadores se comuniquem e troquem dados entre si. O tempo de transferência de dados entre os processadores é normalmente o principal obstáculo dos programas paralelos e origem da maioria dos *overheads*.

Desbalanceamento de carga

Em muitas aplicações paralelas, dependendo do problema (p.ex. busca e otimização), pode ser impossível determinar a quantidade de trabalho e/ou subtarefas associadas aos vários processadores. Já em outros sistemas paralelos, podemos encontrar arquiteturas heterogêneas com diferentes velocidades de processamento. Se diferentes processadores têm diferentes cargas de trabalho, pode ocorrer que, em algum momento, algum processador tenha que ficar aguardando, enquanto que os outros processadores ainda estejam trabalhando no problema. Além disso, em certos momentos, pode ocorrer que os processadores (todos ou apenas uma parte

deles) tenham que se sincronizar. Se nem todos os processadores estiverem prontos para a sincronização ao mesmo tempo, ocorrerá que alguns deles terão que ficar aguardando até que os demais estejam prontos.

Causas comuns de *overheads*

Diversos fatores podem introduzir *overheads* na programação paralela, entretanto, as causas mais comuns são:

(a) tempo de processamento para execução e controle dos programas em paralelo (inicialização, finalização, sincronização, etc.);

(b) tempo de comunicação gasto na troca de dados e/ou controle entre as unidades de processamento;

(c) período de tempo em que uma unidade de processamento fica aguardando por uma comunicação (modo de espera ou ***idle-time***);

(d) tempo gasto em disputas por um mesmo recurso computacional (disco, memória, etc.), também conhecido como **conflito por recursos**;

(e) tempo de espera em que uma unidade de processamento fica aguardando pelo resultado de outra para continuar o seu processamento (**conflito de dados**);

(f) tempo consumido após uma tomada de decisão de parada do sistema (definitiva ou temporária), enquanto as outras ainda estão processando (p.ex. desvios condicionais, interrupções, etc.), conhecido como **conflito de controle**;

(g) sobrecarga imposta pelos compiladores, bibliotecas, ferramentas, sistemas operacionais, etc.; usados na implementação e execução dos programas paralelos, etc.

Efeitos da comunicação no processamento paralelo

O objetivo principal do processamento paralelo é o emprego de várias unidades de processamento na busca de uma solução mais rápida de um problema específico qualquer. Para que as unidades trabalhem de forma conjunta, entretanto, é preciso que haja um canal de comunicação que possibilite a troca de informações entre elas. Nos programas desenvolvidos com o MPI, essa comunicação é feita via troca de mensagens (*message passing*), utilizando uma rede de comunicação qualquer (rede local, internet, etc.), que é significativamente mais lenta que os acessos que uma unidade de processamento faz ao seu próprio hardware como, por exemplo, a

memória. É por isso que, sempre que as unidades enviam e/ou recebem mensagens, elas estarão consumindo, com essa comunicação, um tempo computacional bem maior.

3.2 Decomposição de domínio

Em computação paralela, a decomposição de domínio (*domain decomposition*) é uma técnica de paralelismo de dados empregada para particionar os dados de um problema em unidades de dados menores (ou partições), para serem distribuídas às múltiplas unidades de processamento, de modo que possam ser processadas em paralelo. Nesta técnica existem várias maneiras de se particionar os dados e, dentre elas, podemos citar duas abordagens clássicas:

(a) **Striped Partition** - é a abordagem mais simples, onde uma matriz de dados é dividida em blocos de dados disjuntos menores (partições) e cada um deles é enviado a uma unidade de processamento diferente, separadamente (Figura 3.1).

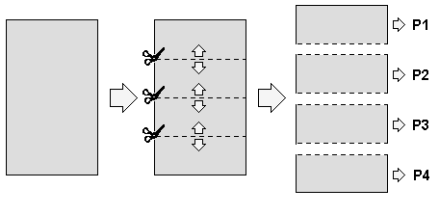


Figura 3.1: *Striped partition.* A matriz de dados é dividida em blocos disjuntos (partições).

(b) ***Overlapping Patition* ou *Multisplittings*** - abordagem, onde uma matriz de dados é dividida em blocos de dados menores sobrepostos (Figura 3.2).

3.2.1 Formas de particionamento

A forma como os blocos de dados são divididos, ou seja como as partições se formam, pode variar muito, de acordo com o problema a ser tratado. Conforme ilustrado na Figura 3.3, as formas de particionamento mais comuns são:

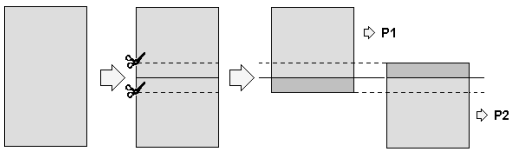


Figura 3.2: *Overlapping partition.* Uma matriz de dados dividida em dois blocos sobrepostos.

(a) **partição 1D ou bloco** - partição formada pela divisão da matriz de dados em uma dimensão apenas. Ex.: partição bloco-linha ou bloco-coluna (Figuras 3.3a e b);

(b) **partição 2D ou bloco-bloco** - partição formada pela divisão da matriz de dados em duas dimensões. Ex.: partição tabuleiro ou bloco-linha-coluna (*checkerboard* - Figura 3.3c).

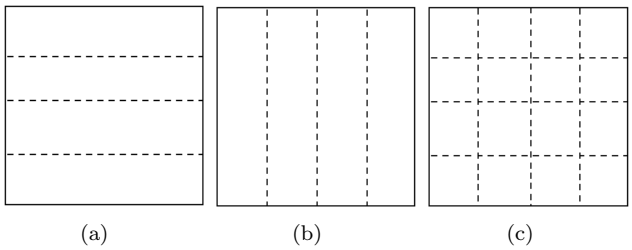


Figura 3.3: Exemplo de partições de domínio: (a) bloco-linha (b) bloco-coluna e (c) tabuleiro.

Quanto ao tamanho dos blocos, eles não precisam ser necessariamente iguais e isto depende muito das características do ambiente de processamento em paralelo utilizado (hardware, software, rede, etc). Quando as unidades de processamento são iguais, ou mais ou menos homogêneas, é sensato utilizar blocos de mesmo tamanho ou de tamanhos aproximados. Por outro lado, se as características dessas unidades forem diferentes (unidades heterogêneas), é aconselhável empregar blocos de tamanhos variados, por exemplo, enviando blocos menores para unidades de menor poder de processamento, que estiverem sobrecarregadas ou que possuírem pouca capacidade de memória, e enviando partições maiores para unidades de maior poder de processamento ou que não apresentem restrições quanto ao tamanho do bloco de dados a ser recebido.

3.2.2 Estratégia de distribuição de partições

Em processamento paralelo, de modo geral, normalmente buscamos empregar um conjunto de unidades de processamento que sejam, mais ou menos, homogêneas em termos de capacidade e processamento. Isso muitas vezes é interessante, pois facilita a distribuição dos dados, uma vez que o processo se resume em distribuir um mesmo volume de dados a todas as unidades de processamento. Entretanto, em algumas situações diferentes volumes de dados podem e/ou devem ser empregados para compensar, por exemplo, as diferenças de capacidade e/ou para minimizar os efeitos do desbalanceamento de cargas entre os processadores. As situações mais comuns nestes casos são:

(a) O emprego de unidades de processamento não-homogêneas; ou

(b) Quando o volume de dados a ser distribuído às unidades homogêneas não é igualmente divisível pelo número de processadores em paralelo (np), etc.

No caso em que unidades heterogêneas são empregadas no processamento paralelo, uma possível solução é fazer uso de heurísticas para avaliar a melhor distribuição, de modo a compensar as diferenças de capacidade e/ou outros fenômenos que venham restringir a velocidade de processamento.

O segundo caso é bastante comum, principalmente, quando a eficiência de um programa em paralelo está sendo estudada, ao se variar o número de processadores. Neste caso, para que os efeitos causados pelo desbalanceamento de carga sejam os mínimos possíveis, uma solução que pode ser aplicada, consiste em realizar uma melhor distribuição das cargas entre as unidades de processamento, de tal forma que um volume v_d de dados, ao ser distribuído a np unidades de processamento, resulte em uma divisão inteira m com resto r ,

$$v_d = m \times np + r,\ 0 \leq r < np, \tag{3.2.1}$$

Uma boa distribuição de cargas, neste caso, seria fazer com que as primeiras $np - r$ unidades de processamento recebam um volume m de dados e as r unidades restantes recebam $m + 1$ (Figura 3.4). Isto faz com que os tempos de espera (*idle-time*) gerados pelo desbalanceamento de carga sejam os mínimos possíveis, visto que eles poderiam ser muito maiores se, por exemplo, uma unidade de processamento recebesse um volume $(m + r)$ de dados e as demais um volume (m) apenas.

3.2.3 Terminologias e convenções

Para uma melhor compreensão das estratégias de implementação do programa paralelo de restauração adotadas, é importante que aqui sejam definidos alguns conceitos e termos que serão empregados neste livro.

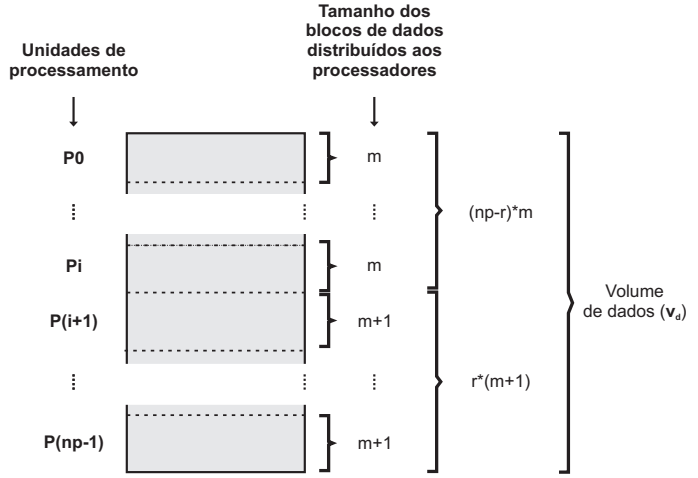


Figura 3.4: Uma estratégia de distribuição de cargas para a minimização de desbalanceamento quando o volume de dados a ser processado não for divisível pelo número de processadores rodando em paralelo.

Partição - resultado da divisão de uma matriz de dados (p.ex. uma imagem) em dois ou mais blocos-linha sobrepostos (*overlapping partition*) de tamanhos aproximadamente iguais, distribuídos para diferentes unidades de processamento para serem restauradas em paralelo (Figura 3.5).

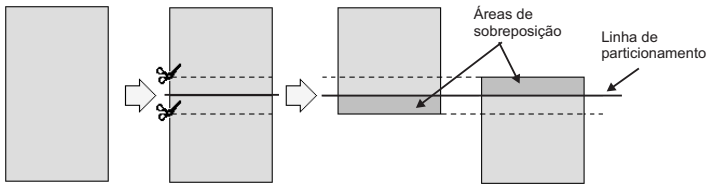


Figura 3.5: Exemplo da divisão de um bloco de dados em duas partições blocos-linha sobrepostas.

Linhas de particionamento - são linhas que dividem uma matriz de dados em um certo número de blocos-linha não sobrepostos de tamanhos aproximadamente iguais (Figuras 3.5 e 3.6).

Áreas de uma partição - conforme ilustrado na Figura 3.6, em uma partição podem ser identificadas algumas áreas específicas. Denominaremos "Área-S", a área da partição que se sobrepõe às partições vizinhas (área de sobreposição),

e "Área-A", o restante dela. A área-S subdivide-se em duas partes: S0 e S1. A área-S0 sobrepõe a partição vizinha acima e S1 aquela abaixo. A área-A, por sua vez, subdivide-se em três partes: A0, A1 e A2. A área-A1 é a parte da área-A que não é sobreposta por nenhuma partição, enquanto que A0 e A2 são, respectivamente, as áreas sobrepostas pelas partições vizinhas. Visto que a primeira e a última partições não possuem, respectivamente, vizinhos acima e abaixo, cabe aqui ressaltar que a primeira partição não tem as áreas S0 e A0 e a última partição não possui as áreas A2 e S1.

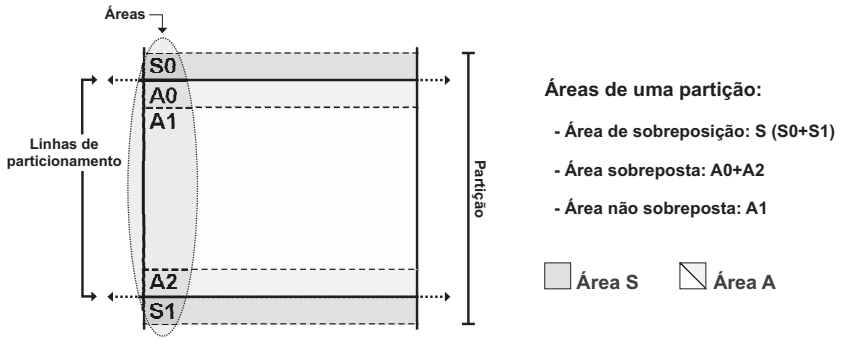


Figura 3.6: Áreas de uma partição.

Classes de partições - para a solução computacional em paralelo do problema de minimização descrito na Seção 2.4, além da imagem degradada outras matrizes de dados são necessárias na aplicação da técnica de restauração. Ao particionarmos a imagem, estamos particionando de algum modo outras matrizes de dados e, para que essas matrizes possam ser identificadas, por um abuso da linguagem, serão definidos aqui alguns termos para as partições de dados com base em seu conteúdo. Denomina-se, portanto:

(a) Y - a partição da imagem experimental (y) a ser restaurada.

(b) X - a partição da imagem estimada ($\hat{x}$), obtida pelo processo iterativo de restauração.

(c) BX - a partição da convolução $B * x$.

(d) $Y - BX$ - a partição da diferença entre as partições Y e BX.

(e) Frs - a partição da derivada primeira F_{rs}.

(f) Fmn - a partição da derivada segunda F_{mn}.

(g) ΔX - a partição com as correções $\Delta\hat{x}$.

Segmento - consiste em uma etapa do processo em paralelo que é responsável pela restauração de uma partição da imagem. Em alguns casos, o segmento pode ser usado também para designar o par: unidade de processamento+partição. Dentro de cada um dos segmentos encontramos os outros tipos de partições citados anteriormente.

3.3 Primeira estratégia computacional paralela

Resumidamente, a Figura 3.7 ilustra como o processo de restauração é feito na primeira versão paralela do programa de restauração:

(a) Primeiramente, uma imagem é dividida em np (número total de processadores) partições bloco-linha sobrepostas;

(b) Cada uma dessas partições é enviada a uma unidade de processamento em particular;

(c) Cada processador recebe uma partição e a processa, independente dos demais. De tempos em tempos, as unidades de processamento podem precisar se comunicar para fazerem atualizações e verificações de convergência;

(d) Ao final do processamento, as áreas de sobreposição são descartadas e as partições são reunidas novamente, compondo a imagem final restaurada.

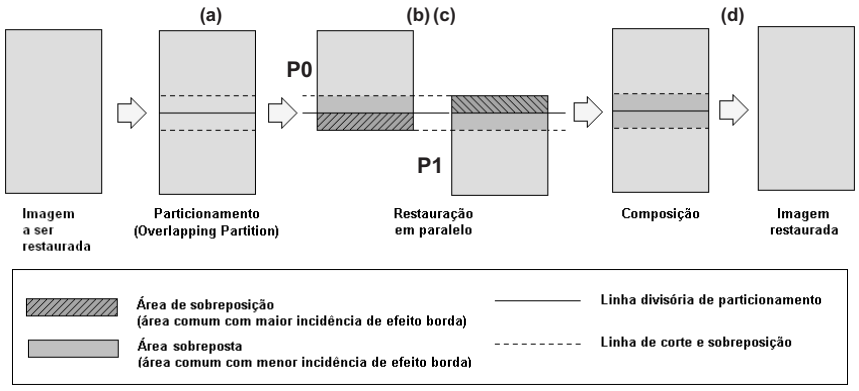


Figura 3.7: Processo de restauração da primeira versão paralela do programa de restauração empregando dois processadores (P0 e P1) e aplicando overlapping partition.

Um aspecto importante dessa primeira estratégia é que a restauração é feita em toda a partição, inclusive nas áreas de sobreposição, resultando em um duplo

processamento de uma mesma região da imagem feito por duas partições vizinhas em paralelo: uma na área de sobreposição de uma partição e a outra na área sobreposta da outra. A vantagem desta estratégia é que ela torna os processadores mais independentes, uns dos outros, além de substituir os tempos de comunicação para troca de dados, cuja execução é mais lenta, por tempos de processamento que é mais rápido. A desvantagem dessa abordagem é a sobrecarga (*overhead*) que é imposta às unidades de processamento ao restaurarem as áreas de sobreposição e o aparecimento de artefatos na imagem restaurada (efeito-borda), cujo problema e proposta de solução são descritos mais adiante neste capítulo.

Nesta estratégia verificou-se também que melhores resultados de restauração podem ser obtidos se em cada iteração as áreas de sobreposição forem atualizadas pela média aritmética simples, ponto a ponto, dessa área com a área sobreposta da partição vizinha (Figura 3.8). Entretanto, ao se fazer isso, a execução torna-se mais lenta, em função dessa troca de dados entre os processadores.

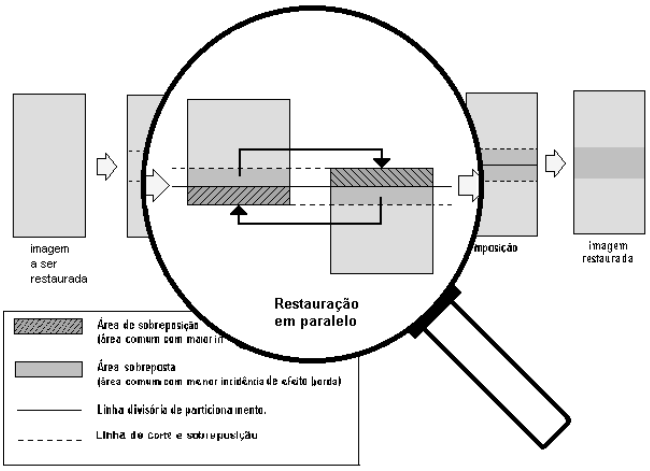


Figura 3.8: Troca de dados entre as unidades de processamento, para atualização por média ponto a ponto das áreas de sobreposição.

Uma vez que a cada iteração as unidades de processamento precisam trocar informações para fazer checagens de convergência, o processo de atualização das áreas de sobreposição pode ser feito nesse momento, aproveitando a comunicação. Um ponto negativo desse processo é que o tempo de processamento e a comunicação gastos (custo operacional) com as atualizações e checagens, pode se tornar prejudicialmente alto, diminuindo o desempenho do programa paralelo.

3.3.1 Fluxograma do algoritmo da primeira estratégia computacional paralela

Na Figura 3.9, é apresentado o fluxograma da primeira estratégia computacional paralela do algoritmo de restauração descrito nesta Seção.

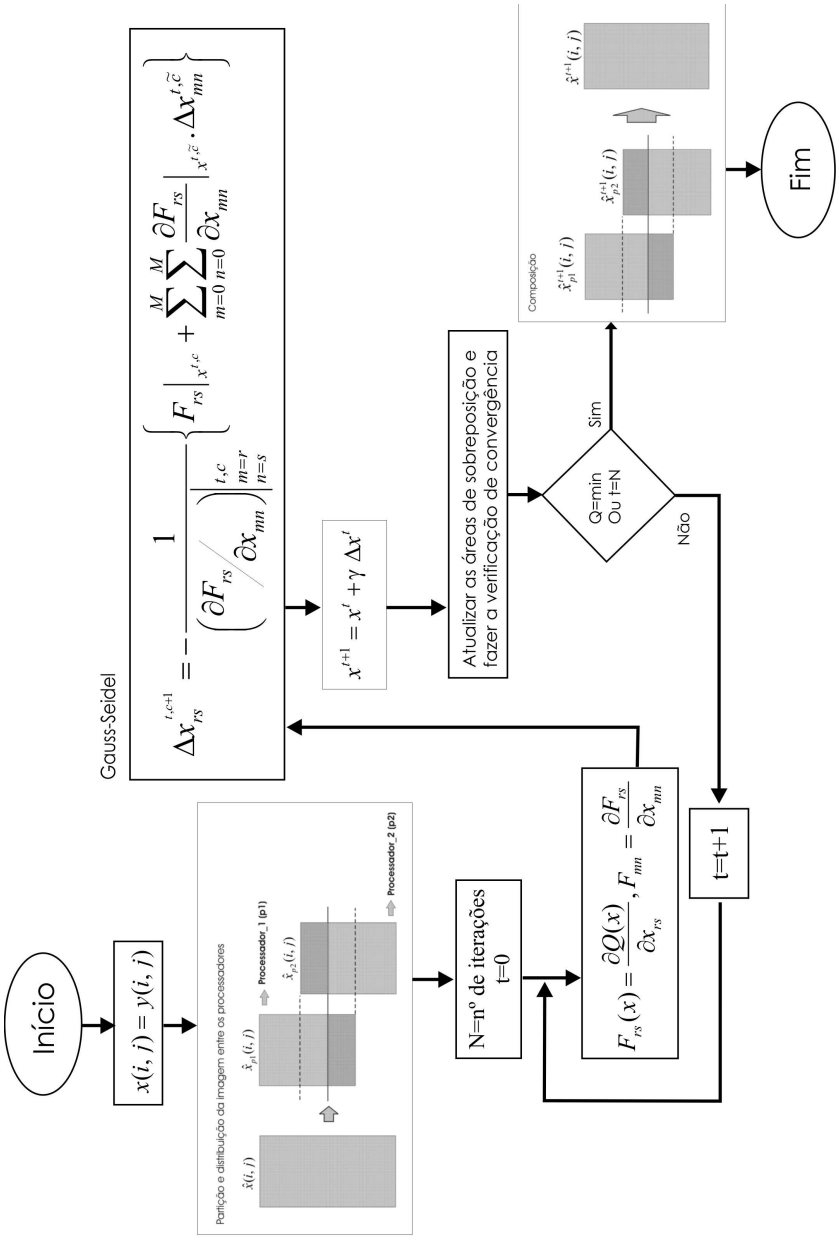


Figura 3.9: Fluxograma do algoritmo da primeira estratégia computacional paralela.

3.3.2 Resultados

Efeito-borda

Na aplicação da primeira estratégia no processo de restauração, entretanto, verificou-se que a decomposição de domínio acabava introduzindo artefatos na imagem restaurada que prejudicam a qualidade das restaurações em paralelo (Figura 3.10).

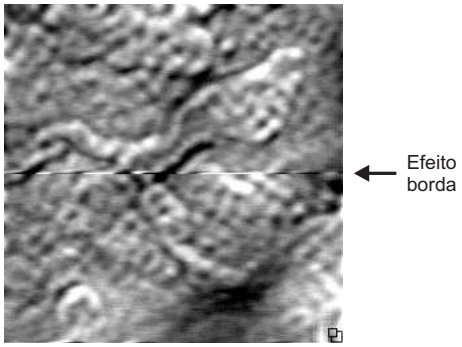


Figura 3.10: Surgimento de artefatos (efeito-borda) em uma imagem restaurada com dois processadores, executando a primeira versão paralela do programa de restauração com a técnica de decomposição de domínio sem sobreposição (*striped partition*), empregando-se os seguintes parâmetros de restauração: $maxIter = 20$, $dimB = 2N + 1 = 21$, $\sigma^2 = 20$, $\alpha = 0,05$, $q = 1$ e $\gamma = 0,1$.

Esse fenômeno, aqui denominado **efeito-borda**, decorre da conjunção de dois fatores: (a) o operador de borramento B que estabelece uma área de influência ao redor dos pontos em uma imagem (Figuras 2.3c e 2.5), e (b) a decomposição de domínio que cria bordas artificiais na imagem original ao particioná-la em blocos de dados menores. A associação desses fatores faz com que os pontos próximos das bordas artificiais em uma partição sofram influências de elementos da imagem que extrapolam estas bordas e que não estão disponíveis na partição (Figura 3.11), visto que elas encontram-se apenas dentro das partições vizinhas.

Solução adotada para minimizar o efeito-borda

Como o tratamento dos pontos próximos das bordas (descrito na Seção 2.5), artificiais ou não, é diferente dos demais pontos internos de uma imagem, a restauração nestas regiões irá sofrer, também, os efeitos desse tratamento, produzindo os artefatos que observamos na imagem restaurada (Figura 3.10). Estes artefatos são ainda maiores quando a matriz de dados é dividida em partições não-sobrepostas (*striped partition*) e menores quando são usadas áreas de sobreposição maiores.

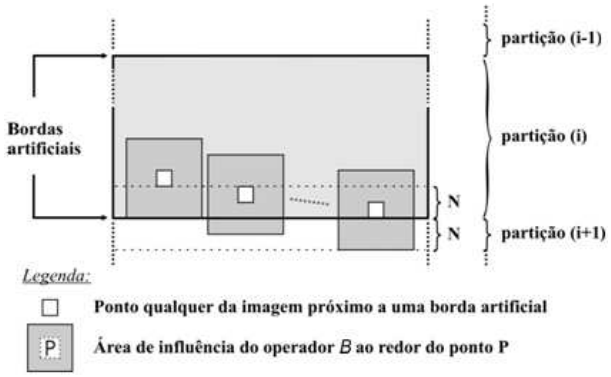
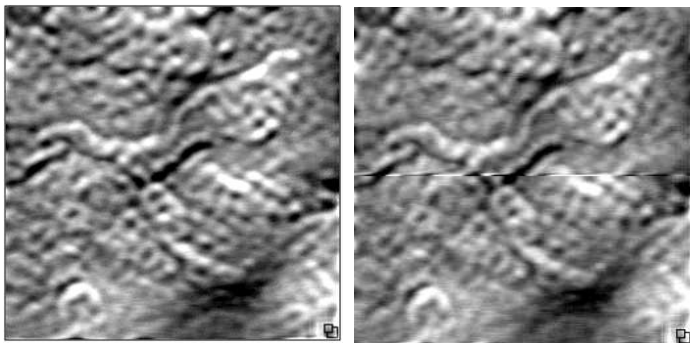


Figura 3.11: Área de influência do operador B próxima a uma borda artificial.

Assim, para reduzir os artefatos observados na imagem restaurada produzidos pelo efeito-borda (Figura 3.10), uma solução é empregar a técnica de sobreposição (*overlapping partition*), dividindo a imagem em vários blocos não-disjuntos sobrepostos, estendendo as bordas artificiais para fora das linhas de particionamento. Como o efeito-borda é mais intenso e ressaltado próximo das bordas, ao se estender as bordas artificiais, também, são estendidas os efeitos-borda. Esta solução faz com que os artefatos observados na imagem restaurada apareçam com uma intensidade maior fora da área não-estendida da partição, ou seja, dentro da área de sobreposição. Assim, após o término do processo de restauração, as áreas de sobreposição são descartadas (área com maior incidência de artefatos) e o restante da partição (com menor incidência) é reunido com aquelas das demais partições vizinhas, formando a imagem final restaurada (Figura 3.12).

Essa estratégia faz com que a imagem restaurada em paralelo sofra menos com os efeitos-borda e o resultado final (a restauração), fique bem próximo da imagem restaurada serialmente. Na Figura 3.13 é apresentado um exemplo de restauração, onde é possível observar estatisticamente essa pequena diferença entre os dois métodos: o serial com MSE=1724,04 e em paralelo apresentando MSE=1744,03. Essa diferença, aparentemente imperceptível a olho nú num primeiro momento, ocorre em função do efeito-borda que foi minimizado, mas não eliminado do processo de restauração em paralelo. Olhando com maior nível de detalhe a área próxima da linha de particionamento de ambas imagens restauradas (Figura 3.14), vê-se mais nitidamente uma diferença visual entre os resultados obtidos com os dois métodos, permitindo perceber uma pequena perturbação na imagem restaurada em paralelo, resultado do efeito-borda. As métricas para a qualidade de restauração MSE e IWMSE são descritas na Seção 4.2.

Dado que o tamanho do operador B ($dimB = 2N + 1$) é conhecido *a priori* e



(a) (b)

Figura 3.12: Imagem restaurada pela primeira versão paralela do programa de restauração usando partições sobrepostas (*overlapping partition*), onde não se observa mais tão claramente o efeito-borda (a), que aparece quando a sobreposição não é empregada (b).

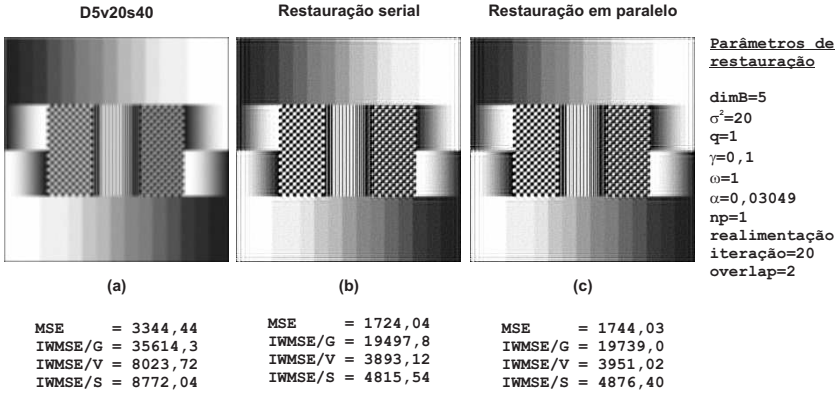
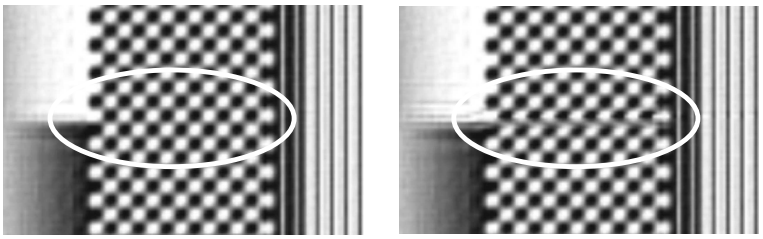


Figura 3.13: Imagem-padrão modificada - D5v20s40 (a) - e resultados de restaurações feitas pelos programas: serial (b) e primeira versão paralela (c). As métricas MSE e IWMSE são descritas na Seção 4.2

que a área de influência se estende N linhas por sobre a partição vizinha, o tamanho da área de sobreposição deve ser de no mínimo N linhas (*overlap area* $\geq N$). Quanto maior for a área de sobreposição (*overlap area*) utilizada, menores são os efeitos (artefatos) observados na imagem final restaurada em paralelo. Porém, as



(a) (b)

Figura 3.14: Comparação visual entre as duas restaurações serial (a) e primeira versão paralela (b). No detalhe, a diferença entre os resultados obtidos com os dois métodos.

sobrecargas de trabalho (*overheads*) serão maiores em cada processador.

3.4 Segunda estratégia computacional paralela

3.4.1 Natureza sequencial dos cálculos

A técnica de restauração baseada na minimização do Funcional de Regularização de Tikhonov, estabelece uma sequência de execução de cálculo no processo de restauração.

Conforme ilustrado na Figura 3.15, observa-se que, para calcular as novas estimativas $\hat{x}^{t+1}$, Equação 2.4.12, é preciso que se calcule antes as correções $\Delta\hat{x}$, Equação 2.4.14. Contudo, para se obter essas correções, é preciso que se calcule antes as derivadas primeira F_{rs} e segunda F_{mn}, Equações 2.4.15 e 2.4.16 e essas, por sua vez, para serem obtidas demandam, respectivamente, o cálculo da diferença $Y - BX$ e do produto $b_{kl}b_{m-r-k,n-s-l}$.

3.4.2 Estratégia de execução dos cálculos

① Cálculo em paralelo do produto $b_{kl}b_{m-r-k,n-s-l}$

Analisando-se o produto $b_{kl}b_{m-r-k,n-s-l}$, Equação 2.4.16, percebe-se que esse termo não varia ao longo da execução do programa de restauração. Ou seja, ele é constante. Como o termo é dependente apenas da matriz de borramento B, cuja forma e tamanho são definidos *a priori*, o seu cálculo pode ser feito uma única vez de modo independente em cada uma das unidades de processamento e armazenado em uma matriz de dados temporária para ser usada toda vez que for preciso calcular a derivada de segunda ordem F_{mn}. Isto é feito desta forma, pois o cálculo deste termo além de apresentar um tempo de processamento insignificante

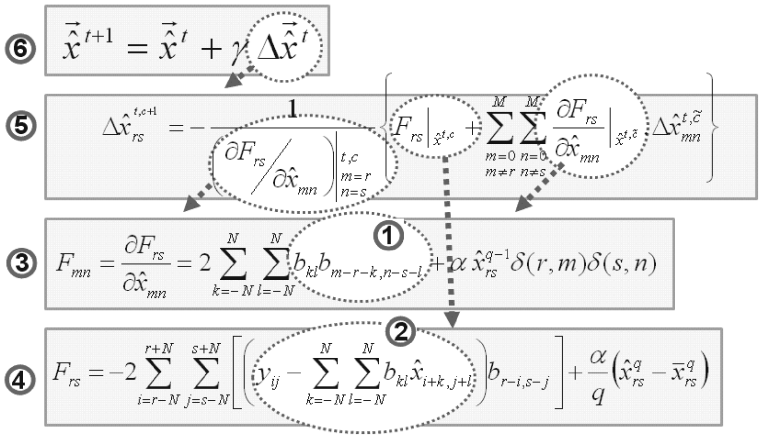


Figura 3.15: Sequência de execução de cálculo dos termos usados no processo de restauração de imagens baseado na minimização do Funcional de Regularização de Tikhonov.

no cômputo do tempo global, ele é mais rápido do que o tempo que seria gasto na comunicação entre as unidades de processamento, se apenas uma delas calculasse o termo e depois o distribuísse às demais unidades.

② Cálculo em paralelo das diferenças $y - B\hat{x}$

Dado que o operador de borramento B cria uma área de influência em torno de um ponto qualquer da imagem (Figura 3.16), nota-se que a partir de um certo limite, conforme esse ponto vai se aproximando da linha de particionamento, os dados necessários ao cálculo da convolução BX vão ultrapassando os limites da própria partição, estendendo-se por sobre a partição vizinha.

Esse limite é dependente da dimensão do operador B e como essa dimensão é determinada por um valor arbitrário N ($dimB = 2N + 1$) definido *a priori*, é fácil concluir que, para se calcular a convolução BX nos pontos próximos da linha de particionamento, é preciso que a partição X seja estendida (sobreposição) em N linhas por sobre as partições vizinhas acima e abaixo (Figura 3.17). Esse alargamento da partição X permite que cada um dos processadores calcule, de forma independente e paralela, a convolução BX e a diferença $Y - BX$ até o limite da linha de particionamento, ou seja, dentro da área-A (Figuras 3.16 e 3.18).

Conforme ilustrado nas Figuras 3.17 e 3.18, o processo de cálculo das diferenças $Y - BX$ pode ser descrito da seguinte forma:

(1) Antes de iniciar o processo de cálculo/restauração, propriamente dito,

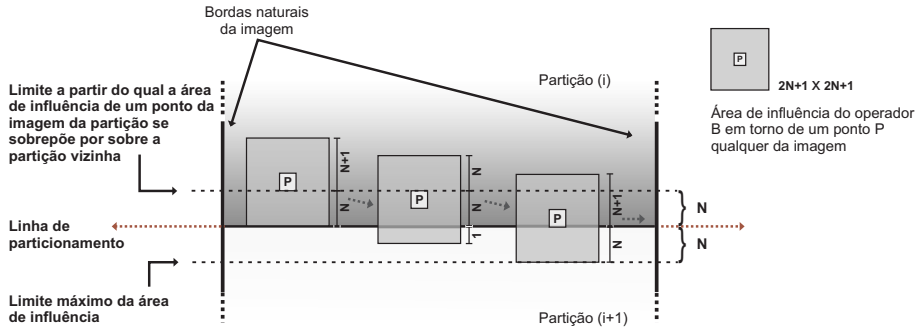


Figura 3.16: Visualização da área de influência em torno do ponto P conforme esse ponto se aproxima da linha de particionamento.

(a) primeiramente, faz-se o particionamento da imagem a ser restaurada em np (número total de processadores) partes com uma sobreposição de N linhas.

(b) A seguir, distribuem-se essas partições resultantes (Y) a cada uma das unidades de processamento (P_j, $j = 0, 1, ...,(np-1)$). Esse processo de carga é executado apenas uma vez, no início da execução do programa.

(c) Como estimativa inicial, o valor da partição X^0 (valor da partição X no instante $t = 0$ de restauração) pode ser uma cópia da partição Y (Figura 3.17). Quanto aos demais valores de X^t ($t = 1, 2, ..$), eles serão obtidos posteriormente, resultantes do processo iterativo de restauração;

(2) Uma vez tendo sido inicializadas as partições X e Y, a unidade de processamento começa o processo paralelo de restauração calculando de forma independente, na área-A, os seguintes termos

(a) BX e, em seguida,

(b) $Y - BX$, faltando apenas o cálculo desse último termo nas áreas de sobreposição (S0 e S1) (Figura 3.18);

De forma a evitar recálculos e novas sobreposições, as diferenças $Y - BX$ da área de sobreposição (área-S) de cada partição, que ficaram faltando no passo (2), podem ser obtidas através da troca de dados entre as unidades de processamento vizinhas, uma vez que as partes que faltam em uma partição podem ser obtidas das outras partições próximas. Detalhadamente, o processo de troca desses dados entre os processadores pode ser descrito da seguinte maneira:

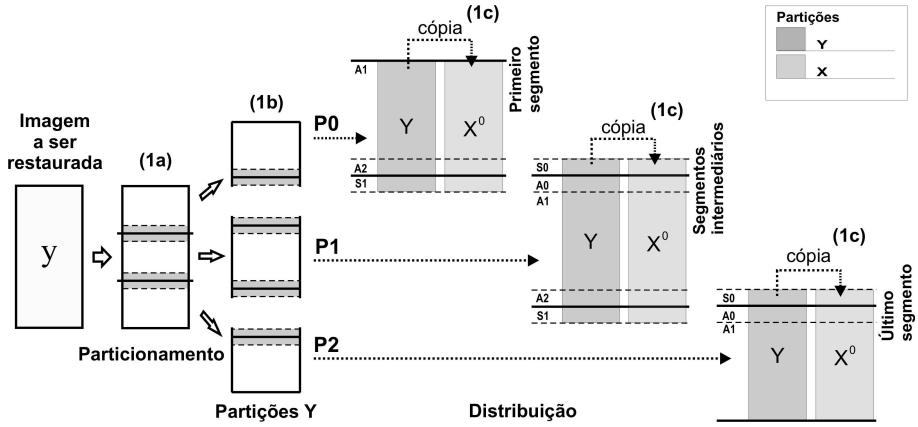


Figura 3.17: Inicialização das matrizes de dados nas unidades de processamento feito antes do início do processo de restauração em paralelo.

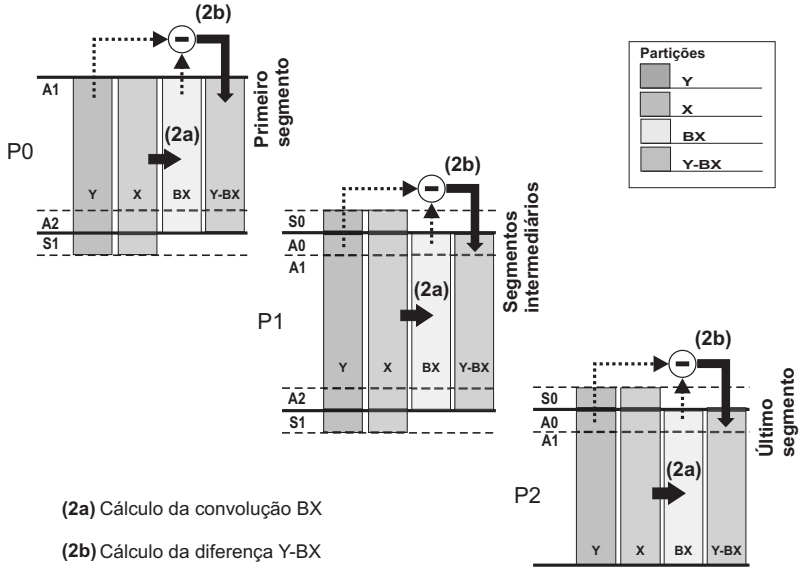


Figura 3.18: Cálculo em paralelo dos termos BX e $Y - BX$.

(3) as áreas A0 e A2 da partição $Y - BX$ de cada unidade de processamento são enviadas para os processadores vizinhos acima e abaixo, de forma a completarem as áreas S1 e S0, respectivamente. O processo de envio de dados

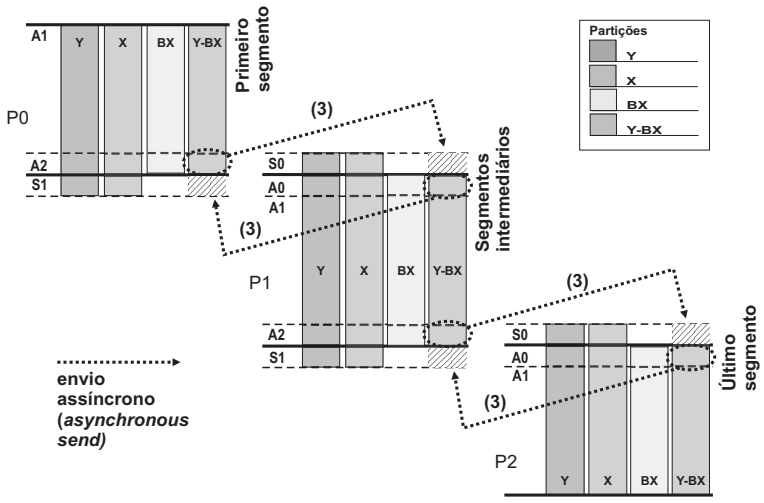
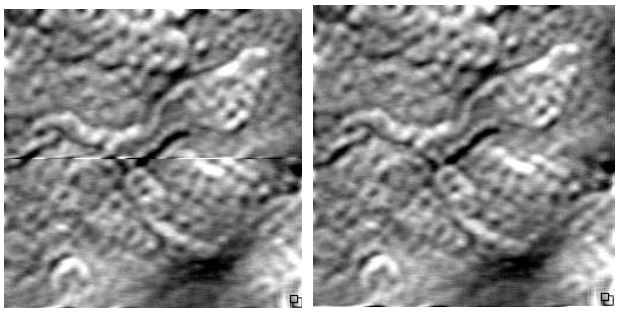


Figura 3.19: Atualização das áreas S0 e S1 necessária ao cálculo da derivada primeira F_{rs}.

pode ser feito de modo assíncrono (*asynchronous send*), aproveitando a área de armazenamento de dados temporário (*buffer*) do canal de comunicação (Figura 3.19). Isso faz com que a operação de envio (*send*) retorne (devolva a execução e o controle aos processadores) antes que os dados tenham sido efetivamente entregues às unidades de processamento vizinhas, possibilitando que cada processador continue o seu processamento, aproveitando o tempo que ficaria aguardando por uma comunicação (*receiver*), para calcular, por exemplo, as derivadas de segunda ordem, reduzindo assim o tempo de latência de comunicação;

A partir deste ponto as duas estratégias paralelas (primeira e segunda) diferem. Na primeira estratégia paralela, descrita na Seção 3.3, a imagem é processada da área-S0 até a área-S1, aplicando-se a técnica de corte nos pontos próximos aos limites da linha de particionamento, o que acaba resultando no efeito-borda observado nas restaurações feitas com a primeira versão paralela do programa de restauração. Enquanto que na segunda estratégia, a restauração é feita apenas na área-A e as áreas S0 e S1 restauradas são obtidas através da troca de dados entre as unidades de processamento vizinhas acima e abaixo, respectivamente. Essa modificação na segunda estratégia faz com que esses pontos não sejam mais tratados de forma diferente, tal como na primeira versão paralela, mas sim tratados de forma igual aos outros pontos internos da imagem. Como resultado dessa mudança, o efeito-borda próximo das linhas de partição é eliminado na segunda

estratégia. Uma comparação dos resultados obtidos com as duas estratégias e apresentada na Figura 3.20.



(a) (b)

Figura 3.20: Comparação dos resultados de restauração obtidos com a primeira e a segunda estrategia.

③ Cálculo em paralelo das derivadas de segunda ordem F_{mn}

Fixando um ponto r, s qualquer dentro da área-A, o cálculo das derivadas de segunda ordem nesse ponto $(\partial F_{rs}/\partial \hat{x}_{mn}\big|^{t}_{\substack{m=r\\n=s}})$ não depende de nenhum dado de fora do segmento. Ao se analisar a Equação 2.4.16, verifica-se que o único termo que a equação precisa encontra-se no próprio segmento, precisamente na área-A da partição X.

Sendo um termo independente das demais partições (exceto de X), o cálculo paralelo das derivadas de segunda ordem pode ser executado tão logo tenham sido enviadas as diferenças $Y - BX$ das áreas A0 e A2, descrito no passo (3). Assim, ao invés de ficar esperando pelos dados das unidades de processamento vizinhas estarem disponíveis para recebimento, o processador pode, enquanto isso, aproveitar o tempo em que ficaria ocioso aguardando a comunicação (*idle-time*) para calcular as derivadas de segunda ordem (F_{mn}).

④ Cálculo em paralelo das derivadas de primeira ordem F_{rs}

Detalhadamente, o processo de cálculo das derivadas de primeira ordem F_{rs}, Equação 2.4.15, pode ser descrito da seguinte maneira:

(4) Após o processo de cálculo das derivadas de segunda ordem,

(a) cada processador aguarda, de forma síncrona, pelo recebimento (*synchronous receive*) dos dados que a ele foram enviados pelas unidades de processamento vizinhas.

(b) O cálculo da partição Frs, limitado apenas à área-A, é iniciado assim que a unidade de processamento receber esses dados (Figura 3.21).

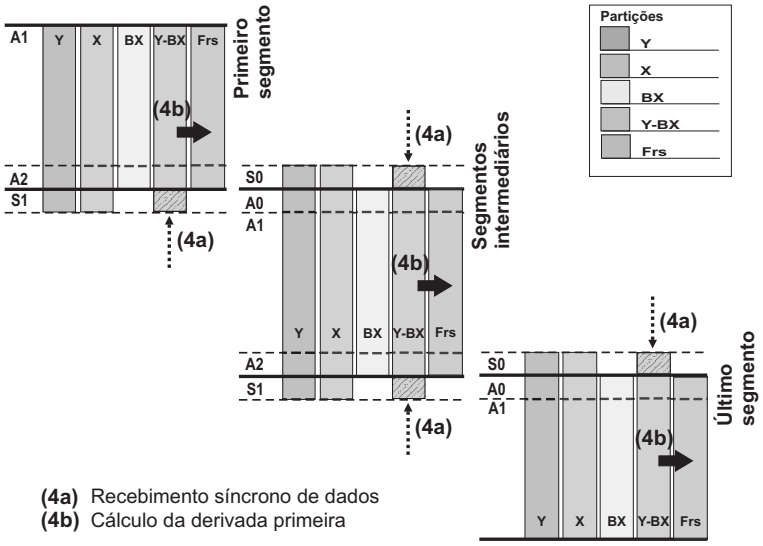


Figura 3.21: Cálculo em paralelo do termo F_{rs}.

⑤ Cálculo em paralelo das correções $\Delta\hat{x}$

O cálculo das correções $\Delta\hat{x}$ exige a solução de um sistema linear pelo método iterativo de Gauss-Seidel, Equação 2.4.14. Esse método estabelece uma ordem em que os elementos da matriz de incógnitas ΔX são calculados. Essa ordem pode ser: (a) de cima para baixo (mais comum); (b) de baixo para cima, ou (c) em ambas as direções, em momentos alternados.

Assumindo-se que a direção de cálculo seja de cima para baixo, por exemplo, também é estabelecida uma sequência de inicialização dos processadores, bem como a ordem de restauração dos pontos da imagem nesse mesmo sentido, ou seja, de cima para baixo. Como efeito dessa escolha, as unidades de processamento com número menor de *rank* começam o processo de restauração primeiro, enquanto que as de número maior ficam aguardando os resultados da partição vizinha acima para iniciarem o seu processamento. Essa espera acaba produzindo um efeito cascata

no processo de cálculo das correções $\Delta\hat{x}$ que, por consequência, acaba introduzindo algum retardo no processamento em paralelo. Na literatura, essa espera por dados é conhecida como **conflito de dados**.

Um outro aspecto a ser observado é que, da mesma forma que o operador B afeta o cálculo da convolução BX ao criar uma área de influência em torno de um ponto da imagem e, consequentemente, a necessidade de se estender algumas partições, o cálculo dos elementos de ΔX também é influenciado por esse operador. Analisando-se as Equações 2.4.14 e 2.4.15, verifica-se que as áreas de sobreposição da partição ΔX são maiores $(2N)$ que as demais partições (N), a partir desse ponto, as derivadas de segunda ordem são nulas. Portanto, para o cálculo das correções, precisamente aquelas que deverão ser obtidas na área de sobreposição (área-S), a partição ΔX deverá ter uma área de sobreposição duas vezes maior $(2N)$ que as demais partições estendidas (N).

De modo a minimizar o conflito de dados, pode-se simplificar o processo de solução do sistema linear em paralelo, assumindo uma estimativa inicial igual a zero $(\Delta\hat{x}^0 = 0)$ e um número máximo de iterações igual a um para o método iterativo de Gauss-Seidel.

Admitindo-se uma estimativa inicial igual a zero, elimina-se a necessidade de sobreposição da partição ΔX com a partição vizinha abaixo. E, ao se estabelecer um número máximo de iterações igual a um, elimina-se a necessidade da partição de baixo transmitir os dados da área-A0 de ΔX para a partição acima, reduzindo a necessidade de transmissão de dados entre as unidades de processamento e, consequentemente, reduzindo o conflito de dados, e minimizando o seu impacto no tempo de processamento e no cálculo das correções ΔX.

Conforme ilustrado na Figura 3.22, o processo de cálculo das correções ΔX empregando Gauss-Seidel pode ser descrito da seguinte forma:

(5) Inicia-se o processo com a primeira unidade de processamento calculando as correções ΔX na área-A, enquanto as outras unidades aguardam;

(6) ao terminar o cálculo, a unidade transmite de modo assíncrono um bloco de $2N$ linhas da área-A2 de ΔX com as correções atualizadas para a unidade de processamento seguinte;

(7) a próxima unidade recebe os dados do processador anterior e,

(8) logo a seguir, é iniciado o cálculo das correções ΔX dentro da área-A em sua partição;

(9) os passos (6), (7) e (8) são repetidos pelas demais unidades de processamento, uma a uma, até que a última unidade de processamento termine o cálculo das correções em sua partição ΔX.

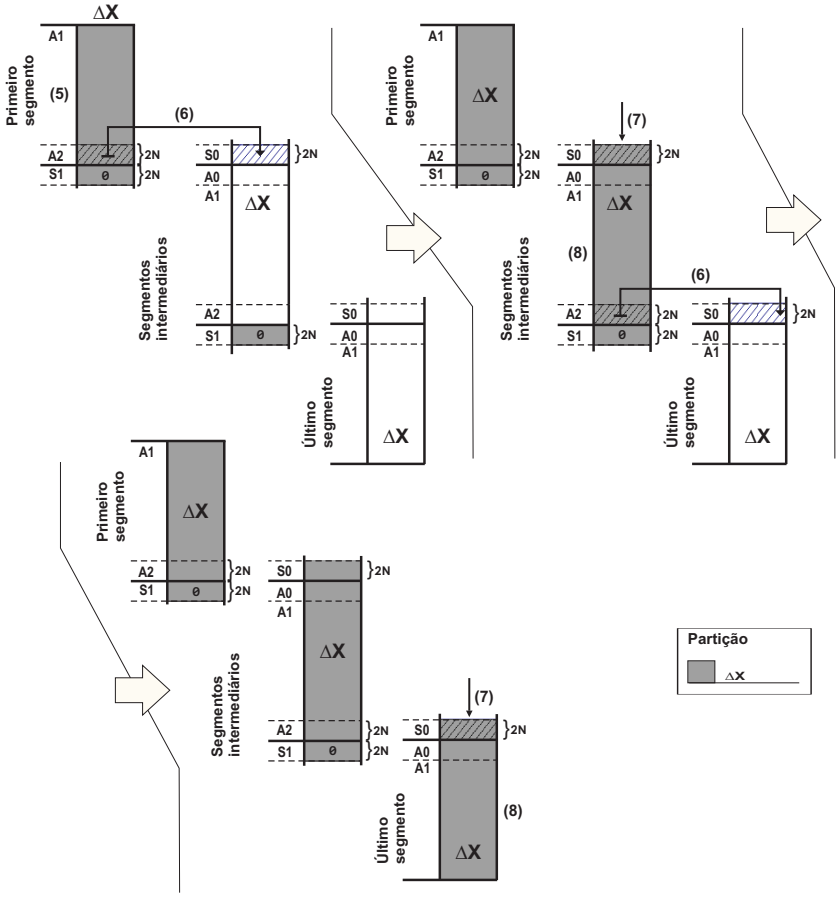


Figura 3.22: Processo de cálculo das correções $\Delta\hat{x}$ em paralelo, empregando Gauss-Seidel.

⑥ Cálculo em paralelo das novas estimativas $\hat{x}^{t+1}$

Conforme ilustrado na Figuras 3.23 e 3.24, o processo de cálculo das novas estimativas pode ser descrito da seguinte forma:

(10) Uma vez obtidas as correções ΔX (passos 5 a 9) e tendo transmitido de forma assíncrona os dados para o processador vizinho abaixo (passo 6), a unidade de processamento pode, então, continuar a execução em paralelo, calculando as novas estimativas X^{t+1} $(t = 0, 1, ..., ni)$ das áreas S0 e A (A0+A1+A2), enquanto os demais processadores inferiores continuam o processamento cal-

culando as correções ΔX.

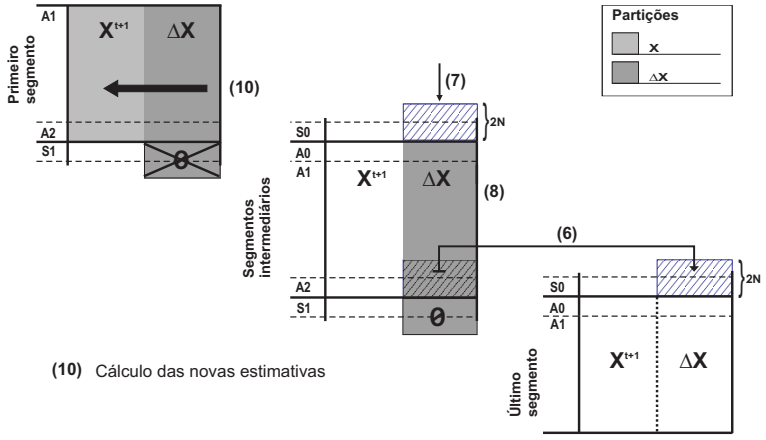


Figura 3.23: Início do processo de cálculo das novas estimativas em paralelo.

(11) Assim que as novas estimativas tiverem sido calculadas,

(a) a unidade de processamento transmite, então, a área-A0 ao processador acima,

(b) para que esta atualize os dados da área-S1.

Ao final do processo, todas as unidades de processamento terão em sua partição X o valor atualizado com as novas estimativas $\hat{X}^{t+1}$. A partir desse momento, os processadores podem fazer os testes de convergência e determinar se o processo de restauração termina ou se uma nova iteração começa (passos 2 a 11).

3.4.3 Resultados

Gauss-Seidel × Jacobi

Apesar dos cuidados tomados para se minimizar o conflito de dados no processo de cálculo das correções $\Delta\hat{x}$ em paralelo, nos testes que foram realizados, detectou-se que o efeito cascata do método iterativo de Gauss-Seidel ainda era muito forte, impactando negativamente nos tempos de processamento da nova estratégia. Conforme se pode observar nos gráficos da Figura 3.25, os tempos obtidos na restauração de uma imagem de dimensão 256×256 com a segunda estratégia paralela

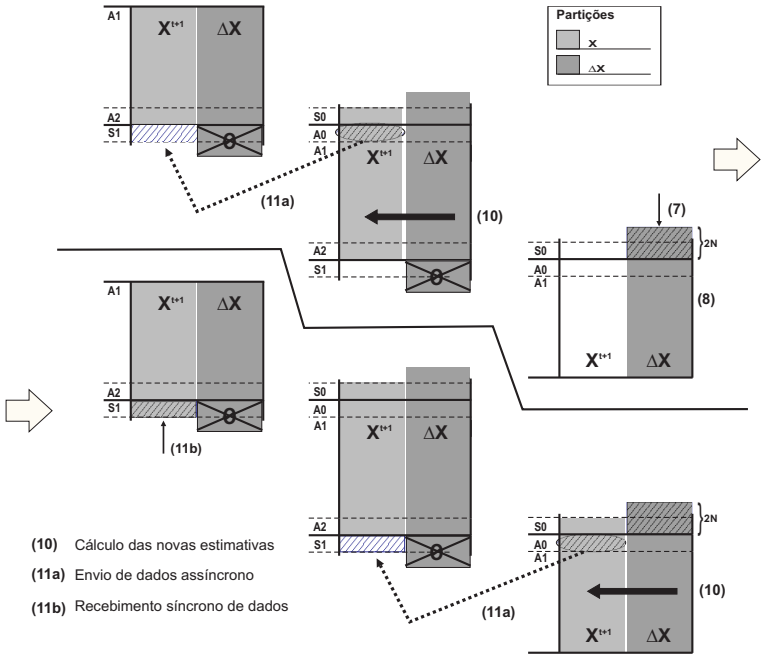


Figura 3.24: Continuação do processo de cálculo em paralelo das novas estimativas.

empregando Gauss-Seidel, até um máximo de oito unidades de processamento, foram piores que os tempos obtidos com a primeira versão paralela.

Ao empregarmos uma imagem de resolução maior (512×512 pixels), uma melhora nos tempos computacionais de ambos os programas em paralelo pode ser observada em relação aos tempos obtidos com o programa serial [52]. Apesar disso, entretanto, o desempenho da segunda estratégia paralela empregando Gauss-Seidel, quando comparado com a primeira versão paralela, ainda é inferior (Figura 3.26).

Após os testes, verificou-se que apesar da sobrecarga de computação extra que existe na abordagem paralela anterior ter sido eliminada, o efeito cascata que ocorre no cálculo das correções $\Delta\hat{x}$ em paralelo introduz um conflito de dados entre as unidades de processamento, o que torna a nova estratégia, até aqui, sem efeito e com baixo desempenho. Uma boa parcela da culpa desse problema pode ser atribuída ao método iterativo de Gauss-Seidel, Equação 2.4.14, empregado na solução do sistema linear, que cria uma relação de dependência entre os pontos e, consequentemente, entre processadores (conflito de dados). Essa dependência aumenta, então, o tempo em que uma unidade de processamento fica aguardando

por comunicação (*idle-time*), impactando diretamente nos tempos de execução do novo programa em paralelo e, por sua vez, inviabilizando o uso da segunda estratégia tal como ela se apresenta até aqui neste momento.

Para viabilizar o uso da nova estratégia é preciso melhorar o seu desempenho, solucionando o problema do conflito de dados causado pelo método de Gauss-Seidel. Uma solução que é comumente adotada consiste em empregar o método iterativo de Jacobi ao invés de Gauss-Seidel na solução do sistema linear que calcula as correções $\Delta\hat{x}$. Assim, no lugar da Equação 2.4.14 para a solução do sistema de equações lineares, pode-se empregar:

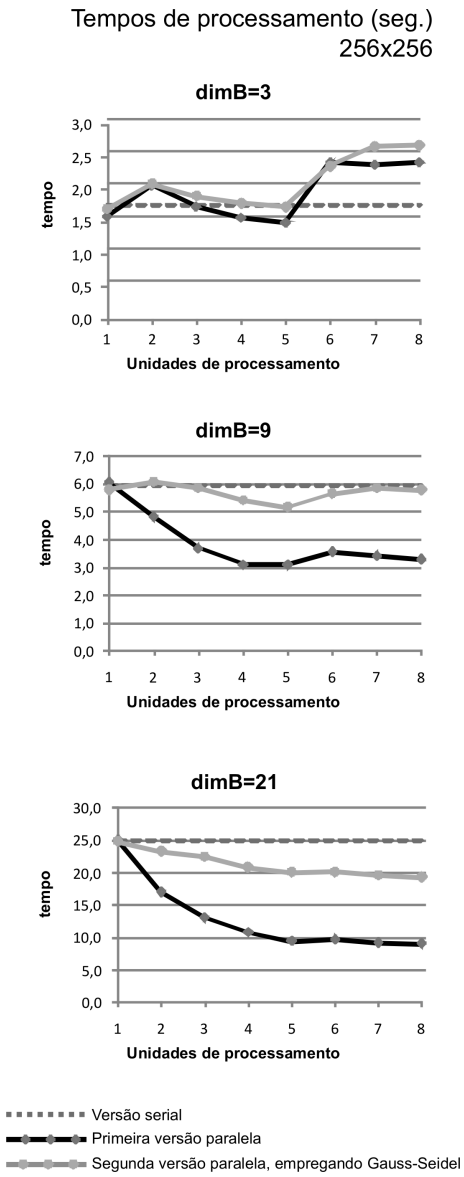


Figura 3.25: Tempos de processamento gastos na restauração de uma imagem de dimensão 256×256 pixels, usando os programas: serial [52] e versões paralelas anterior e a segunda estratégia empregando Gauss-Seidel, para um total de 100 iterações e diferentes dimensões para a matriz de borramento $dimB = 2N + 1 \in \{3; 9; 21\}$.

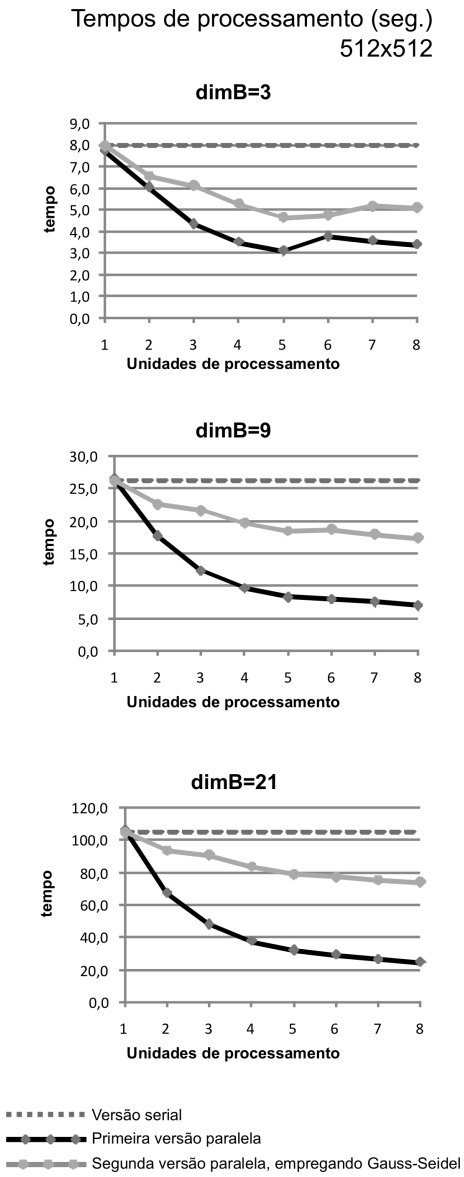


Figura 3.26: Tempos de processamento gastos na restauração de uma imagem de dimensão 512×512 pixels, usando os programas: serial [52] e versões paralelas anterior e a segunda estratégia empregando Gauss-Seidel, para um total de 100 iterações e $dimB = 2N + 1 \in \{3; 9; 21\}$.

$$\Delta\widehat{x}_{rs}^{t,c+1} = -\frac{1}{\left(\dfrac{\partial F_{rs}}{\partial \hat{x}_{mn}}\right)\Big|_{\substack{t,c\\ m=r\\ n=s}}} \left\{ F_{rs}\Big|_{\hat{x}^{t,c}} + \sum_{\substack{m=0\\ m\neq r}}^{M}\sum_{\substack{n=0\\ n\neq s}}^{M} \frac{\partial F_{rs}}{\partial \hat{x}_{mn}}\Big|_{\hat{x}^{t,\tilde{c}}} \Delta\hat{x}_{mn}^{t,c} \right\} \tag{3.4.2}$$

onde $\Delta\widehat{x}_{rs}^{t,0} = 0$ e c é o contador de iterações do método iterativo de Jacobi.

Observe que para a obtenção da Equação 3.4.2 é feito $\tilde{c} = c$ na Equação 2.4.14.

Empregando-se as mesmas restrições que foram adotadas no cálculo em paralelo das correções ΔX para minimizar o conflito de dados causado pelo método iterativo Gauss-Seidel, quais sejam: (a) estimativa inicial $\Delta\hat{x}^0 = 0$ e (b) um número máximo de iterações igual a um, o método iterativo de Jacobi representado na Equação 3.4.2 pode ser simplificado para

$$\Delta\widehat{x}_{rs}^{t,c+1} = -\frac{F_{rs}\Big|_{\hat{x}^{t,c}}}{\left(\dfrac{\partial F_{rs}}{\partial \hat{x}_{mn}}\right)\Big|_{\substack{t,c\\ m=r\\ n=s}}} \tag{3.4.3}$$

Uma vantagem dessa modificação na técnica de restauração original é que o método de Jacobi soluciona o problema de conflito de dados que ocorre quando Gauss-Seidel é empregado na solução do sistema linear para o cálculo das novas correções $\Delta\hat{x}$ em paralelo.

Além disso, essa simplificação no método torna a implementação do programa de restauração mais simples e, consequentemente, o processamento mais rápido ao eliminar do cálculo das correções ΔX um laço de repetição (*loop*) e necessidade de espaço em memória para armazenar as partições F_{rs} e F_{mn}, que agora poderão ser calculadas pontualmente e armazenadas em uma variável comum. Na (Tabela 3.1) são apresentados os tempos computacionais para duas versões seriais do programa de restauração. A primeira utiliza o método de Gauss-Seidel e a segunda utiliza o método de Jacobi Simplificado.

O emprego do método de Jacobi simplificado no cálculo das correções $\Delta\hat{x}$ em paralelo modifica os passos (5) a (11), descritos anteriormente. Conforme ilustrado nas Figuras 3.27, 3.28 e 3.29, com o emprego do método de Jacobi simplificado, esses passos são redefinidos e passam a ser descritos da seguinte forma:

(5-9) Com a eliminação do laço de repetição no Jacobi simplificado, os passos (5) a (9) se juntam em um só: passo (5-9). Uma vez que as interdependências entre os pontos não existem no método de Jacobi simplificado, todas as unidades de processamento podem efetuar os cálculos das correções dentro da área-A em sua partição ΔX, de modo independente dos demais processadores (Figura 3.27). Além disso, como o número máximo de iterações do método de Jacobi

dimB	TS - Tempos Seriais (seg.)		Redução
	Empregando Gauss-Seidel	Empregando Jacobi Simplificado	
1	1,085	0,971	10,5%
3	1,758	1,292	26,5%
5	2,665	1,676	37,1%
9	5,968	2,939	50,7%
13	11,134	4,711	57,7%
21	24,836	10,038	59,6%

Redução=$\frac{|TS_{GS}-TS_J|}{TS_{GS}} \times 100$

Tabela 3.1: Tempos de execução obtidos com duas versões seriais do programa de restauração: uma empregando o método de Gauss-Seidel e outra o método de Jacobi simplificado, obtidos na restauração de uma imagem de 256×256 pixels, empregando um total de 100 iterações e variando a dimensão do operador B: $dimB = 2N + 1 \in \{1; 3; 5; 9; 13; 21\}$.

simplificado é igual a um, as unidades de processamento não precisam mais se comunicar para trocar dados da partição ΔX com os processadores vizinhos.

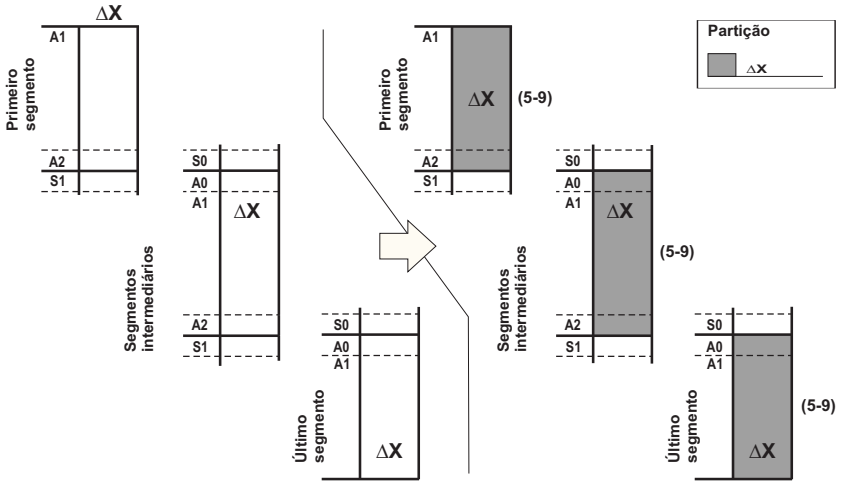


Figura 3.27: Processo de cálculo em paralelo das correções $\Delta\hat{x}$ empregando o método de Jacobi simplificado.

(10) Tendo obtido as correções ΔX no passo (5-9), os processadores podem se-

guir adiante com o seu processamento, de modo independente, calculando as novas estimativas X^{t+1} $(t = 0, 1, ..., ni)$ dentro da área-A, restando apenas o cálculo desse termo nas áreas de sobreposição: S0 e S1 (Figura 3.28).

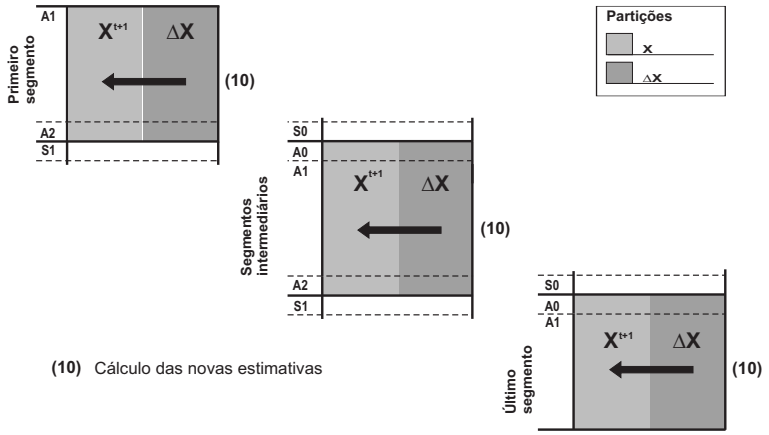


Figura 3.28: Modificação no processo de cálculo em paralelo das novas estimativas X_{t+1} na segunda versão paralela quando o método de Jacobi simplificado é empregado.

(11) As áreas S0 e S1, que faltam no passo (10), são obtidas através da troca de dados entre as unidades de processamento vizinhas (Figura 3.29),

   (a) o envio dos dados pode ser executado de forma assíncrona, mas

   (b) os recebimentos precisam ser feitos de modo síncrono.

Nas Figuras 3.30 e 3.31 são apresentados os gráficos dos tempos de processamento dos mesmos testes feitos anteriormente (Figuras 3.25 e 3.26), porém, substituindo-se aqui os tempos do programa serial empregando Gauss-Seidel [52] por outro, empregando Jacobi simplificado, e incluindo-se os tempos da versão do programa paralelo de restauração da segunda estratégia que emprega, também, o método iterativo de Jacobi simplificado. Pode-se observar que o programa da segunda estratégia, empregando o método iterativo de Jacobi simplificado, apresentou tempos computacionais menores do que aqueles das demais implementações. Apesar do conflito de dados ter sido eliminado do cálculo das novas correções $\Delta\hat{x}$ em paralelo, os gráficos da nova estratégia usando Jacobi simplificado apresentam um comportamento anômalo, por exemplo, quando $np = 3$. Esta anomalia pode ser explicada em razão de problemas de comunicação interprocessadores (*idle-time*).

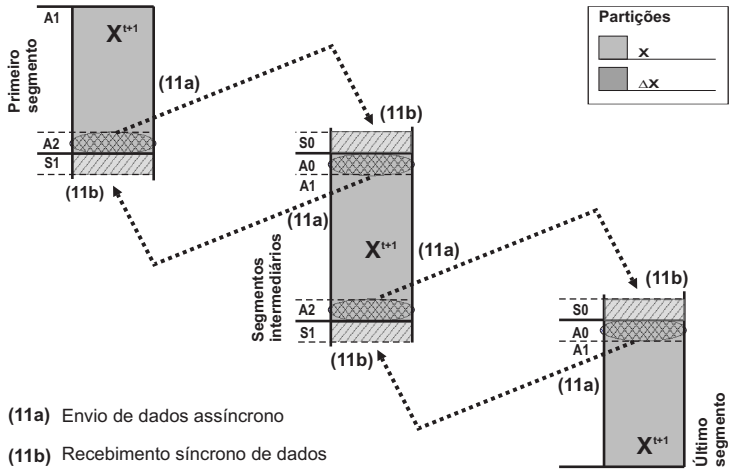


Figura 3.29: Obtenção das áreas S0 e S1 da partição ΔX^{t+1} que ficam faltando do processo de cálculo em paralelo das novas estimativas na nova versão paralela quando o método de Jacobi simplificado é empregado.

Pedido antecipado de recebimento de mensagens (*Message Prefetching*)

Conforme descrito na Seção 3.1.3, as sobrecargas (*overheads*) em um programa paralelo podem se originar de diversas formas. Nem sempre, entretanto, a sua origem é clara e, quando ela é identificada, a sua solução nem sempre é simples e/ou de fácil resolução. Basta observar, por exemplo, todo o esforço que já foi feito até aqui para se otimizar e refinar a segunda estratégia em paralelo (estratégias para o cálculo dos termos em paralelo, adoção do método iterativo de Jacobi no lugar de Gauss-Seidel, etc.), assim como a adoção de outras soluções que foram implementadas na primeira estratégia paralela (técnicas de particionamento, distribuição de cargas, etc.).

Para um melhor entendimento dos processos de comunicação do programa paralelo da segunda estratégia paralela empregando MPI é preciso fazer uso de ferramentas que permitam a visualização do que realmente ocorre dentro de um ambiente de execução em paralelo. Se por um lado o MPI fornece uma base para a construção de programas paralelos, por outro lado, a biblioteca MPE (*Multi-Processing Environment*) [9] fornece uma série de recursos que auxiliam o programador no ambiente MPI, que incluem: a) um conjunto de sub-rotinas para a criação de arquivos de *logs* (arquivos contendo registros das operações de processamento em um computador); b) rotinas para depuração de programas; c) rotinas

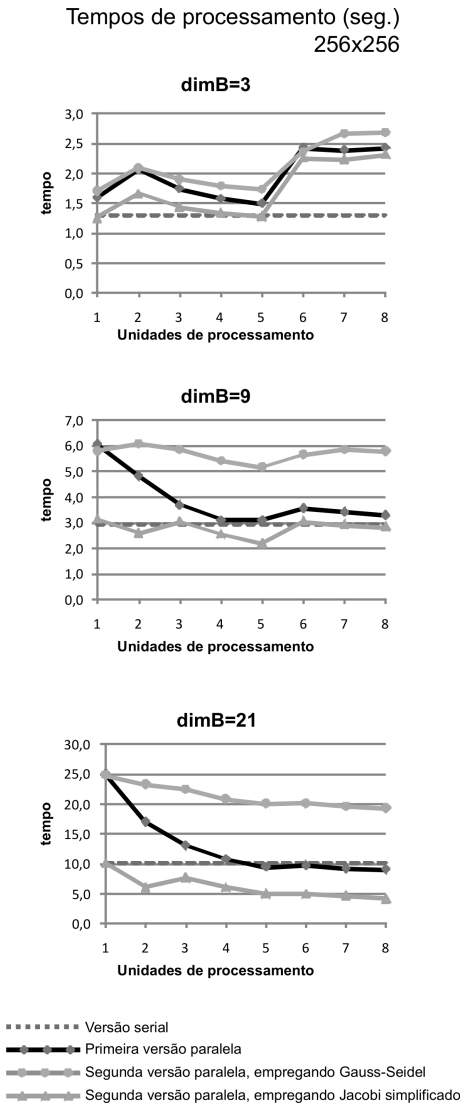


Figura 3.30: Tempos de processamento gastos na restauração de uma imagem de dimensão 256×256 pixels, usando os programas: serial [52] e versões paralelas anterior (Seção 3.3) e a segunda estratégia empregando Jacobi Simplificado, para um total de 100 iterações e $dimB = 2N + 1 \in \{3; 9; 21\}$.

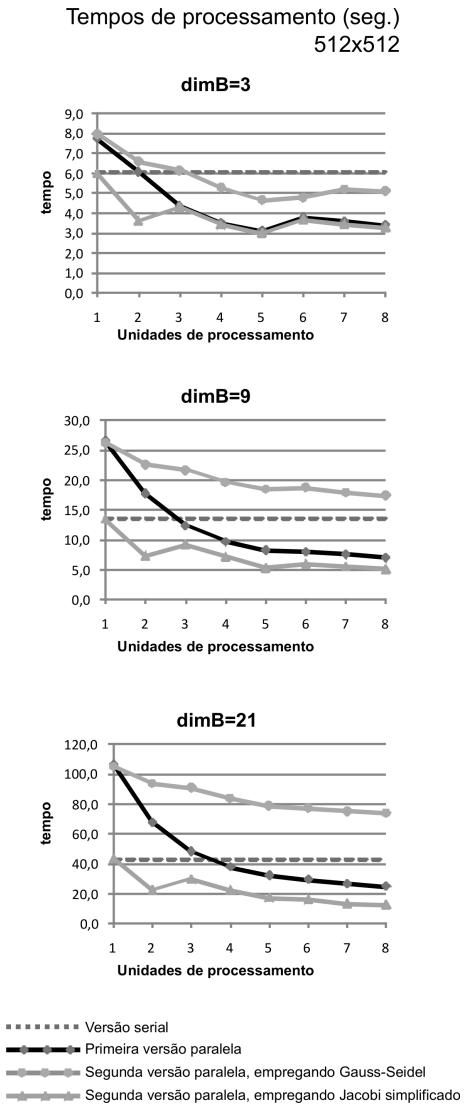


Figura 3.31: Tempos de processamento gastos na restauração de uma imagem de dimensão 512×512 pixels, usando os programas: serial [52] e versões paralelas anterior (Seção 3.3) e a segunda estratégia empregando Jacobi Simplificado, para um total de 100 iterações e $dimB = 2N + 1 \in \{3; 9; 21\}$.

de sequencialização, etc. O MPE oferece diversas maneiras de se gerar arquivos de *log* que descrevem o andamento do processo de execução dos programas paralelos. Esses arquivos podem ser visualizados e examinados por um programa gráfico, o Jumpshot-4 [8], que é distribuído junto com o MPE. Os arquivos de *logs* são gerados em um formato conhecido como CLOG2 e a facilidade com que eles são criados permite que os programas MPI não precisem ser modificados, bastando apenas (re)compilá-los com a biblioteca MPE. Essa biblioteca contém funções com perfis semelhantes àquelas da biblioteca MPI, que interceptam as chamadas MPI da aplicação, gerando os arquivos de *log* de uma forma bem simples e transparente.

Na Figura 3.32 é mostrada a visualização de um *log* de execução do programa paralelo de restauração através do programa Jumpshot-4. No lado esquerdo da figura, tem-se a janela de legendas com uma relação de nomes, representação gráfica dos objetos e outros dados tais como, por exemplo, mensagens, chamadas/esperas de funções MPI e início/término do processamento paralelo, que são visualizados na janela de linha do tempo (lado direito da figura). No centro dessa segunda janela, tem-se a linha de tempo da execução em paralelo da segunda versão paralela do programa de restauração usando Jacobi, empregando-se três unidades de processamento para a restauração de uma imagem de 256×256 pixels - identificados pelos números de 0, 1 e 2, conforme se pode observar no lado esquerdo das linhas de tempo. Abaixo fica o eixo de coordenada-tempo em segundos.

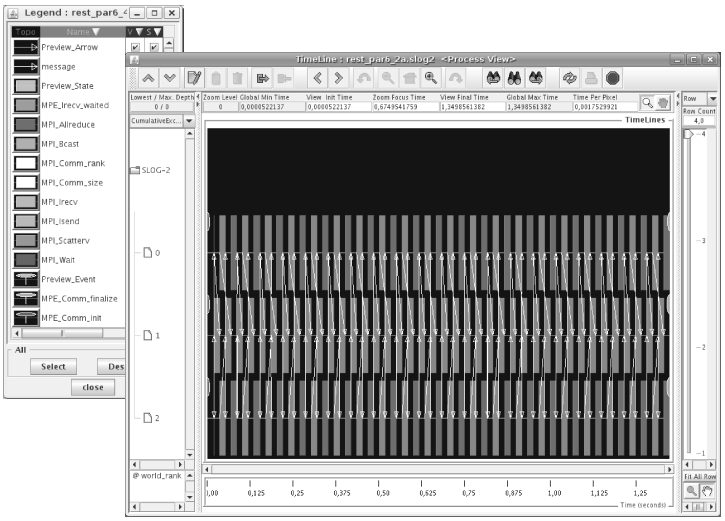


Figura 3.32: Janelas do programa Jumpshot-4: legendas (esquerda) e a de linha de tempo (direita) da execução da segunda versão paralela do programa de restauração de imagens empregando Jacobi, usando três unidades de processamento.

Na Figura 3.33 é mostrado em detalhes (zoom) o processo de comunicação que ocorre, por exemplo, na primeira iteração do laço principal do programa de restauração em paralelo ilustrado na Figura 3.32. Cada um dos pequenos blocos representa a quantidade de tempo consumida por uma chamada de comunicação interprocessadores e os espaços nas linhas de tempo entre esses blocos representam os tempos de execução de instruções do programa de restauração em paralelo que funcionam de forma independente dos outros processadores.

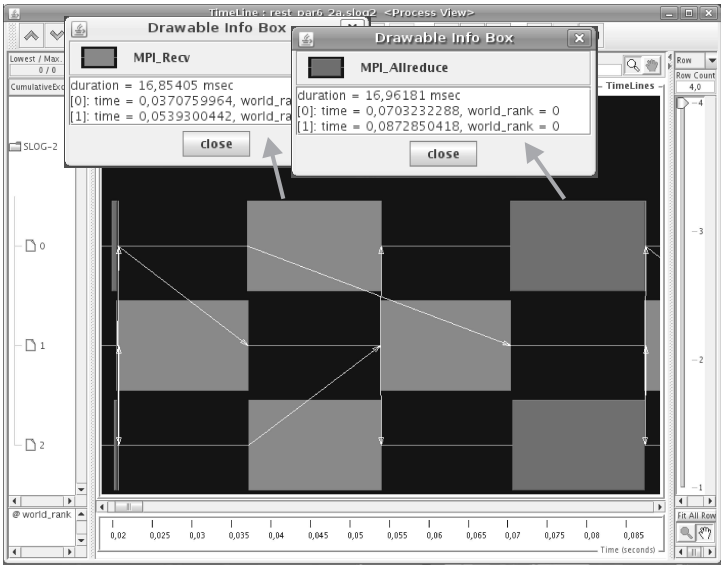


Figura 3.33: Zoom da linha de tempo, mostrando a execução em paralelo da primeira iteração do laço principal da segunda versão paralela do programa de restauração de imagens empregando Jacobi. Os blocos em destaque representam, respectivamente, os tempos de comunicação gastos com as funções *MPI_Recv()* e *MPI_Allreduce()*.

Quanto maiores forem esses blocos, maiores serão os tempos de espera (*idle-time*) por comunicação e, consequentemente, maiores serão os tempos de execução do programa. Por outro lado, quanto menores forem os blocos, mais rápida será a execução do programa em paralelo.

Para se ter uma idéia de proporção, as duas janelas menores na Figura 3.33 (*Drawable Info Box*) mostram em detalhes os tempos gastos (*duration*) pelo processador-0 na execução das funções *MPI_Recv()* (16,85405ms) e *MPI_Allreduce()* (16,96181ms) - a primeira função é responsável pelo recebimento (*receive*) de um bloco de dados e a segunda pelo cálculo do mínimo do funcional Q, Equação 2.2.4, distribuído a todas as unidades de processamento e que é usado como critério para escolha da

imagem recém-processada como solução do problema de restauração.

Analisando-se detalhadamente as Figuras 3.32 e 3.33, observa-se, pelos tamanhos dos blocos de comunicação, que um tempo muito grande (**latência**) ocorre do início ao fim de cada bloco. Isso eleva os tempos de espera (*idle-time*) que consomem uma boa parcela do tempo de execução do programa em paralelo, diminuindo o seu desempenho. Ke *et al.* [28] investigaram esse problema e, como solução, propuseram uma técnica em que a latência de comunicação poderia ser reduzida através do envio antecipado de pedidos de recebimento de mensagens (*message prefetching*) ao remetente (*sender*) antes mesmo deles terem sido efetivamente enviados (*send*).

A biblioteca MPI fornece um bom conjunto de operações para comunicação ponto a ponto, comunicações coletivas e operações sincronizadas. A operação básica de recebimento de dados (*receive*) do MPI é feita através da função *MPI_Recv()* e de envio através da função *MPI_Send()*. A chamada da função *MPI_Recv()* é um processo bloqueante. Ou seja, isso significa que, quando uma função bloqueante é chamada, ela só retorna o controle da execução do programa ao local de chamada quando toda a mensagem tiver sido recebida. Para contornar essa situação, o MPI incluiu em sua biblioteca uma outra função de recebimento não bloqueante, o *MPI_Irecv()*, que retorna imediatamente logo após a sua chamada, tendo ou não completado o recebimento da mensagem. Quando a mensagem (os dados) for realmente necessária, faz-se então uma chamada da função *MPI_Wait()* (bloqueante) que irá aguardar até que a mensagem enviada tenha sido totalmente recebida. Isso permite que se inclua alguma programação entre as chamadas *MPI_Irecv()* e *MPI_Wait()* de modo que algumas instruções possam ser executadas em paralelo, enquanto a mensagem não chega, substituindo, desse modo, tempos de espera por comunicação (*idle-time*) por tempos de processamento, eliminando parte da latência.

Na Figura 3.34, é ilustrada uma comparação entre o método convencional de envio/recebimento e a técnica de envio antecipado de pedido de recebimento de mensagens (*message prefetching*). O padrão vertical representa a fase de computação normal, a horizontal a fase de comunicação mais o tempo de espera (*idle-time* - parte escura) e a diagonal a sobreposição das fases de comunicação e computação. Enquanto uma mensagem bloqueante é feita (*MPI_Recv()* ou *MPI_Wait()*) no método tradicional (normal), a implementação usando *message prefetching* antecipa o processo de recebimento fazendo uma chamada não-bloqueante à função *MPI_Irecv()*, que reserva a área de memória necessária para receber os dados que ainda serão enviados pela função *MPI_Send()* e, depois, libera o processador para que este execute outras instruções em paralelo. Quando a função *MPI_Send()* é finalmente feita pelo remetente (*sender*), uma comunicação mais rápida é feita, uma vez que parte do processo de comunicação já havia sido feito antecipadamente no destinatário (*receiver*), quando este fez uma chamada à função *MPI_Irecv()*. As-

sim, conforme ilustrado na Figura 3.34, a técnica tem o potencial para esconder parte da latência da comunicação, melhorarando o desempenho do programa paralelo [28].

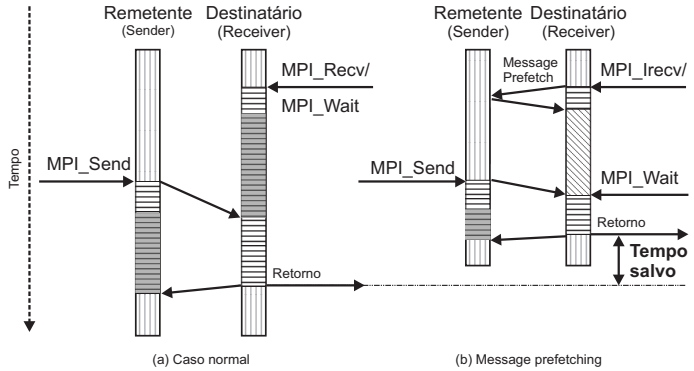


Figura 3.34: Comparação entre (a) uma comunicação paralela normal e outra (b) empregando *message prefetching* [28].

O emprego da técnica *message prefetching* na segunda estratégia paralela modifica os passos (2b) e (10), apresentados na Seção 3.4.2, em razão do emprego do Jacobi simplificado. Com o *message prefetching*, esses passos são agora redefinidos e passam a ser descritos da seguinte maneira:

(2b) Executam-se requisições antecipadas de recebimento (*message prefetching*) das áreas S0 e S1 da partição $Y - BX$ às unidades de processamento vizinhas acima e abaixo, respectivamente. A seguir, calcula-se a área-A da partição $Y - BX$, faltando apenas o cálculo deste termo nas áreas de sobreposição, S0 e S1. Estas áreas serão obtidas através de chamadas à função *MPI_WAIT()* feitas posteriormente no passo (4a), conforme descrito na Seção 3.4.2.

(10) Executam-se requisições antecipadas de recebimento (*message prefetching*) da área S0 e S1 da partição X^{t+1} às unidades de processamento vizinhas acima e abaixo, respectivamente.

Na Figura 3.35, é ilustrado o *log* de execução da nova versão paralela do programa de restauração empregando Jacobi simplificado mais a técnica *message prefetching*, usando um equipamento Dell Precision Workstation T7400, com dois processadores Intel®Xeon®Quad-Core E5405 2 GHz, 2X6 MB L2 cache, 1333 MHz FSB - num total de oito unidades de processamento. Nota-se claramente na figura que houve uma redução significativa no tamanho dos blocos de comunicação. Comparando-se com a Figura 3.32 percebe-se uma redução substancial nos

tempos de comunicação e no tempo de execução do programa paralelo. Sem a aplicação da técnica, a versão do programa paralelo levou em torno de 1,35s para fazer uma restauração de uma imagem de 256×256 pixels, empregando 20 iterações e $dimB = 2N + 1 = 21$. Com a aplicação do *message prefetching*, o mesmo programa gastou aproximadamente 0,70s para fazer a mesma restauração. Isso representa um ganho de velocidade de processamento da ordem de 48,2% com o uso do *message prefetching* (Figura 3.36).

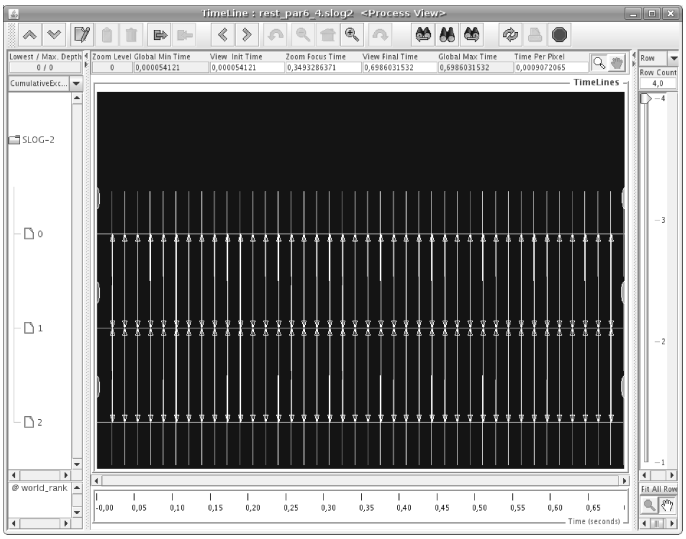


Figura 3.35: Linha de tempo da execução da segunda versão paralela do programa paralelo de restauração de imagens, usando Jacobi e *message prefetching*.

Na Figura 3.37, é mostrado o detalhamento do processo de comunicação que ocorre na primeira iteração do laço principal da segunda versão paralela do programa de restauração em paralelo ilustrado na Figura 3.35. Comparando-se com a Figura 3.33, pode-se notar que, com a aplicação da técnica *message prefetching*, os blocos de comunicação são menores e, portanto, mais rápidos. Uma chamada à função *MPI_WAIT()*, por exemplo, levou 0,52595ms, enquanto que à *MPI_Allreduce()* levou 0,58603ms. Outra chamada à função *MPI_Irecv()* foi tão rápida (5,9046μs) que quase não se nota a presença da mesma no gráfico.

Na Figuras 3.38 e 3.39, são apresentados alguns gráficos dos tempos de processamento dos testes feitos anteriormente (Figuras 3.30 e 3.31), incluindo-se aqui os tempos da segunda versão paralela do programa de restauração empregando Jacobi mais a técnica *message prefetching*.

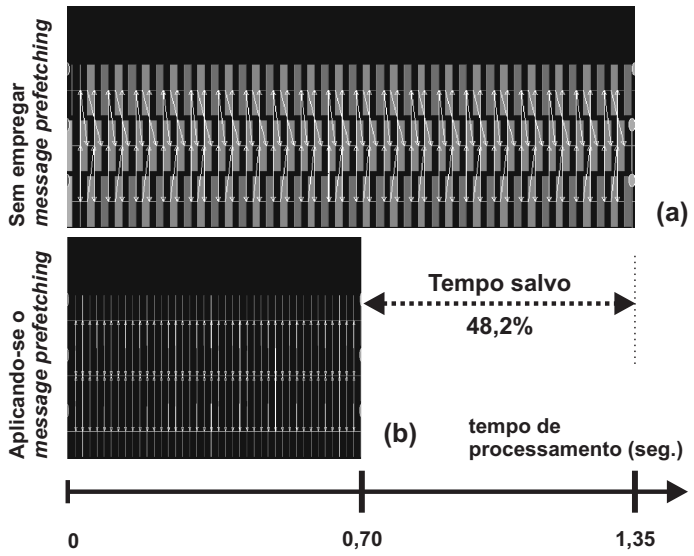


Figura 3.36: Comparação entre os tempos de execução da segunda versão do programa paralelo de restauração sem aplicar a técnica *message prefetching* (a) e com a aplicação dela (b).

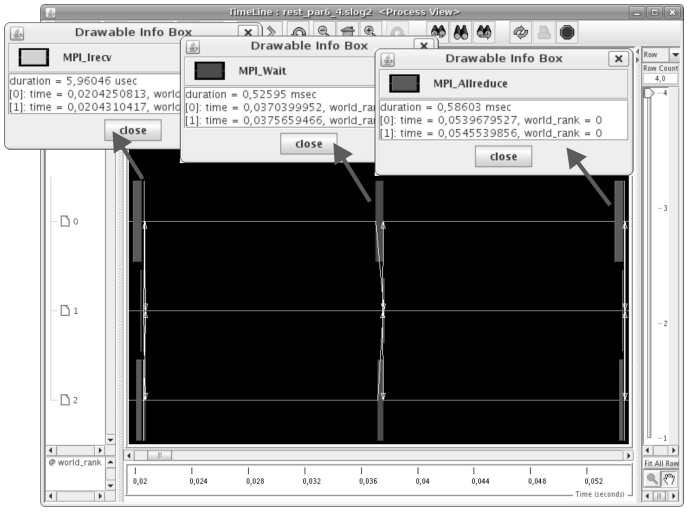


Figura 3.37: Zoom da linha de tempo, mostrando a execução em paralelo da primeira iteração do laço principal da nova versão paralela do programa de restauração de imagens usando Jacobi mais a técnica *message prefetching*.

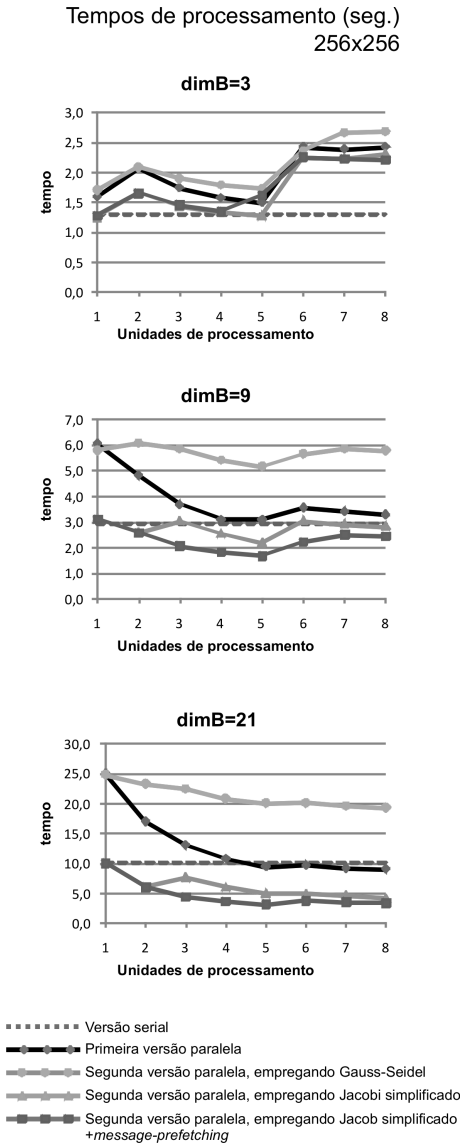


Figura 3.38: Tempos de processamento gastos na restauração de uma imagem de dimensão 256×256 pixels, usando os programas: serial, primeira versão paralela e segunda versão usando Gauss-Seidel, Jacobi e Jacobi+*message prefetching*, para um total de 100 iterações e $dimB = 2N + 1 \in \{3; 9; 21\}$.

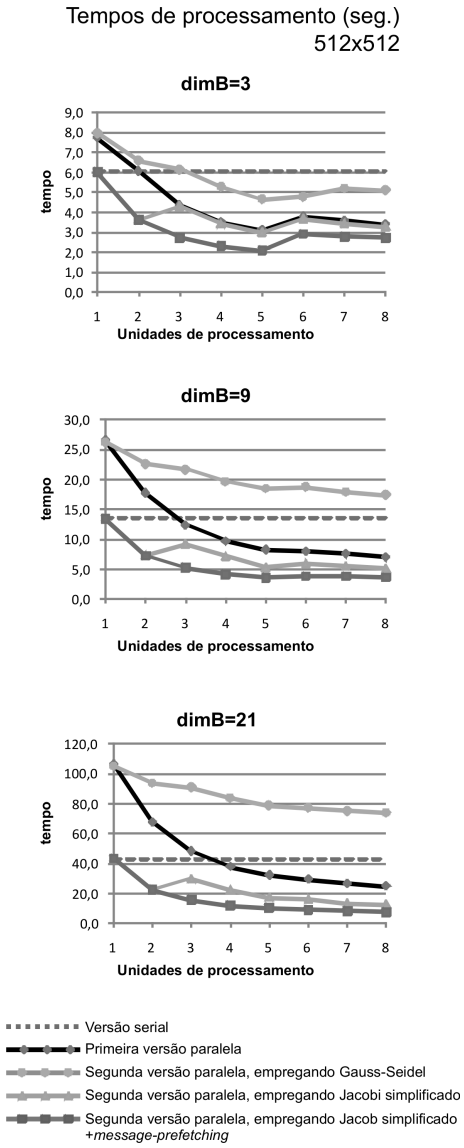


Figura 3.39: Tempos de processamento gastos na restauração de uma imagem de dimensão 512×512 pixels, usando os programas: serial, primeira versão paralela e segunda versão usando Gauss-Seidel, Jacobi e Jacobi+*message prefetching*, para um total de 100 iterações e $dimB = 2N + 1 \in \{3; 9; 21\}$.

Capítulo 4

Métricas de Avaliação

4.1 Medidas de desempenho computacional

4.1.1 Tempo de execução

Em termos gerais, o tempo de execução serial (T_s) é compreendido como o tempo decorrido do início até o término da execução de um programa em um computador sequencial. Porém, dada a existência de arquiteturas paralelas, a definição de tempo serial pode ser estendida como sendo o tempo obtido em um computador onde cada instrução de um programa precisa ser concluída primeiro, antes da próxima instrução ser iniciada, sequencialmente, usando apenas uma unidade de processamento.

Deste modo, o tempo de execução de um programa em paralelo (T_p) é compreendido como sendo o tempo decorrido do início em que o programa é carregado até o momento em que o último processador termina a sua execução.

Apesar de ser uma medida extremamente simples, o tempo de execução nem sempre é a métrica mais conveniente para se avaliar o desempenho de programas paralelos. Como os tempos de execução podem variar de acordo com diversas condições (o tamanho do problema a ser resolvido, volume de dados, velocidade do processador, velocidade de barramento, etc.), eles precisam ser normalizados, de modo que as diferentes dimensões do problema possam ser comparadas e avaliadas com relação a sua eficiência [18].

Portanto, quando se discute a análise de desempenho computacional de programas paralelos, não nos preocupamos apenas com o tempo de execução, frequentemente estamos mais interessamos em saber o quanto estamos ganhando com a paralelização de uma dada aplicação em relação à sua implementação sequencial e, para isso, faz-se necessário o emprego de outras medidas que determinem e

quantifiquem os seus rendimentos. Dentre essas medidas, as mais comuns são: a) aceleração, b) custo e c) eficiência.

4.1.2 Aceleração (*Speed-up*)

A aceleração é uma medida que expressa a melhoria de desempenho de um programa quando é feita alguma modificação no hardware/software ou quando se aplica alguma técnica computacional diferente e/ou otimizada em sua implementação. Essa medida faz referência a quanto um algoritmo paralelo é mais rápido que o seu correspondente serial.

Sendo assim, define-se a aceleração como sendo a relação dos tempos obtidos por um mesmo programa antes e depois de alguma modificação, realizando o mesmo trabalho computacional sobre um mesmo conjunto de dados. Quando essa melhoria é dada somente em função do aumento do número de unidades de processamento, normalmente a aceleração é calculada da seguinte forma (1º caso):

$$S_p = \frac{T_p(V_d, 1)}{T_p(V_d, np)} \tag{4.1.1}$$

onde S_P é a aceleração (*speedup*), $T_p(V_d, 1)$ e $T_p(V_d, np)$ são os tempos de processamento de um programa em paralelo, realizando um trabalho computacional sobre um mesmo volume de dados V_d, empregando, respectivamente, 1 e np processadores.

O valor obtido com a Equação 4.1.1 é também conhecido como **aceleração relativa** (*relative speedup*), pois ele é definido em função do tempo de execução do programa paralelo usando apenas um processador. Esse valor é útil quando se está analisando a escalabilidade de um algoritmo em paralelo. Entretanto, a aceleração relativa nem sempre é muito bem aceita na literatura, visto que os seus valores podem ser disfarçados, mascarando um "aumento" da aceleração [47]. Tome-se o caso em que um algoritmo paralelo seja otimizado para um número específico de processadores: oito, por exemplo. É óbvio que o desempenho do programa usando oito processadores sempre apresentará um desempenho melhor do que quando se emprega um processador apenas. Assim, de modo a evitar uma indicação inadequada de aceleração, o cálculo da Equação 4.1.1 é modificado para (2º caso):

$$S_p^{'} = \frac{T_s(V_d)}{T_p(V_d, np)} \tag{4.1.2}$$

onde $S_p^{'}$ é a **aceleração absoluta** (*absolute speedup*), calculada usando o tempo de processamento $T_s(V_d)$ obtido com um programa serial otimizado e que empregue o melhor algoritmo serial, e $T_p(V_d, np)$ o tempo alcançado com o programa paralelo, usando np processadores. Em muitos casos, esse "melhor" algoritmo deverá ser o algoritmo serial mais rápido.

4.1.3 Custo

O custo de um programa serial (C_s) é basicamente o tempo de execução que se gasta para resolver um dado problema, usando apenas um único processador e empregando o algoritmo serial mais rápido conhecido. Para programas paralelos, o custo (*work ou processor-time product*) é definido como sendo o produto do tempo de execução em paralelo e o número de processadores.

$$C_p = T_p \times np \tag{4.1.3}$$

onde C_p é o custo de um sistema em paralelo, T_p o tempo de execução do programa em paralelo (conforme definido na Seção 4.1.1) e np o número de processadores.

4.1.4 Eficiência

A eficiência representa uma fração do tempo de execução que os processadores gastam para realizar o seu trabalho e fornece uma medida conveniente para se avaliar a qualidade em termos de rendimento dos programas em paralelo, independente do tamanho do problema abordado. O cálculo da eficiência de um programa paralelo pode ser feito, empregando-se as seguintes equações:

(a) para o primeiro caso, Equação 4.1.1, ou **eficiência relativa**:

$$E_p^{'} = \frac{S_p}{np} \tag{4.1.4}$$

(b) para o segundo caso, Equação 4.1.2, ou **eficiência absoluta**:

$$E_p^{'} = \frac{S_p^{'}}{np} \tag{4.1.5}$$

A eficiência (relativa) de um sistema paralelo também pode ser expressa também em função da razão:

$$E_p^{'} = O\left(\frac{C_s}{C_p}\right) \tag{4.1.6}$$

onde $O(\cdot)$ é a a notação de análise assintótica, também conhecida como notação "grande-O" (*big-O notation*).

Se o custo de um algoritmo é $O(C_s)$, isso significa que o algoritmo pode ser convertido em sistema serial com tempo de execução igual a $O(C_s)$. Caso o algoritmo paralelo leve o mesmo tempo ($O(C_s)$) para resolver o mesmo problema em paralelo, ele é dito ser custo-ótimo (*cost-optimal*), uma vez que a sua eficiência é igual a 1 ou $O(1)$ [31].

4.2 Medidas de avaliação de resultados e da qualidade das imagens

Em processamento de imagens (restauração, compressão, realce, etc.), a existência de métricas que permitam a atribuição de um valor que expresse a qualidade de uma imagem é de grande importância. Essas medidas são, de modo geral, usadas tanto para determinar a extensão ou a grandeza da diferença entre duas imagens quanto para se avaliar o desempenho, a eficácia e/ou a validação de um algoritmo computacional. Por exemplo, uma imagem distorcida e uma imagem restaurada por um processo de restauração qualquer podem ser comparadas com a imagem original para determinar a eficácia do método empregado, da mesma forma que os resultados produzidos pelo processo de restauração podem ser utilizados e comparados com os de outros métodos, de modo a se determinar qual deles apresenta o melhor resultado. Geralmente, o método mais empregado e discutido na literatura tem sido o erro médio quadrático [61].

4.2.1 Erro Quadrático Médio - MSE (*Mean Square Error*)

O MSE consiste num valor simples, resultante do somatório dos quadrados das diferenças pontuais das intensidades em escala de cinza dos pontos (pixels) de cada imagem, dado pela seguinte equação:

$$MSE = \frac{\sum_{i=1}^{m}\sum_{j=1}^{n}[f(i,j) - \tilde{f}(i,j)]^2}{m \times n} \tag{4.2.7}$$

onde $f(i,j)$ representa a intensidade do ponto (pixel) de coordenada (i,j) da imagem original/referência de dimensão $m \times n$ e $\tilde{f}(i,j)$ a intensidade do ponto de mesma coordenada em uma outra imagem que se está comparando.

Além do MSE, outras medidas similares podem ou costumam ser empregadas na análise de imagens [4], tais como:

(a) Erro Médio Normalizado - NME (*Normalized Mean Error*)

$$NME = \frac{\left(\sum_{i=1}^{m}\sum_{j=1}^{n}|f(i,j) - \tilde{f}(i,j)|\right)}{\left(\sum_{i=1}^{m}\sum_{j=1}^{n}|f(i,j)|\right)} \tag{4.2.8}$$

(b) Raiz do Erro Quadrático Médio Normalizado - NRMSE (*Normalized Root Mean Square Error*)

$$NRMSE = \frac{\left(\sum_{i=1}^{m} \sum_{j=1}^{n} [f(i,j) - \tilde{f}(i,j)]^2 \right)}{\sqrt{\left(\sum_{i=1}^{m} \sum_{j=1}^{n} [f(i,j) - \bar{f}]^2 \right)}} \tag{4.2.9}$$

onde $\bar{f}$ é a média aritmética das intensidades da imagem original $f(i,j)$.

Uma ampla relação e uma classificação das medidas de qualidade que aparecem na literatura podem ser encontras em [15] e em [41]. No trabalho de dissertação de Romualdo [45], alguns indicadores de desempenho para o algoritmo de restauração de imagens foram usados e avaliados por um grupo de 50 pessoas com o objetivo de investigar e determinar qual deles se aproximava mais das avaliações feitas pelos avaliadores.

4.2.2 Erro Quadrático Médio Relativo - %MSE

Nos casos em que a imagem original $f(i,j)$ é conhecida, pode-se calcular também o erro médio quadrático relativo (%MSE) das restaurações, de modo a obter um valor normalizado em relação ao MSE da imagem modificada $\tilde{f}(i,j)$ [11],

$$\%MSE = \frac{MSE_{imagem\ restaurada}}{MSE_{imagem\ modificada}} \tag{4.2.10}$$

onde $MSE_{imagem\ modificada}$ é o valor do MSE da imagem original modificada artificialmente, por exemplo, pelo processo descrito na Seção 2.7, ou por qualquer outra técnica/ferramenta de processamento digital de imagem, e $MSE_{imagem\ restaurada}$ é o valor do MSE dessa mesma imagem modificada após ser restaurada pelo processo de restauração descrito na Seção 2.4.

Essa informação é extremamente útil, pois ela nos dá uma medida do ganho da restauração com base no MSE. Quanto menor for o valor do %MSE, melhor esse ganho.

4.2.3 Erro Médio Absoluto - MAE (*Mean Absolute Error*)

O MAE é o valor resultante do somatório das diferenças absolutas pontuais das intensidades em escala de cinza de cada ponto (pixel) de duas imagens (p.ex. original e distorcida) dividido pelo total de pontos de cada imagem, dado pela seguinte equação:

$$MAE = \frac{\sum_{i=1}^{m} \sum_{j=1}^{n} |f(i,j) - \tilde{f}(i,j)|}{m \times n} \tag{4.2.11}$$

onde $f(i,j)$ representa a intensidade do ponto (pixel) de coordenada (i,j) da imagem original (ou uma imagem de referência) de dimensão $m \times n$ e $\tilde{f}(i,j)$ a intensidade do ponto de mesma coordenada em uma outra imagem a qual se quer comparar (p.ex. uma imagem resultante de um processo de restauração).

Quanto menor essa medida, mais a imagem que se está comparando ($\tilde{f}$) se aproxima da original f.

4.2.4 Relação Sinal-Ruído - SNR (*Signal to Noise Ratio*)

$$SNR = 10\ log_{10}\left(\frac{\sum_{i=1}^{m}\sum_{j=1}^{n}[f(i,j)]^2}{\sum_{i=1}^{m}\sum_{j=1}^{n}[f(i,j)-\tilde{f}(i,j)]^2}\right) \tag{4.2.12}$$

onde $f(i,j)$ representa a intensidade do ponto (pixel) de coordenada (i,j) da imagem original/referência de dimensão $m \times n$, $\tilde{f}(i,j)$ a intensidade do ponto de mesma coordenada em uma outra imagem a qual se quer comparar.

4.2.5 Relação Sinal-Ruído de Pico - PSNR (*Peak Signal to Noise Ratio*)

$$PSNR = 10\ log_{10}\frac{L_{max}^2}{MSE} \tag{4.2.13}$$

onde L_{max} é o valor máximo de intensidade de cinza (tipicamente, $L_{max} = 255$) e MSE é o erro médio quadrático.

A métrica PSNR é expressa em decibel (dB) e quanto maior for o valor dessa métrica, mais a imagem que se está comparado se aproxima da original.

4.2.6 Relação Sinal-Ruído de Borramento - BSNR (*Blurred Signal-to-Noise Ratio*)

$$BSNR = 10\ log_{10}\left(\frac{\sigma_{BX}^2}{\sigma_\eta^2}\right) \tag{4.2.14}$$

onde σ_η^2 é a variância do ruído aditivo e σ_{BX}^2 é a variância do borramento, sem o ruído, dado por [3]

$$\sigma_{BX}^2 = \frac{1}{(M-1)^2}\sum_{i,j=0}^{M}[g(i,j) - E\{g\}]^2$$

onde $g(i,j) = y(i,j) - \eta(i,j)$ na Equação 2.1.1 e $E\{g\}$ representa o valor esperado, ou a média, de g.

4.2.7 Erro Quadrático Médio Ponderado pela Informação - IWMSE (*Information Weighted Mean Square Error*)

Métodos estatísticos × Métodos heurísticos

Normalmente, os métodos convencionais para avaliação da qualidade de imagens têm como base critérios objetivos obtidos essencialmente por meio de métricas apoiadas em erros estatísticos, tais com: MSE, NME, NRMSE, entre outros. Entretanto, os resultados obtidos através desses métodos (objetivos) diferem bastante de critérios subjetivos que tenham como base características do sistema visual humano (HVS - *Human Visual System*) [34], por exemplo.

Tompa *et al.* [56] afirmam que o MSE, assim como outros métodos convencionais, geralmente falha ao fornecer uma medida das diferenças que não representam corretamente o nível de qualidade observada entre duas imagens. Eles explicam que isso decorre em muitos casos, principalmente, pelo fato do MSE atribuir um mesmo peso às diferenças em todos os pontos da imagem. Porém, ao se analisar uma imagem, percebe-se que nem todos os pontos são igualmente importantes. Ou seja, nem todos os pontos em uma imagem possuem o mesmo peso para um observador humano. Diferenças perceptíveis em regiões mais importantes da imagem irão produzir um efeito negativo muito maior do que aquelas (diferenças) encontradas em outras áreas como o fundo, por exemplo. Para um observador humano é importante que a medida de avaliação corresponda a um critério subjetivo de comparação e expresse a qualidade visual percebida.

Apesar da complexidade do sistema visual humano, à medida que os modelos visuais vão sendo introduzidos nos métodos de avaliação objetivos, o relacionamento entre os métodos objetivos e subjetivos vão se aperfeiçoando e os resultados de avaliação tendem a melhorar, alcançando uma melhor correlação com a resposta dos observadores humanos [16]. Atualmente, muitos trabalhos em diversas áreas do conhecimento têm dado uma maior ênfase aos métodos que se baseiam no sistema visual humano. Trabalhos em visão computacional e de compressão de imagens, por exemplo, têm aceitado as novas abordagens de avaliação, baseadas na percepção/visão humana, como sendo superiores às abordagens convencionais [13].

Medidas de energia de interesse local

Apesar do sistema visual humano ser extremamente seletivo, verificou-se que as áreas de uma imagem que recebem uma maior atenção (interesse visual) são aquelas que contêm alguma característica específica, tal como, por exemplo, uma cur-

vatura elevada, um alto contraste, linhas e bordas, a ocorrência de algum aspecto inesperado, etc. Assim, tomando-se como base alguma dessas peculiaridades, o nível de interesse visual local pode ser medido. Dentre vários aspectos possíveis (Figura 4.1), alguns foram identificados e apresentados por Topper [57] e Topper e Jernigan [58]: (a) o valor do pixel em uma escala de cinza (intensidade); (b) variância local (contraste); (c) magnitude de borda, obtida com um operador de detecção de bordas qualquer (gradiente, Sobel, Canny, etc.) [21, 41], etc.

Entretanto, essas medidas de energia, por si só, não são capazes de ponderar diretamente as diferenças tal como se tem teorizado a respeito do nível de interesse visual, pois o sistema visual humano seleciona a porção do estímulo visual (ou energia) que é menos esperado ou, em termos estatísticos, aquilo que é menos provável [33].

Figura 4.1: Medidas de energia de interesse local: (a) intensidade (escala de cinza) e (b) magnitude de borda (imagem gradiente obtida pela utilização do operador de detecção de bordas de Sobel).

Mapa de informações

Em alguns trabalhos [56, 58, 62] é mostrada a existência de uma relação direta de proporcionalidade entre a medida de informação, também conhecida como auto-informação (*Shannon's self-information*) [48], e o nível de interesse visual humano com relação às diferentes áreas em uma imagem. Isso foi feito através de experimentos em que se fez o mapeamento das áreas de interesse visual em uma imagem a partir do acompanhamento dos movimentos dos olhos das pessoas quando elas observam essa imagem e, a partir disso, verificou-se que a as áreas de maior fixação eram justamente as áreas da imagem que continham os maiores valores de informação. Assim, ao invés de usar as intensidades de energia da imagem diretamente para se ponderar as diferenças entre as imagens, no seu lugar, usa-se a medida de informação calculada com base em uma das medidas de energia de interesse local.

Primeiramente, escolhe-se uma das medidas de energia local e, com base nela, constrói-se um mapa de energias En. A Figura 4.1 ilustra e exemplifica dois desses mapas: (a) escala de cinza e (b) magnitude de borda.

Escolhendo-se, por exemplo, o valor da intensidade do pixel (escala de cinza) como medida de energia de interesse local, o cálculo da medida de informação baseia-se num esquema probabilístico dado por

$$P(e) = \frac{n_e}{N_t} \tag{4.2.15}$$

onde P_e é a probabilidade de ocorrência de um dado evento e (p.ex. valor de cinza), $n_e = H_n(e)$ e N_t o número total de pontos da imagem. Sendo que H_n, também conhecido como histograma, representa um vetor com L elementos, $H_n(e)$ representa o número de vezes que o valor e aparece no mapa de energias e L o número total de eventos e que ocorrem na medida de energia local escolhida. Tomando-se a escala de cinza como medida de energia local, por exemplo, tem-se que $L = 256$, que corresponde ao número total de níveis de cinza que uma imagem digital monocromática pode ter.

Como o nível de interesse visual aumenta à medida que a probabilidade de ocorrência de um dado evento diminui, a relação entre os diferentes eventos e a informação contida em cada um deles é dada da seguinte forma:

$$I_{si}(e) = log\frac{1}{P(e)} \tag{4.2.16}$$

onde $I_{si}(e)$ é a medida de informação (ou auto-informação) de um dado evento e.

Esta medida (auto-informação) fornece um valor não-negativo que representa a quantidade de informação contida em um evento e ele pode ser definido não somente em função dos valores em cada pixel (intensidade), mas também em função de qualquer uma das medidas de energia local citadas anteriormente (escala de cinza, magnitude de borda, variância local, etc.).

Assim, através das medidas de informação obtidas pela Equação 4.2.16, constrói-se um mapeamento da imagem, substituindo-se cada um dos valores associados ao evento e pela sua respectiva medida de informação. Esse mapeamento é referido como mapa de informações, uma vez que mostra a localização e a quantidade de informação contida numa imagem, com

$$I_{map}(i,j) = I_{si}(En(i,j)) \tag{4.2.17}$$

onde $I_{map}(i,j)$ é o mapa de informações, En o mapa de energias da imagem que está sendo mapeada, $I_{si}(\cdot)$ é a medida de informação para um dado evento e (i,j) as coordenadas da imagem.

Erro Quadrático Médio da Informação - IMSE (*Information Mean Square Error*)

Tompa *et al.* [56] descrevem um novo método inspirado na percepção humana, o IMSE (*Information Mean Square Error*), em substituição aos métodos convencionais. Essa nova medida tem as diferenças ponderadas com base no mapa de informações de cada imagem, de modo a garantir que os elementos perceptíveis (pontos da imagem com valores maiores de informação e de maior interesse visual) tenham pesos maiores que os demais elementos da imagem.

$$IMSE = \frac{1}{N_t} \sum_{i=1}^{m} \sum_{j=1}^{n} \left[f(i,j) I^{f}_{map}(i,j) - \tilde{f}(i,j) I^{\tilde{f}}_{map}(i,j) \right]^2 \tag{4.2.18}$$

onde $f(i,j)$ representa a intensidade no ponto de coordenada (i,j) de uma imagem (original ou de referência) com dimensão $m \times n$, $\tilde{f}(i,j)$ a intensidade no ponto de mesma coordenada de uma imagem distorcida ou resultante de um processo de restauração, por exemplo, $N_t = m \times n$ o número total de pontos (pixels) de cada uma destas imagens, e $I^{f}_{map}(i,j)$ e $I^{\tilde{f}}_{map}(i,j)$ são, respectivamente, os mapas de informação das imagens original (ou de referência) e distorcida.

A parte mais importante do uso do IMSE consiste em selecionar uma medida de energia apropriada para se avaliar as diferenças visuais perceptíveis entre as duas imagens. Em seu artigo, Tompa *et al.* [56] deixa em aberto qual medida deve ser empregada e afirmam que essa escolha depende da imagem que se está sendo trabalhada e do método de distorção aplicado.

Porém, analisando-se um pouco mais a fórmula de cálculo do IMSE, Equação 4.2.18, percebe-se que existem alguns problemas com relação à forma como o cálculo das diferenças é feita neste método:

(a) Diferenças onde existem semelhanças nas imagens: mesmo que as intensidades dos pontos sejam iguais ($f(i,j) - \tilde{f}(i,j) = 0$) nas duas imagens, eles poderão apresentar "diferenças" consideráveis no IMSE caso exista alguma diferença em seus mapeamentos naquele ponto ($I^{f}_{map}(i,j) - I^{\tilde{f}}_{map}(i,j) \neq 0$). Isso não faz sentido, pois se as intensidades são iguais (perceptíveis ou não), então não existe diferença entre elas. Para esses casos, a diferença deveria ser zero.

(b) Falsa igualdade para imagens monocromáticas diferentes: tomando-se, por exemplo, uma imagem totalmente branca ($W = \{W_{ij} = 255\}$) e a outra totalmente preta ($P = \{P_{ij} = 0\}$), tem-se que as medidas de informação para ambas as cores, branca na primeira imagem e preta na segunda, são iguais a zero ($I^{W}_{si}(255) = I^{P}_{si}(0) = 0$), uma vez que as probabilidades de cada uma

delas em cada uma das imagens é igual a 1. Disso resulta que, para imagens monocromáticas diferentes, o IMSE será igual a zero, indicando que as "imagens são iguais" quando elas na verdade não são.

Erro Quadrático Médio Ponderado pela Informação - IWMSE (*Information Weighted Mean Square Error*)

Para resolver os problemas descritos acima, é proposta uma alteração no cálculo do IMSE, Equação 4.2.18, de forma a criar um novo método de cálculo das diferenças baseado nas medidas de informação, corrigindo as deficiências do IMSE. Isso leva a uma nova medida aqui denominada "Erro Médio Quadrático Ponderado pela Informação" ou simplesmente IWMSE (*Information Weighted Mean Square Error*) [54], dada por

$$IWMSE = \frac{1}{N_t}\sum_{i=1}^{m}\sum_{j=1}^{n}(1 + I_{map}^{f\tilde{f}}(i,j))[f(i,j) - \tilde{f}(i,j)]^2 \tag{4.2.19}$$

onde $I_{map}^{f\tilde{f}}(i,j) = max(I_{map}^{f}(i,j), I_{map}^{\tilde{f}}(i,j))$ é o valor máximo dos mapas de informação das imagens original e distorcida na coordenada (i,j).

Na Equação 4.2.19, verifica-se que os pontos iguais (perceptíveis ou não) nada acrescentam à medida uma vez que a diferença entre eles resulta em zero. Ou seja, as diferenças ocorrem somente onde não existem semelhanças.

O valor do mapeamento aplicado é dado em função do valor máximo dos mapas de informação das imagens original e distorcida de modo a refletir o nível de interesse máximo em cada um dos pontos em ambas as imagens. O peso é então calculado com base neste valor acrescido de 1 de modo a evitar que imagens monocromáticas diferentes resultem em uma falsa igualdade. Para esses casos, o valor calculado do IWMSE será igual ao do MSE visto que para $I_{map}^{f\tilde{f}}(i,j) = 0$ a equação do IWMSE, Equação 4.2.19, se iguala ao MSE, Equação 4.2.7.

Na Figura 4.2 é apresentado o fluxograma do processo de construção do mapa de informações utilizado no cálculo do IWMSE que é implementado da forma descrita a seguir:

(1) Filtram-se as imagens original e distorcida usando um operador ponto exponencial, Equação 4.2.20, de modo a reduzir a sensibilidade do algoritmo a pequenas variações que por acaso existam nas regiões escuras da imagem [56];

$$\breve{f}_{ij} = d(b^{f_{ij}} - 1), \tag{4.2.20}$$

onde b é uma base qualquer (normalmente, utiliza-se $b = 2$), f_{ij} e $\breve{f}_{ij}$ são respectivamente os valores dos pontos de coordenadas (i,j) das imagens original e filtrada

e d é um fator de correção de escala, escolhido de modo que o valor de maior magnitude em $\breve{f}$ seja 255, dado por

$$d = \frac{255}{log(1 + |R|)} \tag{4.2.21}$$

onde R é o valor de maior magnitude em f.

(2) Usando-se uma das medidas de energia local e (escolhida *a priori*), constrõem-se os mapas de energias En^f e $En^{\tilde{f}}$ das imagens filtradas em (1);

(3) A partir desses mapas (2), constrõem-se os seus respectivos histogramas n_e^f e $n_e^{\tilde{f}}$;

(4) Para cada um dos histogramas do passo (3), calculam-se as funções de densidade de probabilidade $P_f(e)$ e $P_{\tilde{f}}(e)$, Equação 4.2.15;

(5) Calculam-se as medidas de informação $I_{si}^f(e)$ e $I_{si}^{\tilde{f}}(e)$ de cada um dos eventos e, Equação 4.2.16 e, finalmente,

(6) Constroem-se os mapas de informações, substituindo-se os valores associados ao evento e na medida de energia (2) pela sua medida de informação (5), para serem usados no cálculo do IWMSE, Equação 4.2.19.

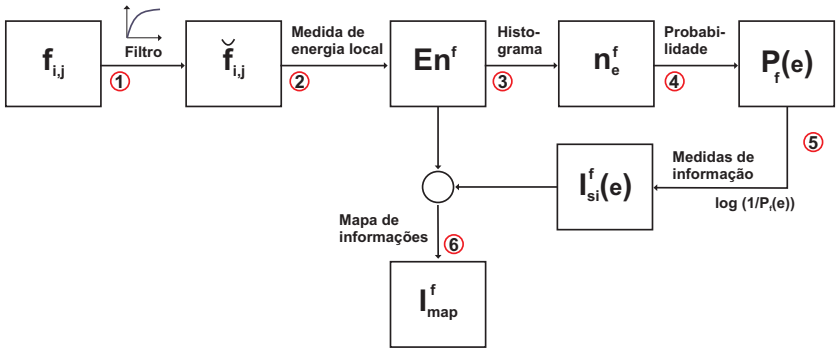


Figura 4.2: Fluxograma do processo de construção do mapa de informações da imagem f_{ij} usado no cálculo do IWMSE.

Na Figura 4.3 é mostrado um exemplo onde as diferenças entre as imagens borrada e restaurada pudessem ser notadamente percebidas e a indicação da qualidade da restauração pelo MSE falha.

Na Tabela 4.1 são apresentados os resultados obtidos com as medidas MSE e IWMSE aplicadas ao exemplo-teste (Figura 4.3). As medidas de energia local usadas nos cálculos do IWMSE foram: a) escala de cinza (/G), b) magnitude de borda (/S) e variância local (/V). Pode-se observar que o MSE das duas imagens com relação à imagem original Figura 4.3(a) são praticamente iguais, nos induzindo erroneamente a achar que ambas imagens (distorciada e restaurada) são qualitativamente equivalentes. O IWMSE, por outro lado, por se basear em critérios subjetivos do sistema visual humano, aponta corretamente para a imagem restaurada (Figura 4.3c) como sendo visualmente melhor do que a imagem borrada (Figura 4.3b).

		IWMSE		
Imagens	**MSE**	**IWMSE/G**	**IWMSE/S**	**IWMSE/V**
Distorcida	773,62	7568,5	820,1	791,5
Restaurada	773,60	6403,9	137,3	622,5

Legenda - Medidas de energias locais: escala de cinza (G=Greyscale), magnitude de borda (S=Sobel) e variância local (V).

Tabela 4.1: Diversas medidas de diferenças das imagens distorcida e restaurada em relação à imagem original.

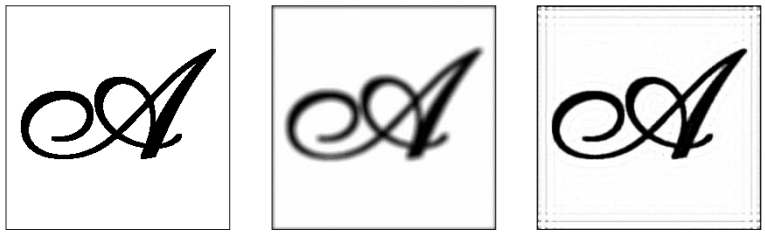


Figura 4.3: (a) Imagem original, (b) distorcida e (c) restaurada.

Capítulo 5

Resultados Computacionais

5.1 Problemas de sobrecarga da primeira estratégia paralela

Apesar da primeira versão paralela do programa de restauração ter alcançado um tempo computacional menor do que aqueles obtidos com o programa serial (Figura 5.1a) e mesmo havendo um aumento na velocidade de processamento (Figura 5.1b), detectou-se uma queda significativa na eficiência computacional dessa primeira versão do programa paralelo. Essa eficiência diminuía à medida que o número de processadores aumentava (Figura 5.1c). Apesar da aceleração (*speedup*) aumentar a cada processador adicionado ao ambiente de execução em paralelo, o ganho relativo decaía de forma considerável (Figura 5.1c) e os custos aumentavam quase que linearmente em uma curva ascendente (Figura 5.1d). Os resultados apresentados na Figura 5.1 foram obtidos empregando-se um cluster de oito microcomputadores com o processador AMD K6II de 450MHz, ligados em uma rede de 100Mbps.

Normalmente, após a inclusão de novas unidades de processamento, dependendo do problema, esse tipo de comportamento é esperado. O simples fato de termos quantidades maiores de processadores e quantidades menores de dados, por si só justificaria em parte essa queda no desempenho. Entretanto, a pequena quantidade de processadores envolvidos, a alta taxa de queda na eficiência e o custo muito alto chamaram a atenção para um outro problema nessa abordagem.

Assim, investigando-se melhor as causas desse problema, verificou-se que, apesar dos blocos de dados distribuídos serem cada vez menores, à medida que a quantidade de processadores aumenta, o tamanho das partições aumenta significativamente e de forma desproporcional. Esse aumento se dá em razão do tamanho das áreas de sobreposição, que permanece fixo apesar do número de processadores

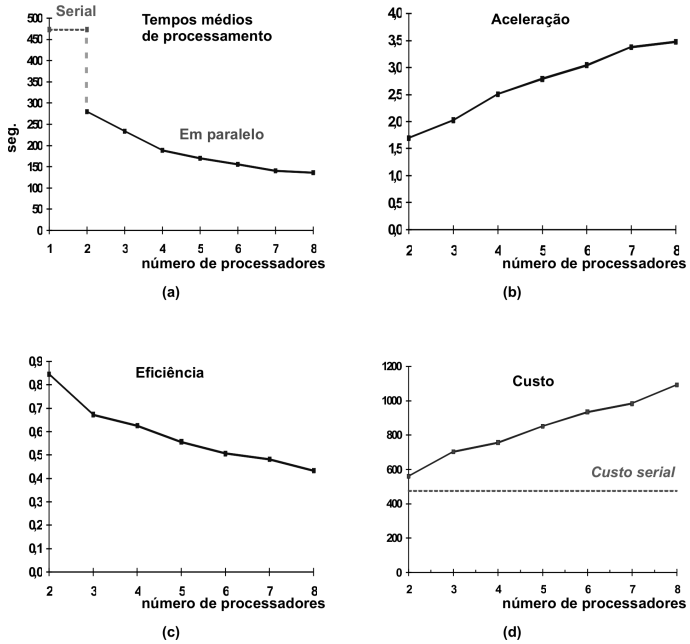


Figura 5.1: Gráficos de desempenho da primeira versão do programa paralelo de restauração no processamento de uma imagem de dimensão 256×256, usando os seguintes parâmetros: $dimB = 21$, área de sobreposição (S0 e S1) igual a 21, maxIter=20, realimentação, $\sigma^2 = 20$, $\alpha = 0,05$, $q = 1$ e $\gamma = 1$.

aumentar.

Conforme ilustrado na Figura 5.2, para a restauração de uma imagem de 256×256 pixels, por exemplo, usando uma área de sobreposição (S0 e S1) igual a 21 e oito processadores, tem-se um aumento na carga total de trabalho de mais de 114% em relação à mesma restauração empregando apenas um processador. Com exceção do primeiro e último processadores que têm uma sobrecarga (*overhead*) de 65,6%, as demais unidades de processamento (intermediárias) têm um aumento de 131,3% no volume de dados em suas partições.

Esse aumento na quantidade de dados a ser processada é significativo e resulta numa sobrecarga de trabalho adicional em cada processador, já que cada um deles processa uma mesma área de imagem (duplo processamento), sobrecarregando o processo de restauração em paralelo como um todo e reduzindo de forma indesejável a eficiência computacional dessa estratégia. Quanto maior o número de

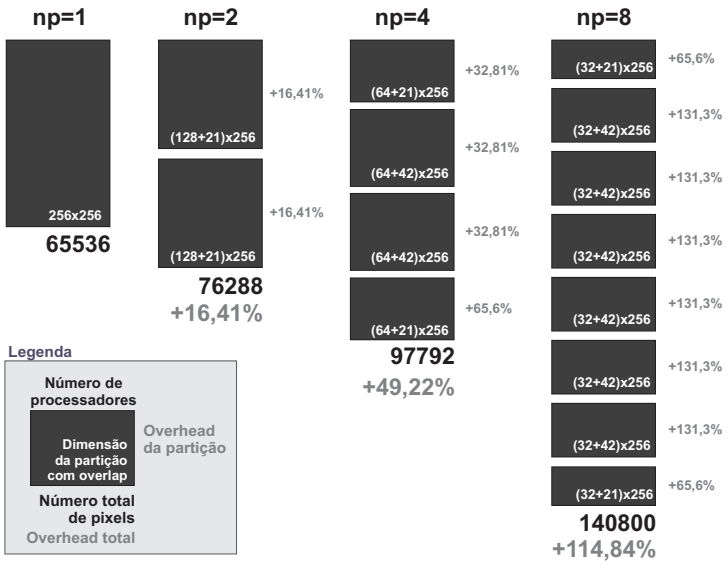


Figura 5.2: Aumento no volume dos dados processados (overhead) que ocorre na primeira versão paralela à medida que se aumenta o número de processadores na restauração de uma imagem de dimensão 256×256, com uma área de sobreposição (S0 e S1) igual a 21.

processadores envolvidos e quanto maior a área de sobreposição (S0 e S1), maiores são as sobrecargas observadas nessa estratégia [52].

5.2 Resultados

Os resultados apresentados nesta Seção foram obtidos com o equipamento descrito a seguir: Dell Precision Workstation T7400, com dois processadores Intel® Xeon® Quad-Core E5405 2 GHz, 2X6 MB L2 cache, 1333 MHz FSB - totalizando oito unidades de processamento -, Memória de 4 GB DDR2 SDRAM 667MHz, placa de vídeo PCIe x16 nVidia Quadro FX570 256MB, 4 discos rígidos de 300 GB SAS (15.000 rpm) em RAID 5, placa de rede 10/100/1000 Gigabit Ethernet Broadcom NetXtreme PCI Express, monitor 22 polegadas UltraSharp® 2208FPW Widescreen, com dual boot: Windows Vista® Business SP1 (instalação original) e o sistema operacional Ubuntu 4.24 (Linux 2.6.24-24 generic), instalado posteriormente. As aplicações serial e em paralelo foram todas elas executadas no ambiente Linux usando os pacotes dos programas OpenMPI 1.2.5 e MPE 2-1.0.6p1.

5.2.1 Metodologia de avaliação do desempenho computacional

Para se avaliar as estratégias de implementação paralelas do programa de restauração e compará-las faz-se necessário estabelecer uma metodologia de avaliação de maneira que os resultados obtidos reflitam de forma fiel e real as medidas de desempenho alcançados com as referidas implementações na arquitetura usada. Para tanto, alguns procedimentos e cuidados foram tomados:

(a) A avaliação do desempenho dos programas em paralelo teve como base de comparação uma versão serial otimizada empregando Jacobi simplificado;

(b) Cada um dos tempos de execução apresentados nos gráficos foi obtido através da média aritmética dos tempos alcançados em três execuções completas dos programas de restauração: (i) serial; (ii) primeira versão paralela e; (iii) segunda versão paralela empregando Jacobi simplificado mais *message prefetching*, em uma mesma arquitetura de hardware, num total de 100 iterações completas, aplicando-se os mesmos parâmetros de restauração;

(c) Na avaliação de desempenho foram empregadas cinco resoluções de imagem diferentes: 128x128 (imagem pequena), 256x256, 512x512, 1024x1024 e 2048x2048 (imagem grande para um AFM). A primeira imagem (baixa resolução) foi usada para se avaliar o comportamento dos programas na restauração para pequenos volumes de dados (p.ex. restauração de partes específicas de uma imagem). A última imagem (alta resolução) foi usada para se analisar o comportamento dos programas na restauração para grandes volumes de dados, uma vez que, à medida que os equipamentos (AFM) têm evoluído, imagens com resoluções maiores tem sido produzidas. As demais resoluções (intermediárias) foram usadas para se avaliar o comportamento dos programas à medida que as dimensões das imagens fossem aumentando;

(d) O esforço computacional exigido para se restaurar essas imagens e o volume de dados comunicado entre os processadores em paralelo foi variado, empregando-se diferentes tamanhos do operador de borramento B: $dimB = 2N + 1 \in \{3; 9; 21\}$. Quanto maior esse valor, maior é o esforço computacional e o volume de dados comunicados entre os processadores;

(e) Por questões de comparabilidade entre os resultados obtidos com as duas versões do programa paralelo de restauração, na execução da primeira versão paralela do programa de restauração empregou-se uma área de sobreposição (S0 e S1) igual a N, cujo valor é obtido de $dimB$, estabelecido *a priori*, onde $N = \frac{dimB-1}{2}$, Equação 2.3.5. Esse valor representa o tamanho mínimo que

deve ser usado para se minimizar o efeito-borda da primeira versão paralela e que é compatível com o tamanho da área de sobreposição (S0 e S1) empregada na segunda versão paralela do programa de restauração.

5.2.2 Análise de desempenho computacional

Na Figura 5.3, são mostrados os gráficos das medidas de desempenho obtidas com três implementações (serial, primeira e segunda versões paralelas) do algoritmo de restauração no processamento de uma imagem com resolução de 128×128 pixels.

Um menor número de processadores observado nos gráficos da Figura 5.3, quando $dimB = 21$, ocorre em razão da quantidade de processadores (np) usada na restauração de uma imagem em paralelo possuir um limite máximo, que varia de acordo com o tamanho da imagem restaurada e a dimensão do operador de borramento B ($dimB$), ambos conhecidos *a priori*. O tamanho de uma partição, descontando-se as áreas sobreposição (S0 e S1), não pode ser inferior a ($dimB$), para que não haja interposição das áreas S0 e S1 e para que exista pelo menos uma linha de imagem na área-A que possa ser restaurada. Ou seja,

$$\frac{M+1}{np} > dimB \Rightarrow 1 \leq np \leq int\left(\frac{M+1}{np}\right) \tag{5.2.1}$$

onde $(M+1) \times (M+1)$ é a dimensão da imagem, np o número de processadores, $dimB = 2N+1$ a dimensão do operador de borramento B e $int(\frac{M+1}{np})$ é o valor resultante da divisão inteira de $M+1$ por $dimB$.

Assim, da Equação 5.2.1 resulta que, para a restauração de imagens de dimensão 128 empregando $dimB = 21$, por exemplo, a quantidade de processadores em paralelo não pode exceder o número máximo de seis unidades ($np \leq 6$).

Analisando-se os gráficos de desempenho dos programas paralelos, observa-se que ambas as estratégias paralelas não são muito eficientes no tratamento de imagens de resoluções pequenas (128×128 ou menores), usando mais de um processador. Esse comportamento é esperado, visto que a aplicação da computação paralela só se justifica nos casos onde há um processamento intenso e/ou um volume de dados grande para ser processado que compense os tempos gastos com as sobrecargas (*overheads*) do processamento em paralelo e que, ainda assim, produza algum ganho no tempo de execução e faça uma melhor aplicação dos recursos computacionais.

Na Figura 5.4, são mostrados os gráficos das medidas de desempenho obtidas com a restauração de uma imagem com resolução de 256×256 pixels.

Comparando-se os gráficos das medidas de desempenho das Figuras 5.3 a 5.7, observa-se que a segunda versão paralela do programa de restauração apresenta um melhor rendimento no processamento de imagens com resolução maiores ou

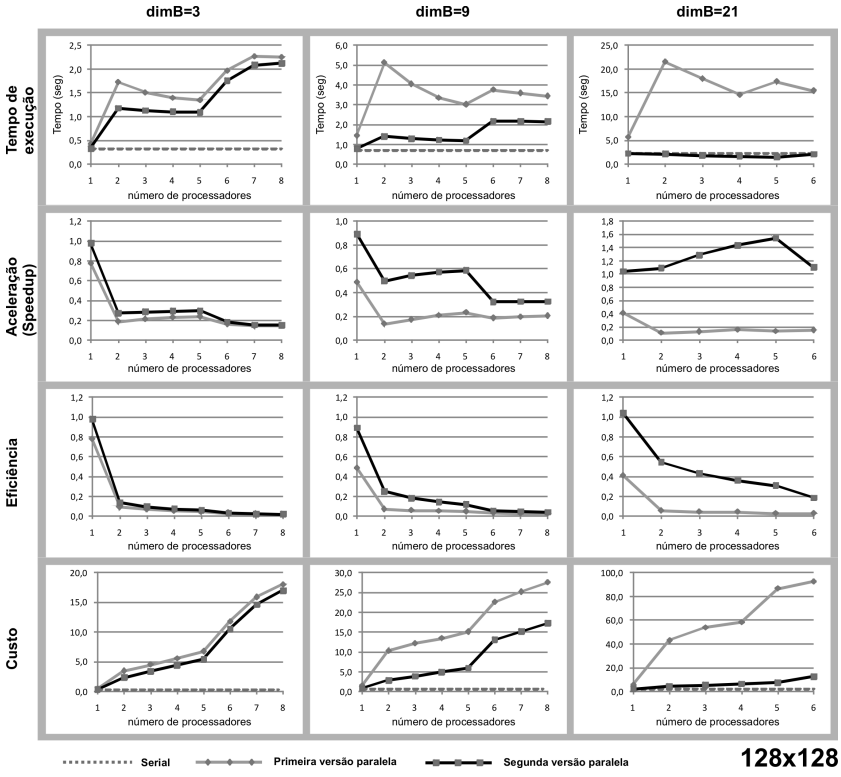


Figura 5.3: Gráficos de medidas de desempenho obtidas com restaurações de uma imagem de 128×128 pixels.

quando o processo de restauração demanda um esforço computacional maior como, por exemplo, quando se aumenta a dimensão do operador de borramento $dimB$. Para estes casos, verifica-se que à medida que o esforço computacional cresce, a segunda versão paralela do algoritmo de restauração torna-se mais eficiente em relação às outras implementações.

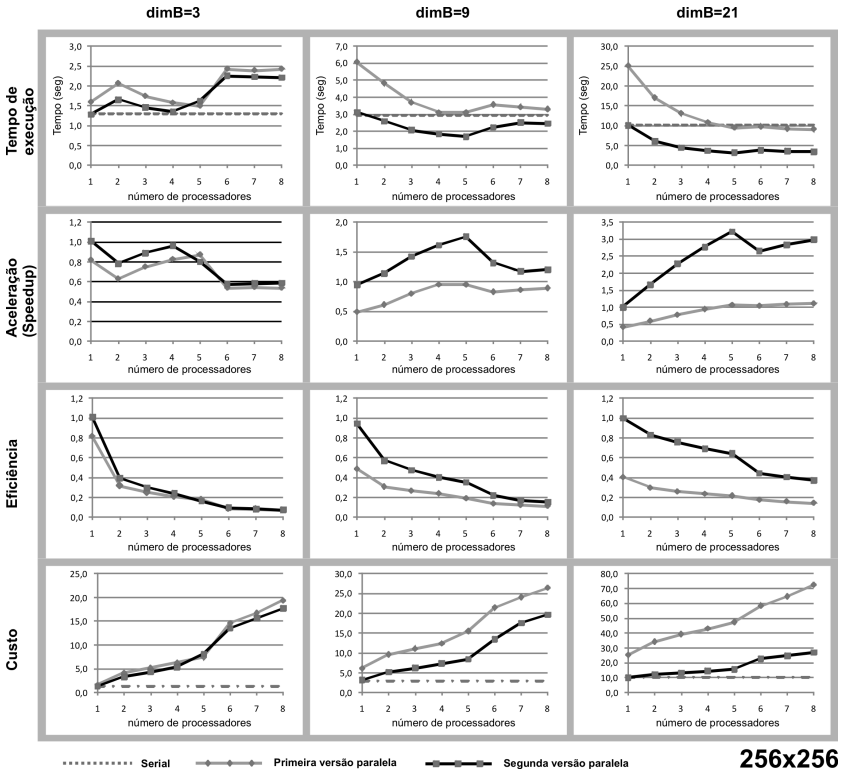


Figura 5.4: Gráficos de medidas de desempenho obtidas com restaurações de uma imagem de 256×256 pixels.

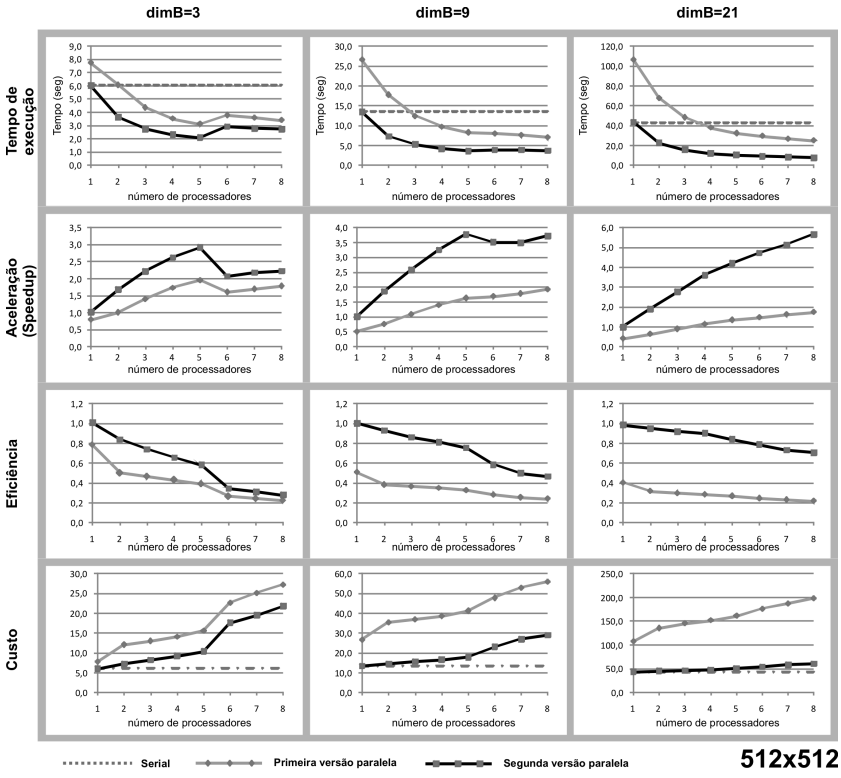


Figura 5.5: Gráficos de medidas de desempenho obtidas com restaurações de uma imagem de 512×512 pixels.

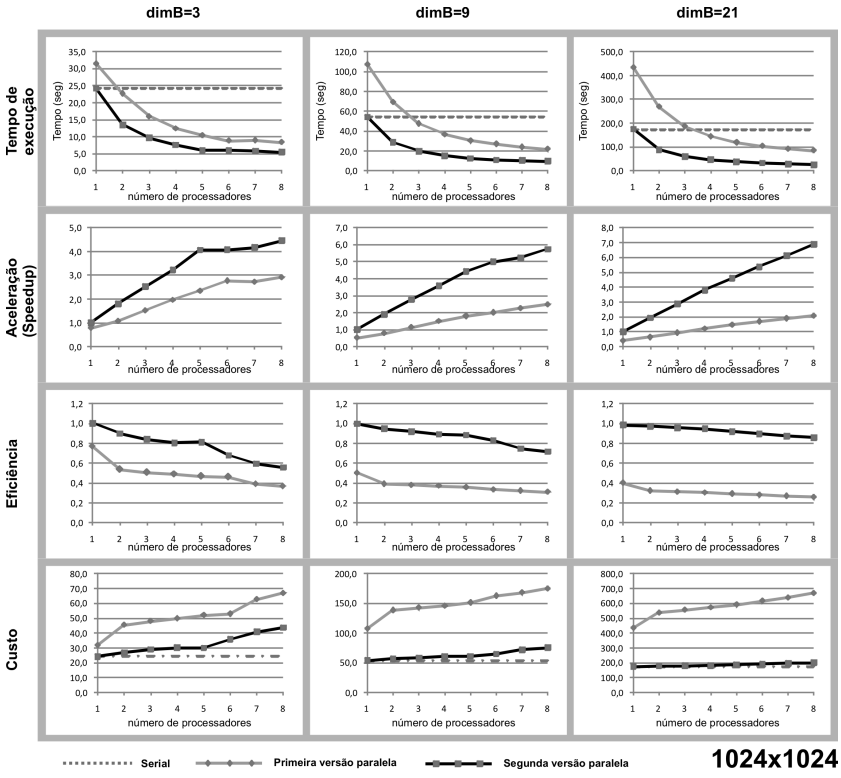


Figura 5.6: Gráficos de medidas de desempenho obtidas com restaurações de uma imagem de 1024×1024 pixels.

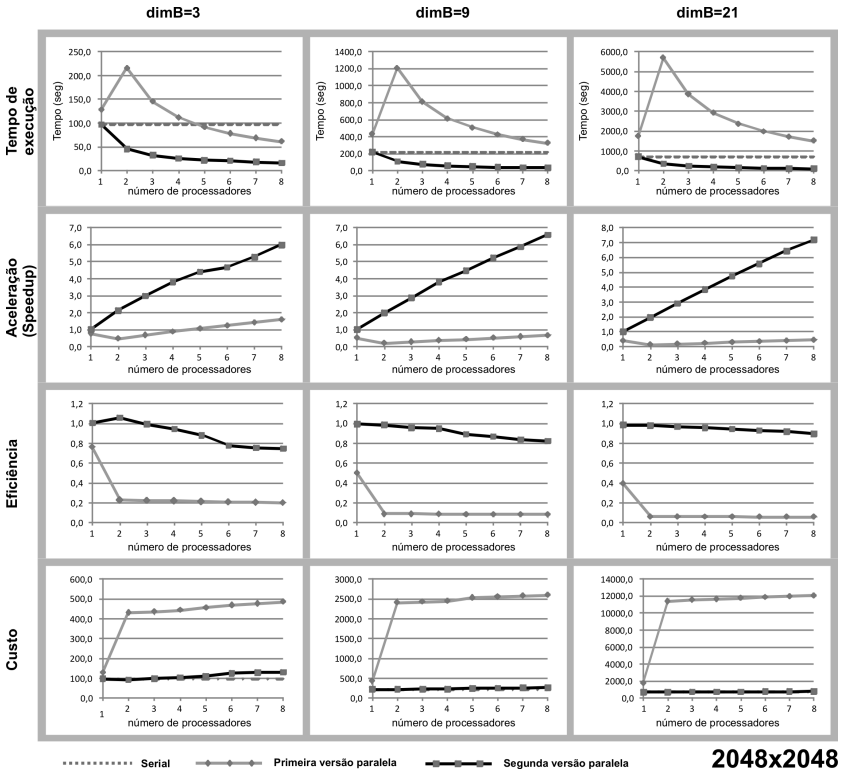


Figura 5.7: Gráficos de medidas de desempenho obtidas com restaurações de uma imagem de 2048×2048 pixels.

5.2.3 Metodologia de avaliação da qualidade das restaurações

Seguindo a mesma linha de pensamento e a preocupação adotada em relação às providências que foram tomadas para se avaliar o desempenho computacional dos programas de restauração, aqui são descritos alguns cuidados e procedimentos que foram tomados para se avaliar a qualidade das restaurações feitas com os programas de restauração:

(a) Para a avaliação dos resultados e da qualidade das imagens recuperadas pelos programas de restauração, foram utilizadas duas imagens conhecidas, a imagem-padrão e uma imagem de texto, como imagens originais e, de modo a simular os efeitos degenerativos produzidos por um AFM durante o processo de aquisição de imagens (borramento e ruído aditivo), com essas imagens produziu-se um conjunto de imagens-teste borradas para que fossem recuperadas pelos programas de restauração e, ao final do processo, os resultados encontrados (restaurações) fossem avaliados e comparados qualitativamente com a imagem original conhecida;

(b) O conjunto de imagens-teste borradas foi gerado a partir das imagens originais, conforme o método descrito na Seção 2.7, variando-se os parâmetros: (i) dimensão da matriz de borramento, $dimB$; (ii) relação sinal-ruído, SNR; e (iii) variância do borramento gaussiano, σ^2, Equação 2.3.6;

(c) Todas as restaurações empregaram um total de 50 iterações completas, com realimentação, sendo que os programas em paralelo utilizaram oito unidades de processamento, i.e. $np = 8$, e a primeira versão paralela empregou uma área de sobreposição igual a $N = \frac{dimB-1}{2}$;

(d) Na restauração das imagens borradas, empregaram-se os mesmos parâmetros usados para se borrar as imagens-teste e o valor do parâmetro de regularização α, Equação 2.4.9, adotado foi escolhido conforme sugerido por Banham e Katsaggelos [3].

5.2.4 Análise de qualidade da restauração

Na Figura 5.8 é mostrado um conjunto de imagens-teste modificadas da imagem-padrão usando uma matriz de borramento de dimensão $dimB = 3$, variando-se $\sigma^2 \in \{20; 40\}$ e $SNR \in \{20; 30; 40\}$. Junto às imagens modificadas estão as restaurações obtidas com os programas paralelos de restauração. Na Tabela 5.1 são mostrados os valores das medidas de avaliação (Seção 4.2) da qualidade das restaurações feitas com as duas versões do programa paralelo.

Métricas	Padrão modificado D3v20s20	Primeira versão	Segunda versão
MAE	32,4759	32,4149	28,9123
MSE	2835,15	2147,08	1753,96
NME	0,214092	0,21369	0,1906
NRMSE	7770,52	5884,66	4807,21
%MSE	N/A	75,7%	61,9%
SNR	10,4895	11,6968	12,5751
PSNR	13,605	14,8123	15,6906
IWMSE/G	31509,1	23840,1	19773,2
IWMSE/S	6331,41	4436,33	3604,84
IWMSE/V	7930,31	4043,84	3347,72

(a)

Métricas	Padrão modificado D3v20s30	Primeira versão	Segunda versão
MAE	26,33	16,4883	15,2305
MSE	2603,81	760,201	671,576
NME	0,173577	0,108697	0,100404
NRMSE	7136,46	2083,54	1840,64
%MSE	N/A	26,8%	23,7%
SNR	10,8592	16,206	16,7443
PSNR	13,9747	19,3215	19,8599
IWMSE/G	27728	8787,41	7815,78
IWMSE/S	5846,66	1641,48	1403,16
IWMSE/V	7538,34	1661,68	1460,57

(b)

Métricas	Padrão modificado D3v20s40	Primeira versão	Segunda versão
MAE	24,6324	10,715	10,5756
MSE	2579,5	561,034	513,556
NME	0,162385	0,070637	0,069718
NRMSE	7069,84	1537,67	1407,54
%MSE	N/A	19,8%	18,1%
SNR	10,8999	17,5254	17,9094
PSNR	14,0154	20,6409	21,0249
IWMSE/G	24917,7	6794,6	6144,34
IWMSE/S	5864,61	1264,42	1095,48
IWMSE/V	7495,55	1313,1	1180,52

(c)

Métricas	Padrão modificado D3v40s20	Primeira versão	Segunda versão
MAE	32,4677	32,3801	28,8573
MSE	2838,12	2144,74	1748,02
NME	0,214038	0,21346	0,190237
NRMSE	7778,66	5878,25	4790,93
%MSE	N/A	75,6%	61,6%
SNR	10,4849	11,7015	12,5898
PSNR	13,6005	14,8171	15,7053
IWMSE/G	31511,8	23809,5	19678,8
IWMSE/S	6338,55	4426,25	3589,89
IWMSE/V	7939,88	4030,72	3329,98

(d)

Métricas	Padrão modificado D3v40s30	Primeira versão	Segunda versão
MAE	26,3261	16,3718	15,1484
MSE	2607,54	752,249	662,872
NME	0,173551	0,107929	0,099863
NRMSE	7146,68	2061,75	1816,78
%MSE	N/A	26,5%	23,4%
SNR	10,853	16,2516	16,801
PSNR	13,9685	19,3672	19,9165
IWMSE/G	27651,3	8679,39	7716,44
IWMSE/S	5858,15	1621,31	1383,45
IWMSE/V	7548,82	1638,07	1438,83

(e)

Métricas	Padrão modificado D3v40s40	Primeira versão	Segunda versão
MAE	24,5783	10,5952	10,4687
MSE	2583,26	554,95	505,032
NME	0,162029	0,069848	0,069013
NRMSE	7080,13	1520,99	1384,18
%MSE	N/A	19,6%	17,8%
SNR	10,8936	17,5727	17,9821
PSNR	14,0091	20,6883	21,0976
IWMSE/G	24819,2	6715,08	6038,04
IWMSE/S	5873,42	1247,99	1072,59
IWMSE/V	7506,52	1296,13	1158,00

(f)

Tabela 5.1: Medidas de qualidade de restaurações realizadas com a primeira e a segunda versões paralelas do programa de restauração, em um conjunto de imagens-padrão modificadas (a-f).

Na Figura 5.9 é mostrado um conjunto de imagens borradas de um texto, empregando-se $dimB = 2N + 1 = 5$, variando-se $\sigma^2 \in \{20; 40\}$ e $SNR \in \{20;\ 30;\ 40\}$ e, junto delas, aparecem as restaurações, obtidas com os programas paralelos de restauração. Na Tabela 5.2 são mostrados os valores das medidas de qualidade.

Métricas	Padrão modificado D5v20s20	Primeira versão	Segunda versão	Padrão modificado D5v20s30	Primeira versão	Segunda versão
MAE	40,0731	35,7066	33,364	36,4014	26,7723	23,881
MSE	3917,44	3448,38	3214,23	3581,41	2185,72	1945
NME	0,182225	0,16237	0,151717	0,165529	0,121742	0,108594
NRMSE	14472,1	12739,3	11874,2	13230,7	8074,66	7185,36
%MSE	N/A	88,0%	82,0%	N/A	55,8%	49,6%
SNR	11,3259	11,8798	12,1852	11,7154	13,86	14,3668
PSNR	12,2008	12,7546	13,06	12,5903	14,7349	15,2416
IWMSE/G	44688,6	43320,5	41003,1	40126,6	28924,6	26233,6
IWMSE/S	7512,8	6812,72	6274,1	7208,42	4180,28	3657,62
IWMSE/V	8977,55	7652,8	7188,69	8131,59	4803,47	4239,19
		(a)			(b)	

Métricas	Padrão modificado D5v20s40	Primeira versão	Segunda versão	Padrão modificado D5v40s20	Primeira versão	Segunda versão
MAE	35,4424	24,8409	21,1502	40,5419	35,786	33,3122
MSE	3546,64	1998,32	1701,3	3975,25	3438,87	3182,72
NME	0,161168	0,11296	0,096177	0,184357	0,162731	0,151481
NRMSE	13102,3	7382,35	6285,07	14685,7	12704,1	11757,9
%MSE	N/A	56,3%	48,0%	N/A	97,0%	89,7%
SNR	11,7578	14,2493	14,9482	11,2623	11,8918	12,228
PSNR	12,6326	15,1242	15,823	12,1372	12,7666	13,1028
IWMSE/G	39263,9	26579,5	23195,6	45255,3	43208	40601
IWMSE/S	7128,59	3768,95	3268,73	7609,88	6818,46	6244,49
IWMSE/V	8046	4441,16	3700,24	9111,99	7366,81	6791,7
		(c)			(d)	

Métricas	Padrão modificado D5v40s30	Primeira versão	Segunda versão	Padrão modificado D5v40s40	Primeira versão	Segunda versão
MAE	36,7809	26,7874	23,9021	35,6451	24,635	21,161
MSE	3630,72	2161,76	1912,47	3593,74	1958,76	1664,89
NME	0,167255	0,121811	0,108691	0,16209	0,112023	0,096226
NRMSE	13412,9	7986,15	7065,21	13276,3	7236,19	6150,55
%MSE	N/A	59,5%	52,7%	N/A	53,9%	45,9%
SNR	11,656	13,9079	14,44	11,7005	14,3362	15,0421
PSNR	12,5309	14,7827	15,3148	12,5753	15,211	15,917
IWMSE/G	40709	28538,4	25732,5	39542	26079,2	22650,6
IWMSE/S	7297,54	4141,77	3610,07	7220,95	3695,5	3201,82
IWMSE/V	8259,81	4700,15	4071,75	8169,38	4321,27	3541,16
		(e)			(f)	

Tabela 5.2: Medidas de qualidade de restaurações realizadas com a primeira e a segunda versões paralelas do programa de restauração, em um conjunto de imagens borradas de um texto (a-f).

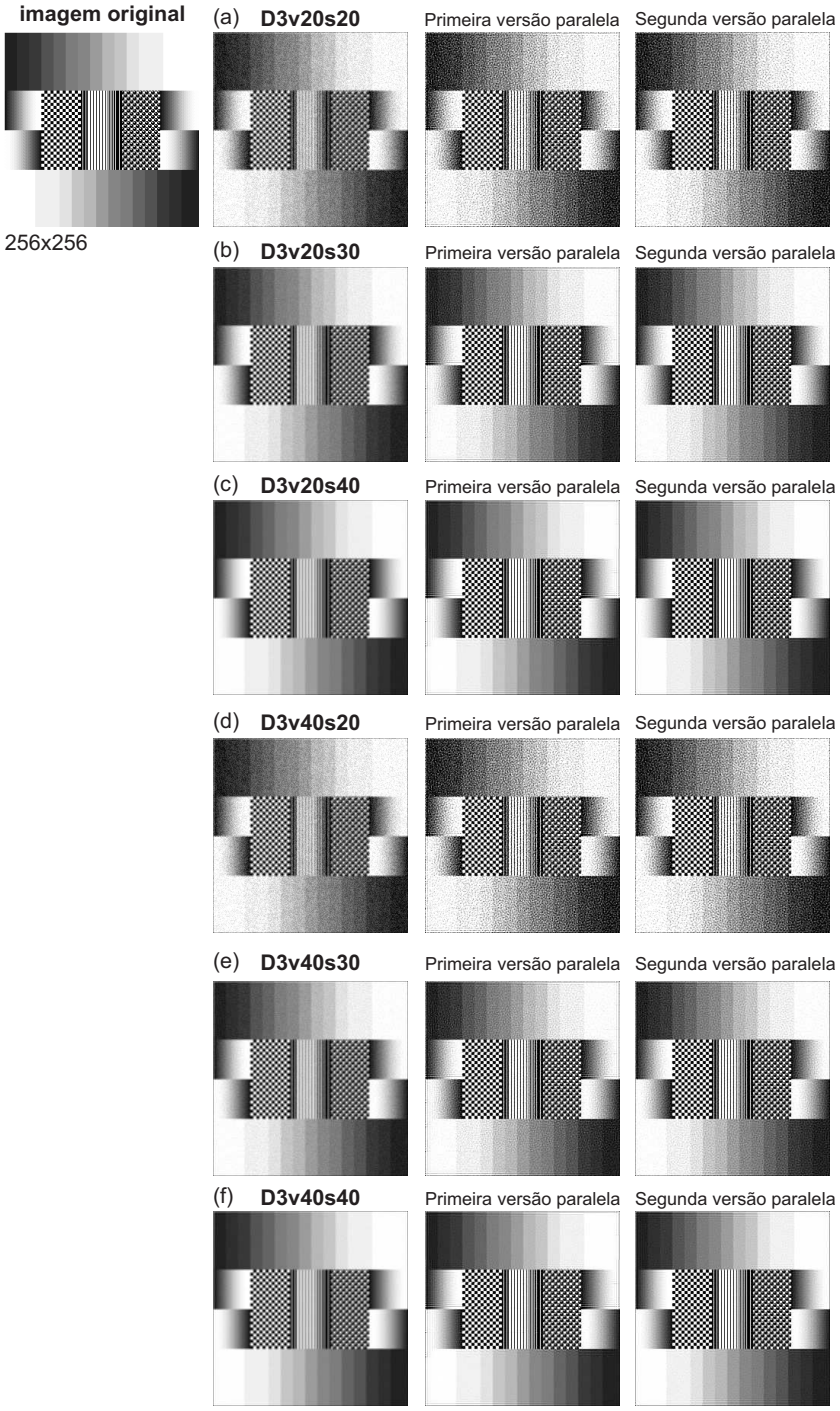


Figura 5.8: Imagens-padrão modificadas e restaurações destas realizadas com a primeira e a segunda versões paralelas do programa de restauração.

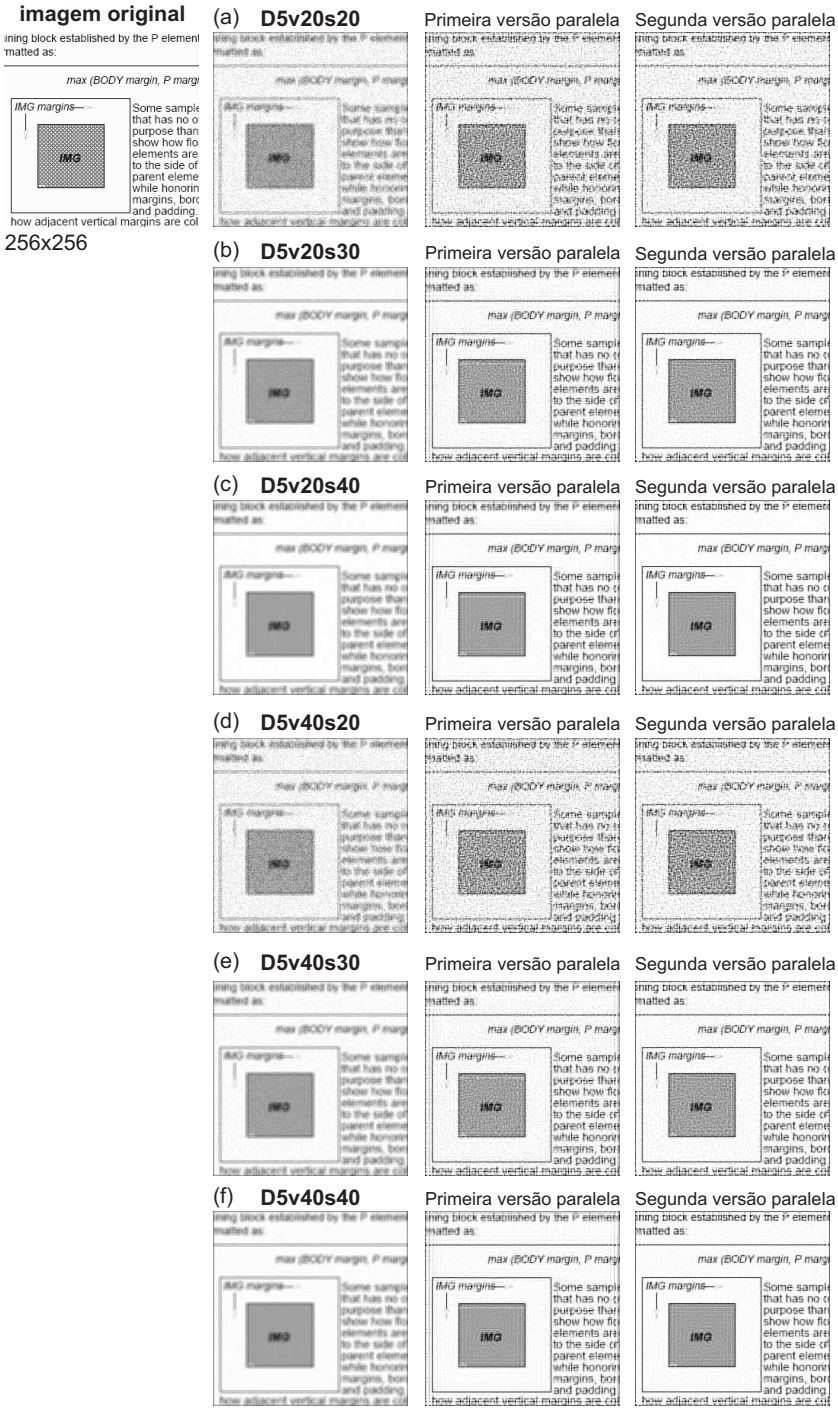


Figura 5.9: Imagem de um texto e restaurações realizadas com a primeira e a segunda versões paralelas do programa de restauração.

5.3 Restauração de imagens de AFM

Para exemplificar um caso real de restauração de imagens de AFM, nas Figuras 5.10, 5.11 e 5.12 são mostradas as restaurações de três imagens de AFM, obtidas com a versão serial de restauração e a segunda versão paralela com oito processadores em paralelo, ambas empregando o método de Jacobi simplificado. As semelhanças nas restaurações, produzidas pelos dois programas, não são meramente visuais, o MSE obtido com as duas imagens restauradas é igual a zero, o que significa que as restaurações produzidas pela segunda versão paralela são exatamente iguais àquelas que obtidas com a versão serial, demonstrando de que o efeito-borda, que ocorre na primeira versão paralela, inexiste na segunda versão do programa de restauração.

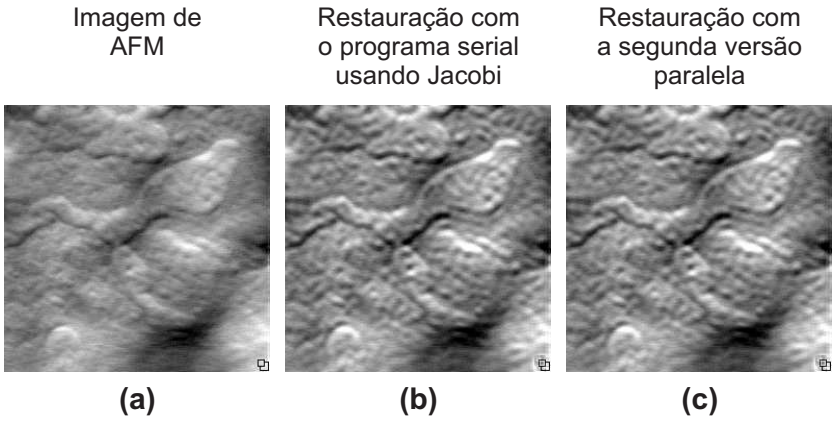


Figura 5.10: Restauração de imagem de AFM, (a) eritroblasto em estado leucêmico de $1000nm \times 1000nm$, obtidas com (b) o programa serial e com (c) a segunda versão paralela, ambos empregando Jacobi simplificado e os seguintes parâmetros de restauração: $dimB = 9; \alpha = 0,05; q = 1; \gamma = 0,1; \sigma^2 = 20; maxIter = 20$ e realimentação.

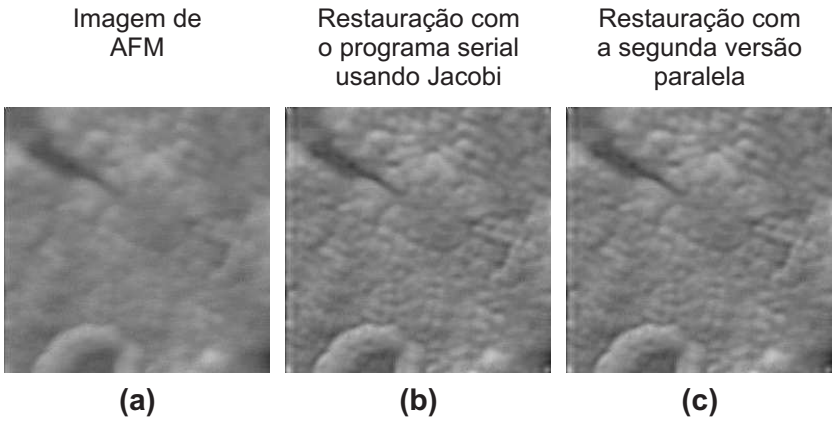


Figura 5.11: Restauração de imagem de AFM, (a) eritroblasto em estado leucêmico de $600nm \times 600nm$, obtidas com (b) o programa serial e com (c) a segunda versão paralela, ambos empregando Jacobi simplificado e os seguintes parâmetros de restauração: $dimB = 13; \alpha = 0,5; q = 1; \gamma = 0,1; \sigma^2 = 10; maxIter = 50$ e realimentação.

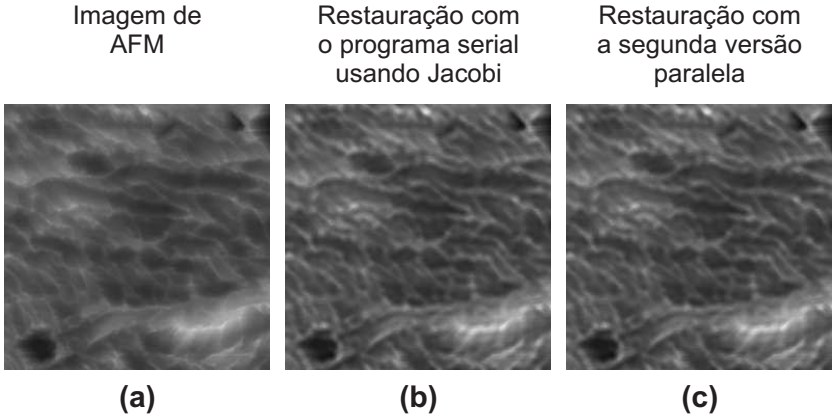


Figura 5.12: Restauração de imagem de AFM da (a) superfície de uma amostra de ferro submetidas a dissolução em H_2SO_4 de $30\mu m \times 30\mu m$ [46], obtidas com (b) o programa serial e com (c) a segunda versão paralela, ambos empregando Jacobi simplificado e os seguintes parâmetros de restauração: $dimB = 9; \alpha = 0,5; q = 0,5; \gamma = 0,2; \sigma^2 = 20; maxIter = 20$ e realimentação.

Capítulo 6

Parâmetro de Regularização

6.1 Parâmetro de regularização ótimo

Uma das questões centrais a respeito da aplicação de regularização em restaurações de imagens está relacionada à escolha apropriada do parâmetro de regularização α, fator determinante que estabelece o compromisso entre a acurácia e a estabilidade da solução. Várias técnicas tem sido desenvolvidas de modo a se determinar este parâmetro [3, 26, 27, 38, 44, 60].

Neste capítulo será apresentado um novo método iterativo para a obtenção de um parâmetro de regularização "ótimo" para ser empregado no problema de restauração de imagens descrito na Seção 2.4, onde o parâmetro considerado ótimo é aquele que é definido em termos da imagem restaurada a cada iteração, com vistas à próxima iteração.

$$\alpha^* = arg_\alpha min Q(\widehat{x}^{t+1}), \alpha^* > 0 \tag{6.1.1}$$

onde, na primeira iteração ($t = 0$), teremos

$$Q(\widehat{x}^1) = \sum_{i=0}^{M}\sum_{j=0}^{M}\left[y_{ij} - \sum_{k=-N}^{N}\sum_{l=-N}^{N} b_{kl}\widehat{x}^1_{i+k,j+l}\right]^2 + \alpha S_q(\widehat{x}^1, \overline{x}) \tag{6.1.2}$$

Na Figura 6.1 são mostradas as diferenças entre os passos de restauração usando um parâmetro de regularização fixo (Figura 6.1a), conhecido *a priori*, e um parâmetro de regularização variável que é determinado aplicando-se o método iterativo proposto (parâmetro de regularização dinâmico ou α^*).

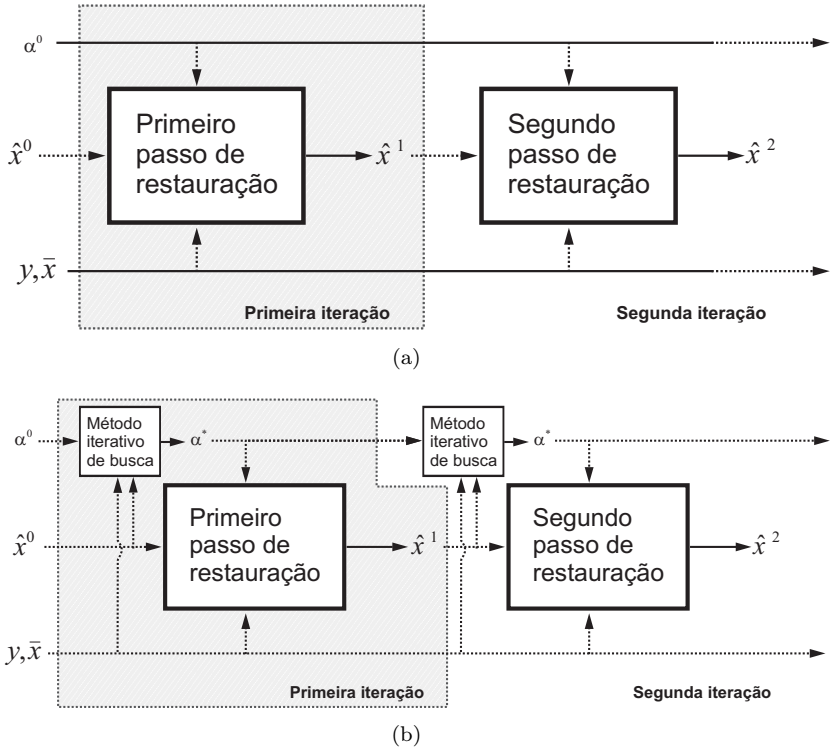


Figura 6.1: Os passos de restauração: (a) usando um parâmetro de regularização fixo; e (b) empregando o método iterativo de busca.

6.2 Método iterativo de busca do parâmetro de regularização ótimo

Aplicando 2.4.12 em 6.1.2, obtém-se

$$Q(\widehat{x}^1) = f(\widehat{x}^0, \alpha) + g(\widehat{x}^0, \alpha) \tag{6.2.3}$$

onde

$$f(\widehat{x}^0,\alpha)=\sum_{i=0}^{M}\sum_{j=0}^{M}\left[\left(y_{ij}-\sum_{k=-N}^{N}\sum_{l=-N}^{N}b_{kl}\widehat{x}^0_{i+k,j+l}\right)+ -\gamma\sum_{k=-N}^{N}\sum_{l=-N}^{N}b_{kl}\Delta\widehat{x}^0_{i+k,j+l}\right]^2 \tag{6.2.4}$$

e

$$g(\widehat{x}^0,\alpha)=\frac{\alpha}{1+q}\sum_{i=0}^{M}\sum_{j=0}^{M}\left\{(\widehat{x}^0_{ij}+\gamma\Delta\widehat{x}^0_{ij})\left[\frac{(\widehat{x}^0_{ij}+\gamma\Delta\widehat{x}^0_{ij})^q-\overline{x}^q_{ij}}{q}\right]+ -\overline{x}^q_{ij}\widehat{x}^0_{ij}-\gamma\overline{x}^q_{ij}\Delta\widehat{x}^0_{ij}+\overline{x}^{q+1}_{ij}\right\} \tag{6.2.5}$$

Na busca do parâmetro de regularização ótimo α que minimize o funcional $Q(\widehat{x}^1)$ na próxima iteração, a equação do ponto crítico pode ser escrita como:

$$\frac{dQ(\widehat{x}^1)}{d\alpha}=\frac{df(\widehat{x}^0,\alpha)}{d\alpha}+\frac{dg(\widehat{x}^0,\alpha)}{d\alpha}=0 \tag{6.2.6}$$

onde

$$\frac{df(\widehat{x}^0,\alpha)}{d\alpha}=-2\gamma\left(R(\widehat{x}^0)-\gamma\sum_{i=0}^{M}\sum_{j=0}^{M}\left[\sum_{k=-N}^{N}\sum_{l=-N}^{N}b_{kl}\Delta\widehat{x}^0_{i+k,j+l}\right]\right) \times\left(\sum_{i=0}^{M}\sum_{j=0}^{M}\left[\sum_{k=-N}^{N}\sum_{l=-N}^{N}b_{kl}\frac{d(\Delta\widehat{x}^0_{i+k,j+l})}{d\alpha}\right]\right) \tag{6.2.7}$$

$$\frac{dg(\widehat{x}^0,\alpha)}{d\alpha}=S_q(\widehat{x}^1,\overline{x})+\alpha\frac{\gamma}{q}\sum_{i=0}^{M}\sum_{j=0}^{M}\left\{\left((\widehat{x}^1_{ij})^q-\overline{x}^q_{ij}\right)\frac{d(\Delta\widehat{x}^0_{ij})}{d\alpha}\right\} \tag{6.2.8}$$

$$\frac{d(\Delta\widehat{x}^t_{ij})}{d\alpha}=\left.\frac{F_{rs}\frac{dF_{mn}}{d\alpha}-\frac{dF_{rs}}{d\alpha}F_{mn}}{(F_{mn})^2}\right|^t_{\substack{m=r\\n=s}} \tag{6.2.9}$$

$$\frac{d(F_{rs})}{d\alpha}=\left.\frac{(\widehat{x}^q_{rs}-\overline{x}^q_{rs})}{q}\right|^t \tag{6.2.10}$$

$$\left.\frac{d(F_{mn})}{d\alpha}\right|_{\substack{m=r\\n=s}}=\left.\widehat{x}^{q-1}_{rs}\right|^t \tag{6.2.11}$$

e

$$\frac{d}{d\alpha}\left(\frac{1}{F_{mn}}\right)\Bigg|_{\substack{m=r\\n=s}}=-\left.\frac{\widehat{x}^{q-1}_{rs}}{F^2_{mn}}\right|^t_{\substack{m=r\\n=s}} \tag{6.2.12}$$

Empregando o método de Newton-Raphson para resolver a Equação 6.2.6, obtém-se o algoritmo do método iterativo de busca para a determinação do parâmetro de regularização dado por

$$\alpha^{k+1} = \alpha^k - \lambda \frac{\varphi(\alpha^k)}{\varphi'(\alpha^k)}, k = 0, 1, 2, \ldots \tag{6.2.13}$$

onde λ é um fator de atenuação,

$$\varphi(\alpha) = \frac{dQ(\widehat{x}^1)}{d\alpha} \tag{6.2.14}$$

$$\varphi'(\alpha) = \frac{d^2 f(\widehat{x}^0, \alpha)}{d\alpha^2} + \frac{d^2 g(\widehat{x}^0, \alpha)}{d\alpha^2} \tag{6.2.15}$$

$$\begin{aligned}\frac{d^2 f(\widehat{x}^0, \alpha)}{d\alpha^2} = -2\gamma & \left(R(\widehat{x}^0) - \gamma \sum_{i=0}^{M} \sum_{j=0}^{M} \left[\sum_{k=-N}^{N} \sum_{l=-N}^{N} b_{kl} \Delta \widehat{x}^0_{i+k,j+l} \right] \right) \\ & \times \left(\sum_{i=0}^{M} \sum_{j=0}^{M} \left[\sum_{k=-N}^{N} \sum_{l=-N}^{N} b_{kl} \frac{d^2(\Delta \widehat{x}^0_{i+k,j+l})}{d\alpha^2} \right] \right) + \\ & + 2\gamma^2 \left(\sum_{i=0}^{M} \sum_{j=0}^{M} \left[\sum_{k=-N}^{N} \sum_{l=-N}^{N} b_{kl} \frac{d(\Delta \widehat{x}^0_{i+k,j+l})}{d\alpha} \right] \right)^2\end{aligned} \tag{6.2.16}$$

$$\begin{aligned}\frac{d^2 g(\widehat{x}^0, \alpha)}{d\alpha^2} = & \frac{\gamma}{q} \left[\sum_{i=0}^{M} \sum_{j=0}^{M} \left\{ \left((\widehat{x}^1_{ij})^q - \overline{x}^q_{ij} \right) \left(2 \frac{d(\Delta \widehat{x}^0_{i,j})}{d\alpha} + \alpha \frac{d^2(\Delta \widehat{x}^0_{i,j})}{d\alpha^2} \right) \right\} \right] + \\ & + \alpha\gamma^2 \sum_{i=0}^{M} \sum_{j=0}^{M} \left\{ (\widehat{x}^1_{ij})^{q-1} \left[\frac{d(\Delta \widehat{x}^0_{i,j})}{d\alpha} \right] \right\}\end{aligned} \tag{6.2.17}$$

e

$$\frac{d^2(\Delta \widehat{x}^t_{rs})}{d\alpha^2} = \frac{2\widehat{x}^{q-1}_{rs}}{q F^2_{mn}} \left\{ (\widehat{x}^q_{rs} - \overline{x}^q_{rs}) - q \frac{\widehat{x}^{q-1}_{rs} F_{rs}}{F_{mn}} \right\} \Bigg|_{\substack{t \\ m=r \\ n=s}} \tag{6.2.18}$$

Critério de determinação do parâmetro de regularização ótimo α^*

A determinação para que o novo valor do parâmetro de regularização α^{k+1} seja aplicado na próxima iteração do algoritmo de restauração de imagens, é sujeito às seguintes condições:

a) $\alpha^{k+1} > 0$; e

b) $Q(\widehat{x}^1, \alpha^{k+1}) < Q(\widehat{x}^1, \alpha^*)$.

Se ambas condições forem satisfeitas, então faz-se $\alpha^* = \alpha^{k+1}$ o novo valor a ser aplicado na próxima iteração. Caso contrário, o valor de α^*, usado na iteração anterior, é mantido.

Critério de parada

O processo iterativo descrito pode ser interrompido se uma das seguintes condições ocorrer:

a) uma tolerância definida *a priori tol* é alcançada: $|\alpha^{k+1} - \alpha^k| < tol$;

b) a condição $\alpha^{k+1} > 0$ não é satisfeita; ou

c) um número máximo de iterações $maxIterAlfa$, previamente definido para o método de busca, é excedido; ou

d) quando $Q(\widehat{x}^1, \alpha^{k+1}) > Q(\widehat{x}^1, \alpha^k)$.

Pseudo-código do método iterativo

Um pseudo-código simplificado do algoritmo de restauração de imagens, empregando o método iterativo de busca (Figura 6.1b), é mostrado abaixo:

1. Faça $\widehat{x}^0_{ij} = y_{ij}$ e $\bar{x}^0_{ij} = y_{ij}$
2. Faça $\widehat{x}^0$ a solução inicial do problema
3. Faça $minQ = Q(\widehat{x}^0)$ o valor mínimo do funcional Q
4. Faça $maxIter$ = número máximo de iterações
5. Faça $t = 0$
6. repetir
7. $t = t + 1$
8. Faça $maxIterAlfa$ = número máximo de iterações do método de busca
9. Calcule o valor do funcional $Q(\widehat{x}^t, \alpha)$ com a Equação 2.4.9
10. Faça $Qmin = Q(\widehat{x}^t, \alpha)$
11. $k = 0$, $\alpha^0 = \alpha$ e $\alpha^* = \alpha$

12. repetir

13. Calcule $\varphi(\alpha^k)$ com a Equação 6.2.14 e $\varphi'(\alpha^k)$ com a Equação 6.2.15

14. Calcule α^{k+1} com a Equação 6.2.13

15. Se $\alpha^{k+1} \geq 0$ então

16. Calcule o valor do funcional $Q(\widehat{x}^t, \alpha^{k+1})$ com a Equação 2.4.9

17. Se $Q(\widehat{x}^t, \alpha^{k+1}) < Qmin$ então

18. Faça $Qmin = Q(\widehat{x}^t, \alpha^{k+1})$

19. Faça $\alpha^* = \alpha^{k+1}$

20. Faça $k = k + 1$

21. até $\alpha^{k-1} < 0$ ou $|\alpha^k - \alpha^{k-1}| < tol$
ou $Q(\widehat{x}^1, \alpha^{k+1}) > Q(\widehat{x}^1, \alpha^k)$ ou $k = maxIterAlfa$

22. Faça $t = t + 1$

23. Faça $\alpha = \alpha^*$

24. Calcule a derivada F_{rs} com a Equação 2.4.15

25. Calcule a derivada F_{mn} com a Equação 2.4.16

26. Calcule as correções $\Delta\widehat{x}$ com a Equação 2.4.14

27. Calcule as novas estimativas $\widehat{x}^{t+1}$ com a Equação 2.4.12

28. Calcule o valor do functional $Q(\widehat{x}^t)$ com a Equação 2.4.9

29. Se $Q(\widehat{x}^t) < minQ$ então

30. Aceite $\widehat{x}^t$ com solução para o problema de restauração de imagens

31. Faça $minQ = Q(\widehat{x}^t)$

32. até $Q(\widehat{x}^t) > Q(\widehat{x}^{t-1})$ ou t=maxIter

6.3 Resultados

Para testar o método iterativo de busca, foi utilizada uma imagem borrada da imagem-padrão (D5v20s40), com MSE=3345,29, e o desempenho do método foi medido, comparando-se os valores do funcional Q a cada iteração, obtidos com duas execuções consecutivas do programa de restauração de imagens: uma usando o parâmetro de regularização fixo e outra empregando o parâmetro dinâmico α^*. Em todas as restaurações foram empregados os mesmos parâmetros de restauração e a medida de avaliação de qualidade usada foi o MSE (Equação 4.2.7), comparando-se a imagem-padrão com as imagens restauradas, após 20 iterações.

Para demonstrar o funcionamento do método de busca do parâmetro de regularização ótimo, o programa de restauração foi executado usando-se diferentes valores iniciais do parâmetro α, $\alpha^0 \in \{0,03049; 0,01; 0,05; 0,1; 0,5\}$. O primeiro valor, $\alpha^0 = 0,03049$, foi obtido [3],

$$\alpha = \frac{1}{BSNR} \tag{6.3.19}$$

onde BSNR (*Blurred Signal-to-Noise Ratio*) é dado na Equação 4.2.14.

O valor do parâmetro α dado pelas Equações 6.3.19 e 4.2.14, só pode ser obtido quando as variâncias do sinal e do ruído da imagem modificada são conhecidos ou, então, quando estas puderem ser estimadas. Neste caso, o valor obtido para a imagem-padrão modificada (D5v20s40) foi $\alpha^0 = 0,03049$.

Na Tabela 6.1, são mostrados os valores do funcional Q obtidos a cada iteração do algoritmo de restauração, com o valor inicial $\alpha^0 = 0,03049$. Na coluna α^*, são mostrados os novos valores do parâmetro de restauração α obtidos com o algoritmo de busca, e na última linha da tabela são mostrados os valores de MSE obtidos no final do processamento do programa de restauração em ambas execuções: uma empregando um valor α constante e a outra com um valor dinâmico. No primeiro caso (Tabela 6.1), podemos observar uma pequena melhora nos resultados obtidos com o algoritmo de busca do parâmetro de regularização ótimo.

Na Tabela 6.2 são mostrados os valores do funcional Q obtidos a cada iteração do programa de restauração com o valor inicial do parâmetro de regularização $\alpha^0 = 0,01$.

Na Figura 6.2 é mostrada a evolução dos valores do funcional Q obtidos em ambas execuções (α constante e dinâmico) com os valores iniciais $\alpha^0 = 0,01$ (Figura 6.2a) e $\alpha^0 = 0,5$ (Figura 6.2b).

Para os valores $\alpha^0 \in \{0,05; 0,1\}$, o programa de restauração produziu resultados similares àqueles mostrados na Tabela 6.1, onde foi empregado $\alpha^0 = 0,03049$.

Para permitir a comparação dos resultados do programa de restauração de imagens empregando diferentes valores de α^0, os resultados bem como o valor dinâmico de α são mostrados na Tabela 6.3, onde o MSE relativo ou %MSE (Equação 4.2.10)

Iter.	const. α	Q(x)	α^*	Q(x)
1	0,03049	98,0961	0,029172	97,8575
2	”	83,4503	0,028499	83,0747
3	”	72,4027	0,027705	71,8424
4	”	64,0233	0,027018	63,3418
5	”	57,3433	0,026283	56,5492
6	”	51,9434	0,025599	51,0733
7	”	47,6253	0,024888	46,7624
8	”	44,3279	0,024222	43,4807
9	”	41,6772	0,023535	40,8064
10	”	39,4209	0,022856	38,5127
11	”	37,4478	0,022165	36,4995
12	”	35,7068	0,021491	34,727
13	”	34,1579	0,020806	33,1487
14	”	32,7765	0,020137	31,7483
15	”	31,5397	0,019461	30,4941
16	”	30,4295	0,018798	29,3756
17	”	29,4302	0,01813	28,3691
18	”	28,5291	0,017475	27,4687
19	”	27,7148	0,016815	26,6554
20	”	**26,9772**	0,016168	**25,9253**
MSE:	**1466,43**		**1429,03**	

Tabela 6.1: Resultados do programa de restauração de imagens, empregando o parâmetro α constante e dinâmico, $\alpha^0 = 0,03049$ e $\lambda = 0,01$. Os valores $Q(x)$ em negrito representam os menores valores obtidos em cada uma das restaurações.

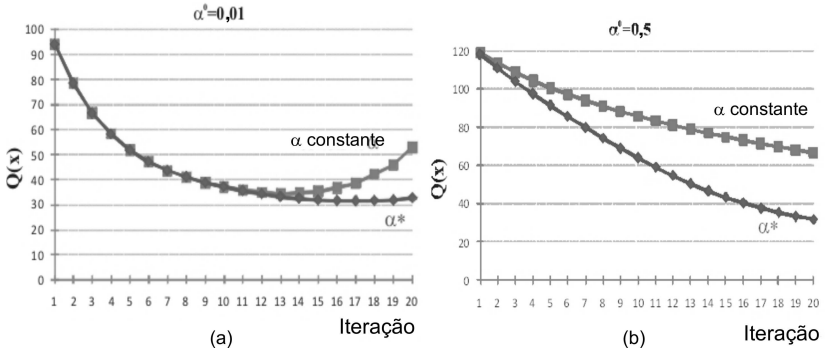


Figura 6.2: Evolução do funcional Q em duas execuções do programa de restauração, uma com α constante e o outro com α^* dinâmico, ambos empregando o parâmetro inicial (a)$\alpha^0 = 0,01$ e (b) $\alpha^0 = 0,5$

Iter.	const. α	Q(x)	α^*	Q(x)
1	0,01	93,9591	0,00983	93,9204
2	”	78,6075	0,00975	78,5572
3	”	66,5784	0,00962	66,5054
4	”	58,2494	0,00954	58,1786
5	”	51,7043	0,00942	51,6423
6	”	47,0093	0,00934	46,9922
7	”	43,5397	”	43,574
8	”	40,9589	”	41,0715
9	”	38,7712	0,01293	38,8214
10	”	37,1847	”	37,0745
11	”	35,8301	0,01298	35,4632
12	”	35,0997	”	34,3063
13	”	**34,5399**	”	33,1912
14	”	34,7729	”	32,5278
15	”	35,1875	”	31,839
16	”	36,7568	”	31,6391
17	”	38,5015	”	**31,3538**
18	”	42,1618	”	31,6452
19	”	45,7977	”	31,7466
20	”	52,8869	”	32,6075
MSE:	**1588,04**		**1454,57**	

Tabela 6.2: Saídas do programa de restauração de imagens, empregando o parâmetro α constante e dinâmico, $\alpha^0 = 0,01$ e $\lambda = 0,01$. Os valores $Q(x)$ em negrito representam os menores valores obtidos em cada uma das restaurações.

representa o ganho da restauração, sendo que quanto menor este valor, maior o ganho.

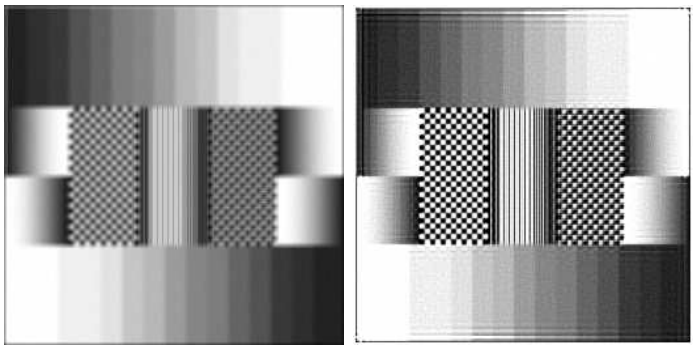
α^0	λ	Q(x) com α const.	Q(x) com α^*	MSE com α const.	MSE com α^*	%MSE com α const.	%MSE com α^*
0,03049	0,01	26,9772	25,9253	1466,43	1429,03	43,8%	42,7%
0,01	0,001	34,5399	31,3538	1588,04	1454,57	47,5%	43,5%
0,05	0,01	29,5004	27,704	1554,12	1492,25	46,5%	44,6%
0,1	0,1	35,5027	24,8483	1752,49	1382,46	52,4%	41,3%
0,5	0,1	66,7226	31,7726	2510,25	1631,6	75,0%	48,8%

Tabela 6.3: Resultados finais das execuções do programa de restauração de imagens, empregando diferentes valores iniciais do parâmetro de regularização.

Para os valores $\alpha^0 \in \{0,03049; 0,01; 0,05\}$, podemos observar na Tabela 6.3 que as restaurações de imagens obtidas com o algoritmo de busca do parâmetro de

regularização ótimo são ligeiramente melhores do que os resultados obtidos com o parâmetro de regularização constante. Entretanto, para os valores $\alpha^0 \in \{0,1; 0,5\}$ as restaurações empregando α^* são significamente melhores.

Dependendo do valor inicial usado para α^0, o processo de restauração de imagens empregando o parâmetro de regularização dinâmico α^* produz um ganho melhor, ou no mínimo melhor. Na Figura 6.3 é mostrada a imagem restaurada com o melhor valor MSE (MSE=1382,46).



(a) (b)

Figura 6.3: (a) Imagem borrada e (b) restaurada empregando $\alpha^0 = 0,1$, $\lambda = 0,1$ e o parâmetro de regularização dinâmico α^*.

Capítulo 7

Uso de GPUs (Graphics Processing Units)

7.1 Introdução

Para que seja efetivo, o desenvolvimento de programas paralelos e a estratégia de implementação paralela adotada devem levar em consideração as características da arquitetura empregada na execução dos programas paralelos. Para tanto, alguns aspectos importantes devem e/ou precisam ser observados, tais como:

(a) a natureza do problema,

(b) as características do algoritmo proposto,

(c) as adaptações dele à arquitetura paralela,

(d) a forma como a memória será acessada, distribuída e/ou compartilhada entre os processadores,

(e) a organização das unidades de processamento (topologia),

(f) o número de unidades de processamento,

(g) a forma como elas irão se comunicar,

(h) a existência (ou não) de concorrência de acesso a memória e/ou a dispositivos de I/O, etc.

Deste modo, para se tirar o máximo proveito dos recursos computacionais paralelos, faz-se necessário que a programação paralela seja muito bem empregada

nestas arquiteturas, o que faz dela um ponto-chave essencialmente importante na busca para se alcançar ganhos de desempenho.

7.2 Memória distribuída × compartilhada

Conforme a taxonomia proposta por Flynn [17], que classifica os sistemas de computadores com base na análise do número de fluxos de instruções e dados que podem ser manuseados simultaneamente pela máquina, as arquiteturas paralelas podem ser classificadas em:

(a) **SISD (*Single Instruction, Single Data*)** - categoria de máquinas que executam apenas uma única instrução por vez e essa instrução manipula apenas um único dado (p.ex. computadores seriais - Máquinas de Von Neumann).

(b) **SIMD (*Single Instruction, Multiple Data*)** - máquinas paralelas com várias unidades operacionais que executam uma mesma instrução que é aplicada, simultaneamente, à diversos dados (p.ex. processadores vetoriais/matriciais).

(c) **MIMD (*Multiple Instruction, Single Data*)** - Arquiteturas de múltiplos processadores que executam, simultaneamente, diferentes instruções sobre diferentes dados.

Particularmente, as máquinas MIMD ainda podem ser divididas em duas outras classes, dependendo da forma como a memória é implementada:

(a) **Máquinas com memória distribuída ou Multicomputadores**: neste tipo de arquitetura, os processadores não dipõem de uma memória única, a interligação entre os vários componentes é feita através de um barramento (ou um outro canal de comunicação qualquer) e a comunicação entre eles é realizada via troca de mensagens, usando por exemplo uma biblioteca de comunicação como MPI.

(b) **Máquinas com memória compartilhada ou Multiprocessadores**: máquinas cujas unidades de processamento são interligadas através de uma única memória (p.ex. placas gráficas ou GPUs - *Graphics Processing Units*).

7.3 GPGPU (*General-Purpose computation on Graphics Processing Units*

O uso de programação paralela em GPUs para a solução de problemas mais genéricos, que não só o processamento gráfico, e de uso geral é um fenômeno relati-

vamente recente, conhecido como Computação de Propósito Geral em GPU´s ou GPGPU (*General-Purpose computation on Graphics Processing Units*). Originalmente, as GPUs eram apenas blocos de hardwares, otimizados para um pequeno conjunto de operações, mas que foram se tornando cada vez mais especializados e programáveis à medida que as novas demandas iam surgindo (p.ex. shaders - conjunto de instruções para calcular efeitos de renderização, alocação de memórias para buffers de arrays, etc.). Vários projetos de pesquisa, então, procuraram desenvolver linguagens que simplificassem o uso de GPUs e, em 2006, a NVIDIA introduziu a sua arquitetura CUDA [37] e as ferramentas que fizeram da computação paralela de dados em GPUs algo bastante simples.

A GPU conecta-se à Unidade de Processamento Central ou CPU (*Central Processing Unit*) através de um barramento de dados de alta velocidade, normalmente uma PCI-Express, e possui um banco de memória próprio, de alguns megabytes (são poucos os modelos tem mais que 1Gb), disponíveis nas configurações atuais. Os dados são transferidos entre as memórias da GPU e da CPU usando a DMA (*Direct Memory Access*), que pode operar de modo concorrente entre os dois elementos, sob certas restrições.

Comparando-se a montagem de um cluster de computadores usando a biblioteca MPI, por exemplo, o custo de uma GPU é relativamente mais baixo. Apesar disso, entretanto, a forma de paralelismo utilizada pelo MPI difere daquela que é empregada numa GPU. Ao contrário do MPI, que trabalha em uma arquitetura empregando memória distribuída, a GPU emprega uma arquitetura do tipo SIMT (*Single Instruction Multiple Thread*), utilizando memória compartilhada. Além disso, a GPU foi desenvolvida para trabalhar em alto rendimento, ou seja, computação intensiva e altamente paralela.

A SIMT gerencia um conjunto de *threads* (interrupções), onde cada *thread* opera de forma independente com o seu próprio endereço de memória e estado dos registradores. A arquitetura SIMT é semelhante à organização SIMD, porém difere-se desta, pois em uma SIMD uma única instrução controla múltiplas unidades de processamento, enquanto que as instruções SIMT especificam o comportamento de desvio e a execução de uma única *thread* [37].

7.4 Resultados

Foi observado em [1] que uma boa escolha para o problema de restauração de imagens empregando o funcional de regularização de Tikhnov seria o uso da biblioteca CUDA em GPUs da Nvidia. Porém, como o CUDA não funciona de mesma forma que o MPI, o algoritmo paralelo desenvolvido por Stutz [53] não serve de base para a implementação em CUDA. Assim, baseado no algoritmo da Figura 2.4 é possível constatar dois pontos críticos no processamento: (a) o cálculo da derivadas; e (b) a solução de um sistema de equações lineares. Sendo que o cálculo das derivadas, no

Dimensão do Sistema	Arquitetura		
	CPU	GPU	Desempenho
256×256	0,27s	0,11s	2×
512×512	1,41s	0,28s	5×
1024×1024	9,31s	0,76s	12×
2048×2048	58,34s	4,78s	12×
4096×4096	424,54s	16,57s	26×
8192×8192	3279,95s	82,74s	40×

Tabela 7.1: Desempenho CPU x GPU, tempos de processamento medidos em segundos

entanto, não representa um peso tão grande quando comparado com a necessidade de resolver um sistema linear.

Dado que a escalabilidade do método de Jacobi para arquiteturas paralelas é superior quando comparada àquela de Gauss-Seidel paralelo [59], aliado ao fato o uso de Jacobi melhora a qualidade final da imagem restaurada [2], então em função desses dois fatos foram escritas duas implementações de Jacobi, uma para CPU e outra para GPU. Utilizando um modelo de sistema com dominância da diagonal, de tamanhos 256×256, 512×512, 1024×1024, 2048×2048, 4096×4096 e 8192×8192, quantidade máxima de iterações 5000, e tolerância de erro 10^{-8}, foram obtidos os resultados apresentados na Tabela 7.1.

Conforme pode-se observar, a GPU atingiu um desempenho até 40 vezes superior quando comparado com a CPU. O computador utilizado foi equipado com processador Core i7 de 1,2GHz, 3GB de RAM, e a placa de vídeo uma Nvidia 335M com 1GB de RAM dedicada, 72 núcleos, clock de 1088MHz e barramento de 128bits, na programação em CUDA foram utilizadas 16 threads por bloco e uma malha de dimensão n=16 com (n = 256; 512; 1024; 2048; 4096; 8192).

Considerações finais

Finalizando, cabe aqui resaltar que o processo de restauração de imagens em paralelo, usando GPU e memória compartilhada, encontra-se em fase de desenvolvimento, demandando mais alguns trabalhos de pesquisa. Os resultados apresentados neste Capítulo ainda são parciais e correspondem apenas à solução de uma parte do problema de restauração que é a solução de um sistema linear de equações, faltando ainda desenvolver o algoritmo paralelo que implemente o restante do processo de restauração, para ser executado em paralelo usando GPU e memória compartilhada.

Bibliografia

[1] A.G Almeida, D. Stutz, G.A.G. Cidade, A.J. Silva Neto, Uso de Graphics Processing Unit (GPU) na Restauração de Imagens de Microscopia de Força Atômica com Regularização de Tikhonov, **32** (2009), *Congresso Nacional de Matemática Aplicada e Computacional (CNMAC).*

[2] H.C. Andrews, B. R. Hunt, "Digital Image Restoration", Prentice-Hall, 1977.

[3] M.R. Banham, A.K. Katsaggelos, Digital Image Restoration, *IEEE Signal Process. Magazine*, 24-41, March 1997.

[4] A. Bevilacqua, E.L. Piccolomini, Parallel image restoration on parallel and distributed computers, *Parallel Computing*, **26** (2000), 495-506.

[5] G. Binnig; E.C.F. Quate; C.H. Gerber, Atomic Force Microscope, *Phys. Rev. Lett.*, **56** (1986), n.9, 930-933.

[6] L.M. Bregman, The Relaxation Method of Finding the Common Point of Convex Sets and its Application to the Solution of Problems in Convex Programming, *Zh. vijchisl. Mat. mat. Fiz.*, **7** (1967), n.3, 620-631.

[7] R.F. Carita Montero, N.C. Roberty, A.J. Silva Neto, Absorption Coefficient Estimation in Heterogeneous Media Using a Domain Partition Consistent with Divergent Beams, *Inverse Problems in Engineering*, **9** (2001), 587-617.

[8] A. Chan, D. Ashton, R. Lusk, W. Gropp, "Jumpshot-4 Users Guide", Mathematics and Computer Science Division, University of Chicago, Argonne National Laboratory, July, 2007.

[9] A. Chan, W. Gropp, E. Lusk, "User's Guide for MPE: Extensions for MPI Programs", Mathematics and Computer Science Division, University of Chicago, Argonne National Laboratory, 1998.

[10] G.A.G. Cidade, N.C. Roberty, A.J. Silva Neto, Reconstrução de Imagens com Regularização de Tikhonov e Determinação do Parâmetro de Regularização Ótimo, *In: II Simpósio Interdisciplinar em Tecnologia e Saúde*, **2** (1998), Rio de Janeiro, RJ, Anais...

[11] G.A.G. Cidade, A.J. Silva Neto, N.C. Roberty, A Generalized Approach for Atomic Force Microscopy Image Restoration with Bregman Distances as Tikhonov Regularization Terms, *Inverse Problems in Engineering*, **8** (2000), 457-472.

[12] G.A.G. Cidade, A.J. Silva Neto, N.C. Roberty, "Restauração de Imagens com Aplicações em Biologia e Engenharia - Problemas Inversos em Nanociência e Nanotecnologia", SBMAC (Notas em Matemática Aplicada), São Carlos, SP, **1**, 2003.

[13] N. Damera-Venkata, T.D. Kite, W.S. Geisler, B.L. Evans, Image Quality Assessment Based on a Degradation Model, *IEEE Trans. on Image Processing*, **9** (2000), n.4.

[14] S. Dongmo, M. Troyon, P. Vautrot, E. Delain, N. Bonnet, Blind Restoration Method of Scanning Tunneling and Atomic Force Microscopy Images, *J. Vac. Sci. Technol.*, **B 14(2)** (1996), 1552-1556.

[15] A.M. Eskicioglu, P.S. Fisher, Image Quality Measures and their Performance, *IEEE Trans on Communication*, **43** (1995), 2959-2965.

[16] A.M. Eskicioglu, Quality Measurement for Monochrome Compressed Images in the Past 25 Years. *In: IEEE International Conference on Acoustics, Speech, and Signal Processing*, **4** (2000), ICASSP '00, Proceedings..., 1907-1910.

[17] M.J. Flynn, Very High Speed Computing Systems, *Proc. IEEE*, **54** (1966), 1901-1909.

[18] I. Foster, "Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering", Addison Wesley, 1994.

[19] V. Furtado, Avaliação do uso do Funcional de Tikhonov com uma Família de Funções de Regularização a um Parâmetro para a Restauração de Imagens Biológicas. 2002. *Dissertação (mestrado)*, Pós-graduação em Modelagem Computacional, Instituto Politécnico/UERJ, Nova Friburgo.

[20] R.C. Gonzalez, P. Wintz, "Digital Image Processing", Addison-Wesley, Boston, 2ed., 2001.

[21] R.C. Gonzalez, R.E. Woods, "Processamento de Imagens Digitais", 3ed., Pearson, 2010.

[22] P.K. Hansma, V.B. Ellings, O. Marti, C.E. Bracker, Scanning Tunneling Microscopy and Atomic Force Microscopy: Application to Biology and Technology. *Science*, **242** (1988), 209-242.

[23] F.C. Jeng, J.W. Woods, Inhomogeneous Gaussian Image Models for Estimation and Restoration, *IEEE Trans. Acoust.*, Speech, Signal Proc., **36** (1988), n.8, 1305-1312.

[24] M.G. Kang, A.K. Katsaggelos, General Choice of the Regularization Functional in Regularized Image Restoration. *IEEE Trans. Image Process.*, **4** (1995), n.5, 594-602.

[25] N. B. Karayiannis, A. N. Venetsanopoulos, Regularization Theory in Image Restoration: The Regularizing Operator Approach, *Optical Engineenring*, **28(7)** (1989), 761-780.

[26] A.K. Katsaggelos, Iterative Image Restoration Algorithm, *Optical Engineering Special issue on Visual Comunications and Image Processing*, **28** (1989), n.7, 735-748.

[27] A.K. Katsaggelos, M.G. Kang, Iterative evaluation of the regularization parameter in regularized image restoration, *Journal of Visual Communication and Image Representation*, **3** (1992), n.4, 446-455.

[28] J. Ke, M. Burtscher, E. Speight, Reducing Communication Time through Message Prefetching. *In: The 2005 International Conference on Parallel and Distributed Processing Techniques and Applications*, 557-563, 2005.

[29] A.C. Kokaram, N. Persad, J. Lasenby, W.J. Fitzgerald, A. Mckinnon, M. Welland, Restoration of Images from Scanning-Tunneling Microscope, *Applied Optics*, **34** (1995), n.23, 5121-5132.

[30] E. Kreyszig, "Advanced Engeneering Mathematics", 3. ed, Wiley International, New York, 1972.

[31] V. Kumar, A. Grama, A. Gupta, G. Karypis, "Introduction to Parallel Computing: Design and Analysis of Algorithm", Benjamin/Cummings, California, 1994.

[32] J.S. Lim, "Two-Dimensional Signal and Image Processing", Prentice-Hall, 1990.

[33] N. Mackworth, A. Morandi, The Gaze Selects Informative Details within Pictures, *Perception and Psychophysics*, **2** (1967), 547-552.

[34] J. Ming, S. Shui-Fa, D. Fangmin, L. Bangjun, Image Quality Evaluation Based on HVS and Energy Weighted Subband of Wavelet Transform, *IEEE Comp. Soc.*, (2008), 491-495, Proceedings. Second International Symposium on Intelligent Information Technology Application.

[35] MPI Forum, MPI: A Message-Passing Interface Standard, *The International Journal of Supercomputer Applications and High Performance Computing*, **8(3/4)** (1994), 165-414.

[36] J.P. Noonan, P. Natarajan, A General Formulation for Iterative Restoration Methods, *IEEE Trans. on Signal Process.*, **45** (1997), n.10, 2590-2593.

[37] NVIDIA Corporation, "NVIDIA CUDA Programming Guide", 2010.

[38] S. Oraintara, W.C. Karl, D.A. Castanon, T.Q. Nguyen, A method for choosing the regularization parameter in generalized Tihkonov regularized linear inverse problems, *IEEE Image Process*,**1** (2000), Proc. Int. Conference on, n. 2000, 93-96.

[39] P.S. Pacheco, "Parallel Programing with MPI", M. Kaufmann, San Francisco, 1997.

[40] J.K. Paik, A.K. Katsaggelos, Parallel Iterative Image Restoration Algorithms, *IEEE Circuits and Systems*, **1** (1989), Proceedings of the 32nd Midwest Symposium on, 63-66.

[41] H. Pedrini, W.R. Schwartz, "Análise de Imagens Digitais: Princípios, Algoritmos e Aplicações", Thompson, São Paulo, 2008.

[42] W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, "Numerical Recipes in Fortran 77 - The Art of Scientific Computing". 2. ed., Cambridge Univ. Press, New York, **1**, 1996.

[43] R.C. Pueter, T.R. Gosnell, A. Yahil, Digital Image Reconstruction: Deblurring and Denoising, *Annu. Rev. Astrophys.*, **43** (2005), 139-194.

[44] S.J. Reeves, R.M. Mersereau, Optimal estimations of the regularization parameter and stabilizing functional for regularized image restoration, *Optical Enginnering*, **29** (1990), n.5.

[45] K.V. Romualdo, Proposição e Avaliação de Indicadores de Desempenho para Algoritmos de Restauração de Imagens, 2006, *Dissertação (mestrado)*, Pós-graduação em Modelagem Computacional, Instituto Politécnico de Nova Friburgo/UERJ, Nova Friburgo.

[46] A.C. Santos, Restauração de Imagens de Microscopia Óptica com o Funcional de Regularização de Tikhonov Visando a Avaliação de Superfícies Metálicas. 2008. Dissertação (mestrado), Pós-graduação em Modelagem Computacional, Instituto Politécnico/UERJ, Nova Friburgo.

[47] E. Schlemer, "Sistemas Distribuídos", Gravataí, RS, 2003.

[48] C.E. Shannon, A Mathematical Theory of Communication, *The Bell System Technical Journal*, **27** (1948), 379-423, 623-656.

[49] A.J. Silva Neto, F.D. Moura Neto, "Problemas Inversos - Conceitos Fundamentais e Aplicações", EdUerj, Rio de Janeiro, 2005.

[50] S. Smith, "The Scientific and Engineer´s Guide to Digital Signal Processing", 2. ed., California Tech. Pub. 1999. Disponível em: http://www.dspguide.com/. Acesso em 12/04/2011.

[51] M. Snir, S. Otto, S. Huss-Lederman, D.E. Walker, J. Dongarra, "MPI: The Complete Reference", Cambridge: The MIT Press, 1996.

[52] D. Stutz, Restauração de Imagens em Escala Nanométrica com Funcional de Regularização e Tikhonov e Computação Paralela. 2004. Dissertação (mestrado), Pós-Graduação em Modelagem Computacional, Instituto Politécnico/UERJ, Nova Friburgo.

[53] D. Stutz, Estratégias de Computação Paralela para a Restauração de Imagens com o Funcional de Regularização de Tikhonov. 2009. Tese (doutorado), Pós-Graduação em Modelagem Computacional, Instituto Politécnico/UERJ, Nova Friburgo.

[54] D. Stutz, A.J. Silva Neto, R.C. Farias, Information Weighted Mean Square Error (IWMSE): Uma medida de comparação de imagens baseada na percepção, *In: Encontro de Modelagem Computacional*, 10, Anais. Nova Friburgo, RJ, 2007.

[55] A.N. Tikhonov, V.Y. Arsenin, "Solutions of Ill-Posed Problems", Wiley, New York. 1977.

[56] D. Tompa, J. Morton, E. Jernigan, Perceptually Based Image Comparison, *IEEE Image Processing*, **1** (2000), Proceedings... 2000 International Conference on., 489-492.

[57] T. Topper, M. Jernigan, On the Informativeness of Edges, *IEEE Internat. Conf. on Systems, Man and Cybernetics*, **3** (1989), 909-914.

[58] T. Topper, Selection Mechanisms in Human and Machine Vision, *PhD thesis*, University of Waterloo, Waterloo Ontario Canada, 1991.

[59] J.N. Tsitsiklis, A Comparison of Jacobi and Gauss-Seidel Parallel Iterations, *Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, Massachusetts*, **02139** (1988).

[60] R. Wallis, *Approach to space variant restoration and enhancement of images*, In: Proc. Symp. Current Math. (1976), Problems in Image Science, Naval Postgraduate School, Monterey, California.

[61] A.D. Weisman, E.R. Dougherty, H.A Mizes, R.J.D. Miller, Nonlinear Digital Filtering of Scanning-Probe-Microscopy Images by Morphological Pseudo-Convolutions, *J. Appl. Phys.*, **71** (1992), n.4, 1565-1578.

[62] A. Wong, M. Vogel, Resolution-dependent Information Measures for Image Analysis, *IEEE Trans. on Systems, Man and Cybernetics*, **SMC-7(1)** (1977), 49-61.

Índice