

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ELETRÔNICA
E DE COMPUTAÇÃO

**Projeto de um algoritmo de restauração de
imagens usando processamento paralelo**

Autor: _____
Luciana Habib Abi Ghosn

Orientador _____
Eugenius Kaszkurewicz

Examinador _____
Antônio Cláudio G. De Sousa

Examinador _____
Sérgio Barbosa Villas Boas

Examinador _____
Amit Bhaya

DEL

Mai de 2005

Agradecimentos

Gostaria de agradecer a colaboração das pessoas sem as quais a realização deste projeto não seria possível.

Ao meu Orientador Eugenius Kaszkurewicz, pela sua atenção, compreensão e motivação, assim como meu co-orientador, Antônio Cláudio G. De Sousa.

Aos meus companheiros de curso, que em muito contribuíram para a minha formação.

Resumo

Este projeto tem como objetivo viabilizar a restauração de imagens que sofreram efeito de degradação, usando para isso, implementações com redes neurais e algoritmos paralelos implementados em computadores paralelos. ___

A necessidade de se criar um algoritmo que pudesse restaurar imagens que sofreram degradações em um ambiente de computação paralela, se deve ao alto custo computacional dessas restaurações num ambiente seqüencial para grandes dimensões da imagem.

A quantidade de dados a serem processados é elevada, o que torna o processamento seqüencial excessivamente lento em ambiente computacional disponível. Assim o algoritmo paralelo mostra-se como uma possível solução.

As degradações de uma imagem podem ser de diversos tipos: borrão (fora de foco), deslocamento (moção), ausência ou excesso de iluminação e ruído. Como a tecnologia de imagem não é perfeita, toda imagem gravada constitui-se em uma imagem distorcida de algum modo.

Neste trabalho consideramos que as distorções sofridas pelas imagens são invariantes no espaço, todos os pixels da imagem sofreram o mesmo tipo de distorção e podem ser descritas por modelos lineares com perturbações os quais podem ser resolvidos usando o método iterativo de Jacobi, que na realidade constitui-se uma especialização de redes neurais.

A preferência pelos métodos iterativos aos diretos se deve à facilidade e conveniência de paralelização.

O texto presente encontra-se dividido em 7 capítulos. No capítulo 1 é apresentada a motivação, um breve histórico e a descrição do projeto. No capítulo 2, é explicada detalhadamente a fase de análise do problema apresentado. São apresentados a equação integral que descreve o modelo contínuo e linear de formação da imagem, e a discretização da equação recaindo numa equação que descreve um modelo de sistema linear, a seguir são introduzidas as classes de métodos iterativos que podem solucionar o problema e que atendem ao critério de

convergência e as variações da matriz degradação de acordo com o tipo de distorção sofrida pela imagem.

No capítulo 3 é feito um estudo da viabilidade da solução seqüencial do sistema linear de recuperação de imagem usando o método iterativo de Jacobi, abrangendo o processo de discretização da imagem e modelagem da solução seqüencial.

No capítulo 4 é dada uma introdução ao processamento paralelo e à biblioteca MPI que foi utilizada. Além disso, trata da análise de requisitos, isto é, considerações de desempenho que devem ser levadas em conta num programa paralelo.

No capítulo 5 é apresentada e analisada a proposta de uma solução paralela para o sistema linear de recuperação de imagens, utilizando o método iterativo de Jacobi.

No capítulo 6 são apresentados os resultados computacionais com as curvas de desempenho do programa paralelo quando aplicado a imagens submetidas a diferentes tipos degradação.

No capítulo 7 são apresentadas as conclusões relativas ao trabalho bem como sugestões de continuidade do projeto.

Palavras-chave

Restauração de imagens

Distorção de imagens

Processamento paralelo

Método de Jacobi

MPI

SUMÁRIO

Capítulo 1 – Introdução _____	01
1.1 – Motivação _____	01
1.2 – Histórico _____	01
1.3 – Descrição do projeto _____	02
Capítulo 2 – Embasamento teórico _____	04
2.1 – Restauração de imagens _____	04
2.2 – Causas da distorção _____	04
2.3 – Modelo matemático de formação da imagem _____	07
2.4 – Associação feita do valor dos pixels das imagens às respectivas cores _____	10
2.5 – Diferentes configurações da matriz distorção H _____	10
2.6 – Exemplo ilustrativo _____	19
2.7 – Diferentes matrizes de distorção H _____	24
2.8 – Métodos de recuperação de imagens distorcidas _____	32
2.8.1 – Métodos Diretos _____	32
2.8.2 – Métodos Iterativos _____	33
Capítulo 3 – Análise da solução seqüencial do sistema linear de recuperação da imagem usando o método iterativo de Jacobi _____	41
3.1 – Discretização da imagem _____	41
3.2 – Modelagem da solução seqüencial _____	42
3.3 – Cálculo do erro _____	42
3.4 – Algoritmo seqüencial _____	43
3.3 – Justificativa do uso de soluções paralelas _____	46

Capítulo 4 – Processamento paralelo e medida de desempenho	47
4.1 – Introdução ao processamento paralelo	47
4.1.1 – Paralelismo	47
4.1.2 – Necessidade de Processamento mais Rápido	47
4.1.3 – MPI (“Message Passing Interface”)	48
4.2 – Processamento paralelo	49
4.3 – Medida do desempenho do programa paralelo	49
4.3.1 – Balanceamento de carga	49
4.3.2 – Considerações de desempenho	50
4.3.3 – “Speed-up” e eficiência	51
4.3.4 – Lei de Amdahl – medidas de desempenho	53
4.3.5 – Escalonamento do programa	55
4.3.6 – Escalabilidade	56
4.3.7 – Tempo de Computação	56
4.3.8 – Tempo Ocioso	56
 Capítulo 5 – Proposta de uma solução paralela do sistema linear de recuperação de imagem usando o método iterativo de Jacobi	 57
5.1 – Modelagem Paralela	57
5.1.1 – Desafios para resolução em uma máquina paralela para diferentes dimensões da imagem distorcida	62
5.2 – Cálculo do Erro	63
5.3 – Etapas do Algoritmo Paralelo	66
5.4 – Especificações da máquina cluster utilizada e dos processadores	34

Capítulo 6 – Análise dos resultados computacionais _____	68
6.1 – Aplicação do algoritmo a diferentes imagens distorcidas __	68
6.2 – Curvas de desempenho e verificação do controle de qualidade_____	72
Capítulo 7 – Conclusão _____	76
Referências Bibliográficas _____	79
Apêndice A – Processamento Paralelo _____	81
A.1 – Introdução _____	81
A.2 – Objetivos do Paralelismo _____	81
A.3 – Principais Áreas de Aplicação _____	81
Apêndice B – MPI (“Message Passing Interface”) _____	83
B.1 – O que é MPI? _____	83
B.1 – Características _____	83
B.2 – Histórico _____	84
B.3 – Conceitos e Definições _____	85
B.4 – Rotinas do MPI de Comunicação “Point-to-Point ” _____	86
B.4 – Rotinas do MPI de Comunicação Coletiva _____	88
Apêndice C – Algoritmo Seqüencial _____	102
Apêndice D – Algoritmo Paralelo _____	113

Lista de Figuras

2.1 Esquema do sistema de formação de imagem.....	7
2.2 Associação de valores de pixels da imagem a cores feita pelo Matlab.....	10
2.3 Imagem da Lena em escala de cinza 256X256.....	13
2.4 Imagem da Lena corrompida pela matriz deslocamento H_1	13
2.5 Imagem da Lena corrompida pela matriz H_3	13
2.6 Imagem da Lena corrompida pela matriz H_4	13
2.7 Formação da imagem de objetos pela penetração de radiação.....	14
2.8 Fonte de erro na formação de imagens de radiografias.....	16
2.9 Imagem de uma radiografia em escala cinza.	17
2.10 Imagem da radiografia corrompida pela matriz distorção H_5 com $n=1.2$	17
2.11 Imagem de uma impressão digital em escala de cinza.....	17
2.12 Imagem da impressão digital corrompida pela matriz distorção H_5 com $n=1.4$	18
2.13 Imagem da Lena corrompida pela matriz distorção H_5 com $n=1.2$	18
2.14 Imagem da Lena rotacionada pela matriz H_6	18
2.15 Imagem utilizada como exemplo ilustrativo.....	19
2.16 Imagem original.....	20
2.17 Imagem original deslocada de sete pixels para a esquerda.....	22
2.18 Imagem resultante da sobreposição das figuras 2.16 e 2.17.....	22
2.19 Imagem original deslocada de dez pixels para a esquerda e escurecida por um fator 0.3.....	22
2.20 Imagem original deslocada de dez pixels para a direita e escurecida por um fator 0.3.....	23
2.21 Imagem resultante da sobreposição das figuras 2.19 e 2.20.....	24
2.22 Figura corrompida pela matriz de Toeplitz H_5 com $n=1.2$	25
2.23 Imagem original do pássaro (256X256).....	26
2.24 Imagem deslocada de 7 pixels para a esquerda.....	27
2.25 Resultante da sobreposição das Figuras 2.23 e 2.24 (com as sete últimas colunas zeradas)..	28
2.26 Resultante da sobreposição das Figuras 2.23 e 2.24.....	28
2.27 Resultante da distorção causada pela matriz H_4 na Figura 2.23.....	29
2.28 Imagem 2.23 que foi corrompida pela matriz Toeplitz H_5 com $n=1.1$	30
2.29 Imagem 2.23 que foi corrompida pela matriz Toeplitz H_5 com $n=1.2$	31
2.30 Imagem 2.23 que foi corrompida pela matriz Toeplitz H_5 com $n=1.4$	31
2.31 Imagem 2.23 que foi corrompida pela matriz Toeplitz H_5 com $n=1.8$	31
2.32 Imagem 2.23 que foi corrompida pela matriz Toeplitz H_5 com $n=2.0$	31
2.33 Imagem 2.22 que foi corrompida pela matriz Toeplitz H_5 com $n=3.0$	32
3.1 Fluxograma de funcionamento do algoritmo seqüencial.....	44
3.2 Loop principal do algoritmo seqüencial feito em linguagem C.....	45

4.1 Típico gráfico de “speed-up”.....	52
4.2 Típico gráfico de eficiência.....	53
5.1 Particionamento das matrizes.....	59
5.2 Cálculo da expressão iterativa de Jacobi, $f_{aux} = D_p^{-1}N_p f_{antigo} + D_p^{-1}g$, em cada processador.....	60
5.3 Cálculo da expressão que define o critério de parada: $\ z_p\ = \ f_{novop}\ - \ f_{antigop}\ $ e da norma infinito dos subvetores z_p em cada processador.....	60
5.4 Envio das normas infinito parciais calculadas em cada processador para o mestre.....	62
5.5 Fluxograma das etapas do processo de recuperação de imagens.....	65
5.6 Cluster utilizado para executar o programa paralelo de restauração de imagens distorcidas.....	66
6.1 Imagem distorcida que tentaremos restaurar.....	68
6.2 Imagem restaurada utilizando a matriz H_1	70
6.3 Imagem restaurada utilizando a matriz H_2	70
6.4 Imagem restaurada utilizando a matriz H_3	71
6.5 Imagem restaurada utilizando a matriz H_4	71
6.6 Imagem restaurada utilizando a matriz H_5	71
6.7 Imagem restaurada utilizando a matriz H_6	71
6.8 Imagem restaurada utilizando a matriz H_7	73
6.9 Imagem inicialmente distorcida.....	73
6.10 Curva de “speed-up”.....	74
6.11 Curva de eficiência.....	75
7.1 Cálculo de $f_{novo} = D^{-1}N*f_{antigo} + D^{-1}g$, particionando as matrizes D e N não somente por linhas mas também por colunas e particionando os vetores f_{antigo} e g.....	78
B.6.1 Funcionamento da rotina Broadcast.....	94
B.6.2 Funcionamento da rotina Scatter.....	95
B.6.3 Funcionamento da rotina Gather.....	96
B.6.4 Processos sendo enviados para o buffer.....	97
B.6.5 Buffer reenviando os processos.....	97
B.6.6 Funcionamento da rotina All to All.....	99
B.6.7 Exemplo de rotina Reduce.....	101
B.6.8 Exemplo de rotina Reduce.....	101

Lista de Tabelas

5.1 Principais características de cada um dos nós que compõem a máquina cluster.....	67
6.1 Tabela de “speed-up”.....	73
6.2 Tabela de eficiência.....	73
B.6.1 Definições do Mpi em C e Fortran.....	100

Capítulo 1 – Introdução

1.1 – Motivação

Este projeto surgiu em função da demora em tempo de processamento, da implementação de uma solução em ambiente seqüencial existente, de um algoritmo que pudesse restaurar imagens que sofreram degradações. A demora de processamento está relacionada às dimensões dos sistemas a serem processados, que requerem velocidade de cálculo e capacidade elevada de memória e recursos. As degradações mencionadas podem ser de diversos tipos: borrão (fora de foco), ausência ou excesso de iluminação, deslocamento (moção) e ruído. Como a tecnologia de imagem não é perfeita, toda imagem gravada é uma imagem distorcida de algum modo.

Como estado da arte em métodos de restauração de imagens temos variadas implementações utilizando redes neurais. Como existe uma correspondência entre métodos iterativos e redes neurais, neste trabalho, optou-se por utilizar métodos iterativos diretamente para recuperar imagens distorcidas.

1.2 – Histórico

Neste trabalho estudaremos restaurações de imagens que sofreram distorções invariantes no espaço utilizando o método iterativo de Jacobi que consiste, em uma especialização de redes neurais. Portanto, apresentaremos abaixo o que foi descoberto e estudado até o presente momento sobre implementações neurais de métodos de restauração iterativa:

- Zhou, em seu artigo [15], introduz a restauração de imagens usando uma rede neural de Hopfield, na qual os pixels da imagem são codificados por uma soma de um conjunto de elementos binários (0 ou 1). Neste tipo de esquema de representação, proposto por Takeda em seu artigo [16], e

referenciado como bit-density coding (BDC), várias configurações diferentes podem ser representadas pela mesma solução.

- Paik em seu livro [13] sugeriu uma rede com neurônios de valores discretos e com conexões diferentes de zero, com dois esquemas de atualização (um seqüencial e o outro paralelo) que já provou convergir. Mais recentemente, os mesmos autores J. Paik e A. Katsaggelos estenderam seu trabalho com a publicação de um artigo [11] propondo outra rede de Hopfield modificada com uma lei de atualização diferente, baseada no fato que diversos algoritmos são implementados e estudados em termos de propriedades de convergência.
- Abbiss, seguindo uma outra direção, em seu artigo [12] e Yeh em seu livro [11], introduziram redes do tipo Hopfield, de elementos binários com uma lei de atualização modificada utilizando dois níveis limites ao invés de somente um. Esta modificação garante o declínio da energia, mesmo com conexões (“pesos da rede”) negativas (como é o caso neste caso), ao custo de elementos ligeiramente mais complexos.
- Abbiss ainda em sua publicação mencionada acima [12] sugere uma rede de elementos graduados. Esta estrutura implementa um esquema de restauração com um prévio conhecimento das propriedades de convergência (o algoritmo de Papoulis [17]). As questões de precisão numérica e aceleração de convergência são levantadas, porém não qualificadas.

1.3 – Descrição do projeto

O projeto consiste de restaurar imagens que sofreram distorções de diversos tipos usando para isso métodos iterativos para sistemas lineares num algoritmo paralelo, implementado em computadores paralelos.

O sistema de formação de imagens consiste num sistema linear que pode ser resolvido por métodos diretos ou iterativos. Porém optaremos pela solução usando

este último, já que os iterativos nos apresentam diversas vantagens como serem facilmente paralelizáveis.

O processamento de imagens envolve a manipulação de uma quantidade muito grande de dados, tanto durante a discretização da imagem, quanto durante a recuperação através da resolução de um sistema linear usando métodos iterativos. Assim o algoritmo paralelo mostra-se como uma solução conveniente para grandes dimensões da imagem a ser processada, no ambiente de computação disponível.

Dentre os métodos iterativos optamos pelo de Jacobi, pois este é mais fácil de ser resolvido usando um algoritmo paralelo.

Para restaurar imagens distorcidas primeiramente ela será discretizada numa matriz, usando para isso o software MATLAB, e logo depois armazenada num arquivo auxiliar. O programa paralelo fará a leitura deste arquivo e por meio de um arranjo lexicográfico chegará num vetor e de posse de uma provável matriz distorção H , resolverá o sistema linear de recuperação de imagens utilizando o método iterativo de Jacobi, restaurando assim a imagem original.

Capítulo 2 – Embasamento teórico

2.1 – Restauração de imagens

A obtenção da imagem original, a partir de uma imagem degradada e algum conhecimento dos fatores de degradação é chamado de *restauração de imagens*. O problema de restaurar uma imagem original, quando é fornecida a distorcida, com ou sem conhecimento da função degradação ou grau e tipo de ruído presente é um problema que pode ser resolvido de diferentes maneiras. Em todos os casos utilizados, é gerado um conjunto de equações simultâneas, que estão muito distantes de serem resolvidas analiticamente. Aproximações comuns para este problema podem ser divididas em duas categorias, filtragem inversa ou técnicas relativas de transformação, e técnicas algébricas. Sendo esta última explorada neste trabalho.

2.2 – Causas da distorção

Imagens são adquiridas sob uma grande variedade de circunstâncias. Como a tecnologia de imagens está avançando rapidamente, nosso interesse em registros incomuns aumenta também. Nós geralmente forçamos a tecnologia de adquirir imagens ao seu limite mais alto. Por essa razão nós sempre teremos que lidar com imagens que foram objeto de algum tipo de degradação.

Como nossa tecnologia de imagens não é perfeita, toda imagem registrada é distorcida de algum modo. Todo sistema de aquisição de imagens tem um limite para sua resolução e a velocidade na qual imagens podem ser registradas. Na maioria das vezes, os problemas de resolução finita e velocidade não são cruciais para as aplicações das imagens produzidas, mas há sempre casos nos quais isto não é verdade.

A restauração de imagem consiste num processo de recuperar imagens que foram gravadas em presença de uma ou mais fontes de degradação.

Existe um grande número de possíveis degradações que uma imagem pode sofrer. Degradações comuns são: borrão (fora de foco), deslocamento (moção), ausência ou excesso de iluminação e ruído.

Borrão (fora de foco) pode ser causado quando o objeto, na imagem, está fora do campo de profundidade, foco, em algum momento durante a aquisição. Por exemplo, uma árvore em primeiro plano, deve resultar num borrão quando temos que acertar a camera com a imagem num segundo plano de uma montanha. Um objeto fora de foco perde algum detalhe em pequena escala e o processo de borrão pode ser modelado como se as componentes de alta frequência tivessem sido atenuadas de alguma maneira na imagem.

Este tipo de distorção se dá também quando o sistema de lentes se encontra com algumas impurezas que influenciaram na captura da imagem, impedindo assim a reprodução exata da imagem original.

Outra distorção de imagem, muito comum encontrada é a tipo deslocamento. Esta pode ser causada quando um objeto se move em relação à câmera durante a captura da imagem, assim como a imagem de um carro se movendo ao longo de uma pista. Na imagem resultante, o objeto parece ter sido “espalhado” ou ter “deslizado” em uma direção. Essa distorção pode ser resultante também quando a câmera se move durante a aquisição da imagem.

As imagens adquiridas são também podem ter sido contaminadas por ruído aleatório, cuja natureza depende do problema particular que temos em mãos.

Ruído é geralmente uma distorção devida mais ao sistema de imagens do que à cena registrada. Resulta em variações aleatórias nos valores dos pixels da imagem. Este pode ser causado pelo próprio sistema de imagens (por impurezas presentes nas lentes da câmera) ou meio de transmissão ou gravação.

Isso é muito comum principalmente quando há transferência de dados (no caso imagens) de um transmissor para um receptor, havendo adição de ruído aos dados e com isso distorcendo a imagem.

Um exemplo onde duas fontes de distorção se fazem presentes é no caso da captura de imagens via satélite e sonares. No momento em que é feita a aquisição da imagem a poluição atmosférica, distorção do ar, poeira espacial e a névoa branca

já a distorcem. Posteriormente para transmiti-la a um receptor, há adição de ruído que contribui para uma maior distorção.

Às vezes as definições não são claras como no caso onde a imagem é distorcida por turbulência atmosférica, assim como névoa quente. Neste caso, a imagem aparece borrada porque a distorção atmosférica causou seções no objeto gravado que se movem de maneira randômica. Esta distorção pode ser descrita como borrão causado por deslocamento randômico, mas pode, na maioria das vezes, ser modelada como um processo de borrão padrão.

Alguns tipos de distorção de imagens, assim como alguns tipos de degradação atmosférica podem ser melhor descritas como distorções na fase do sinal.

O problema também é relevante numa variedade de aplicações: em astronomia, onde as imagens são degradadas por turbulência atmosférica, aberrações do sistema óptico e movimentação relativa entre a câmera e o objeto a ser capturado. Em medicina, no reconhecimento arterial, onde a radiografia de imagens é de baixo contraste devido à natureza dos sistemas de imagem que usam raio-X e para melhorar a resolução espacial das imagens resultantes da tomografia computadorizada, onde a degradação é devido a natureza divergente e variante no espaço do raio-X também utilizado nestes sistemas.

Qualquer que seja o processo de degradação ele se encaixa em duas categorias:

- Algumas distorções podem ser descritas como invariantes no espaço. Neste caso todos os pixels sofreram a mesma forma de distorção. Este é geralmente causado por problemas com o sistema de aquisição de imagens assim como distorções em sistemas ópticos, ausência de foco global ou movimentação da câmera.
- Geralmente as distorções são o que denominamos variante no espaço. Neste caso, a degradação sofrida por cada pixel da imagem depende da sua localização na imagem. Esta degradação é geralmente causada por fatores internos, como a distorções no

sistema óptico, ou por fatores externos, como movimentação de objetos em sentido e velocidade aleatórios.

Além do que já foi dito, degradações de imagens podem ser descritas como lineares ou não-lineares [1]. Neste trabalho, consideraremos somente as distorções descritas por modelos lineares. Para estas, um modelo matemático adequado é apresentado a seguir.

2.3 – Modelo matemático de formação e distorção da imagem

Matematicamente o modelo linear contínuo de formação da imagem pode ser representado pela seguinte equação integral de *Fredholm* [1]:

$$g(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} h(x, \xi, y, \eta) f(\xi, \eta) d\xi d\eta + \varepsilon(x, y) \quad (1)$$

onde $f(\xi, \eta)$ e $g(x, y)$ são funções que descrevem o objeto (no plano de coordenadas (ξ, η) , que é referido como plano do objeto) e a imagem gravada (no plano de coordenadas (x, y) , que é referido como plano da imagem, onde esta é gravada), $h(x, \xi, y, \eta)$ é a forma analítica da função de degradação e $\varepsilon(x, y)$ representa o ruído presente na imagem. No esquema abaixo temos o objeto e a imagem em seus respectivos sistemas de coordenadas.

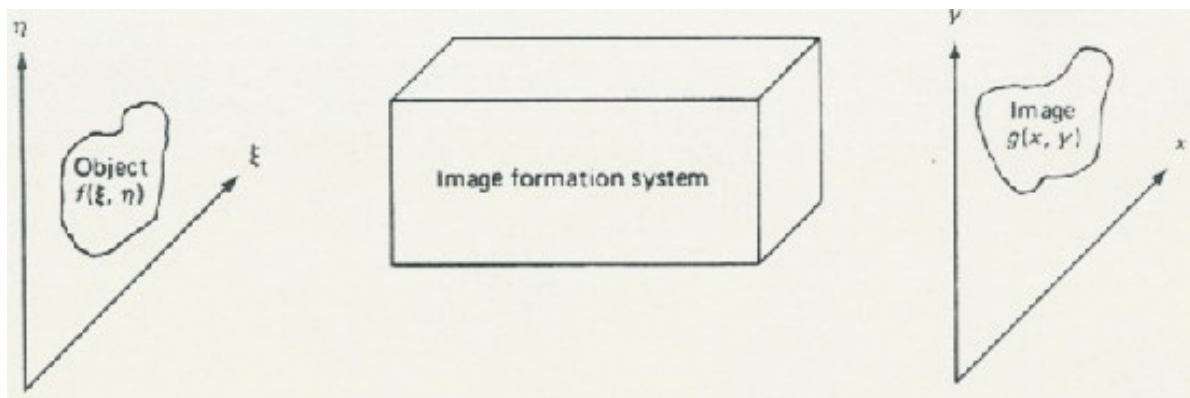


Figura 2.1: Esquema do sistema de formação de imagem

O conhecimento de $\varepsilon(\mathbf{x},\mathbf{y})$ é geralmente limitado pela informação estatística. Em restauração de imagens, a função de degradação $\mathbf{h}(\mathbf{x},\xi,\mathbf{y},\eta)$ é conhecida *a priori* e representa a resposta do sistema a um impulso unitário nas coordenadas (x,y) bidimensional (2D). Como em ótica o impulso é um ponto de luz, a função h é também chamada de “Point Spread Function” (PSF). A forma analítica de PSF assume um papel fundamental no processo de restauração de imagens e na escolha de um método para resolver a equação (1).

Uma grande variedade de métodos, ambos estatísticos e determinísticos, para resolver o problema pode ser encontrada na literatura. Métodos determinísticos são usualmente mais precisos que os estatísticos, mas na sua formulação geral eles são de grande complexidade e de tempo computacional proibitivo para computadores escalares. Como por exemplo, a restauração de uma imagem com 512×512 pixels requer a resolução de um sistema linear com matriz de coeficientes de dimensão 512^2 .

O objetivo deste trabalho é apresentar algoritmos paralelos para a solução de problemas de restauração de imagem invariantes no espaço, usando métodos determinísticos.

Arquiteturas paralelas apresentaram um grande desenvolvimento nos anos recentes e diferentes abordagens foram propostas. Neste trabalho exploramos o paralelismo da informação, baseado na decomposição da imagem de maneira uniforme para todos os processadores.

O modelo de formação de imagem descrito representado acima, no caso discreto é :

$$g(x, y) = \sum_{\alpha}^N \sum_{\beta}^M f(\xi, \eta) h(x, y; \xi, \eta) + \varepsilon(x, y) \quad (2)$$

Se $\mathbf{h}(\mathbf{x},\xi,\mathbf{y},\eta)$ é uma função linear então a equação (1) pode ser reescrita como um arranjo lexicográfico de $\mathbf{g}(\mathbf{x},\mathbf{y})$, $\mathbf{f}(\mathbf{x},\mathbf{y})$ e $\varepsilon(\mathbf{x},\mathbf{y})$ em vetores coluna de dimensão \mathbf{NM} . Para representar de maneira lexicográfica uma imagem, podemos simplesmente rearranjar cada pixel da imagem linha por linha e empilhá-los um após o outro, formando um único vetor coluna. Por exemplo, assumamos que a imagem $\mathbf{f}(\mathbf{x},\mathbf{y})$ seja representada pela seguinte matriz:

$$f(x,y) = \begin{bmatrix} 11 & 12 & 13 & 14 \\ 21 & 22 & 23 & 24 \\ 31 & 32 & 33 & 34 \\ 41 & 42 & 43 & 44 \end{bmatrix}$$

Após o arranjo lexicográfico resulta no seguinte vetor coluna:

$$f = [11 \ 12 \ 13 \ 14 \ 15 \ 21 \ 22 \ 23 \ 24 \ 31 \ 32 \ 33 \ 34 \ 41 \ 42 \ 43 \ 44]^T$$

Se formos coerentes e ordenarmos $\mathbf{g}(x,y)$, $\mathbf{f}(x,y)$ e $\boldsymbol{\varepsilon}(x,y)$ da mesma maneira, podemos reescrever (2) como uma operação de matrizes:

$$\mathbf{g} = \mathbf{Hf} + \boldsymbol{\varepsilon}, \quad (3)$$

onde \mathbf{g} e \mathbf{f} consistem em arranjos lexicográficos respectivamente, dos vetores originais e degradados da imagem, $\boldsymbol{\varepsilon}$ é uma componente de ruído aditivo e \mathbf{H} (matriz degradação da imagem) é uma matriz operador cujos elementos são um arranjo dos elementos de $\mathbf{h}(x,\xi,y,\eta)$ de tal maneira que a multiplicação das matrizes \mathbf{f} e \mathbf{H} executa a mesma operação que convoluir $\mathbf{f}(x,y)$ com $\mathbf{h}(x,\xi,y,\eta)$.

Analisando essa expressão observamos que recaímos numa equação $\mathbf{Ax} + \boldsymbol{\delta} = \mathbf{b}$, que nada mais é que a forma geral de um sistema linear escrito na forma matricial ($\mathbf{Ax}=\mathbf{b}$) acrescida por um fator $\boldsymbol{\delta}$, que neste trabalho iremos desconsiderar. Consideraremos somente distorções presentes no momento em que é feita a aquisição da imagem (representada pela matriz \mathbf{H}), desconsiderando as causadas pelo ruído.

2.4 – Associação feita do valor dos pixels das imagens às respectivas cores

Neste trabalho foram utilizadas somente como exemplo imagens em preto e branco. Para discretização dessas imagens utilizamos o software Matlab. Este carrega a imagem numa matriz de pixels cujos elementos são valores associados às cores dos pixels que compõem a imagem. Abaixo temos um esquema que exhibe a associação de cores aos respectivos valores, em ponto flutuante, feito pelo “software”:

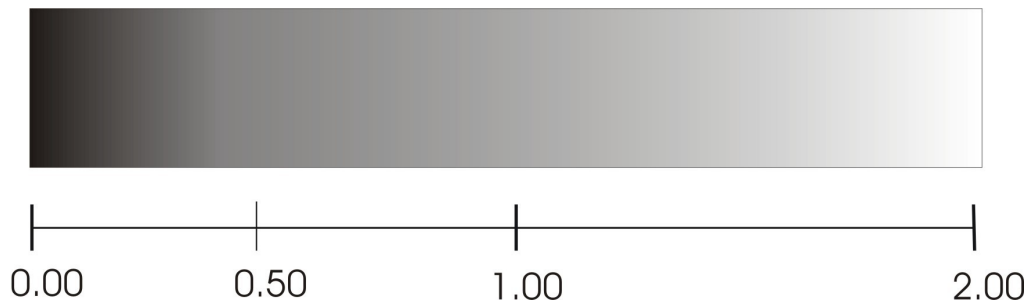


Figura 2.2: Associação de valores de pixels da imagem a cores feita pelo Matlab.

onde o preto absoluto possui valor próximo de zero, o cinza entre 0.50 e 1.80 e quanto mais a cor do pixel se aproxima de branco mais próximo seu valor fica de 2.0.

2.5 – Diferentes configurações da matriz distorção H

Retomando a equação que representa o modelo linear de formação da imagem:

$$\mathbf{g} = \mathbf{Hf} + \boldsymbol{\varepsilon},$$

A matriz que representa as distorções sofridas pela imagem, \mathbf{H} , pode assumir diversas configurações dependendo do tipo de degradação que a imagem sofre.

- “Deslocamento”: representa a distorção causada pelo deslocamento do sensor ou do objeto durante a aquisição. A imagem distorcida, neste caso, apresenta-se duplicada e deslocada uma em relação à outra. A matriz degradação então, pode apresentar a seguinte configuração:

A diagonal principal composta de ‘1’ conferindo nenhuma distorção (representando a imagem original como ela é), somente repetindo a imagem original, e numa outra diagonal secundária, cuja distância desta para a principal depende do deslocamento apresentado entre as imagens original e a duplicada, também composta de ‘1’. O que reproduz duas imagens: a imagem inicial na sua posição original, e a mesma, deslocada de n pixels (distância da diagonal secundária repleta de ‘1’ à principal) para a direita ou esquerda dependendo da posição da diagonal secundária em relação à principal (à esquerda ou à direita respectivamente).

$$H_1 = \begin{bmatrix} 1 & 0 & \dots & 1 & 0 & \dots & \dots & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & \dots & 0 \\ \dots & 0 & 1 & \dots & 0 & 1 & 0 & \dots \\ 0 & 0 & \dots & 1 & 0 & 0 & 1 & 0 \\ 0 & \dots & 0 & 0 & \dots & 0 & \dots & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 & 1 & 0 \\ 0 & 0 & \dots & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad H_2 =$$

$$\begin{bmatrix} 1 & 0 & \dots & 0 & 0 & \dots & \dots & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & \dots & 0 \\ \dots & 0 & 1 & \dots & 0 & 0 & 0 & \dots \\ 1 & 0 & \dots & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & \dots & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & \dots & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

A matriz distorção acima onde a diagonal secundária repleta de ‘1’ está à esquerda da principal, caracteriza uma imagem deslocada adiantada em relação à

principal, isto é caracteriza um deslocamento para a direita. Analogamente a matriz distorção acima à direita, caracteriza um deslocamento para a esquerda.

Temos que garantir, que a matriz distorção H preserve a energia em f , assim:

$$\|H^*f\| = \|f\|,$$

por isso normalizamos cada linha da matriz, dividindo-a pelo somatório do valor dos elementos da linha.

Aplicando a matriz distorção H_1 à uma imagem que consiste num “benchmark” em restauração de imagens, “Lena.gif” retida de [19], obtemos a imagem mostrada na Figura 2.4.



Figura 2.3: Imagem da Lena em escala de cinza 256X256.



Figura 2.4: Imagem da Lena corrompida pela matriz deslocamento H_1 .

Ausência de iluminação: distorção causada pela falta de iluminação durante a aquisição da imagem, conferindo a esta um aspecto escurecido.

A matriz distorção apresenta a seguinte configuração: diagonal principal multiplicada por um fator que seja inferior a um e maior que zero, o que reproduz a imagem principal escurecendo-a. Como mostra a matriz exemplo abaixo:

$$H_3 = \begin{bmatrix} 0.7 & 0 & 0 & 0 & 0 \\ 0 & 0.7 & 0 & 0 & 0 \\ 0 & 0 & 0.7 & 0 & 0 \\ 0 & 0 & 0 & 0.7 & 0 \\ 0 & 0 & 0 & 0 & 0.7 \end{bmatrix}$$

Aplicando-a a imagem descrita pela Figura 2.3 obtemos a imagem distorcida abaixo:



Figura 2.5: Imagem da Lena corrompida pela matriz H_3 .

- Excesso de iluminação: distorção que confere um aspecto esbranquiçado à imagem causada, pelo excesso de iluminação durante a aquisição desta.

Onde a matriz distorção que a caracteriza apresenta: diagonal principal preenchida por um fator que seja maior que um, reproduzindo a imagem inicial, porém esmaecida, conforme mostra a matriz exemplo abaixo:

$$H_4 = \begin{bmatrix} 1.5 & 0 & 0 & 0 & 0 \\ 0 & 1.5 & 0 & 0 & 0 \\ 0 & 0 & 1.5 & 0 & 0 \\ 0 & 0 & 0 & 1.5 & 0 \\ 0 & 0 & 0 & 0 & 1.5 \end{bmatrix}$$



Figura 2.6: Imagem da Lena corrompida pela matriz H_4 .

- Toeplitz: Em geral H pode tomar qualquer forma. Porém, se a função que descreve a degradação sofrida pela imagem no momento de sua captura, $h(x, \xi, y, \eta)$, é invariante no espaço então essa função pode ser reescrita como $h(x - \xi, y - \eta)$ na equação (2) e a matriz H toma a forma de uma matriz bloco Toeplitz.

A matriz Toeplitz é uma matriz onde todo elemento que se situa na mesma diagonal possui o mesmo valor. Abaixo temos um exemplo de uma matriz de Toeplitz:

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 1 & 2 & 3 & 4 \\ 3 & 2 & 1 & 2 & 3 \\ 4 & 3 & 2 & 1 & 2 \\ 5 & 4 & 3 & 2 & 1 \end{bmatrix}$$

Além disto, é visível uma outra propriedade: a matriz Toeplitz é igual a sua transposta.

A matriz Toeplitz gera um tipo de distorção em que o valor de cada pixel da imagem formada é influenciado pelo valor dos demais pixels vizinhos. Na matriz exemplo abaixo o valor de cada pixel é influenciado em intensidade cada vez menor, por valores dos demais pixels, à medida que a localização dos pixels vizinhos vai se distanciando.

Temos que garantir, porém, que a matriz distorção H preserve a energia em f ; assim:

$$\|H^*f\| = \|f\|,$$

por isso normalizamos cada linha da matriz, dividindo-a pelo somatório do valor dos elementos da linha.

$$H_5 = \begin{bmatrix} 1 & 1/n & 1/n^2 & 1/n^3 & \dots & \dots \\ 1/n^2 & 1 & 1/n & 1/n^2 & 1/n^3 & \dots \\ 1/n^3 & 1/n^2 & 1 & 1/n & 1/n^2 & 1/n^3 \\ \dots & 1/n^3 & 1/n^2 & 1 & 1/n & 1/n^2 \\ \dots & \dots & 1/n^3 & 1/n^2 & 1 & 1/n \\ \dots & \dots & \dots & 1/n^3 & 1/n^2 & 1 \end{bmatrix}$$

Uma matriz de Toeplitz por blocos consiste numa matriz que pode ser dividida em blocos de um mesmo tamanho. Cada bloco consistindo numa matriz de Toeplitz, e blocos que se situam na mesma diagonal são idênticos. Abaixo segue um exemplo de uma matriz de Toeplitz por blocos 6X6:

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 1 & 4 & 3 & 6 & 5 \\ 3 & 4 & 1 & 2 & 3 & 4 \\ 4 & 3 & 2 & 1 & 4 & 3 \\ 5 & 6 & 3 & 4 & 1 & 2 \\ 6 & 5 & 4 & 3 & 2 & 1 \end{bmatrix} = \begin{bmatrix} H_{11} & H_{22} & H_{33} \\ H_{22} & H_{11} & H_{22} \\ H_{33} & H_{22} & H_{11} \end{bmatrix}$$

onde:

$$H_{11} = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}, H_{22} = \begin{bmatrix} 3 & 4 \\ 4 & 3 \end{bmatrix}, H_{33} = \begin{bmatrix} 5 & 6 \\ 6 & 5 \end{bmatrix}$$

Perceba que uma matriz Toeplitz é também uma matriz Toeplitz por blocos com um bloco de tamanho um por um, mas uma matriz Toeplitz por blocos geralmente não é

Toeplitz. A estrutura de uma matriz H , Toeplitz por blocos, se sucede devido à estrutura em bloco dos vetores f , g e ε gerados por arranjo lexicográfico.

A matriz Toeplitz pode representar distorções invariantes no espaço como, por exemplo, causadas por impurezas no sistema de lentes da câmera durante a aquisição da imagem, ou no caso de imagens de radiografia que apresentam baixo contraste devido à natureza do sistema de imagens de raio-X, prejudicando assim a análise destas pelos profissionais da área médica.

Na Figura 2.7 temos um esquema de como se forma uma imagem pela penetração da radiação no objeto. Uma fonte pontual de radiação penetra no objeto e é interceptada no plano da imagem. A imagem é uma projeção da sombra do interior do objeto. Tais imagens são usualmente denominadas de radiografias.

Uma fonte de radiação de tamanho finito se espalha no plano de projeção da imagem, como mostrado na Figura 2.8. A técnica de modelar estes efeitos na formação da imagem é através da função fonte de radiação linear e invariante no espaço, que é matricialmente representada por uma matriz Toeplitz.

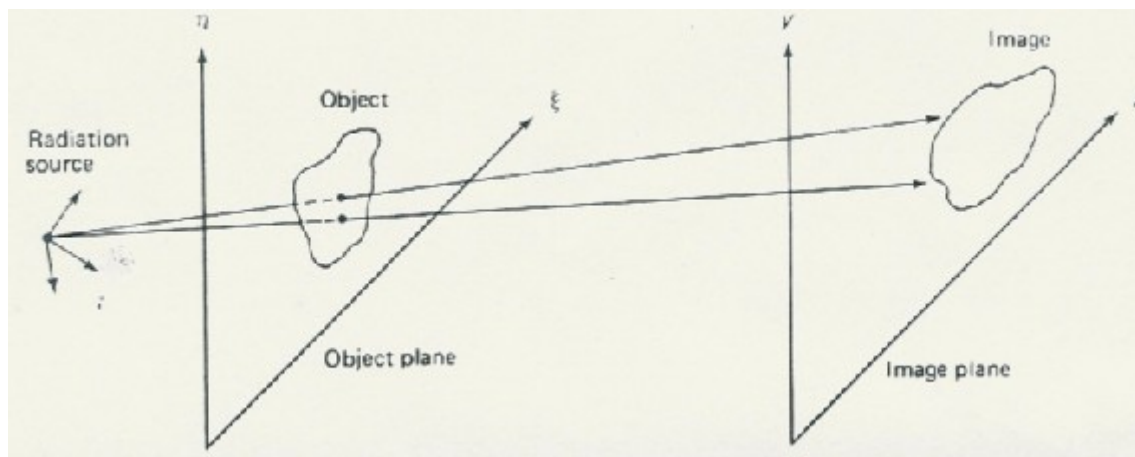


Figura 2.7: Formação da imagem de objetos pela penetração de radiação

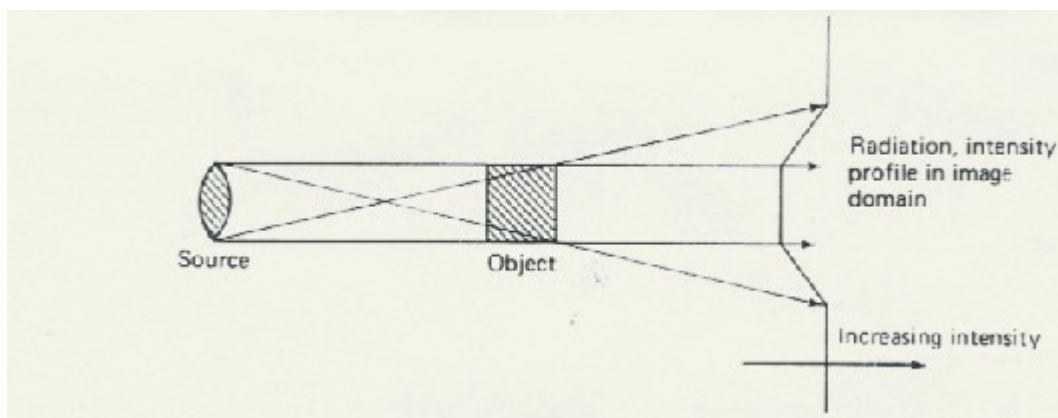


Figura 2.8: Fonte de erro na formação de imagens de radiografias.

Abaixo temos as imagens distorcidas resultantes da aplicação da matriz H_5 sobre as figuras: 2.9 de uma radiografia retirada de [19], 2.11 de uma impressão digital retirada de [19] e sobre a imagem da Lena.



Figura 2.9: Imagem de uma radiografia em escala cinza.

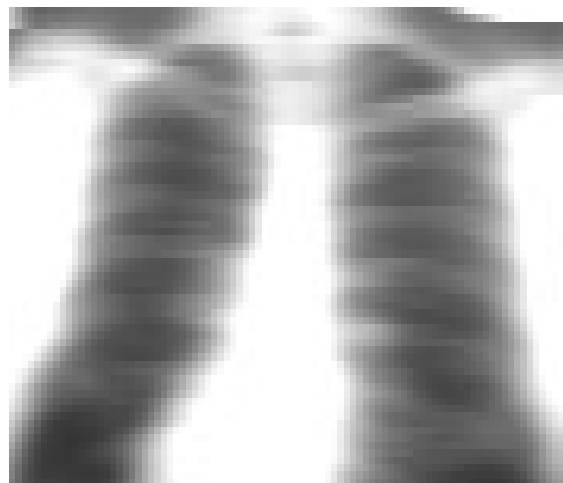


Figura 2.10: Imagem da radiografia corrompida pela matriz distorção H_5 com $n=1.2$.

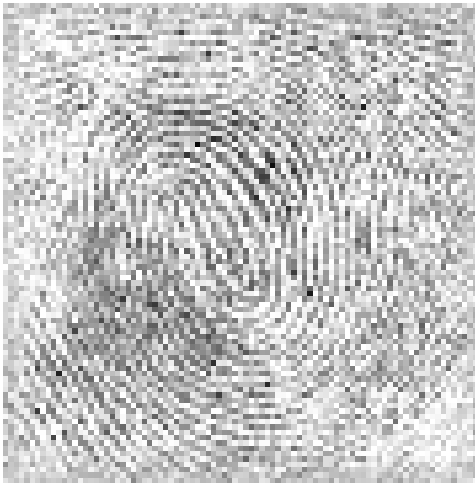


Figura 2.11: Imagem de uma impressão digital em escala de cinza.

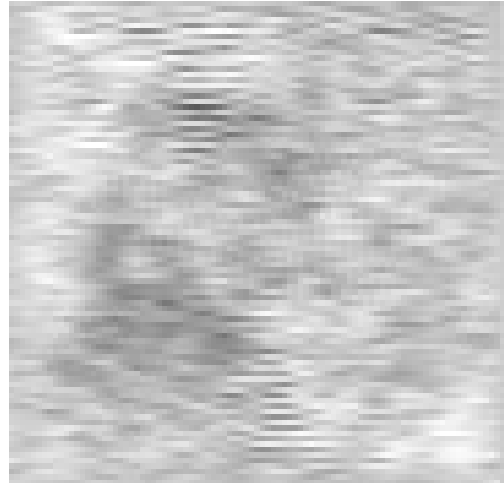


Figura 2.12: Imagem da impressão digital corrompida pela matriz distorção H_5 com $n=1.4$.



Figura 2.13: Imagem da Lena corrompida pela matriz distorção H_5 com $n=1.2$.

- Rotação: podemos ainda gerar diversos efeitos sobre a imagem, temos matrizes de distorção que tem como característica produzir rotação na imagem original. Têm como configuração a matriz abaixo:

$$H_6 = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Aplicando a matriz à imagem descrita pela Figura 2.2, obtemos a seguinte imagem distorcida:



Figura 2.14: Imagem da Lena rotacionada pela matriz H_6 .

Este trabalho será focado na análise dos seguintes tipos de distorção: excesso de iluminação, ausência de iluminação, borrão (fora de foco) e deslocamento.

2.6 – Exemplo ilustrativo

Tomemos como exemplo ilustrativo a imagem de um quadrado, equidistante dos quatro vértices, com 20 pixels de lado e mais 20 pixels de distância em relação a cada um dos vértices, como mostra a figura seguinte.

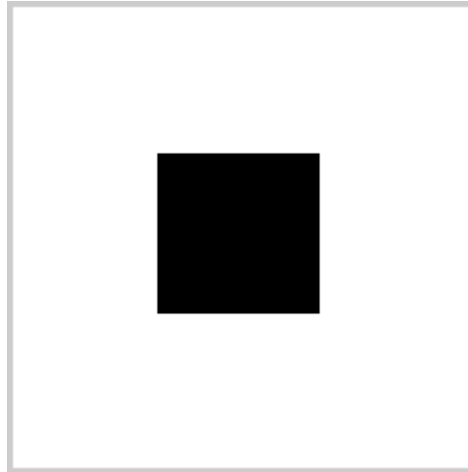


Figura 2.15: Imagem utilizada como exemplo ilustrativo.

Submetendo a imagem à matriz de distorção abaixo, que além de repetir a imagem original inseri uma outra imagem idêntica a primeira só que deslocada sete pixels ($d=7$) para a esquerda, isto é adiantada em relação à original.

Conforme a equação linear de formação da imagem, é observado que a diagonal secundária à direita da principal é responsável por repetir os pixels que formam a imagem original só que com um adiantamento de “d” pixels na horizontal, adiantamento este que equivale à distância (em colunas) desta em relação à diagonal principal, como segue a explicação abaixo.

$$\begin{array}{c}
 \mathbf{g} = \mathbf{Hf} \\
 \begin{bmatrix}
 1 & 0 & \dots & 0 & 1 & 0 & 0 & 0 & 0 \\
 0 & 1 & 0 & \dots & 0 & 1 & 0 & 0 & 0 \\
 0 & 0 & 1 & 0 & \dots & 0 & 1 & 0 & 0 \\
 \dots & 0 & 0 & 1 & 0 & \dots & 0 & 1 & 0 \\
 0 & \dots & 0 & 0 & 1 & 0 & \dots & 0 & 1 \\
 0 & 0 & \dots & 0 & 0 & 1 & 0 & \dots & 0 \\
 0 & 0 & 0 & \dots & 0 & 0 & 1 & 0 & \dots \\
 0 & 0 & 0 & 0 & \dots & 0 & 0 & 1 & 0 \\
 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 1
 \end{bmatrix}
 \begin{array}{c}
 \text{imagem original} \\
 \downarrow \\
 \begin{bmatrix}
 f_{11} \\
 f_{12} \\
 \dots \\
 f_{18} \\
 \dots \\
 \dots \\
 f_{1n}
 \end{bmatrix}
 \end{array}
 =
 \begin{array}{c}
 \text{imagem deslocada} \\
 \downarrow \\
 \begin{bmatrix}
 f_{11} \\
 f_{12} \\
 \dots \\
 f_{18} \\
 \dots \\
 f_{1n-7} \\
 \dots \\
 f_{1n}
 \end{bmatrix}
 \end{array}
 +
 \begin{array}{c}
 \begin{bmatrix}
 f_{18} \\
 f_{19} \\
 \dots \\
 f_{10} \\
 \dots \\
 f_{1n} \\
 0 \\
 \dots \\
 0
 \end{bmatrix}
 \end{array}
 \end{array}$$

Observando o vetor da imagem deslocada vemos que ela começa a partir do sétimo pixel da original em diante e termina sete pixels antes do último da original (os últimos sete pixels são zero, por isso o detalhe em preto na base direita da figura 3). É como se fosse deslocada a imagem original toda para a esquerda de sete colunas e as sete primeiras colunas passam a ser as sete últimas.

Abaixo temos a imagem original, a deslocada de sete pixels para a esquerda (adiantada) e a resultante da sobreposição das Figuras 2.16 e 2.17.

A imagem resultante da sobreposição de duas figuras é determinada pela soma aritmética do valor dos pixels de cada uma das imagens em cada ponto (conforme explicado mostra o esquema abaixo)

Conforme dito na seção 2.5, temos que a cor preta caracteriza um valor de pixel próximo de 0.0, a cinza próximo de 1.0 e a branca acima de 2.0, portanto:

Preto (0.0) + Branco (1.0) = Branco (1.0)

Preto (0.0) + Preto (0.0) = Preto (0.0)

Branco (2.0) + Branco (2.0) = Branco (4.0)

Seguindo a lógica aritmética acima, vemos que de fato a sobreposição da Figura 2.16 com a 2.17 resulta na 2.18.

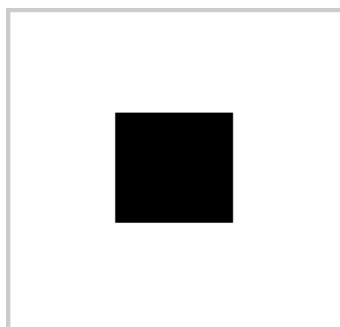


Figura 2.16: Imagem original.

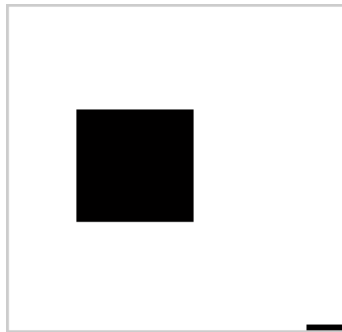


Figura 2.17: Imagem original deslocada de sete pixels para a esquerda.

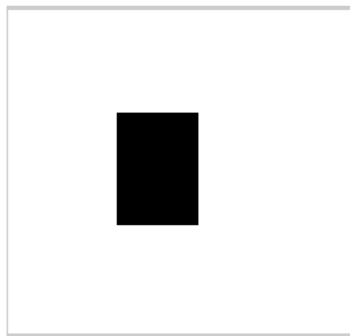
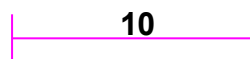


Figura 2.18: Imagem resultante da sobreposição das figuras 2.16 e 2.17.

Desta vez, submetendo-a a matriz distorção abaixo, que simula um objeto em movimento para a esquerda e para a direita, além de repetir a imagem original inseri duas outras imagens, uma idêntica a primeira só que deslocada dez pixels para a esquerda (Figura 2.19), isto é adiantada e a outra também idêntica à original só que deslocada dez pixels para a direita (Figura 2.20), isto é atrasada.

As imagens atrasada e adiantada são multiplicadas por um fator 0.3, o qual confere a estas um clareamento (uma coloração acinzentada), como mostram as figuras abaixo.



$$H = \begin{bmatrix} 1 & 0 & \dots & 0 & 0.3 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & \dots & 0 & 0.3 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 & 0.3 & 0 & 0 \\ \dots & 0 & 0 & 1 & 0 & \dots & 0 & 0.3 & 0 \\ 0.3 & \dots & 0 & 0 & 1 & 0 & \dots & 0 & 0.3 \\ 0 & 0.3 & \dots & 0 & 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 0.3 & \dots & 0 & 0 & 1 & 0 & \dots \\ 0 & 0 & 0 & 0.3 & \dots & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0.3 & \dots & 0 & 0 & 1 \end{bmatrix}$$



Figura 2.19: Imagem original deslocada de dez pixels para a esquerda e escurecida por um fator 0.3.

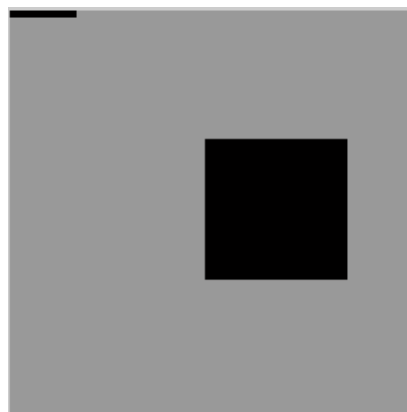


Figura 2.20: Imagem original deslocada de dez pixels para a direita e escurecida por um fator 0.3.

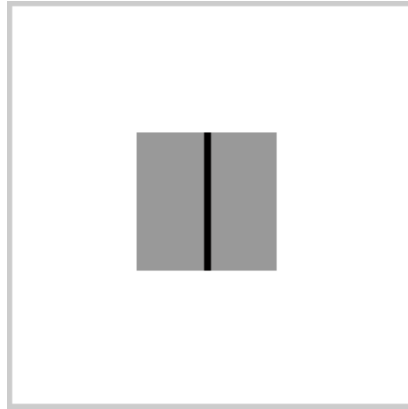


Figura 2.21 – Imagem resultante da sobreposição das figuras 2.19 e 2.20.

Submetemos agora a imagem 2.15 à matriz Toeplitz abaixo, que gera uma distorção do tipo borrão. Neste caso, valor de cada pixel da imagem formada é influenciado pelo valor dos demais pixels vizinhos, no caso da matriz abaixo é influenciado em intensidade cada vez menor, por valores dos demais pixels, a medida que a localização dos pixels vizinhos vai se distanciando. Por isso na figura abaixo observamos um leve “degradê” do preto para o branco nas laterais do quadrado. Obtemos como imagem resultante a representada na Figura 2.22.

$$H_5 = \begin{bmatrix} 1 & 1/n & 1/n^2 & 1/n^3 & \dots & \dots \\ 1/n^2 & 1 & 1/n & 1/n^2 & 1/n^3 & \dots \\ 1/n^3 & 1/n^2 & 1 & 1/n & 1/n^2 & 1/n^3 \\ \dots & 1/n^3 & 1/n^2 & 1 & 1/n & 1/n^2 \\ \dots & \dots & 1/n^3 & 1/n^2 & 1 & 1/n \\ \dots & \dots & \dots & 1/n^3 & 1/n^2 & 1 \end{bmatrix}$$

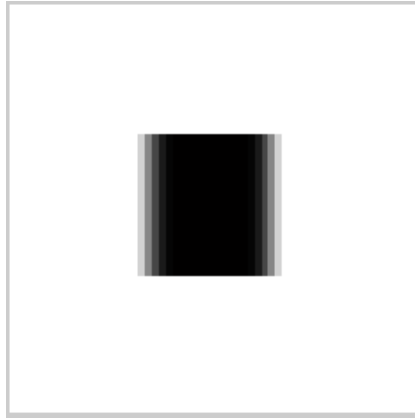


Figura 2.22: Figura corrompida pela matriz de Toeplitz H_5 com $n=1.2$.

Com base nisso poderemos analisar as demais imagens distorcidas que tentaremos restaurar.

2.7 – Diferentes matrizes de distorção H

Dentre as diversas configurações possíveis que a matriz H pode assumir, vamos nos focar mais neste trabalho nas associadas às distorções do tipo deslocamento, ausência, excesso de iluminação da imagem e borrão (fora de foco).

Utilizamos a Figura 2.23, que é um “benchmark”, retirada de [10] como nosso objeto de testes.



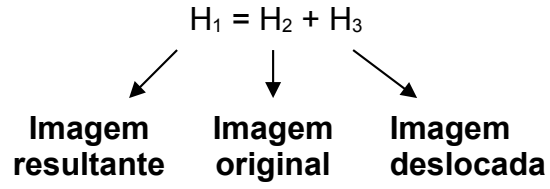
Figura: 2.23: Imagem original do pássaro (256X256).

A distorção do tipo deslocamento, como já mencionada no capítulo1, é caracterizada por um deslocamento, pela imagem adquirida de um objeto em movimento.

A matriz distorção H_1 , conforme utilizada na sessão anterior, simula um objeto em movimento para a esquerda, que além de repetir a imagem original (conferida pela diagonal principal com elemento '1', matriz H_2) inseri uma outra imagem idêntica à primeira só que deslocada 7 pixels para a esquerda, conforme mostra a matriz H_3 isto é adiantada em relação à original, conforme mostra a Figura 2.24.

Esta tarja acinzentada que aparece na figura 2.24, são as sete primeiras colunas da imagem original que passaram a ser as sete últimas com o deslocamento da imagem de sete colunas para a esquerda, conforme explicado na seção anterior.

Para resolvermos esse problema desta “tarja” que aparece na minha imagem deslocada, escurecerei as sete últimas colunas desta imagem, pois a cor preta tem valor neutro (zero) e qualquer valor de cor somada a ela, não se altera. Com isso ao submeter à matriz abaixo à Figura 2.23, terei como resultado a Figura 2.25, que representa fielmente uma distorcao do tipo deslocamento, é como se o pássaro realmente estivesse em movimento, ao ser capturada sua imagem. Já se não tivesse zerado as sete últimas colunas da imagem deslocada, ao submeter à matriz abaixo à imagem original teria como resultado a Figura 2.26.



$$H_1 = \begin{bmatrix} \overbrace{1 & 0 & \dots}^7 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & \dots & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 & 1 & 0 & 0 \\ \dots & 0 & 0 & 1 & 0 & \dots & 0 & 1 & 0 \\ 0 & \dots & 0 & 0 & 1 & 0 & \dots & 0 & 1 \\ 0 & 0 & \dots & 0 & 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 & 1 & 0 & \dots \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 1 \end{bmatrix} =$$

$$\overbrace{\hspace{10em}}^7$$

$$\begin{bmatrix} 1 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & \dots & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 & 0 & 0 & 0 \\ \dots & 0 & 0 & 1 & 0 & \dots & 0 & 0 & 0 \\ 0 & \dots & 0 & 0 & 1 & 0 & \dots & 0 & 0 \\ 0 & 0 & \dots & 0 & 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 & 1 & 0 & \dots \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 & \dots & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \dots & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 & 0 & 0 \\ \dots & 0 & 0 & 0 & 0 & \dots & 0 & 1 & 0 \\ 0 & \dots & 0 & 0 & 0 & 0 & \dots & 0 & 1 \\ 0 & 0 & \dots & 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & \dots \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 \end{bmatrix}$$

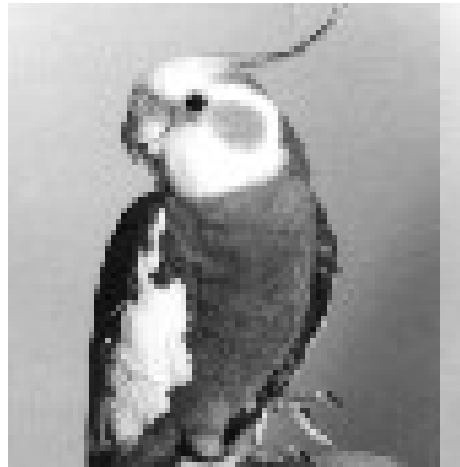


Figura 2.24 – Imagem deslocada de 7 pixels para a esquerda.

Aplicando esta matriz à figura do pássaro, obtemos a seguinte imagem como mostra a Figura 2.24:



Figura 2.25: Resultante da sobreposição das Figuras 2.23 e 2.24 (com as sete últimas colunas zeradas).

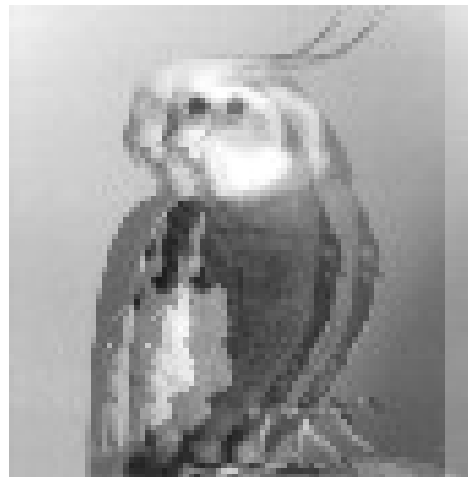


Figura 2.26: Resultante da sobreposição das Figuras 2.23 e 2.24.

Na figura resultante dessa distorção, as duas imagens a deslocada e a original ficaram com a mesma intensidade, isto é para a formação de cada pixel da Figura 2.25, utilizamos uma combinação do valor real do pixel da imagem original e da imagem deslocada sem alterar a sua intensidade.

De fato a Figura 2.25 simula um efeito deslocamento do pássaro no movimento em que a aquisição da imagem foi feita.

Tendo como base que a imagem que resulta de uma sobreposição é determinada por uma simples soma de valores dos pixels de cada uma das imagens ponto a ponto, será submetida agora, a Figura 2.23 à matriz H_4 abaixo. Com isto a imagem deslocada será escurecida, multiplicando-a por um fator 0.3, a fim de que quando houver a sobreposição, prevaleça a imagem original enquanto que da deslocada, fique somente um leve sombreamento à esquerda, conforme mostra a Figura 2.27.

$$H_4 = \begin{bmatrix} 1 & 0 & \dots & 0 & 0.1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & \dots & 0 & 0.1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 & 0.1 & 0 & 0 \\ \dots & 0 & 0 & 1 & 0 & \dots & 0 & 0.1 & 0 \\ 0 & \dots & 0 & 0 & 1 & 0 & \dots & 0 & 0.1 \\ 0 & 0 & \dots & 0 & 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 & 1 & 0 & \dots \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 1 \end{bmatrix}$$



Figura 2.27 – Resultante da distorção causada pela matriz H_4 na Figura 2.23.

Há ainda algumas configurações da matriz H , que geram outros tipos de distorções, além de deslocamentos e falta de iluminação. Submetendo a Figura 2.23 à matriz Toeplitz H_5 , onde os valores dos pixels vizinhos influenciam em baixa intensidade no valor de cada pixel, com o valor de n (fator Toeplitz) igual a 1.1, resulta na Figura 2.28.

$$H_5 = \begin{bmatrix} 1 & 1/n & 1/n^2 & 1/n^3 & \dots & \dots \\ 1/n^2 & 1 & 1/n & 1/n^2 & 1/n^3 & \dots \\ 1/n^3 & 1/n^2 & 1 & 1/n & 1/n^2 & 1/n^3 \\ \dots & 1/n^3 & 1/n^2 & 1 & 1/n & 1/n^2 \\ \dots & \dots & 1/n^3 & 1/n^2 & 1 & 1/n \\ \dots & \dots & \dots & 1/n^3 & 1/n^2 & 1 \end{bmatrix}$$

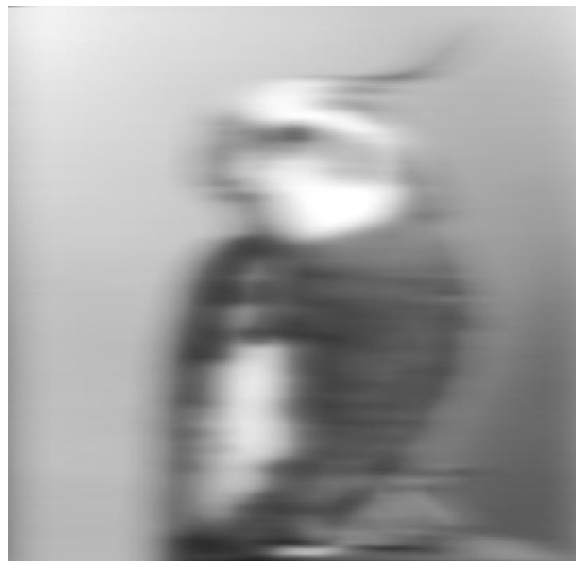


Figura 2.28: Imagem 2.23 que foi corrompida pela matriz Toeplitz H_5 com $n=1.1$.

Testando as distorções geradas na Figura 2.23 com valores maiores de n , maiores que 1.1, observa-se que fatores de Toeplitz acima de 2.0 praticamente não geram mais distorção.



Figura 2.29: Imagem 2.23 que foi corrompida pela matriz Toeplitz H_5 com $n=1.2$.



Figura 2.30: Imagem 2.23 que foi corrompida pela matriz Toeplitz H_5 com $n=1.4$.



Figura 2.31: Imagem 2.23 que foi corrompida pela matriz Toeplitz H_5 com $n=1.8$.



Figura 2.32: Imagem 2.23 que foi corrompida pela matriz Toeplitz H_5 com $n=2.0$.



Figura 2.33: Imagem 2.22 que foi corrompida pela matriz Toeplitz H_5 com $n=3.0$.

2.8 – Métodos de recuperação de imagens distorcidas

Dentre os métodos utilizados para resolver sistemas lineares, temos os diretos e os iterativos.

2.8.1 – Métodos diretos

- **Fatorização LU:**

Consiste numa variação do método de Gauss que decompõe uma matriz como um produto de uma matriz triangular inferior com uma matriz triangular superior. Essa decomposição nos leva ao algoritmo mais usado para resolver um sistema linear $\mathbf{Ax}=\mathbf{b}$ em um computador. A razão principal para a popularidade desse método é que ele fornece a maneira mais barata de resolver um sistema linear para o qual se faz necessário mudar, respectivamente, o lado direito. Esse tipo de situação aparece freqüentemente em problemas aplicados.

Suponha que uma matriz **A** pode ser escrita como um produto de uma matriz triangular inferior **L** com uma matriz triangular superior **U**, isto é,

$$\mathbf{A} = \mathbf{LU}.$$

Nesse caso afirma-se que **A** tem uma fatorização LU ou uma decomposição LU. A fatorização LU de uma matriz **A** pode ser usada para se resolver eficientemente o sistema linear **Ax=b**. Substituindo **A** por **LU**, obtém-se:

$$(\mathbf{LU})\mathbf{x}=\mathbf{b}$$

ou

$$\mathbf{L}(\mathbf{Ux})=\mathbf{b}.$$

Fazendo **Ux=z**, essa equação matricial fica

$$\mathbf{Lz}=\mathbf{b}.$$

Como **L** é triangular inferior, resolvemos diretamente para **z** por substituição de cima para baixo. Uma vez determinado **z**, como **U** é triangular superior, resolvemos **Ux=z** por substituição de baixo para cima. Resumindo, se matriz **A**, $n \times n$, tem uma fatorização LU, então a solução de **Ax=b** pode ser determinada por uma substituição de cima para baixo, seguida de uma substituição de baixo para cima [9]. O exemplo seguinte ilustra esse procedimento.

Considerando o sistema linear

$$\begin{aligned} 6x_1 - 2x_2 - 4x_3 + 4x_4 &= 2 \\ 3x_1 - 3x_2 - 6x_3 + x_4 &= -4 \\ -12x_1 + 8x_2 + 21x_3 - 8x_4 &= 8 \\ -6x_1 - 10x_3 + 7x_4 &= -43 \end{aligned}$$

cuja matriz de coeficientes

$$\mathbf{A} = \begin{bmatrix} 6 & -2 & -4 & 4 \\ 3 & -3 & -6 & 1 \\ -12 & 8 & 21 & -8 \\ -6 & 0 & -10 & 7 \end{bmatrix}$$

Tem uma decomposição LU, onde

$$\mathbf{L} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1/2 & 1 & 0 & 0 \\ -2 & -2 & 1 & 0 \\ -1 & 1 & -2 & 1 \end{bmatrix} \quad \text{e} \quad \mathbf{U} = \begin{bmatrix} 6 & -2 & -4 & 4 \\ 0 & -2 & -4 & -1 \\ 0 & 0 & 5 & -2 \\ 0 & 0 & 0 & 8 \end{bmatrix}$$

Para resolver o sistema dado usando essa decomposição LU, faz-se da seguinte maneira. Seja

$$\mathbf{b} = \begin{bmatrix} 2 \\ -4 \\ 8 \\ -43 \end{bmatrix}$$

Então, é resolvido $\mathbf{Ax}=\mathbf{b}$ escrevendo essa equação na forma $\mathbf{LUx}=\mathbf{b}$. Primeiro, é feito $\mathbf{Ux}=\mathbf{b}$ e resolvido $\mathbf{Lz}=\mathbf{b}$:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1/2 & 1 & 0 & 0 \\ -2 & -2 & 1 & 0 \\ -1 & 1 & -2 & 1 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix} = \begin{bmatrix} 2 \\ -4 \\ 8 \\ -43 \end{bmatrix}$$

por substituição de cima para baixo. Obtém-se:

$$z_1 = 2$$

$$z_2 = -4 - \frac{1}{2} z_1 = -5$$

$$z_3 = 8 + 2 z_1 + 2 z_2 = 2$$

$$z_4 = -43 + z_1 - z_2 + 2 z_3 = -32$$

Agora será resolvido $\mathbf{Ux}=\mathbf{z}$,

$$\begin{bmatrix} 6 & -2 & -4 & 4 \\ 0 & -2 & -4 & -1 \\ 0 & 0 & 5 & -2 \\ 0 & 0 & 0 & 8 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 2 \\ -5 \\ 2 \\ -32 \end{bmatrix},$$

por substituição de baixo para cima. Obtém-se

$$x_4 = \frac{-32}{8} = -4$$

$$x_3 = \frac{2 + 2x_4}{5} = -1,2$$

$$x_2 = \frac{2 + 2x_3 + 4x_4}{-2} = -6,9$$

$$x_1 = \frac{2 + 2x_2 + 4x_3 - 4x_4}{6} = 4,5.$$

Logo, a solução do sistema linear dado é

$$x = \begin{bmatrix} 4,5 \\ 6,9 \\ -1,2 \\ -4 \end{bmatrix}.$$

- **Fatorização QR:**

Esse tipo de fatorização é muito utilizado nos programas de computador para encontrar os autovalores de uma matriz, resolver sistemas lineares e encontrar aproximações de mínimos quadráticos.

Teorema: Se a matriz **A** é uma matriz $m \times n$ com colunas linearmente independentes, então **A** pode ser fatorada como **A=QR**, onde **Q** é uma matriz $m \times n$ cujas colunas formam uma base ortonormal para o espaço coluna de **A** e **R** é uma matriz triangular superior invertível $n \times n$ [9].

A matriz $R = [r_{ij}]$, é resultado do produto dos elementos da matriz **A** pelos da matriz **Q**:

$$r_{ij} = u_i \cdot w_j.$$

Primeiramente deve-se representar as colunas de **A** por u_1, u_2, \dots, u_n e seja W o subespaço de R^n que tem esses valores como base. Em seguida, usa-se o processo de Gram-Schmidt para transformar a base $\{ u_1, u_2, \dots, u_n \}$ para W em uma base ortonormal $\{ w_1, w_2, \dots, w_n \}$, que irão compor as colunas da matriz **Q**:

$$Q = [w_1 \ w_2 \ \dots \ w_n]$$

Pode-se obter uma base ortonormal w_1, w_2, \dots, w_n para o espaço coluna de **A**.

Apartir das matrizes **A** e **Q** chega-se à matriz **R**, onde

$$R = [r_{ij}] \quad \text{e} \quad r_{ij} = u_i \cdot w_j.$$

2.8.2 – Métodos Iterativos

Os métodos iterativos para resolver sistemas lineares começam com uma estimativa inicial para a solução e sucessivamente aprimoram isto até que a solução fique tão acurada quanto desejada. Em teoria, um número infinito de iterações se faz necessário para que haja convergência para a solução exata, mas na prática a iteração termina quando um valor de erro, tipicamente uma norma do resíduo, é tão pequeno quanto desejado.

Os métodos iterativos têm muitas vantagens significantes sobre os métodos diretos, principalmente quando implementados em paralelo como será apresentado. São facilmente paralelizáveis o que não se verifica no caso dos métodos diretos, por isso serão utilizados para resolver o sistema linear de recuperação de imagens.

- **Métodos Iterativos Estacionários:**

Provavelmente o método mais simples para resolver o sistema linear representado pela equação abaixo

$$\mathbf{Ax} = \mathbf{b}, \quad (4)$$

tem a seguinte forma:

$$\mathbf{x}_{(k+1)} = \mathbf{G} \mathbf{x}_{(k)} + \mathbf{c}, \quad \mathbf{k} = 0, 1, 2, \dots \quad (5)$$

onde a matriz \mathbf{G} e o vetor \mathbf{c} são escolhidos tal que um ponto fixo da equação acima seja a solução para $\mathbf{Ax} = \mathbf{b}$. Onde $\mathbf{x}_{(k)}$ significa o vetor resultante da iteração anterior e $\mathbf{x}_{(k+1)}$ o resultante da iteração atual (versão atualizada de $\mathbf{x}_{(k)}$). Tal método é dito ser estacionário se \mathbf{G} e \mathbf{c} são constantes em todas as iterações.

Uma maneira de se obter uma matriz \mathbf{G} adequada é através da decomposição, na qual a matriz \mathbf{A} possa ser escrita como:

$$\mathbf{A} = \mathbf{M} - \mathbf{N},$$

com \mathbf{M} não-singular. Podemos então escrever $\mathbf{G} = \mathbf{M}^{-1}\mathbf{N}$ e $\mathbf{c} = \mathbf{M}^{-1}\mathbf{b}$, de maneira que os esquemas de iteração fiquem:

$$\mathbf{x}_{(k+1)} = \mathbf{M}^{-1}\mathbf{N} \mathbf{x}_{(k)} + \mathbf{M}^{-1}\mathbf{b}, \quad (6)$$

que é implementado como

$$\mathbf{M} \mathbf{x}_{(k+1)} = \mathbf{N} \mathbf{x}_{(k)} + \mathbf{b},$$

de maneira que é resolvido um sistema linear com a matriz \mathbf{M} a cada iteração. Formalmente, este esquema de decomposição é uma iteração com um ponto fixo usando a função iteração:

$$\mathbf{g}(\mathbf{x}) = \mathbf{M}^{-1}\mathbf{N} \mathbf{x} + \mathbf{M}^{-1}\mathbf{b},$$

cuja matriz Jacobiana é

$$\mathbf{G}(\mathbf{x}) = \mathbf{M}^{-1}\mathbf{N}.$$

Então, o esquema de iteração é convergente se o raio espectral for menor que 1:

$$\rho(\mathbf{G}) = \rho(\mathbf{M}^{-1}\mathbf{N}) < 1,$$

e quanto menor $\rho(\mathbf{G})$, mais rápida é a convergência.

Para rápida convergência, devemos escolher \mathbf{M} e \mathbf{N} de maneira que $\rho(\mathbf{M}^{-1}\mathbf{N})$ seja tão pequeno quanto possível. Existe um “*trade-off*”, no entanto, de maneira que o custo de iteração é determinado pelo custo de resolver um sistema linear usando a matriz \mathbf{M} . Num exemplo extremo, se $\mathbf{M} = \mathbf{A}$, então o esquema converge em uma única iteração (isto é, temos um método direto). Na prática, \mathbf{M} é escolhida de modo a se aproximar de \mathbf{A} de alguma maneira, mas é geralmente restrito a ter uma forma simples, tal como diagonal ou triangular, tal que o sistema linear a cada iteração seja fácil de resolver [7].

- **Método de Jacobi:**

A escolha mais simples para \mathbf{M} na decomposição da matriz $\mathbf{A} = \mathbf{M} - \mathbf{N}$ é a matriz diagonal, especificamente a diagonal de \mathbf{A} . Faça \mathbf{D} ser uma matriz diagonal com a mesma diagonal de \mathbf{A} , e \mathbf{L} e \mathbf{U} sera as porções triangular inferior e superior, respectivamente, da matriz \mathbf{A} de maneira que:

$$\mathbf{M} = \mathbf{D}, \quad \mathbf{N} = -(\mathbf{L} + \mathbf{U})$$

mostram a decomposição de \mathbf{A} . Se \mathbf{A} não tem elementos nulos na diagonal principal, e assim \mathbf{D} seja não-singular, substituindo esses termos na equação (6), obtém-se um método iterativo chamado *método de Jacobi*, representado pela equação abaixo:

$$\mathbf{x}_{(k+1)} = \mathbf{D}^{-1}(\mathbf{b} - (\mathbf{L} + \mathbf{U}) \mathbf{x}_{(k)}). \quad (7)$$

onde desenvolvendo a expressão acima obtém-se:

$$\mathbf{x}_{(k+1)} = \mathbf{D}^{-1}\mathbf{b} - \mathbf{D}^{-1}(\mathbf{L} + \mathbf{U}) \mathbf{x}_{(k)}. \quad (8)$$

Passando este esquema para cada componente individual da solução, observa-se que, começando com uma atribuição inicial $\mathbf{x}_{(0)}$, o método de Jacobi computa a próxima iteração ($\mathbf{x}_{(k+1)}$) resolvendo para cada componente de \mathbf{x} em termos das demais ($\mathbf{x}_{(k)}$):

$$x_{i(k+1)} = \frac{b_i - \sum_{j \neq i} a_{ij} x_{j(k)}}{a_{ii}}, \quad i=1, \dots, n. \quad (9)$$

Note que o método de Jacobi requer um armazenamento duplo para o vetor \mathbf{x} , já que todos os valores das componentes anteriores são necessários na varredura, portanto os valores das componentes novas não podem sobrescrevê-los até que a varredura seja completada.

O método de Jacobi nem sempre converge, mas é garantido que convirja sob condições que são freqüentemente satisfeitas na prática (isto é se a matriz é diagonalmente dominante por linhas). Infelizmente, a velocidade de convergência do método de Jacobi é geralmente lenta.

- **Método de Gauss-Seidel:**

Uma razão para a lentidão de convergência do método de Jacobi é que este não faz uso da última informação disponível: os valores das componentes novas são somente usados após o fim da varredura completa. O método de Gauss-Seidel remedia esta desvantagem fazendo uso de cada nova componente da solução assim que esta tiver sido computada:

$$x_{i(k+1)} = \frac{b_i - \sum_{j<i} a_{ij}x_{j(k+1)} - \sum_{j>i} a_{ij}x_{j(k)}}{a_{ii}}, \quad i=1,\dots,n. \quad (10)$$

Utilizando a mesma notação do método anterior, o método de Gauss-Seidel pode ser escrito em termos de matrizes como:

$$\begin{aligned} \mathbf{x}_{(k+1)} &= \mathbf{D}^{-1}(\mathbf{b} - \mathbf{L} \mathbf{x}_{(k+1)} - \mathbf{U} \mathbf{x}_{(k)}) \\ &= (\mathbf{D} + \mathbf{L})^{-1}(\mathbf{b} - \mathbf{U} \mathbf{x}_{(k)}), \end{aligned} \quad (11)$$

e portanto corresponde a decomposição:

$$\mathbf{M} = \mathbf{D} + \mathbf{L}, \quad \mathbf{N} = -\mathbf{U}.$$

Adicionalmente acelerando a convergência, um outro benefício do método de Gauss-Seidel é que o armazenamento duplo, que acontecia no de Jacobi, não se faz necessário para o vetor \mathbf{x} , desde então os valores da componente mais recente podem sobrescrever os imediatamente anteriores. Porém, por outro lado, a atualização dos desconhecidos deve ser feita sucessivamente, em contraste com o de Jacobi, no qual os valores desconhecidos devem ser atualizados em qualquer

ordem ou até mesmo simultaneamente. A última característica pode fazer o método de Jacobi preferível num ambiente de processamento paralelo.

O método de Gauss-Seidel nem sempre converge, mas é garantido que convirja sob condições que são freqüentemente satisfeitas na prática, que são de alguma maneira, menos restritas que as do método de Jacobi (por exemplo, se a matriz é simétrica e positiva definida). Embora o método de Gauss-Seidel convirja mais rapidamente que o de Jacobi, é ainda muitas vezes lento para ser prático, principalmente em ambiente de computação seqüencial [7].

Com base nas afirmações acima, optou-se pela resolução do sistema linear de recuperação de imagens, implementado em ambiente paralelo, utilizando o método iterativo de Jacobi.

Capítulo 3 – Análise da solução seqüencial do sistema linear de recuperação de imagem usando o método iterativo de Jacobi

Primeiramente será desenvolvida e estudada a viabilidade da solução seqüencial para o problema apresentado.

3.1 – Discretização da imagem

Para resolver o problema de restauração de imagens, deve-se começar primeiramente discretizando-a e transformando-a numa matriz de pixels, para que se possa trabalhar com valores em ponto flutuante, correspondente a uma escala de intensidade de cor da imagem.

Para isso, foi feito uso do software MATLAB e de algumas funções que este nos oferece e montou-se um script. Este script é apresentado abaixo:

```
f=imread('C:\Documents and Settings\Luciana\MATLAB\bin\bird.gif','gif');
f=double(f);
f=f./200;
f=imresize(f,0.2734375);
imshow(f);
save('C:\Documents and Settings\Luciana\MATLAB\bin\imageDistorted','f','-
ascii');
```

- Primeiramente é transformada a imagem distorcida numa matriz de pixels e a denomina de 'f', depois os elementos desta matriz são transformados em double (tipo básico de dado em C que é ponto flutuante de precisão dupla) para que se possa trabalhar.
- É reduzida a intensidade da imagem para que possa ser observada pela função "imshow" do MATLAB,
- É redimensionada então a imagem que originalmente era 256 X 256 para outras dimensões (70X70, 80X80, etc...) a fim de verificar o ganho de desempenho do algoritmo paralelo sobre o sequencial à medida em que aumenta-se a dimensão da imagem e o número de processadores.

- Por final, essa matriz é salva como um arquivo texto chamado “gImageFile”, o qual será submetido ao programa desenvolvido que fará o arranjo lexicográfico desta matriz e terá como objetivo restaurar essa imagem.

3.2 – Modelagem da solução seqüencial

Retomando a equação (3) que representa um modelo de um sistema linear de formação da imagem, vimos que ela é análoga à equação (4) que modela sistemas lineares. E que a equação (7) traduz a solução de sistemas lineares usando o método iterativo de Jacobi. Logo, substituindo variáveis vemos que:

$$\mathbf{f}_{(k+1)} = \mathbf{D}^{-1}(\mathbf{g} - (\mathbf{L} + \mathbf{U}) \mathbf{f}_{(k)}), \quad (12)$$

como inicialmente havíamos denominando de \mathbf{N} , o termo $-(\mathbf{L} + \mathbf{U})$, temos:

$$\mathbf{f}_{(k+1)} = \mathbf{D}^{-1}\mathbf{N} \mathbf{f}_{(k)} + \mathbf{D}^{-1} \mathbf{g}, \quad (13)$$

que descreve a solução do sistema de recuperação de imagem usando o método iterativo de Jacobi.

O vetor \mathbf{g} , que consiste no vetor de pixels que forma a imagem distorcida, leremos do arquivo “gImage” e a matriz \mathbf{N} , será a matriz distorção \mathbf{H} (a qual terá uma configuração diferente de acordo com o tipo de degradação sofrida pela imagem) com a diagonal principal nula e os demais elementos pertencentes às diagonais secundárias com sinal trocado.

3.3 – Cálculo do erro

Estamos resolvendo neste trabalho um sistema de equações lineares da forma $\mathbf{Ax}=\mathbf{b}$. Uma maneira de medir o erro a cada iteração é utilizando a norma do resíduo, da seguinte forma:

1. O resíduo é definido por $\mathbf{r} = \mathbf{Ax}-\mathbf{b}$

Nesse caso, quando o resíduo for zero, significa que encontramos a solução do sistema de equações lineares, pois teremos $\mathbf{Ax}=\mathbf{b}$;

Será utilizado que a norma do resíduo como medida de erro, isto é, $\text{erro}=\|\mathbf{r}\|$. Ou seja, quando o erro for zero, significa que $\|\mathbf{r}\|=0$, o que implica que $\mathbf{r}=\mathbf{0}$ e nesse caso teremos a solução do sistema de equações.

Retomando a equação que define nosso sistema de recuperação de imagem agora modelada pela solução de Jacobi,

$$\mathbf{f}_{(k+1)} = \mathbf{D}^{-1}\mathbf{N} \mathbf{f}_{(k)} + \mathbf{D}^{-1} \mathbf{g}$$

será definido então como critério de parada quando a expressão abaixo,

$$\mathbf{z} = \|\mathbf{f}_{(k+1)} - \mathbf{f}_{(k)}\|,$$

alcançar determinado valor estabelecido durante o processamento. O valor de \mathbf{z} determina o quão longe o resultado da iteração anterior, $\mathbf{f}_{(k)}$, está do da próxima iteração, $\mathbf{f}_{(k+1)}$.

Se o critério de parada for atendido, é calculado então o resíduo, pois somente o cálculo do resíduo pode oferecer-nos uma medida certa e precisa do quão longe encontra-se da solução ideal ($\mathbf{Ax}=\mathbf{b}$). Se este for menor que o erro máximo estabelecido no algoritmo, o processamento é parado e é apresentada a solução aproximada.

3.4 – Algoritmo seqüencial

Fluxograma do algoritmo seqüencial

Sabendo que $\mathbf{H} = \mathbf{D} - \mathbf{N}$ e que a solução de Jacobi ficou da forma $\mathbf{f}_{(k+1)} = \mathbf{D}^{-1}\mathbf{N} \mathbf{f}_{(k)} + \mathbf{D}^{-1}\mathbf{g}$, abaixo é apresentado um diagrama de funcionamento do algoritmo seqüencial:

$\mathbf{f}_{(k)}$

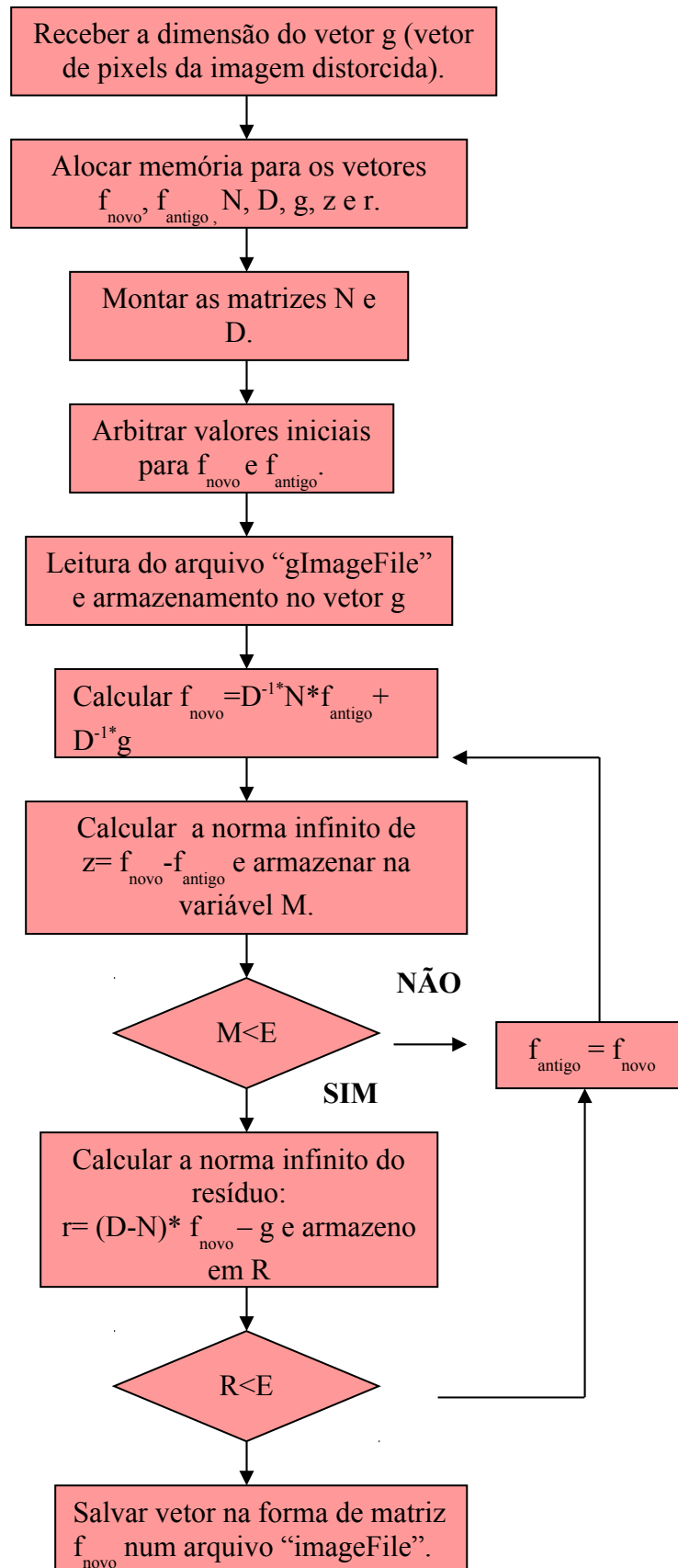


Figura 3.1: Fluxograma de funcionamento do algoritmo seqüencial.

Considerações:

Erro mínimo (E): 0.001 (valor absoluto)

Número máximo de iterações (count): 100

$f_{\text{novo}} = f_{(k+1)}$

$f_{\text{antigo}} = f_{(k)}$

y = dimensão do vetor g (vetor de pixels que descreve a imagem distorcida)

Loop principal (cálculo de $f_{\text{novo}} = D^{-1}N f_{\text{antigo}} + D^{-1}g$)

```
while ((stop==0) &&(count<100))
{
for(i=0;i<y;i++)
{
f_antigo [i]= f_novo [i];
f_novo [i]=0;
}
for(i=0;i<y;i++)
{
for(j=0;j<y;j++)
f_novo [i]+=matrixN[i][j]/matrixD[i][i]* f_antigo [j];
f_novo [i]= f_novo [i]+g[i]/matrixD[i][i];
}
E=0;
for(i=0;i<y;i++)
z[i]=fabsf(f_novo [i]- f_antigo [i]);
M=0;
for(i=0;i<y;i++)
if(z[i]>M)
M=z[i];
if(M<E)
{
for(i=0;i<y;i++){
for(j=0;j<y;j++)
r[i]=fabsf((-N[i][j]+matrixD[i][i])* f_novo [i]-g[i]);}
R=0; for(i=0;i<y;i++)
if(r[i]>R)
R=r[i];
if(R<E)
stop=1;
}
```

Figura 3.2: Loop principal do algoritmo seqüencial feito em linguagem C.

3.5 – Justificativa do uso de soluções paralelas

Executando então o algoritmo acima para uma imagem simples, em preto e branco, de dimensão a partir de 60X60 observa-se que não foi possível alocar memória suficiente para as matrizes (a partir de $150^2 \times 150^2$) que foram utilizadas durante o processamento. Já isso não se verifica durante o processamento paralelo, pois não foi utilizada somente a memória de um processador para alocar matrizes e vetores, mais sim de vários, alocando em cada um as respectivas partições das matrizes e vetores. Além disso, o tempo demandado de processamento paralelo mostra-se claramente menor em relação ao seqüencial, para elevadas dimensões da imagem distorcida sendo processada por muitos processadores.

A matriz **N** será uma matriz quadrada. Se a imagem tiver uma dimensão $n \times n$, o vetor **g** apresentar dimensão n^2 e a matriz em questão terá dimensão $n^2 \times n^2$.

O que torna o processamento seqüencial inviável para fins práticos e imagens mais robustas, no contexto seqüencial disponível.

Capítulo 4 – Processamento paralelo e medida de desempenho

4.1 – Introdução ao processamento paralelo

4.1.1 – Paralelismo

Paralelismo é uma estratégia para obter resultados mais rápidos, de grandes e complexas tarefas. Suas principais funções são: divisão de tarefas em várias outras menores, entre vários processadores que irão executá-las simultaneamente e a coordenação desses processadores.

Um exemplo disto é o processamento de imagens e recuperação destas, que conta com uma grande quantidade de dados e matrizes o que inviabiliza o processamento seqüencial. Assim faz-se necessário o processamento paralelo, o paralelismo, dividindo os dados e as tarefas entre os processadores disponíveis, otimizando assim o tempo de processamento.

Deste modo resolveremos o sistema linear de recuperação de imagens usando o método iterativo de Jacobi, implementado em algoritmo paralelo utilizando linguagem de programação C e a biblioteca MPI.

4.1.2 – Necessidade de Processamento mais Rápido

Existem várias classes de problemas que necessitam de processamento mais rápido:

- Problemas de modelagem e simulação, baseados em sucessivas aproximações e cálculos cada vez mais precisos;
- Problemas que dependem de manipulação de grandes bases de dados;
- Processamento de sinais e imagens;
- Visualização de dados;
- Banco de Dados.

Grandes desafios computacionais:

- Modelagem de Clima;
- Turbulência de Fluidos;
- Dispersão de Poluição;
- Engenharia Genética;
- Circulação de Correntes Marítimas;
- Modelagem de Semicondutores;
- Sistema de Combustão.

4.1.3 – MPI (“Message Passing Interface”)

Para resolver o algoritmo de restauração de imagens será usado o processamento paralelo, mediante a grande quantidade de dados a ser processada, e a biblioteca MPI.

O modelo *Message-Passing* é um dos vários modelos computacionais para conceituação de operações de programa. O modelo *Message-Passing* é definido como:

- Conjunto de processos que possuem acesso à memória local;
- Comunicação dos processos baseados no envio e recebimento de mensagens;
- A transferência de dados entre processos que requer operações de cooperação entre cada processo.
- Consiste numa biblioteca de *Message-Passing*, desenvolvida para ser padrão em ambientes de memória distribuída, em *Message-Passing* e em computação paralela.
- *Message-Passing* portátil para qualquer arquitetura, tem aproximadamente 125 funções para programação e ferramentas para se analisar o desempenho.
- Utilizado por programas em C e FORTRAN.
- A plataforma alvo para o MPI, são ambientes de memória distribuída, máquinas paralelas massivas, *clusters* de estações de trabalho.

- Todo paralelismo é explícito: o programador é responsável por identificar o paralelismo e implementar um algoritmo utilizando construções com o MPI.

Para mais informações a respeito da biblioteca MPI vide Apêndice B.

4.2 – Processamento Paralelo

Necessidades para programar em paralelo:

- Decomposição do algoritmo, ou dos dados, em "pedaços";
- Distribuição dos "pedaços" por entre os diferentes processadores, que trabalharão simultaneamente;
- Coordenação do trabalho e da comunicação entre esses processadores.

Para mais informações a respeito do processamento paralelo vide Apêndice A.

4.3 – Medida de desempenho do programa paralelo

4.3.1 - Balanceamento de carga

A distribuição das tarefas por entre os processadores deve ser de maneira que o tempo da execução paralela seja eficiente, se as tarefas não forem distribuídas de maneira balanceada é possível que ocorra a espera pelo término do processamento de uma única tarefa para dar prosseguimento ao programa.

Fine-Grain

- Tarefas executam um pequeno número de instruções entre ciclos de comunicação;
- Facilita o balanceamento de carga;
- Baixa computação, alta comunicação;
- É possível que ocorra mais comunicação do que computação, diminuindo o desempenho.

Coarse-Grain

- Tarefas executam um grande número de instruções entre cada ponto de sincronização possibilitando o aumento de desempenho;
- Difícil de obter um balanceamento de carga eficiente;
- Alta computação, baixa comunicação;

Associado ao grain size está a razão entre a COMPUTAÇÃO e a COMUNICAÇÃO. Quanto maior a razão implica numa maior granularidade.

4.3.2 - Considerações de desempenho

Existem algumas indicações que devem ser seguidas a fim de obter um bom desempenho num programa paralelo, dentre elas temos:

- Evitar engarrafamento de mensagens
- Observar que o tempo gasto em computação deve ser maior que o gasto em comunicação
- Definir um conjunto de pequenas tarefas, produzindo uma decomposição de pequena granularidade (“fine-grain”)
- Evitar replicação de dados e computação
- Iniciar pelo programa serial otimizado
- Controlar o processo de *Granularity* (Aumentar o número de computação em relação a comunicação entre processos);

- Utilize rotinas com comunicação *non-blocking* (vide Apêndice B);
- Evite utilizar rotinas de sincronização de processos;
- Evite, se possível, *buffering* (vide Apêndice B);
- Evite transferência de grande quantidade de dados.

4.3.3 – “Speed-up” e eficiência

Consideramos T_p o tempo de execução para um programa paralelo em np processadores. Definimos as seguintes expressões:

- S_p , a razão de “speed-up” em p processadores, é dada por

$$S_p = T_0/T_p,$$

Onde T_0 é o tempo do algoritmo serial mais rápido em um único processador. Deve ser notado que esta definição de razão de speed-up compara o algoritmo paralelo com o algoritmo serial mais rápido para um dado problema. Isto mede o benefício a ser ganho em mover a aplicação de uma máquina serial com um único processador para uma máquina paralela com p processadores idênticos. Mesmo o algoritmo sendo empregado normalmente, é esperado que o tempo demandado para a execução da implementação paralela num único processador (T_1) exceda o tempo que leva uma implementação serial no mesmo processador (T_0) devido às despesas gerais associadas com a execução de processos paralelos.

- S_p , a razão de “speed-up” algorítmica em p processadores, é dada por

$$S_p = T_1/T_p.$$

Esta quantidade mede o “speed-up” a ser ganho com a paralelização do algoritmo. Isto então mede diretamente os efeitos da sincronização e atrasos de comunicação no desempenho do algoritmo paralelo, e é a definição da razão de “speed-up” que será usada neste trabalho. Idealmente seria desejável que S_p crescesse linearmente com p , com gradiente unitário. Infelizmente, mesmo para

um “bom” algoritmo paralelo é possível esperar como o melhor que o “speed-up” crescesse próximo a velocidade linear e então eventualmente declinasse à medida que o problema se saturasse com os processadores e despesas gerais de sincronização começassem a dominar (vide Figura 4.1).

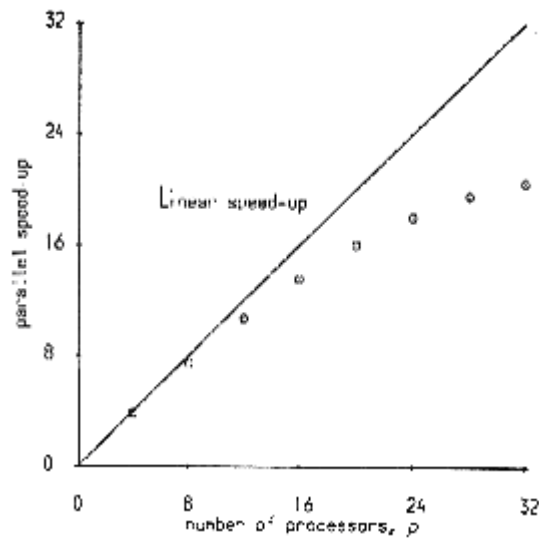


Figura 4.1: Típico gráfico de “speed-up”.

- E_p , a eficiência em p processadores, é dada por

$$E_p = 100 \times S_p / p.$$

Note que a eficiência é medida como uma porcentagem e que numa situação ideal poderia-se esperar por 100% de eficiência para todos os p processadores. Na prática é esperado que a eficiência decresça com o aumento de p . (vide Figura 4.2).

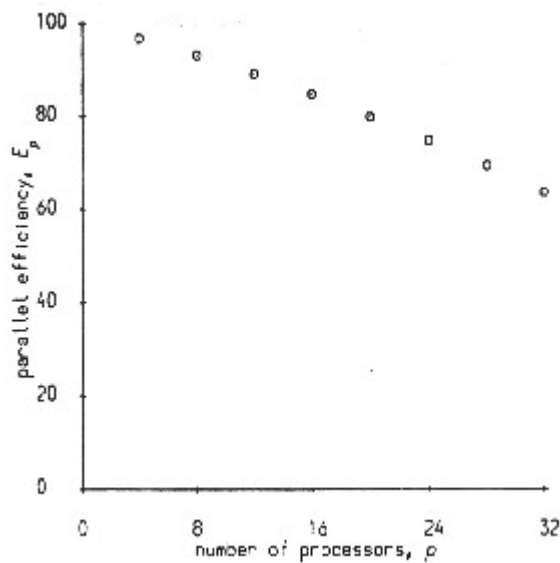


Figura 4.2: Típico gráfico de eficiência.

Deve ser observado que, como definido acima, ambas, medidas de “speed-up” e eficiência são as forças que claramente dirigem o desenvolvimento por trás do algoritmo paralelo. Porém, não podemos perder de vista outras características, como a acurácia.

4.3.4 - Lei de Amdahl – medidas de desempenho

Já foi mencionado que uma situação ideal seria aquela na qual a razão de “speed-up” crescesse linearmente com o número de processadores p , com inclinação 1, provendo uma eficiência de 100%. Tal situação raramente ocorre na prática, em parte devido à necessidade de sincronizar processos paralelos. Existe uma consideração adicional que emerge do fato inelutável que existem prováveis porções do programa que são inerentemente seqüenciais. Este resultado está encapsulado na Lei de Amdahl (Amdahl, 1967).

Lei de Amdahl: Suponha que r é a fração do programa que é paralelizável e que $s=1-r$ é a fração seqüencial inerente restante. Então em p processadores a razão de speed-up algorítmica, S_p satisfaz:

$$\bar{S}_p \leq \frac{1}{s + r/p}.$$

A prova deste resultado é simples e direta. Inicia-se com a definição da razão algorítmica do “speed-up” dada pela equação (5.1). Não importa quantos processadores sejam usados para resolver o problema, o tempo demandado para a parte seqüencial continua constante (sT_1), e o melhor que se pode esperar da parte paralela é que o tempo decresça inversamente com p e seja dado por rT_1/p . Portanto

$$T_p \geq sT_1 + rT_1/p,$$

e

$$\bar{S}_p = \frac{T_1}{sT_1 + rT_1/p} = \frac{1}{s + r/p}.$$

A lei de Amdahl aparenta ter sérias conseqüências até o ponto que a paralelização do algoritmo interessa, pois esta impõe um limite superior na razão de “speed-up”. Por exemplo, se somente 50% do programa é paralelizável, então $s=r=1/2$, e

$$\bar{S}_p \leq \frac{2}{1 + 1/p},$$

tal que $p \rightarrow \infty$, $S_p \leq 2$. Assim, para este programa em particular, a razão de “speed-up” fica limitada a 2, apesar do número de processadores usado.

Uma visão alternativa para a lei de Amdahl, que assume que a dimensão do programa cresce com o número de processadores, é por argumentação, mais apropriada. É assumido então, que o trecho do programa que somente pode ser executado de maneira seqüencial, T_s , é fixo, e o trecho que pode ser executado paralelamente, cresce com a dimensão do problema, n , e é dada por nT_p , onde T_p é fixo. A suposição que o tempo serial é independente da dimensão do problema é razoável somente em alguns casos. Para muitos outros problemas a dependência, em n , para os tempos de execução serial e paralelo são diferentes, como por exemplo, o tempo serial dado por nT_s e o tempo paralelo dado por n^2T_p . Em ambos os casos a fração seqüencial, s , é dada por

$$s = \frac{T_s}{T_s + nT_p},$$

com a fração paralela r dada por

$$r = \frac{nT_p}{T_s + nT_p}.$$

Logo

$$\bar{S}_p \leq \frac{T_s + nT_p}{T_s + nT_p} = \frac{1 + n\tau}{p + n\tau}$$

Onde $\tau = T_p/T_s$ é fixo. Portanto

$$\bar{S}_p \leq \frac{1 + n\tau}{p + n\tau}.$$

Pode-se concluir que $\bar{S}_p \leq p$. Adicionalmente, para uma dada escolha de p , podemos escolher n suficientemente grande que $n\tau \gg p$ e logo $S_p \rightarrow p$. Por outro lado se, para grandes valores de p , $n\tau = 1$ então $S_p \leq 2$. Esta visão da lei de Amdahl sugere que a razão para mover para uma máquina paralela maior seria para resolver exemplos maiores de um dado problema do que para resolver problemas da mesma dimensão mais rapidamente.

4.3.5 – Escalonamento do programa

Lei de Amdahl é relevante somente quando o problema é fixo, ou quando a fração serial independe do tamanho do problema, o que se verifica raramente.

Computadores maiores são utilizados para resolver problemas maiores, e a fração serial geralmente diminui quando o tamanho do problema aumenta.

Taxa de aumento do problema pode ser caracterizada pela manutenção de alguma grandeza invariante enquanto o número de processadores varia.

Candidatos plausíveis incluem

- Tamanho total do problema [Amdahl]
- Trabalho por processador [Gustafson]
- Tempo total de execução [Worley]
- Memória por processador [Sun]
- Eficiência [Grama]
- Erro computacional [Singh]

4.3.6 – Escalabilidade

Escalabilidade se refere à eficácia de um algoritmo paralelo na utilização de processadores adicionais.

Um algoritmo é denominado **escalável** em função do aumento do número de processadores se sua eficiência pode ser mantida constante (ou no mínimo limitada acima de zero) pelo aumento do tamanho do problema.

Um algoritmo escalável neste sentido poderia, no entanto, não ser prático se a taxa de aumento do tamanho do problema resulta em tempo total de execução inaceitável.

4.3.7 – Tempo de Computação

Tempo de computação é o tempo de execução serial mais o tempo gasto em qualquer computação adicional de execução paralela.

Taxa de computação pode variar em função do tamanho de problema por causa de efeitos de cache, assincronismo, etc.

4.3.8 – Tempo Ocioso

Tempo ocioso se deve à falta de tarefa alocada ou falta de dados necessários (p.ex. na espera da chegada de uma mensagem).

Tempo ocioso decorrente de falta de tarefa pode ser reduzido pela melhoria no balanceamento de carga.

Tempo ocioso decorrente da falta de dados pode ser reduzido pela utilização de computação e comunicação sobrepostas ou assincronismo.

“Multithreading” é um enfoque para sobrepor comunicação e computação.

Capítulo 5 – Proposta de uma solução paralela do sistema linear de recuperação de imagem usando o método iterativo de Jacobi

5.1 – Procedimentos de paralelização

Retomando a equação que modela a solução do sistema linear de formação de imagens usando método iterativo de Jacobi (13) é visto que:

$$\mathbf{f}_{(k+1)} = \mathbf{D}^{-1}\mathbf{N} \mathbf{f}_{(k)} + \mathbf{D}^{-1} \mathbf{g},$$

que descreve a solução da degradação de imagem usando o método iterativo de Jacobi.

5.1.1 – Desafios para resolução em uma máquina paralela para diferentes dimensões da imagem distorcida

1º Determinar a dimensão da matriz H:

Isto é determinar o grau de refinamento da minha malha o que está intimamente ligado à memória do meu processador.

2º Particionar as matrizes D e N e o vetor g:

Distribuindo entre os processadores as linhas desta, de modo que cada um fique responsável pelo bloco que recebeu desta e a partir daí compute a expressão da equação de Jacobi iterativa por blocos de maneira paralela.

3º Montagem das matrizes D e N:

Consiste em escrever as matrizes D e N, que compõem a matriz H (matriz que determina a distorção sofrida pela imagem) e o vetor g independente da dimensão (isto é, do número de pixels associados à discretização).

4º Processar em paralelo e obter a solução

Computar a expressão da equação de Jacobi iterativa em cada processador, simultaneamente, e por final concatenar os resultados parciais (vetores calculados em cada processador) formando um único vetor.

5º Calcular o erro do método de Jacobi

Apartir do vetor solução, com o valor dos pixels da imagem restaurada, calcular o erro por meio da norma infinita parcial.

No algoritmo paralelo, será usada alocação dinâmica de memória visto que as matrizes usadas para armazenar os valores dos pixels que compõe a imagem distorcida tomarão dimensões variadas de acordo com o número de processadores e dimensão das imagens processadas.

Seguindo as regras de uma modelagem paralela, primeiramente será tratada a decomposição do problema em subproblemas.

No programa paralelo há etapas que são executadas em paralelo, isto é tarefas que são particionadas, distribuídas e executadas em todos os processadores simultaneamente e outros que somente podem ser executados seqüencialmente.

Etapas do procedimento que podem ser processadas em paralelo

- Alocação de memória para as matrizes D , N e vetores g , f_{nov} , f_{antigo} , erros, faux , z e r . As matrizes N e D e os vetores faux , z e r são divididos em blocos, isto é submatrizes e subvetores, (N_p , D_p , faux_p , z_p e r_p respectivamente) de maneira que cada processador fique “responsável” pela manipulação de determinadas partições das matrizes e vetores, cujo número de linhas de cada submatriz alocada em cada processador, depende da dimensão da imagem processada e do número de processadores.
- Determinar a linha de início e fim de cada bloco pelo qual cada processador ficará responsável das matrizes D , N e dos vetores faux , z e r , respectivamente as submatrizes N_p e D_p e os subvetores faux_p , z_p e r_p conforme mostra a Figura 5.1.

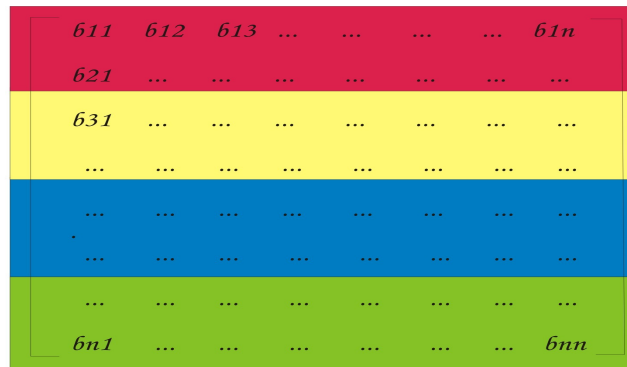


Figura 5.1: Particionamento das matrizes.

- Montagem das matrizes N e D em cada processador (respectivamente N_p e D_p), cada um com sua respectiva partição das matrizes.
- Cálculo da expressão em cada processador: $\text{faux} = D_p^{-1} N_p f_{\text{antigo}} + D_p^{-1} g$, de maneira que cada processador fique “responsável” pelo cálculo da expressão em sua respectiva partição, armazenando o resultado localmente no vetor faux , para que posteriormente haja a concatenação dos faux de todos os processadores, conforme mostra a Figura 5.2.
- Cálculo da expressão que define o critério de parada: $\|z_p\| = \|f_{\text{novop}}\| - \|f_{\text{antigop}}\|$ e da norma infinito dos subvetores z_p (através da seleção do maior dos elementos em cada subvetor) em cada processador (norma infinito parcial), conforme mostra a Figura 5.3.
- Cálculo da expressão que define o erro em cada processador: $\|r_p\| = \|Hf - g\| = \|(D_p - N_p) * f_{\text{novop}}\|$ e da norma infinito dos subvetores r_p em cada processador (norma infinito parcial).

$$\begin{array}{c}
 \begin{array}{c}
 \text{faux}_{00} \\
 \dots \\
 \text{faux}_{100} \\
 \dots \\
 \text{faux}_{200} \\
 \dots \\
 \text{faux}_{n0}
 \end{array}
 =
 \begin{array}{c}
 \begin{array}{c}
 d_{p00} \ 0 \ \dots \ \dots \ \dots \ \dots \ 0 \\
 0 \ d_{p11} \ 0 \ \dots \ \dots \ \dots \ \dots \\
 \dots \ 0 \ \dots \ 0 \ \dots \ \dots \ \dots \\
 \dots \ \dots \ 0 \ d_{p100} \ 0 \ \dots \ \dots \\
 \dots \ \dots \ \dots \ \dots \ 0 \ \dots \ \dots \\
 \dots \ \dots \ \dots \ \dots \ \dots \ 0 \ \dots \ 0 \\
 0 \ \dots \ \dots \ \dots \ \dots \ 0 \ d_{pn0}
 \end{array}
 \begin{array}{c}
 C \ \dots \ \dots \ \dots \ -r_{p0n} \ \dots \ \dots \ C \\
 \dots \ C \ \dots \ \dots \ \dots \ -r_{p2n} \ \dots \ \dots \\
 \dots \ \dots \ \dots \ \dots \ C \ \dots \ \dots \ -r_{p10n} \\
 \dots \ \dots \ \dots \ \dots \ \dots \ \dots \ \dots \\
 -r_{p3n} \ \dots \ \dots \ \dots \ \dots \ \dots \ \dots \\
 \dots \ -r_{p21n} \ \dots \ \dots \ C \ \dots \ \dots \\
 \dots \ \dots \ \dots \ \dots \ \dots \ \dots \ \dots \\
 C \ \dots \ \dots \ -r_{pn0} \ \dots \ \dots \ \dots \ C
 \end{array}
 \begin{array}{c}
 f_antigo_{00} \\
 \dots \\
 \dots \\
 f_antigo_{100} \\
 \dots \\
 f_antigo_{200} \\
 \dots \\
 f_antigo_{n0}
 \end{array}
 +
 \end{array}
 +
 \begin{array}{c}
 \begin{array}{c}
 d_{p00} \ 0 \ \dots \ \dots \ \dots \ \dots \ 0 \\
 0 \ d_{p11} \ 0 \ \dots \ \dots \ \dots \ \dots \\
 \dots \ 0 \ \dots \ 0 \ \dots \ \dots \ \dots \\
 \dots \ \dots \ 0 \ d_{p100} \ 0 \ \dots \ \dots \\
 \dots \ \dots \ \dots \ \dots \ 0 \ \dots \ \dots \\
 \dots \ \dots \ \dots \ \dots \ 0 \ \dots \ 0 \\
 0 \ \dots \ \dots \ \dots \ \dots \ 0 \ d_{pn0}
 \end{array}
 \begin{array}{c}
 g_{00} \\
 \dots \\
 \dots \\
 g_{100} \\
 \dots \\
 g_{200} \\
 \dots \\
 g_{n0}
 \end{array}
 \begin{array}{l}
 \text{Processador 0} \\
 \text{Processador 1} \\
 \text{Processador 2} \\
 \text{Processador 3} \\
 \text{Todos}
 \end{array}
 \end{array}$$

Figura 5.2: Cálculo da expressão iterativa de Jacobi, $\text{faux} = D_p^{-1} N_p f_{\text{antigo}} + D_p^{-1} g$, em cada processador.

$$Z = \text{abs} \left(\begin{array}{c}
 \begin{array}{c}
 x_novo_{00} \\
 \dots \\
 x_novo_{100} \\
 \dots \\
 x_novo_{200} \\
 \dots \\
 x_novo_{n0}
 \end{array}
 \begin{array}{c}
 x_antigo_{00} \\
 \dots \\
 x_antigo_{100} \\
 \dots \\
 x_antigo_{200} \\
 \dots \\
 x_antigo_{n0}
 \end{array}
 \begin{array}{c}
 z_{00} \\
 \dots \\
 z_{100} \\
 \dots \\
 z_{200} \\
 \dots \\
 z_{n0}
 \end{array}
 \end{array} \right)
 \begin{array}{l}
 \rightarrow Z_0 \\
 \rightarrow Z_1 \\
 \rightarrow Z_2 \\
 \rightarrow Z_3
 \end{array}$$

Figura 5.3: Cálculo da expressão que define o critério de parada: $\|z_p\| = \|f_{\text{novop}}\| - \|f_{\text{antigop}}\|$ e da norma infinito dos subvetores z_p em cada processador.

Etapas do procedimento que somente podem ser executados sequencialmente

- Dividir o total de linhas das matrizes e vetores pelo número de processadores disponíveis a fim de determinar a quantidade de linhas que caberá a cada um manipular
- Leitura da matriz de pixels da imagem distorcida, que se encontra armazenada num arquivo “glImage” e arranjo lexicográfico desta carregando-a num vetor g . Somente um processador, o “mestre” (processador zero), deve executar essa leitura.
- Cálculo da norma infinito do vetor erro (vetor resultante da concatenação da norma infinito dos subvetores z_p calculada em cada processador), no processador “mestre” (norma infinito total) e comparação com o erro máximo definido no algoritmo. O cálculo da norma infinito nada mais é que a seleção do maior dos elementos do vetor.
- Cálculo da norma infinito total do resíduo (calculada a partir das normas infinito parciais recebidas de cada processador) no processador “mestre”, e comparação com o erro máximo definido, conforme mostra a Figura 5.4.

Etapas do procedimento onde se faz necessária a comunicação entre processadores

- Após a divisão do total de linhas das matrizes e vetores por entre os processadores disponíveis, determinando a quantidade de linhas pela qual cada processador ficará responsável por manipular (trecho executado sequencialmente) deve haver uma comunicação a todos os processadores da quantidade que cabe a cada um.
- Concatenar todos os subvetores faux (resultantes do cálculo da expressão $\text{faux} = D_p^{-1} N_p f_{\text{antigo}} + D_p^{-1} g$ em cada processador) no vetor f_{novo} e enviar a todos os processadores.
- Envio das normas infinito parciais dos subvetores z_p , calculadas em cada processador, para o “processador mestre”.

- Envio das normas infinito parciais dos subvetores r_p , calculadas em cada processador, para o “processador mestre”.

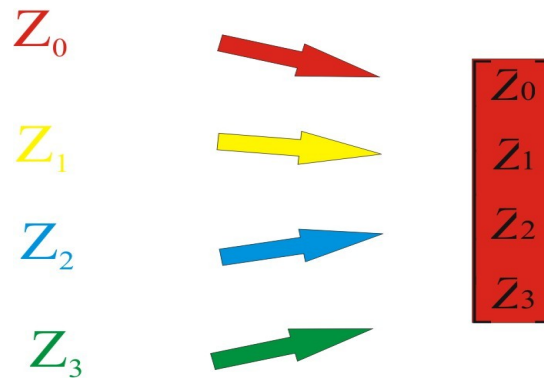


Figura 5.4: Envio das normas infinito parciais calculadas em cada processador para o mestre.

- Envio de um sinal “stop” para todos os processadores, a fim de que sejam encerrados todos os processos e saiam do loop principal, caso o resultado da norma infinito total do resíduo (calculada a partir das normas infinito parciais recebidas de cada processador) seja menor que o valor de erro máximo definido.

5.2 – Cálculo do Erro

Conforme dito e exemplificado no esquema da Figura 5.3, inicialmente calcularemos em cada processador a norma infinito parcial do vetor $z = f_{(k+1)} - f_{(k)} = f_{\text{novo}} - f_{\text{antigo}}$, correspondente ao bloco pelo qual cada um ficou responsável ($z = f_{\text{novo}} - f_{\text{antigo}}$), enviando a seguir o resultado para o “processador mestre”. Este por sua vez calcula a norma infinito total a partir de todas as normas parciais recebidas de todos os outros processadores, conforme mostra a Figura 5.4.

Apartir daí, caso o resultado da norma seja menor que o valor de erro máximo considerado no problema, calcula-se o resíduo ($\|r\| = \|Hf-g\| = \|(D-N)*f_{\text{novo}}\|$) e a norma infinito parcial correspondente ao bloco pelo qual cada processador ficou responsável por calcular ($\|r_p\| = \|Hf-g\| = \|(D_p-N_p)*f_{\text{novo}}\|$) enviando a seguir o resultado

para o “processador mestre”. Este por sua vez calcula a norma infinito total, a partir de todas as normas parciais recebidas de todos os outros processadores. Caso contrário retorna-se ao cálculo da expressão iterativa de Jacobi.

Caso o resultado da norma do resíduo seja menor que o valor máximo de erro considerado no problema termina-se o loop e toma-se o vetor $f_{\text{nov}}o$ como solução. Caso contrário retorna-se ao cálculo da expressão iterativa de Jacobi.

5.3 – Passos do Algoritmo Paralelo

Sabendo que $H = D - N$ e que a solução de Jacobi ficou da forma

$f_{\text{nov}}o = D^{-1} * N * f_{\text{antigo}} + D^{-1} * g$, pode ser implementada uma solução paralela do problema da seguinte forma:

- a. Receber a dimensão da imagem distorcida.
- b. Em cada processador é alocado espaço de memória para as matrizes D, N e vetores g, $f_{\text{nov}}o$, f_{antigo} , erros, faux, z e r.
- c. Dividir o total de linhas da matriz D, N e dos vetores faux, z e r e comunicar a todos os processadores a quantidade de linhas que cabe a cada um manipular.
- d. Determinar a linha de início e fim de cada partição das matrizes N, D e vetores g, $f_{\text{nov}}o$, f_{antigo} , erros, faux, z e r que caberá a cada processador manipular.
- e. Montagem em cada processador das respectivas partições das matrizes N, D e vetores g, $f_{\text{nov}}o$, f_{antigo} , erros, faux, z e r (N_p , D_p).
- f. Arbitrar valores iniciais para $f_{\text{nov}}o$ e zerar f_{antigo} em cada processador.
- g. Calcular $\text{faux} = D_p^{-1} N_p f_{\text{antigo}} + D_p^{-1} g$ em cada processador.
- h. Concatenar todos os subvetores faux no vetor $f_{\text{nov}}o$ e enviar a todos os processadores.
- i. Cálculo da expressão que define o critério de parada: $\|z\| = \|f_{\text{nov}}o\| - \|f_{\text{antigo}}\|$ em cada processador ($\|z_p\| = \|f_{\text{nov}}o_p\| - \|f_{\text{antigo}}_p\|$) e da norma infinito dos subvetores z_p em cada processador (norma infinito parcial).

- j. Enviar todas as normas infinito parciais (z_p) para o “processador mestre” (processador ‘0’).
- k. No processador mestre, calcular a norma infinito total de z a partir das normas infinito parciais, recebidas de cada processador.
- l. Se o resultado da norma for maior que uma tolerância definida, calcula-se o resíduo $r=Hf-g$ em cada processador ($\|r_p\| = \|Hf-g\|=\|(D_p-N_p)*f_{novo}\|$) e da norma infinito dos subvetores r_p em cada processador (norma infinito parcial)
- m. Caso contrário volta-se à etapa da letra (f).
- n. Enviar todas as normas infinito parciais (z_p) para o “processador mestre” (processador ‘0’) .
- o. No processador mestre, calcular a norma infinito total do resíduo a partir das normas infinito parciais, recebidas de cada processador.
- p. Se a norma do resíduo, por sua vez, for menor que um valor máximo de erro estabelecido volta-se à etapa da letra (f).
- q. Caso contrário são encerrados todos os processos e é salvo o vetor f_{novo} na forma de matriz num arquivo “imageFile”.

Abaixo temos um esquema mais geral das etapas pelas quais o processo de restauração de imagens distorcidas passa.

Fluxograma de etapas do processo de recuperação de imagens

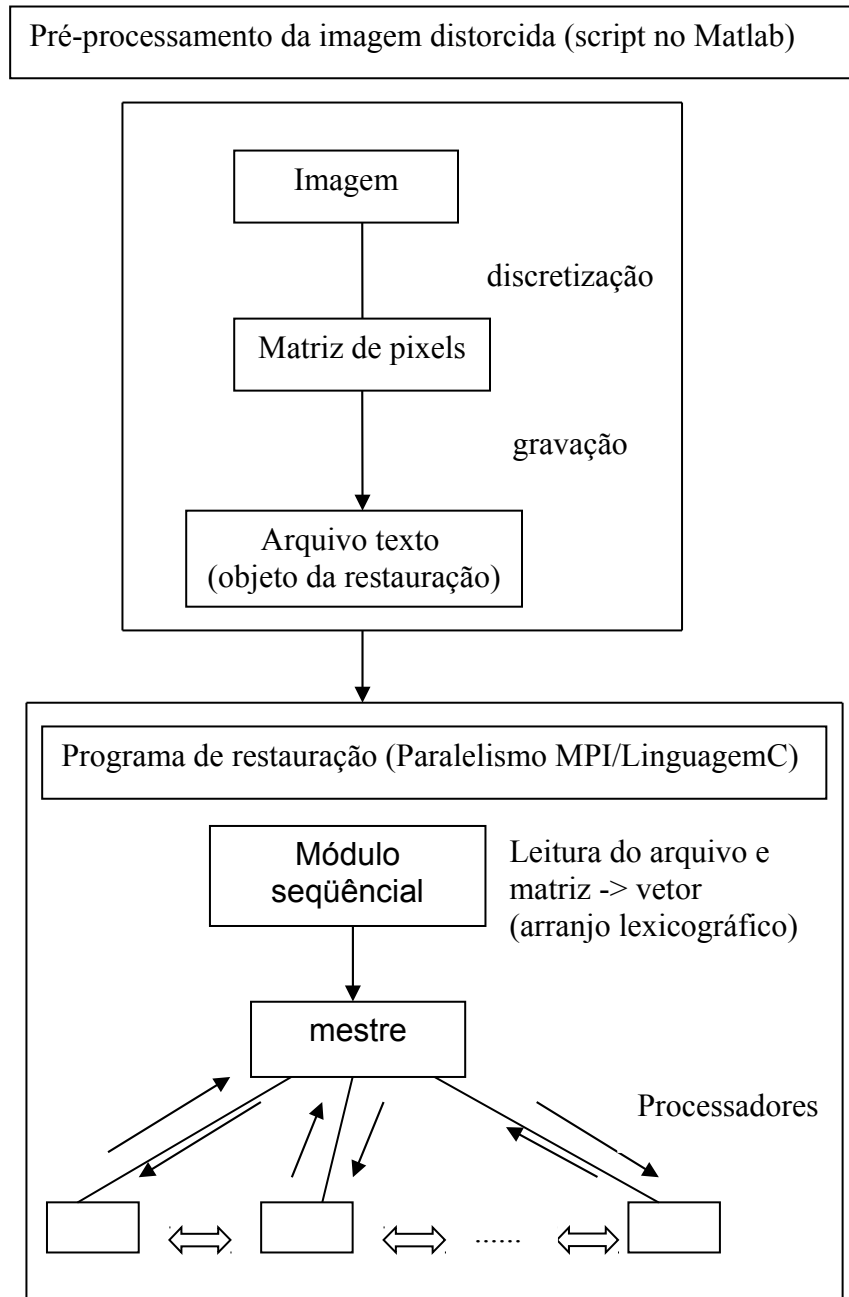


Figura 5.5: Fluxograma das etapas do processo de recuperação de imagens.

5.4 – Especificações da máquina cluster utilizada e dos processadores

Será utilizada a máquina 'Cluster Itaotec' (do laboratório do NACAD), que é um cluster de processadores para executar os experimentos deste projeto.

Este cluster possui arquitetura de memória distribuída, na tabela abaixo encontramos toda especificação do cluster e informações mais detalhadas a respeito.

Cluster InfoServer-Itaotec:

Este cluster possui 16 nós duais de processamento (cada nó com dois processadores) cujos processadores são Intel Pentium III de 1GHz.

Memória principal por nó: 512MB de memória RAM e cache de 256KB por CPU.

Memória Total: 8.0 Gbytes (distribuída).

Rede: dedicada com tecnologia Fast-Ethernet (100 Mbits/s).

Sistema Operacional: Linux distribuição RedHat 7.3.

Compiladores: Fortran-77, Fortran-90 e C/C++.

Ferramentas para desenvolvimento de aplicações paralelas.



Figura 5.6 – Cluster utilizado para executar o programa paralelo de restauração de imagens distorcidas.

O Cluster Infoserver Mercury instalado no NACAD possui 16 nós, denominados node1 a node16, 1 estação de administração, denominada adm e 1 estação de acesso, denominada acc1.

Cada um dos nós, incluindo as estações **adm** e **acc1**, é uma estação de trabalho completa, com duas CPUs, memória RAM, disco local e suas próprias interfaces de rede.

Os 16 nós são interligados por uma rede fast ethernet, dedicado exclusivamente para a execução de programas paralelos.

A tabela abaixo ilustra as principais características de cada um dos nós:

---	processador1 a processador32	processadores da estação adm	Processadores da estação acc1
processor type	Pentium III	Pentium III	Xeon
clock frequency	1 Ghz	1 Ghz	1 Ghz
cache size	256 KB	256 KB	256 KB
RAM memory	512 MB	512 MB	512 MB
disk storage	18.0 GB	160 GB	18.0 GB

Tabela 5.1: Principais características de cada um dos nós que compõem a máquina cluster.

As informações acima foram retiradas do site do laboratório NACAD [18].

Capítulo 6 – Análise de resultados computacionais

6.1 – Aplicação do algoritmo à diferentes imagens distorcidas

Tomando a seguinte imagem distorcida como problema exemplo, tentar-se-á restaurá-la.



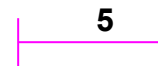
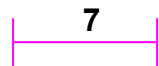
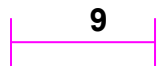
Figura 6.1: Imagem distorcida que tentaremos restaurar.

Observa-se que apartir do que foi concluído na seção 2.6 a respeito de imagens distorcidas e sobrepostas, analisando a imagem acima se verifica que é resultado de uma sobreposição de três imagens: a imagem original, uma outra imagem semelhante à original, porém deslocada alguns pixels para a direita e uma última imagem, bem enfraquecida quase imperceptível ao fundo branco, deslocada alguns pixels para a esquerda. Trata-se de uma distorção do tipo deslocamento, isto é provavelmente o pássaro deveria estar em movimento quando sua imagem foi capturada.

Nota-se também que a intensidade das imagens deslocadas, isto é dos sombreamentos que aparecem à direita e à esquerda da imagem central do pássaro, é bem inferior à da imagem do pássaro central.

A fim de tentar restaurá-la, considerando que houve um deslocamento em dois sentidos e pela aparência desta imagem, arbitramos uma matriz de distorção com deslocamento de 10 pixels para cada um dos lados em relação à imagem original, porém o algoritmo utilizado demorou muito tempo executando e não convergiu. Optou-se então por diminuir o número mínimo de iterações para o cálculo do algoritmo de Jacobi.

Foi executado então o algoritmo de recuperação com matrizes de distorção do tipo deslocamento, porém com deslocamentos variados e intensidades também variadas para as imagens deslocadas. Não esquecendo, porém de preservar a energia do sistema, conforme dito na seção 2.5, por isso foi normalizada cada linha da matriz, dividindo-a pelo somatório do valor dos elementos da linha.

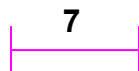


$$H_4 = \begin{bmatrix} 1 & 0 & \dots & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & \dots & 0 & 1 & 0 & 0 & 0 \\ \dots & 0 & 1 & 0 & \dots & 0 & 1 & 0 & 0 \\ 1 & \dots & 0 & 1 & 0 & \dots & 0 & 1 & 0 \\ 0 & 1 & \dots & 0 & 1 & 0 & \dots & 0 & 1 \\ 0 & 0 & 1 & \dots & 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 0 & 1 & \dots & 0 & 1 & 0 & \dots \\ 0 & 0 & 0 & 0 & 1 & \dots & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & \dots & 0 & 1 \end{bmatrix}, H_5 =$$

$$H_6 = \begin{bmatrix} 1 & 0 & \dots & 0 & 0.5 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & \dots & 0 & 0.5 & 0 & 0 & 0 \\ \dots & 0 & 1 & 0 & \dots & 0 & 0.5 & 0 & 0 \\ 0.5 & \dots & 0 & 1 & 0 & \dots & 0 & 0.5 & 0 \\ 0 & 0.5 & \dots & 0 & 1 & 0 & \dots & 0 & 0.5 \\ 0 & 0 & 0.5 & \dots & 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0.5 & \dots & 0 & 1 & 0 & \dots \\ 0 & 0 & 0 & 0 & 0.5 & \dots & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.5 & \dots & 0 & 1 \end{bmatrix}, H_6 =$$

$$\begin{bmatrix} 1 & 0 & \dots & 0 & 0.4 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & \dots & 0 & 0.4 & 0 & 0 & 0 \\ \dots & 0 & 1 & 0 & \dots & 0 & 0.4 & 0 & 0 \\ 0.4 & \dots & 0 & 1 & 0 & \dots & 0 & 0.4 & 0 \\ 0 & 0.4 & \dots & 0 & 1 & 0 & \dots & 0 & 0.4 \\ 0 & 0 & 0.4 & \dots & 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0.4 & \dots & 0 & 1 & 0 & \dots \\ 0 & 0 & 0 & 0 & 0.4 & \dots & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.4 & \dots & 0 & 1 \end{bmatrix}$$

7



$$e \ H_7 = \begin{bmatrix} 1 & 0 & \dots & 0 & 0.3 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & \dots & 0 & 0.3 & 0 & 0 & 0 \\ \dots & 0 & 1 & 0 & \dots & 0 & 0.3 & 0 & 0 \\ 0.3 & \dots & 0 & 1 & 0 & \dots & 0 & 0.3 & 0 \\ 0 & 0.3 & \dots & 0 & 1 & 0 & \dots & 0 & 0.3 \\ 0 & 0 & 0.3 & \dots & 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0.3 & \dots & 0 & 1 & 0 & \dots \\ 0 & 0 & 0 & 0 & 0.3 & \dots & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.3 & \dots & 0 & 1 \end{bmatrix}$$

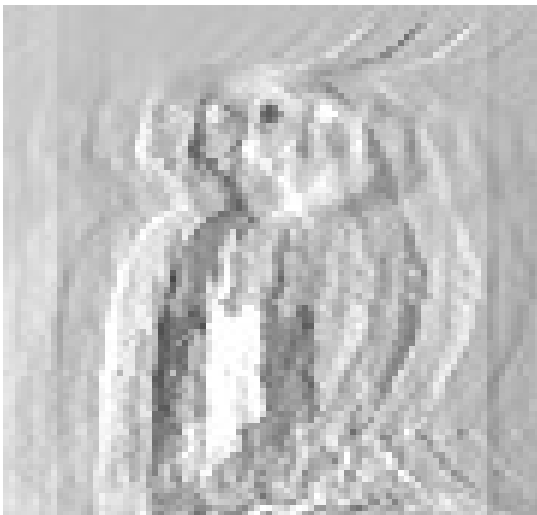


Figura 6.2: Imagem restaurada utilizando a matriz H_1 .

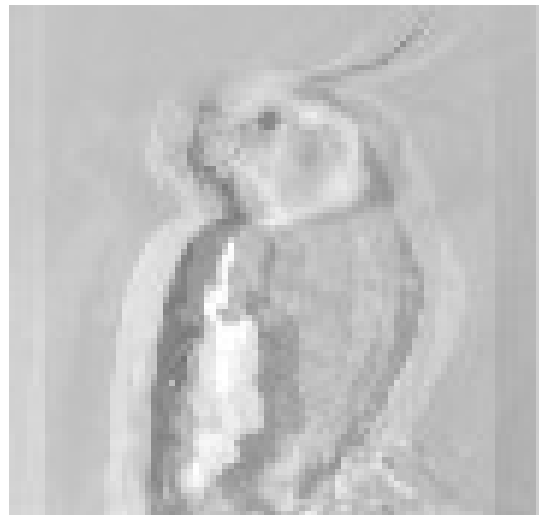


Figura 6.3: Imagem restaurada utilizando a matriz H_2 .

Utilizando a matriz H_1 em que caracteriza uma matriz deslocamento de 9 colunas entre as imagens original e deslocadas observa-se pela Figura 6.2 que não são mais três imagens, mas agora 5 (a original e mais 2 pares de deslocadas cada uma com um deslocamento diferente), ao invés de recuperar deslocou-se mais uma vez a imagem original. Portanto 9 colunas foi um deslocamento muito grande além disso a imagem ficou muito clara logo o fator de intensidade das imagens deslocadas deve ser menor que 1.0.

Tenta-se então agora com a matriz H_2 , com um deslocamento de 7 colunas observa-se pela Figura 6.3 uma grande melhora na imagem distorcida. Provavelmente deve ser esse o número de colunas deslocadas entre a imagem original e as deslocadas.

Submetendo a Figura 6.1 agora, à matriz H3 e H4, com um deslocamento de 5 e 3 colunas respectivamente observamos como na Figura 6.2 a formação de mais 1 par de imagens deslocadas. Portanto utilizando o deslocamento de 7 colunas, que recuperou melhor a imagem da Figura 6.1, e escurecendo mais as imagens deslocadas (diminuindo o fator de multiplicação), sera submetida agora à matriz H5.

Observando a Figura 6.6 em comparação à 6.3, o contraste entre o fundo cinza da imagem e o pássaro vai aumentando, o detalhe branco da asa aparece mais claro e o contraste do branco e preto, no rosto do pássaro também se fez mais nítido. E por fim submetendo às matrizes H6 e H7 que escurecem as imagens deslocadas com os fatores 0.4 e 0.3 respectivamente, se verifica uma recuperação muito boa da imagem, inicialmente distorcida.

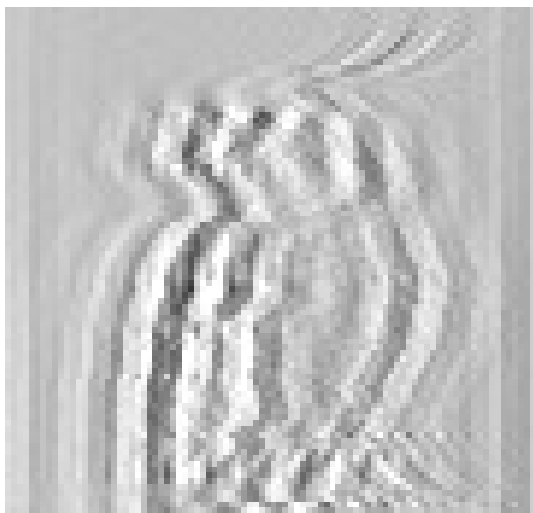


Figura 6.4: Imagem restaurada utilizando a matriz H_3 .

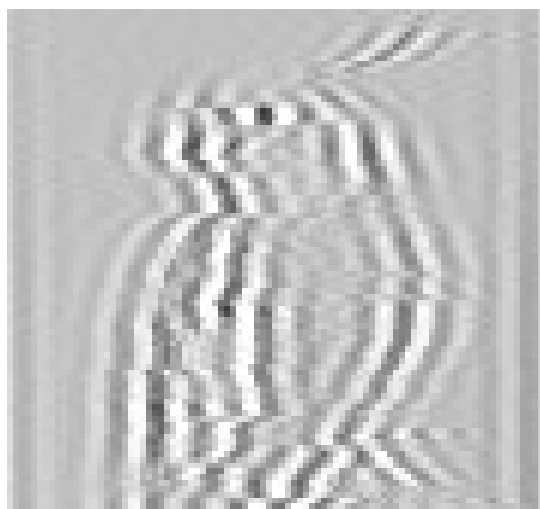


Figura 6.5: Imagem restaurada utilizando a matriz H_4 .



Figura 6.6: Imagem restaurada utilizando a matriz H_5 .



Figura 6.7: Imagem restaurada utilizando a matriz H_6 .

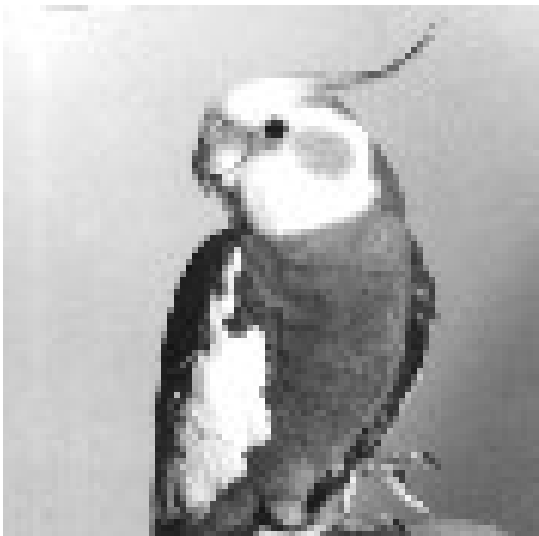


Figura 6.8: Imagem restaurada utilizando a matriz H_7 .



Figura 6.9: Imagem inicialmente distorcida.

A restauração de imagens nada mais é do que apartir de uma primeira impressão da imagem, inferir o tipo de distorção que esta sofreu e daí em diante tentar restaurá-la com diversas configurações de matriz distorção, arbitrando diferentes valores de deslocamento e diferentes fatores de clareamento ou escurecimento, se for o caso.

Tentando encontrar assim, pelo método das tentativas, uma matriz distorção que melhor se encaixa a distorção sofrida pela imagem, obtendo assim a melhor imagem restaurada, dentro do possível.

6.2 – Curva de desempenho e verificação do controle de qualidade

Utilizando como objeto de testes a imagem distorcida da seção 6.1, será testado o ganho de desempenho do algoritmo paralelo sobre o seqüencial, de restauração de imagens, conforme aumenta a dimensão da imagem a ser restaurada e o número de processadores em que será executado o algoritmo.

Abaixo é apresentada uma tabela, com os valores do tempo demandado pelos algoritmos paralelos e seqüenciais em função da dimensão da imagem distorcida e do número de processadores utilizados para tal. E em seguida o gráfico das curvas de desempenho, “speed-up” X número de processadores e eficiência X número de processadores respectivamente, curva de “speed-up” e de eficiência. O resultado nos mostra claramente a vantagem de utilizar o algoritmo paralelo neste caso.

Número de processadores	dimensao 60	dimensao 90	dimensao 120	dimensao 150	“Speed-up” ideal
	Ts/Tp	Ts/Tp	Ts/Tp	Ts/Tp	Ts/Tp
4	3.590	3.880	3.850	3.900	4.000
5	4.400	4.802	4.750	4.880	5.000
6	5.140	5.770	5.720	5.825	6.000
8	5.970	6.703	7.390	7.583	8.000
10	7.090	8.110	9.050	9.159	10.000
12	8.290	9.972	10.830	11.300	12.000

Tabela 6.1: Tabela de “speed-up”.

Número de processadores	dimensao 60	dimensao 90	dimensao 120	dimensao 150	“Speed-up” ideal
	Ep	Ep	Ep	Ep	Ep
4	89.750	97.000	96.250	97.500	100.000
5	88.000	96.040	95.000	97.600	100.000
6	85.667	96.167	95.333	97.083	100.000
8	74.625	83.788	92.375	94.788	100.000

10	70.900	81.100	90.500	91.590	100.000
12	69.083	83.100	90.250	94.167	100.000

Tabela 6.2: Tabela de eficiência.

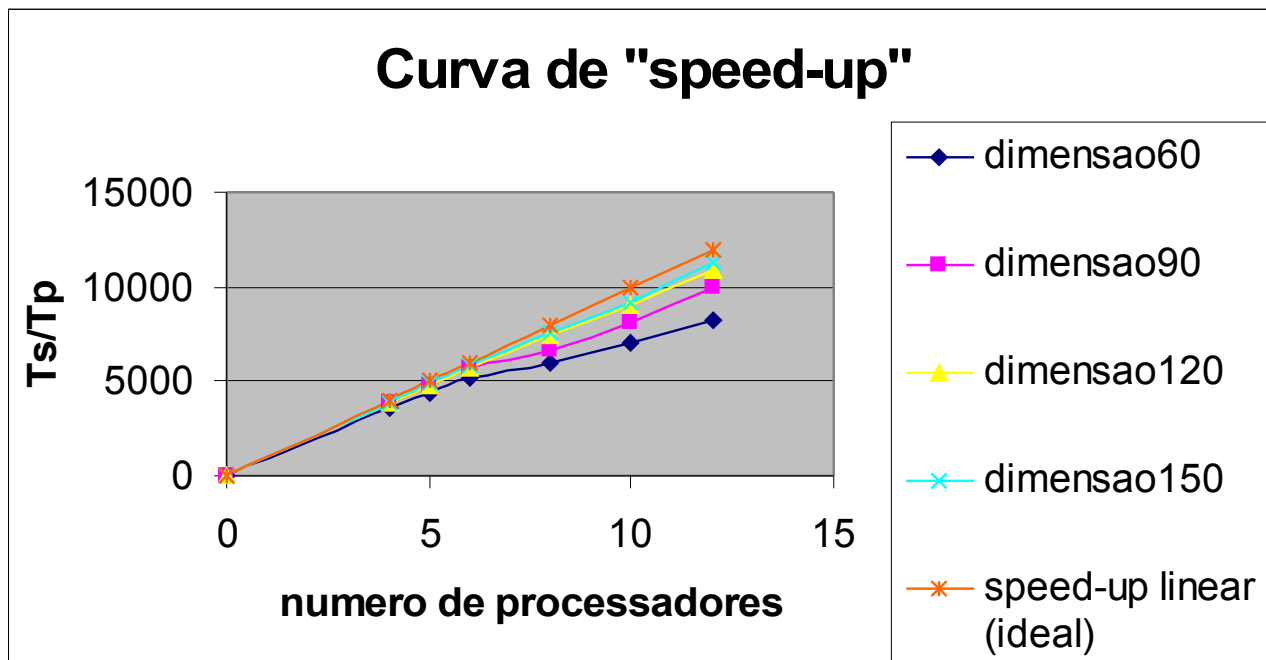


Figura 6.10: Curva de "speed-up".

Na Figura 6.7 acima é possível verificar que o "speed-up" vai aumentando conforme aumenta o número de processadores utilizado para uma dimensão fixa da imagem.

Analisando as curvas para imagens de dimensão 60X60 até 256X256, observa-se que o "speed-up" aumenta conforme aumenta o número de processadores, isto se deve ao fato de que quanto maior o número de processadores melhor se verifica a divisão de tarefas e o paralelismo.

Observa-se também através dos gráficos que para um dado número de processadores quanto maior a dimensão da imagem maior o "speed-up", isto é, melhor o desempenho do algoritmo paralelo sobre o seqüencial. Isto se faz verdadeiro já que com uma dimensão baixa da imagem, é observado mais tempo de comunicação do que de computação (apresentando baixo desempenho neste caso).

Conforme mostra a curva de eficiência da Figura 6.8, foi alcançada uma eficiência bem próxima do ideal.

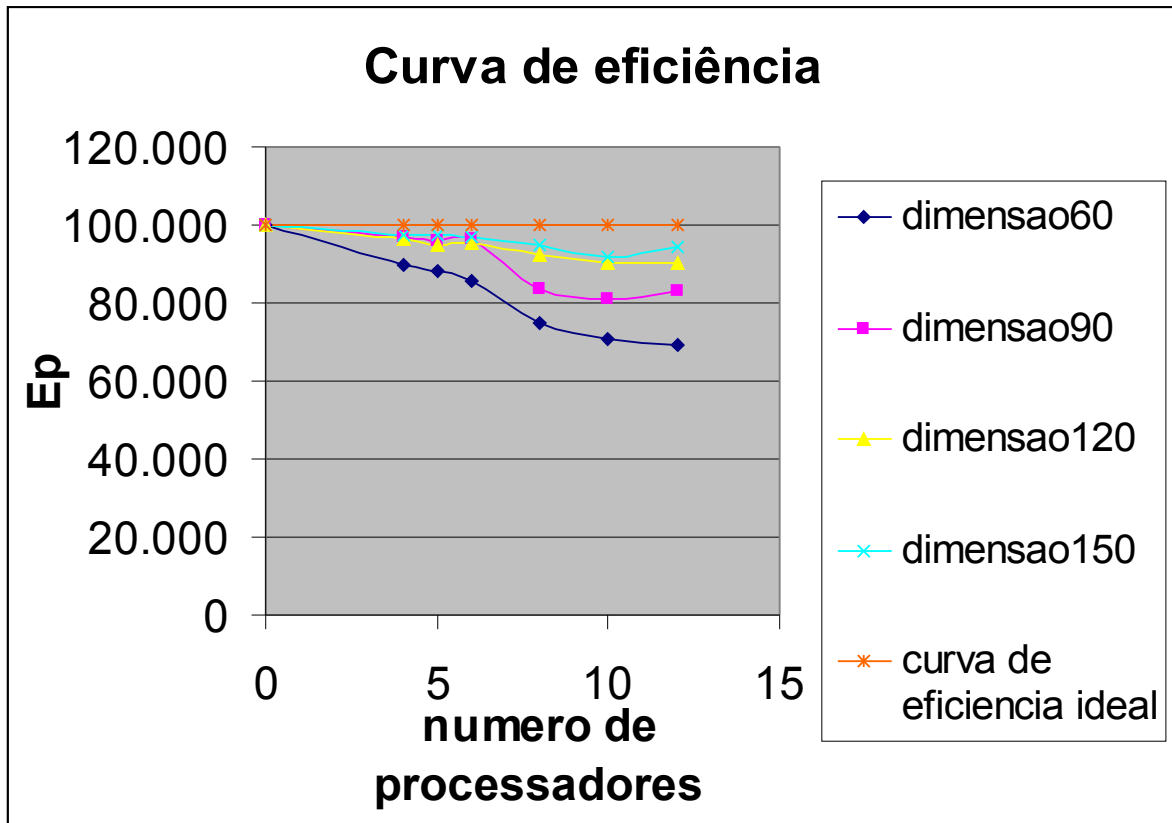


Figura 6.11: Curva de eficiência.

Capítulo 7 – Conclusão

Neste trabalho foi proposto viabilizar a restauração de imagens que sofreram efeito de degradação, utilizando para isso implementações com redes neurais (método iterativo de Jacobi, que consiste em uma especialização de redes neurais) e algoritmos paralelos implementados em computadores paralelos.

Este algoritmo deve ser capaz de recuperar qualquer imagem que sofreu degradações de natureza conhecida, no momento de sua captura.

Este processo de recuperação de imagens foi dividido em três etapas. Primeiramente o pré-processamento da imagem distorcida através de um script feito no Matlab, em que é feita a discretização da imagem a ser processada transformando-a numa matriz de pixels. Este por sua vez é submetido a um programa de restauração que faz uso da biblioteca MPI é feito utilizando a linguagem C.

O programa paralelo é composto de um módulo seqüencial, que faz a leitura do arquivo onde está a matriz de pixels da imagem a ser restaurada e arranjo lexicográfico desta. Logo depois os dados são processados de maneira distribuída e paralela em tarefas, pelos processadores controlados pelo “mestre” (processador zero) e por fim após a recuperação da imagem dentro de um padrão de erro aceitável, os pixels restaurados são armazenados na forma de uma matriz num arquivo texto, o qual será submetido ao Matlab para que possibilite a visualização da imagem restaurada.

Com base nos resultados alcançados no capítulo anterior foi comprovada a viabilidade e também a eficácia do algoritmo paralelo para o problema de restauração de imagens distorcidas, principalmente para grandes dimensões da imagem, mediante a grande quantidade de dados a ser processada.

Foi verificado através de resultados, a restauração das imagens distorcidas, utilizadas as matrizes distorção do tipo deslocamento, ausência ou excesso de iluminação.

Porém, apesar dos resultados, outros métodos, dentre eles “B.T. Polyak”, se mostram mais eficazes na restauração de imagens com outros tipos de distorção

como àquelas causadas por uma matriz distorção do tipo Toeplitz. O método de Jacobi iterativo utilizado, ao tentar restaurar uma imagem cuja distorção é caracterizada por uma matriz Toeplitz, o valor do erro diverge.

Dentre as dificuldades encontradas durante a execução deste trabalho estão a mudança de paradigma de uma programação seqüencial para paralela e chegar às diferentes configurações das matrizes de distorção.

A programação seqüencial é a maneira mais usual de programar, porém atualmente com o aumento da quantidade de dados a ser processada a programação em paralelo se mostrou mais vantajosa. Porém o modelo, padrão de programação paralela difere significativamente do seqüencial.

Quanto ao "performance debugging" este não foi feito. Poderiam ter sido usadas técnicas de "esparsidade de matrizes" (técnicas que consideram a existência de zeros nas operações entre matrizes, por exemplo, não fazendo multiplicações desnecessárias por zero). Se, neste trabalho, tivessem sido processadas imagens de dimensão bem mais elevada, não seria possível alocar memória suficiente em cada processador, mesmo utilizando o algoritmo paralelo, nas máquinas disponíveis para o processamento. A quantidade de memória necessária para o processamento, mais do que o tempo de processamento, obriga a utilizar a esparsidade de matrizes e com isso viabilizando assim o processamento de imagens muito maiores, o que não é possível tratar com o algoritmo seqüencial.

Além disso, poderia ter havido uma maior paralelização, isto é um maior particionamento das tarefas durante o cálculo da expressão de Jacobi, $f_{\text{novo}} = D^{-1} * N * f_{\text{antigo}} + D^{-1} * g$, particionando as matrizes D e N não somente por linhas mas por colunas também, assim o que antes formava uma partição em cada matriz agora teria várias partições e por conseqüente conseguindo também, particionar os vetores f_{antigo} e g por linhas conforme mostra a Figura 7.1.

Certamente se tivesse sido feito um "performance debugging" melhoraria o desempenho da implementação paralela.

Como sugestões para trabalhos futuros além do já mencionado no parágrafo anterior, que fosse melhorado o algoritmo considerando a influência não só das fontes de distorção que degradam no momento da captura da imagem, mas a influência do ruído também como fonte de distorção, o qual pode ser causado pelo

próprio sistema de imagens (por impurezas presentes nas lentes da câmera) ou pelo meio de transmissão ou gravação.

$$\begin{bmatrix} f_{aux00} \\ \dots \\ f_{aux100} \\ \dots \\ f_{aux200} \\ \dots \\ f_{auxn0} \end{bmatrix} = \begin{bmatrix} d_{p00} & 0 & \dots & \dots & \dots & \dots & \dots & 0 \\ 0 & d_{p11} & 0 & \dots & \dots & \dots & \dots & \dots \\ \dots & 0 & \dots & 0 & \dots & \dots & \dots & \dots \\ \dots & \dots & 0 & d_{p100} & 0 & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & 0 & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & 0 & d_{p200} & 0 & \dots \\ 0 & \dots & \dots & \dots & \dots & 0 & d_{pn0} & \dots \end{bmatrix}^{-1} \begin{bmatrix} C & \dots & \dots & \dots & -r_{p100} & \dots & \dots & 0 \\ \dots & C & \dots & \dots & \dots & -r_{p121} & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & C & \dots & \dots & \dots & -r_{p10n} \\ -r_{p200} & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & -r_{p21} & \dots & \dots & \dots & C & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ C & \dots & \dots & -r_{pn0} & \dots & \dots & \dots & 0 \end{bmatrix} \begin{bmatrix} f_{antigo00} \\ \dots \\ f_{antigo100} \\ \dots \\ f_{antigo200} \\ \dots \\ f_{antigo_n0} \end{bmatrix} + \begin{bmatrix} d_{p00} & 0 & \dots & \dots & \dots & \dots & \dots & 0 \\ 0 & d_{p11} & 0 & \dots & \dots & \dots & \dots & \dots \\ \dots & 0 & \dots & 0 & \dots & \dots & \dots & \dots \\ \dots & \dots & 0 & d_{p100} & 0 & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & 0 & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & 0 & d_{p200} & 0 & \dots \\ 0 & \dots & \dots & \dots & \dots & 0 & d_{pn0} & \dots \end{bmatrix}^{-1} \begin{bmatrix} g_{00} \\ \dots \\ g_{100} \\ \dots \\ g_{200} \\ \dots \\ g_{n0} \end{bmatrix}$$

- Processador 0
- Processador 1
- Processador 2
- Processador 3

Figura 7.1 – Cálculo de $f_{novo} = D^{-1} * N * f_{antigo} + D^{-1} * g$, particionando as matrizes D e N não somente por linhas mas também por colunas e particionando os vetores f_{antigo} e g .

Referências Bibliográficas

- [1] Perry W. S., Wong H.-S. e Guan L., “Adaptive Image Processing”, Spie Press, Bellingham, WA USA, 2002.
- [2] A.Cichocki e R. Unbehauen, “Neural Networks for Optimization and Signal Processing”, John Wiley & Sons, New York, 1993.
- [3] Figueiredo A. T., Leitão M.N. J.; “Sequential and Parallel Image Restoration, Neural Network Implementations”, IEEE Transactions On Image Processing, vol.3, nº 6, November 1994.
- [4] E. Loli Piccolomini and F. Zama, “Parallel Image Restoration with Domain Decomposition”, Department of Mathematics – Piazza Porta S. Donato 5, 40127 Bologna, Italy.
- [5] Freeman T.L., Philips C., “Parallel Numerical Algorithms”, Prentice Hall Internacional, UK, 1992.
- [6] Ananth G., Anshul G., George K. e Vipin K., “Introduction to Parallel Computing”, Addison-Wesley, 1994, 2ª ed.
- [7] Strang G., “Linear Algebra and Its Applications”, Academic Press, United Kingdom (UK), 1976.
- [8] Andrews H.C. e B.R. Hunt, “Digital Image Restoration”, Prentice-Hall, Englewood Cliffs, New York, 1997.
- [9] Kolman B., “Introdução à Álgebra Linear com aplicações”, LTC, Rio de Janeiro, RJ, 1998, 6ª ed.

[10] "www.images.com.br/bird".

[11] J.Paik e A.Katsaggelos, "Image restoration using a modified Hopfield network," *IEEE Trans. Image Processing*, vol.1, pp. 49-63, Jan. 1992.

[12] J.Abbiss, B.Brames e M. Fiddy, "Superresolution algorithms for a modified Hopfield neural network", *IEEE Trans. Signal Processing*, vol.39, pp. 1516 -1523, Julho 1991.

[13] J.Paik e A.Katsaggelos, "Image restoration using the Hopfield network with nonzero autoconnections," in *Proc. Int. Conf. ASSP-ICASSP* (Albuquerque, NM), 1990, pp. 1909-1912.

[14] S.Yeh, H.Stark e M.Sezan, "Hopfield-type neural networks," in *Digital Image Restoration*, A. Katsaggelos, Ed. New York: Springer Verlag, 1991, pp.57-88.

[15] Y.Zhou et al., "Image restoration using a neural network," *IEEE Trans. Acoust., Speech, Signal Processing*, vol.36, pp. 1141-1151, Julho 1988.

[16] M.Takeda e J.Goodman, "Neural networks for computation.Number representation and programing complexity," *Appl. Optics*, vol. 25, pp. 3033-3046, 1986.

[17] A. Papoulis, "A new algorithm in spectral analysis and band-limited extrapolation", *IEEE Trans. Circuits, Syst.*, vol. CAS-22, pp. 737-742, 1975.

[18] "<http://www.nacad.ufrj.br>".

[19] "<http://ultima.cs.unr.edu/getimages.htm>".

Apêndice A

Processamento Paralelo

Aqui daremos uma breve explicação do conceito e objetivos do paralelismo, além das principais áreas de aplicação.

A.1 Introdução

Paralelismo consiste na divisão de tarefas em várias outras menores, as quais serão distribuídas por entre os processadores que irão processá-las e executá-las simultaneamente.

A.2 Objetivos do Paralelismo

- Reduzir tempos_Possibilidades de cálculos em “tempo real” / simulação em tempo real -- e.g. Meteorologia. Previsão de tempo em tempo hábil.
- Viabilizar a resolução de problemas que não poderiam ser considerados anteriormente
- Atingir maior precisão no mesmo tempo (e.g. refinar malhas)
- Superar limites físicos na velocidade de processamento sequencial

A.3 Principais Áreas de Aplicação

- Previsão do tempo e simulação de fenômenos globais (El Niño, La Niña): “Earth Simulator”

- Projeto racional de remédios
- Genoma humano (mineração de dados)
- Engenharia: modelos de turbulência/aviões
- Física do plasma: gases ionizados em altas temperaturas
- Ciência dos Materiais (modelagem de semicondutores)
- Raciocínio automático (e.g., xadrez: Deep Blue)
- Economia: modelos nacionais/internacionais
- Inteligência Artificial

Apêndice B

MPI (“Message-Passing-Interface”)

Neste apêndice serão apresentadas características, histórico e alguns conceitos e definições do Mpi (Message Passing Interface), além dos diversos tipos de comunicação “Point-to-Point” e coletiva.

B.1 - O que é MPI?

- Uma biblioteca de *Message-Passing*, desenvolvida para ser padrão em ambientes de memória distribuída, em *Message-Passing* e em computação paralela.

B.2 – Características:

- Especificada por um fórum internacional aberto, constituído por representantes da indústria, universidades e laboratórios do governo.
- Especificação de uma biblioteca de troca de mensagens
 - Modelo de troca de mensagens
 - Não é especificação de compilador
 - Não é um produto específico
- Apropriada para computadores paralelos, clusters e redes heterogêneas
- Projetada para permitir o desenvolvimento de bibliotecas de programas paralelos que sejam portáteis e eficientes
- Projetada para fornecer o acesso a computadores paralelos avançados para:
 - Usuários finais
 - Projetistas de bibliotecas
 - Desenvolvedores de ferramentas
- Especificada em C, C++ , Fortran 77 e 90

- Razões para o uso do MPI
 - Padronização
 - Portabilidade
 - Desempenho
 - Funcionalidade
 - Disponibilidade
- MPI é grande ou pequeno?
- MPI é grande
 - Possui em torno de 150 funções:
Funcionalidade sem complexidade
- MPI é pequeno
 - Muitos programas paralelos podem ser escritos com somente 6 funções
- Tem o tamanho que for necessário:
 - Usuário pode aumentar o conjunto de funções que vai usar
- Todo o paralelismo é explícito: o programador é responsável pela correta identificação do paralelismo e sua implementação
- O programador deve:
 - Inicializar os processos
 - Sincronizar os processos
 - Alocar e liberar buffer para as mensagens
 - Etc.

B.3 - Histórico

No final da década de 80 foi dado o início do conceito de memória distribuída, o desenvolvimento da computação paralela, ferramentas para desenvolver programas em ambientes paralelos, problemas com portabilidade, desempenho, funcionalidade e preço, determinaram a necessidade de se desenvolver um padrão.

Já em Abril de 92 foi iniciado o *Workshop* de padrões de *Message-Passing* em ambientes de memória distribuída (Centro de Pesquisa em Computação Paralela, Williamsburg, Virginia); iniciada a discussão das necessidades básicas e essenciais para se estabelecer um padrão *Message-Passing* e com isso a criação de um grupo de trabalho para dar continuidade ao processo de padronização.

Em novembro do mesmo ano houve uma reunião em Minneapolis do grupo de trabalho e apresentação de um primeiro esboço de interface *Message-Passing* (MPI1). O Grupo adota procedimentos para a criação de um MPI Fórum; MPIF se consiste eventualmente de aproximadamente 175 pessoas de 40 organizações, incluindo fabricantes de computadores, empresas de softwares, universidades e cientistas de aplicação.

Em novembro de 1993 na Conferência de Supercomputação 93 há a apresentação do esboço do padrão MPI.

Em maio de 94 a disponibilização, como domínio público, da versão padrão do MPI (MPI1).

E em dezembro de 1995 a Conferência de Supercomputação 95 onde houve a reunião para discussão do MPI2 e suas extensões. (Lastovetsky, 2003).

B.4 - Conceitos e Definições

Rank - Todo processo tem uma única identificação, atribuída pelo sistema quando o processo é iniciado. Essa identificação é contínua e começa no zero até n-1 processos.

Group - Grupo é um conjunto ordenado de N processos. Todo e qualquer grupo é associado a um *communicator* e, inicialmente, todos os processos são membros de um grupo com um *communicator* já pré-estabelecido (MPI_COMM_WORLD).

Communicator - O *communicator* define uma coleção de processos (grupo), que poderão se comunicar entre si (contexto). O MPI utiliza essa combinação de grupo e contexto para garantir uma comunicação segura e evitar problemas no envio de mensagens entre os processos.

- É possível que uma aplicação de usuário utilize uma biblioteca de rotinas, que por sua vez, utilize *Message-Passing*.

- Essa rotina pode usar uma mensagem idêntica a mensagem do usuário.

As rotinas do MPI exigem que seja especificado um *communicator* como argumento.

MPI_COMM_WORLD é o comunicador pré-definido que inclui todos os processos definidos pelo usuário, numa aplicação MPI.

Application Buffer - É um endereço normal de memória aonde se armazena um dado que o processo necessita enviar ou receber.

System Buffer - É um endereço de memória reservado pelo sistema para armazenar mensagens. Dependendo do tipo de operação de *send* e *receive*, o dado no *application buffer* pode necessitar ser copiado de/para o *system buffer* (*Send Buffer* e *Receive Buffer*). Neste caso tem-se comunicação assíncrona.

Blocking Communication - Uma rotina de comunicação é dita *bloking*, se a finalização da chamada depender de certos eventos.

Ex: Numa rotina de envio, o dado tem que ter sido enviado com sucesso, ou, ter sido salvo no *system buffer*, indicando que o endereço do *application buffer* pode ser reutilizado.

Numa rotina de recebimento, o dado tem que ser armazenado no *system buffer*, indicando que o dado pode ser utilizado.

Non-Blocking Communication - Uma rotina de comunicação é dita *Non-blocking*, se a chamada retorna sem esperar qualquer evento que indique o fim ou o sucesso da rotina.

Ex: Não espera pela cópia de mensagens do *application buffer* para o *system buffer*, ou a indicação do recebimento de uma mensagem.

Standard Send - Operação básica de envio de mensagens usada para transmitir dados de um processo para outro.

Synchronous Send - Bloqueia até que ocorra um *receive* correspondente no processo de destino.

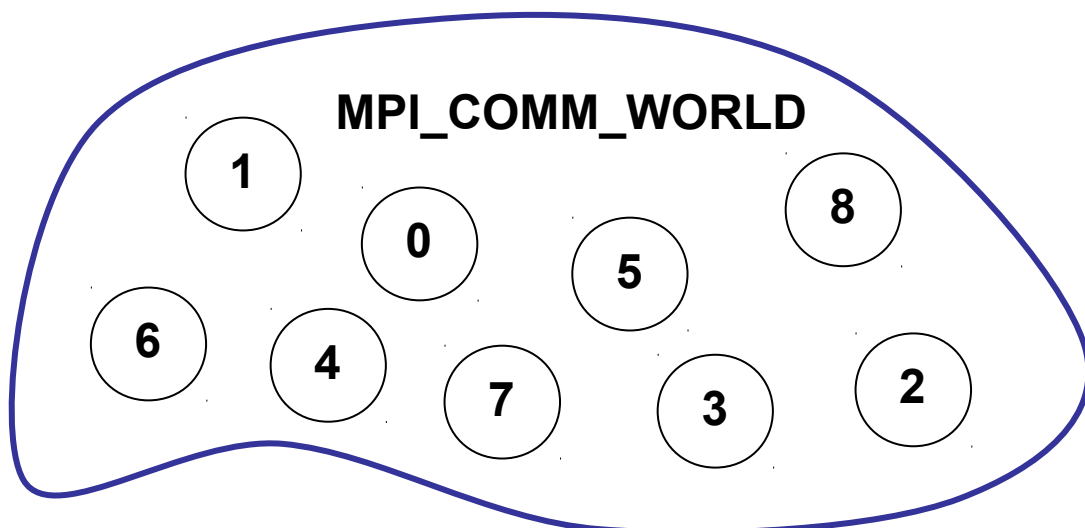
Buffered Send - O programador cria um *buffer* para o dado, antes deste ser enviado. Necessidade de se garantir um espaço disponível para um *buffer*, na incerteza do espaço do *System Buffer*.

Ready Send - Tipo de *send* que pode ser usado se o programador tiver certeza de que exista um *receive* correspondente, já ativo.

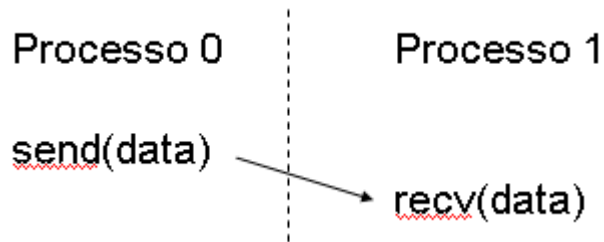
Standard Receive - Operação básica de recebimento de mensagens usado para aceitar os dados enviados por qualquer outro processo. Pode ser *blocking* e *non-blocking*.

Return Code - Valor inteiro retornado pelo sistema para indicar a finalização da sub-rotina.

- Noção de grupo e contexto combinados em um objeto chamado comunicador ou comm
- O comunicador aparece como argumento na maioria das funções de comunicação coletiva e ponto-a-ponto
- Cada processo MPI pertence a 1 ou mais grupos/comunicador
- Inicialmente todos os processos pertencem ao grupo MPI_COMM_WORLD
- Cada processo possui um identificador chamado rank
- Cada processo possui um identificador chamado rank em relação a um grupo
- rank varia de 0 a n-1,
sendo n = número de processos do grupo
- Grupos adicionais podem ser criados pelo programador



B.5 – Rotinas do MPI de Comunicação “Point-to-Point”



Existem alguns aspectos que devem ser levados em consideração numa comunicação Point-to-Point:

- Para onde a mensagem está sendo enviada
- Onde está o dado
- Qual o tamanho da mensagem
- Qual o tipo da mensagem
- Como o receptor identifica a mensagem

A mensagem é composta pelo dado e “envelope” que transporta a mensagem.

- Especificação da mensagem:
 - endereço – localização da memória onde a mensagem começa
 - contador – número de itens contidos na mensagem
 - datatype – tipos do MPI
 - fonte – rank do processo que envia
 - destino – rank do processo que recebe
 - tag – identificador da mensagem
 - comm – comunicador utilizado

Os componentes básicos de qualquer biblioteca de *Message-Passing* são as rotinas de comunicação *Point-to-Point* (transferência de dados entre dois processos):

- Envia:

MPI_SEND

- *Blocking send*.
- A rotina só retorna após o dado ter sido enviado.
- Após retorno, libera o *system buffer* e permite acesso ao *application buffer*.

int MPI_Send (*sdbuf, count, datatype, dest, tag, comm)

FORTRAN call *MPI_SEND (sdbuf, count, datatype, dest, tag, comm, mpierr)*

sdbuf - Endereço inicial do dado que será enviado. Endereço do *application buffer*.

Count - Número de elementos a serem enviados.

Datatype - Tipo do dado.

dest - Identificação do processo destino.

Tag - Rótulo da mensagem.

Comm - MPI communicator.

mpierr - Variável inteira de retorno com o status da rotina.

➤ Recebe:

MPI_RECV

- *Blocking receive*.

- A rotina retorna após o dado ter sido recebido e armazenado.

- Após retorno, libera o *system buffer*.

C int MPI_Recv(*recvbuf, count, datatype, source, tag, *status, comm)

FORTRAN call *MPI_RECV (recvbuf, count, datatype, source, tag, comm, status, mpierr)*

Recvbuf - Variável indicando o endereço do *application buffer*.

count - Número de elementos a serem recebidos.

Datatype - Tipo do dado.

source - Identificação da fonte. OBS: *MPI_ANY_SOURCE*

tag - Rótulo da mensagem. OBS: *MPI_ANY_TAG*

comm - MPI communicator.

status - Vetor com informações de source e tag.

Mpierr - Variável inteira de retorno com o status da rotina.

• Parâmetros Comuns das Rotinas de Send e Receive

Buf - Endereço do dado a ser enviado, normalmente o nome da variável, do vetor ou da matriz;

Count - Variável inteira que representa o número de elementos a serem enviados;

datatype - Tipo do dado;

source - Variável inteira que identifica o processo de origem da mensagem, no contexto do *communicator*;

dest - Variável inteira que identifica o processo destino, no contexto do *communicator*;

tag - Variável inteira com o rótulo da mensagem;

comm - *Communicator* utilizado;

status - Vetor com informações sobre a mensagem;

ierorr - Código de retorno com o status da rotina.

B.6 – Rotinas do MPI de Comunicação Coletiva:

Consiste numa comunicação realizada entre um grupo de processos em um comunicador. E tem as seguintes características:

- Mensagens não possuem tag
- As comunicações são bloqueantes

➤ Todos os processos chamam a função

Existem, basicamente, três tipos de rotinas de comunicação coletiva com as respectivas funções:

- Movimentação de dados: *Broadcast, Scan, Allreduce/Reduce, Scatter/Gather, dentre outras*
- Computação coletiva
- Sincronização

Dentre as rotinas do MPI relacionadas ao tema abordado neste trabalho temos:

MPI_INIT

- Primeira rotina MPI utilizada.
- Define e inicia o ambiente necessário para executar o MPI.
- Sincroniza todos os processos no início de uma aplicação MPI.

C: *int MPI_Init (*argc, *argv)*

FORTRAN: *call MPI_INIT (mpierr)*

argc - Apontador para um parâmetro da função main;

argv - Apontador para um parâmetro da função main;

mpierr - Variável inteira de retorno com o status da rotina.

mpierr=0, Sucesso

mpierr<0, Erro

MPI_COMM_RANK

- Identifica o processo, dentro de um grupo de processos.

- Valor inteiro, entre 0 e n-1 processos.

C: *int MPI_Comm_rank (comm, *rank)*

FORTRAN: *call MPI_COMM_RANK (comm, rank, mpierr)*

Comm - MPI communicator.

rank - Variável inteira de retorno com o número de identificação do processo.

Mpierr - Variável inteira de retorno com o *status* da rotina.

MPI_COMM_SIZE

- Retorna o número de processos dentro de um grupo de processos.

C: *int MPI_Comm_size (comm, *size)*

FORTRAN: *call MPI_COMM_SIZE (comm, size, mpierr)*

comm - MPI Communicator.

Size - Variável inteira de retorno com o número de processos inicializados durante uma aplicação MPI.

Mpierr - Variável inteira de retorno com o status da rotina

MPI_FINALIZE

- Finaliza o processo para o MPI.

- Última rotina MPI a ser executada por uma aplicação MPI.

- Sincroniza todos os processos na finalização de uma aplicação MPI.

C *int* MPI_Finalize()

FORTRAN *call* MPI_FINALIZE (*mpierr*)

- *mpierr*: Variável inteira de retorno com o status da rotina.

Data Movement

BROADCAST

C *int* MPI_Bcast(**buffer, count, datatype, root, comm*)

FORTRAN *call* MPI_BCAST (*buffer, count, datatype, root, comm, ierr*)

buffer - Endereço inicial do dado a ser enviado;

count - Inteiro que indica o número de elementos no *buffer*;

datatype - Constante MPI que identifica o tipo de dado dos elementos no *buffer*;

root - Inteiro com a identificação do processo que irá efetuar um *broadcast*;

comm - Identificação do *communicator*.

Rotina que permite a um processo enviar dados, de imediato, para todos os processos de um grupo. Todos os processos do grupo, deverão executar um *MPI_Bcast*, com o mesmo *comm* e *root*.

Veja a figura B.6.1:

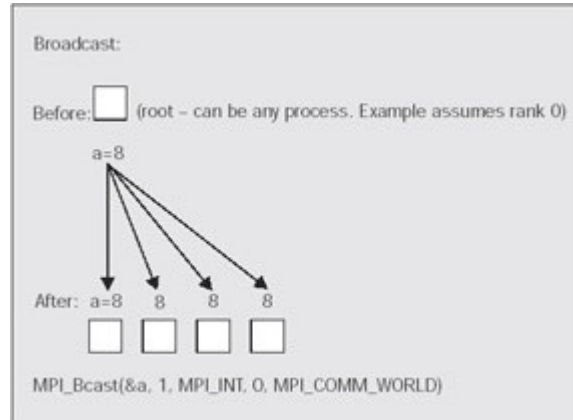


Figura B.6.1 – Funcionamento da rotina Broadcast.

Gather e Scatter

Se um processo necessita distribuir dados em n segmentos iguais, onde o n ésimo segmento é enviado para n ésimo processo num grupo de n processos, utiliza-se a rotina de *SCATTER*. Por outro lado, se um único processo necessita coletar os dados distribuídos em n processos de um grupo. Utiliza a rotina de *GATHER*.

- *SCATTER*

C `int MPI_Scatter(*sbuf, scount, stype, *rbuf, rcount, rtype, root, comm)`

FORTRAN `call MPI_SCATTER(sbuf ,scount, stype, rbuf, rcount, rtype, root, comm, ierr)`

sbuf - Endereço dos dados que serão distribuídos *send buffer*;

scount - Número de elementos que serão distribuídos para cada processo;

stype - Tipo de dado que será distribuído;

rbuf - Endereço aonde os dados serão coletados *receive buffer*;

rcount - Número de elementos que serão coletados;

rtype - Tipo de dado que será coletado;

root - Identificação do processo que irá distribuir os dados;
comm - Identificação do "communicator".

Observe a figura B.6.2:

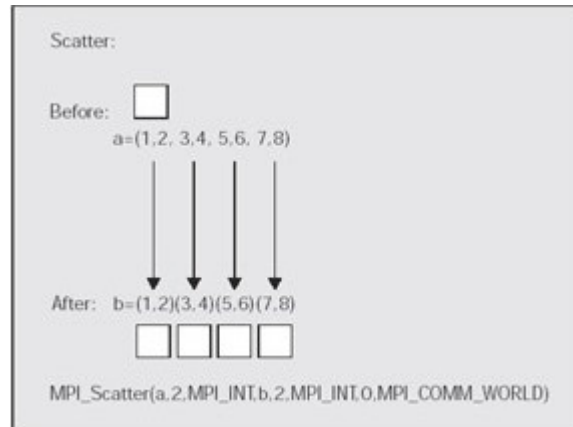


Figura B.6.2 – Funcionamento da rotina Scatter.

- **GATHER**

C `int MPI_Gather(*sbuf, scount, stype, *rbuf, rcount, rtype, root, comm)`

FORTRAN `call MPI_GATHER(sbuf, scount, stype, rbuf, rcount, rtype, root, comm, ierr)`

sbuf - Endereço inicial dos dados que serão distribuídos *send buffer*;
scount - Número de elementos que serão distribuídos para cada processo;
stype - Tipo de dado que será distribuído;
rbuf - Endereço aonde os dados serão coletados *receive buffer*;
rcount - Número de elementos que serão coletados;
rtype - Tipo de dado coletado;
root - Identificação do processo que ira coletar os dados;
comm - Identificação do *communicator*

Veja a figura B.6.3:

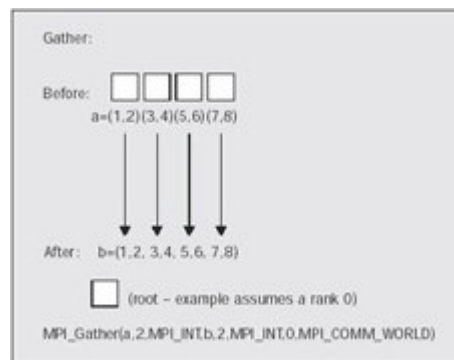


Figura B.6.3 – Funcionamento da rotina Gather.

Exemplo:

DIMENSION A(25,100), b(100), cpart(25), ctotal(100)

INTEGER root

DATA root/0/

DO I=1,25

cpart(I)=0.

DO K=1,100

cpart(I) = cpart(I) + A(I,K)*b(K)

END DO

END DO

call MPI_GATHER(cpart,25,MPI_REAL,ctotal,25,MPI_REAL,root,

MPI_COMM_WORLD,ierr)

Observe o esquema apresentado na figura B.6.4:

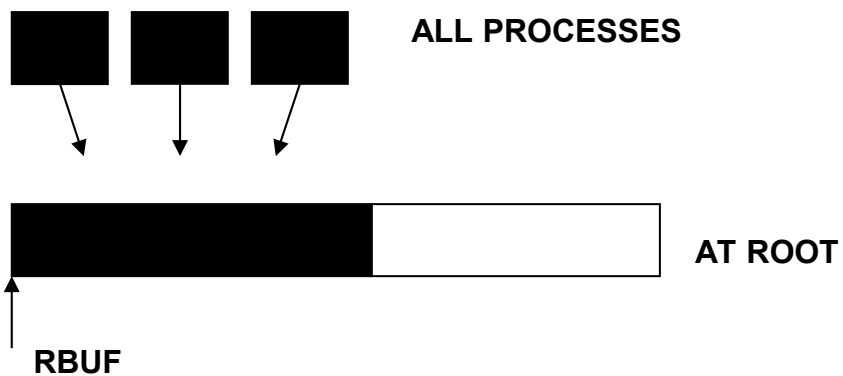


Figura B.6.4 – Processos sendo enviados para o buffer.

```
real a(100), rbuf(MAX)
```

```
...
```

```
...
```

```
...
```

```
call mpi_gather(a,100,MPI_REAL,rbuf,100,MPI_REAL,root,comm, ierr)
```

Observe o esquema na figura B.6.5:

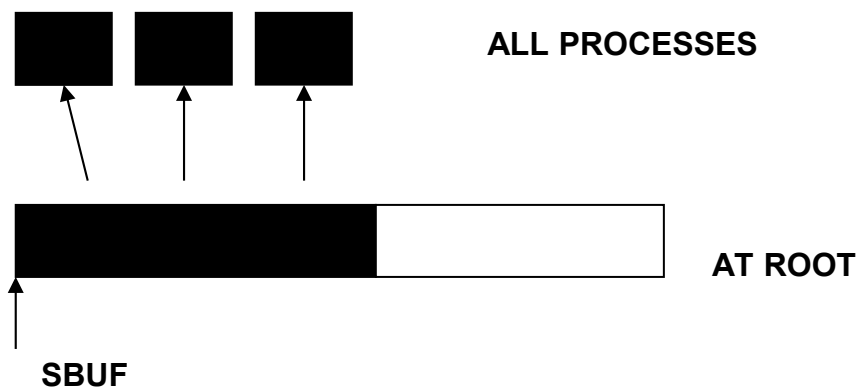


Figura B.6.5 – Buffer reenviando os processos

```
real sbuf(MAX), rbuf(100)  
...  
...  
...  
call mpi_scatter(sbuf,100,MPI_REAL,rbuf,100,MPI_REAL,  
root,comm,ierr)
```

ALLGATHER

```
C int MPI_Allgather( *sbuf, scount, stype, *rbuf, rcount, rtype, comm)
```

```
FORTRAN call MPI_ALLGATHER( sbuf, scount, stype, rbuf, rcount, rtype,  
comm, ierr)
```

Sbuf - Endereço inicial dos dados que serão distribuídos *send buffer*;

Scount - Número de elementos que serão distribuídos;

stype - Tipo de dado que será distribuído;

rbuf - Endereço aonde o dado será coletado *receive buffer*;

rcount - Número de elementos que serão coletados;

rtype - Tipo de dado coletado;

comm - Identificação do *communicator*.

Essa rotina ao ser executada faz com que todos os processos colem os dados de cada processo da aplicação. Seria similar a cada processo efetuar um *brodcast*.

ALL TO ALL

C int MPI_Alltoall(*sbuf, scount, stype, *rbuf, rcount, rtype, comm)

FORTRAN call MPI_ALLTOALL(sbuf, scount, stype, rbuf, rcount, rtype, comm, ierr)

- *sbuf*: Endereço inicial dos dados que serão distribuídos *send buffer*;
- *scount*: Número de elementos que serão distribuídos;
- *stype*: Tipo de dado que será distribuído;
- *rbuf*: Endereço aonde o dados serão coletados *receive buffer*;
- *rcount*: Número de elementos que serão coletados;
- *rtype*: Tipo de dado coletado;
- *comm*: Identificação do *communicator*.

Esta rotina ao ser executada faz com que cada processo envie seus dados para todos os outros processos da aplicação. Seria similar a cada processo efetuar um *scatter*.

Observe a figura B.6.6:

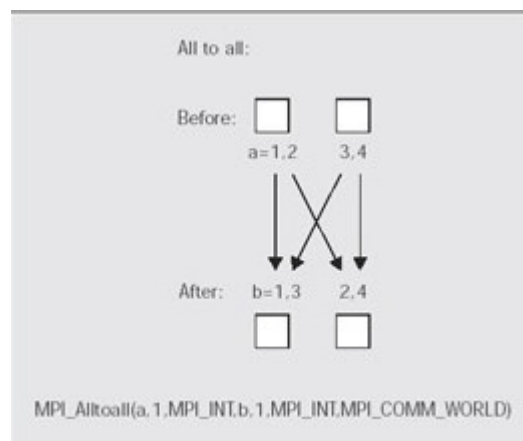


Figura B.6.6 – Funcionamento da rotina All to All

- Rotinas de Computação Global

Uma das ações mais úteis em operações coletivas são as operações globais de redução ou combinação de operações. O resultado parcial de um processo, em um grupo, é combinado e retornado para um específico processo utilizando-se algum tipo de função de operação.

Veja a tabela B.6.1 abaixo:

FUNÇÃO	RESULTADO	C	FORTRAN
MPI_MAX	valor máximo	integer,float	Integer,real,complex
MPI_MIN	valor mínimo	integer,float	Integer,real,complex
MPI_SUM	Somatório	integer,float	integer,real,complex
MPI_PROD	Produto	integer,float	integer,real,complex

Tabela B.6.1 – Definições do Mpi em C e Fortran

REDUCE

C `int MPI_Reduce(*sbuf, *rbuf, count, stype, op, root, comm)`

FORTRAN `call MPI_REDUCE(sbuf,rbuf,count,stype,op,root,comm,ierr)`

sbuf - Endereço do dado que fará parte de uma operação de redução *send buffer*;

rbuf - Endereço da variável que coletará o resultado da redução *receive buffer*;

count - Número de elementos que farão parte da redução;

stype - Tipo dos dados na operação de redução;

op - Tipo da operação de redução;

root - Identificação do processo que irá receber o resultado da operação de redução;

comm - Identificação do *communicator*.

Veja as figuras B.6.7 e B.6.8:

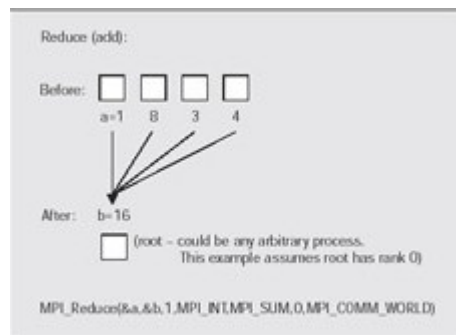


Figura B.6.7 – Exemplo de rotina Reduce

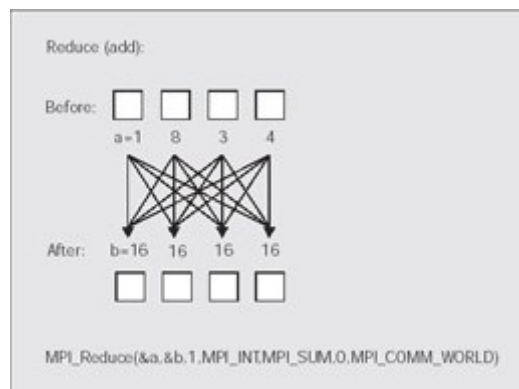


Figura B.6.8 – Exemplo de rotina Reduce

Apêndice C

Algoritmo seqüencial

Neste apêndice mostraremos o código, feito na linguagem C, relativo ao algoritmo seqüencial utilizado neste trabalho.

```
/*
*restoration_sequencial.c
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <time.h>

int main(int argc, char **argv)
{
    float **matrixB;
    float *f1, *f2, *g, *N, *r, *soma;
    double M, R;
    long int imageMatrixLen, hMatrixLen;
    long int indexG, indexBufNum;
    int count, stop;
    //float t0, t1;
    long int index01, index02, rol, col, i, j;
    long int buffPointer;
    char bufferLine[16*180+4+1];
```

```

char buffNumber[20];
FILE *input, *output;

M=0;

//t0=time(NULL);
if (argc!=2)
{
    printf("Uso: %s [dimensao da imagem]\n", argv[0]);
    exit(1);
}

imageMatrixLen=atoi(argv[1]);
hMatrixLen=imageMatrixLen*imageMatrixLen;

//-----Alocacao de memoria-----//
if ((N=(float *)malloc(sizeof(float)*hMatrixLen))==NULL)
{
    printf("Erro alocando mem para vetor N.\n");
    exit(1);
}

if ((r=(float *)malloc(sizeof(float)*hMatrixLen))==NULL)
{
    printf("Erro alocando mem para vetor r.\n");
    exit(1);
}

if ((f2=(float *)malloc(sizeof(float)*hMatrixLen))==NULL)
{
    printf("Erro alocando mem para vetor f2.\n");
    exit(1);
}

```

```

if ((f1=(float *)malloc(sizeof(float)*hMatrixLen))==NULL)
{
    printf("Erro alocando mem para vetor f1.\n");
    exit(1);
}

if ((soma=(float *)malloc(sizeof(float)*hMatrixLen))==NULL)
{
    printf("Erro alocando mem para vetor soma.\n");
    exit(1); // fecha MPI e desloca outras mem.
}

if ((g=(float *)malloc(sizeof(float)*hMatrixLen))==NULL)
{
    printf("Erro alocando mem para vetor g.\n");
    exit(1);
}

if ((matrixB=(float**)malloc(hMatrixLen*sizeof(float*)))==NULL)
{
    printf("Erro alocando linhas da matriz B.\n");
    exit(1);
}
for (index01=0; index01<hMatrixLen; index01++)
{
    if ((matrixB[index01]=(float*)malloc(hMatrixLen*sizeof(float)))==NULL)
    {
        printf("Erro alocando colunas da matriz B.\n");
        exit(1);
    }
}

```

```

    }
}

if ((matrixD=(float**)malloc(hMatrixLen*sizeof(float*)))==NULL)
{
    printf("Erro alocando linhas da matriz D.\n");
    exit(1);
}
for (index01=0; index01<hMatrixLen; index01++)
{
    if
((matrixD[index01]=(float*)malloc(hMatrixLen*sizeof(float)))==NULL)
    {
        printf("Erro alocando colunas da matriz D.\n");
        exit(1);
    }
}

//-----fim Alocação de memória-----//

//-----Encontrar matrixD-----//

for (rol=0; rol<hMatrixLen; rol++)
{
    for (col=0; col<hMatrixLen; col++)
    {
        if (col==rol)
        {
            matrixD[rol][col]=1.0;
        }
        else matrixD[rol][col]=0.0;
    }
}

```

```

    }

//-----fim Encontrar matrixD-----//

//-----Encontar matriz B-----//
for (rol=0; rol<hMatrixLen; rol++)
    {
        soma[rol]=0.0;
    }

for (rol=0; rol<hMatrixLen; rol++)
{
    for (col=0; col<hMatrixLen; col++)
    {
        if (col==rol)
        {
            matrixB[rol][col]=0.0;
            if((col<(hMatrixLen-7))&&(rol<(hMatrixLen-7)))
            {
                matrixB[rol][col+7]=-0.4;
                matrixB[rol+7][col]=-0.4;
            }
        }
        //else matrixB[rol][col]=0.0;
    }
}

for (rol=0; rol<hMatrixLen; rol++)
    {
        for (col=0; col<hMatrixLen; col++)

```



```

        {
            soma[rol]+=matrixB[rol][col];
        }
    }

//-----fim Encontrar matrix B-----//

//-----Chutes iniciais para f1 e inicializacao para f2-----//

for(i=0;i<hMatrixLen;i++)
{
    f1[i]=0.755000;
    f2[i]=0.000000;
}

//-----fim Chutes iniciais para f1-----//

//-----abrindo o arquivo glImageFile----
if ((input=fopen("glImageFile","r"))==NULL)
{
    printf("Arquivo glImageFile não existe.\n");
    exit(1);
}

//-----leitura do arquivo glImageFile e armazenamento no vetor g//
indexG=0;
for (rol=0; rol<imageMatrixLen; rol++)

```

```

    {
buffPointer=0;
        fgets(bufferLine,(16*imageMatrixLen+4),input);
        if (bufferLine[sizeof(bufferLine)]=='\n')
            bufferLine[sizeof(bufferLine)]='\0';
buffPointer=0;
for (col=0;col<imageMatrixLen;col++)
{
    while(bufferLine[buffPointer]==' ')
        buffPointer++;
    indexBufNum=0;
    while((bufferLine[buffPointer]!=' ')&&(bufferLine[buffPointer]!='\0'))
    {
        buffNumber[indexBufNum]=bufferLine[buffPointer];
        buffPointer++;
        indexBufNum++;

        }
        buffNumber[indexBufNum]='\0';
        g[indexG++]=atof(buffNumber);
    }

}

printf("Numero de elementos de g: %li\n",indexG);

//-----fim leitura do arquivo gimageFile e armazenamento no vetor g//

//-----loop principal-----//
stop=0;

```

```

//t0=time(NULL);
count=0;
while((stop==0)&&(count<6))
{

    for (i=0;i<hMatrixLen;i++)
    {
        f2[i]=f1[i];
        f1[i]=0;
        // printf("Entrei1");
    }

    for(i=0;i<hMatrixLen;i++)
    {
        for(j=0;j<hMatrixLen;j++)

            f1[i]+=matrixB[i][j]/matrixD[i][i]*f2[j];
                f1[i]=f1[i]+g[i]/matrixD[i][i];
                // printf("f1[i]=%e\n",f1[i]);
    }
//for(i=0;i<hMatrixLen;i++)
//{
//f1[i]=-f1[i]*(1.0/soma[i]);
//printf("f1[i]=%e\n",f1[i]);
//}

    // if((count%15)==0)
//    {
for(i=0;i<hMatrixLen;i++)
{
    N[i]=(fabsf(f2[i]-f1[i]));
    printf("%e",N[i]);
}
}

```

```

    }

M=0;

for(i=0;i<hMatrixLen;i++)
if(N[i]>M)
M=N[i];

if(M<0.1)
{
for(i=0;i<y;i++)
{
for(j=0;j<y;j++)
r[j]=fabsf((-N[i][j]+matrixD[i][i])* f2[i])-g[i];
}
R=0;

for(i=0;i<y;i++)
if(r[i]>R)
R=r[i];

if(R<0.1)
stop=1;

}

//M+=abs(N[i]);
// }

count++;

```

```

}/*while*/

//    t1=time(NULL);
//-----fim loop principal-----//
    if ((output=fopen("imageFile","w"))==NULL)
    {
        printf("Arquivo imageFile não foi criado.\n");
        exit(1);
    }
for(i=0;i<hMatrixLen;i++)
f2[i]=-(1.0/soma[i])*f2[i];
printf("f1[i]=%e\n",f2[i]);

//-----rearranjo da matriz f2-----
    //printf("f2[4900]=%f\n",f2[4899]);
    for (index01=0;index01<imageMatrixLen;index01++)
    {
        for (index02=0;index02<imageMatrixLen;index02++)
        {
            fprintf(output," %f", f1[index01*imageMatrixLen+index02]);
        }
        fprintf(output,"\n");
    }

    //printf("erro=%f, iteracoes=%d\n",M,count);

//    printf(asctime(localtime(&t1-t0)));
//
//    for(i=0;i<hMatrixLen;i++)
//    {

```

```
    // printf("%6.5f",f2[i]);  
    // printf("\n");  
    // }  
  
    free(f1);  
    free(f2);  
    free(g);  
    free(soma);  
    free(matrixB);  
    free(matrixD);  
    free(r);  
    free(N);  
    fclose(input);  
    fclose(output);  
  
    return(0);  
}
```

Apêndice D

Algoritmo paralelo

Neste apêndice mostraremos o código, feito na linguagem C, relativo ao algoritmo paralelo utilizado neste trabalho.

```
/*
*malha.c
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include "mpi.h"

int main(int argc, char **argv)
{
    int i,j,x,u,w,y,m2,*nu_elem,rest,dispf,iteracoes;
    double inicio,fim;

    float *aux,*h,*x_novo,*x_antigo,**N;
    float *erros;
    float max[1],max1[1];
    int eu,np,condicao,stop,cont,*disp;
    long int indexG, indexBufNum, index01, index02, rol, col, k;
    long int buffPointer;
    char bufferLine[16*512+4+1];
    char buffNumber[20];
    FILE *input,*output,*hOutput;
```

```

//Inicio do programa paralelo

MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD,&np);
MPI_Comm_rank(MPI_COMM_WORLD,&eu);

if(argc!=2)
{
    printf("Uso:./nomedoprograma dimensao\n");
    MPI_Finalize();
    exit(1);
}

else
{
    condicao=0;
    x=atoi(argv[1]);
    y=x*x;
}

/*****/
nu_elem =(int*)malloc(sizeof (int)*np);
disp=(int *)malloc(sizeof(int)*np);
// g=(float *)malloc(sizeof(float)*y);
x_novo=(float *)malloc(sizeof(float)*y);
x_antigo=(float *)malloc(sizeof(float)*y);
erros=(float *)malloc(sizeof(float)*np);

if ((soma=(float *)malloc(sizeof(float)*hMatrixLen))==NULL)

```



```

        {
            printf("Erro alocando mem para vetor soma.\n");
            exit(1); // fecha MPI e desloca outras mem.
        }
    }
/*****

```

```

if((eu==0)&&(condicao==0))
{ //01
/* Distribuindo as linhas para os np processadores*/

rest = y % np;
for(i=0;i<np;i++)
    nu_elem[i]=y/np;
if (rest)
{
    while(rest--)
    {
        nu_elem[i-1]++;
        i--;
    }
}
//Comunico essa distribuicao para os demais processadores

MPI_Bcast(nu_elem,np,MPI_INT,0,MPI_COMM_WORLD);
} //!01

else if(condicao==0)

{
    MPI_Bcast(nu_elem,np,MPI_INT,0,MPI_COMM_WORLD);
}

```

```

    }

    /*****/

    /* Aloco matrizes*/

    aux =(float *)malloc(sizeof(float)*nu_elem[eu]);
    r =(float *)malloc(sizeof(float)*nu_elem[eu]);
    // aux1 =(float *)malloc(sizeof(float)*nu_elem[eu]);

    h=(float*)malloc(sizeof(float)*nu_elem[eu]);

    /* if ((N=(float**)malloc(y*sizeof(float)))==NULL)
        {
            printf("Erro alocando linhas da matriz N.\n");
            MPI_Finalize();
            exit(1);//fecha MPI e desloca outras mem.
        }
        for (index01=0; index01<y; index01++)
        {
            if ((N[index01]=(float*)malloc(y*sizeof(float)))==NULL)
            {
                printf("Erro alocando colunas da matriz N.\n");
                MPI_Finalize();
                exit(1);//fecha MPI e desloca outras mem.
            }
        }
    */

    if ((N=(float**)malloc(nu_elem[eu]*sizeof(float)))==NULL)
        {

```

```

        printf("Erro alocando linhas da matriz N.\n");
        MPI_Finalize();
        exit(1);//fecha MPI e desloca outras mem.
    }
    for (index01=0; index01<nu_elem[eu]; index01++)
    {
        if ((N[index01]=(float*)malloc(y*sizeof(float)))==NULL)
        {

            printf("Erro alocando colunas da matriz N.\n");
            MPI_Finalize();
            exit(1);//fecha MPI e desloca outras mem.
        }
    }
}

/*fim*/

/*****/

//Determino a linha inicio e fim do bloco que cada processador vai calcular

disp[0]=0;
for (i=1;i<np;i++)
{
    disp[i] = disp[i-1] + nu_elem[i-1];
}
dispf=disp[eu]+nu_elem[eu];

//printf("sou: %d, tenho: %d, %d, %d\n",eu,nu_elem[eu],disp[eu],dispf);

```

```

//indexG=0;
    if(eu==0)
    {
        indexG=0;
//-----abrindo o arquivo glImageFile----
        if ((input=fopen("glImageFile","r"))==NULL)
        {
            printf("Arquivo glImageFile não existe.\n");
            exit(1);
        }

        //-----leitura do arquivo glImageFile e armazenamento no vetor g//

        for (rol=0; rol<x; rol++)
        {
            buffPointer=0;
            fgets(bufferLine,(10*x+4),input);
            if (bufferLine[sizeof(bufferLine)]=='\n')
                bufferLine[sizeof(bufferLine)]='\0';
            buffPointer=0;
            for (col=0;col<x;col++)
            {
                while(bufferLine[buffPointer]!=' ')
                    buffPointer++;
                indexBufNum=0;
                while((bufferLine[buffPointer]!=' ')&&(bufferLine[buffPointer]!
                ='\0')&&(bufferLine[buffPointer]!='\n'))
                {
                    buffNumber[indexBufNum]=bufferLine[buffPointer];

```

```

buffPointer++;
indexBufNum++;

        }
        buffNumber[indexBufNum]='\0';
        x_antigo[indexG++]=atof(buffNumber);
        buffPointer++;
    }

}

//    printf("Numero de elementos de g: %li\n",indexG);
//MPI_Bcast(g,y,MPI_FLOAT,0,MPI_COMM_WORLD);
    } //if eu==0
    //else
    //MPI_Bcast(g,y,MPI_FLOAT,0,MPI_COMM_WORLD);

//-----fim leitura do arquivo gimageFile e armazenamento no vetor g//

//-----distribuo o vetor g particionado para os demais processadores-----//

for (j=0;j<nu_elem[eu];j++)
    h[j]=0;

MPI_Scatterv(x_antigo,nu_elem,disp,MPI_FLOAT,h,nu_elem[eu],MPI_FLOAT,0,MPI
_COMM_WORLD);

//-----fim distribuicao -----//

```

```

for (i=0; i<nu_elem[eu]; i++)
    {
        soma[i]=0.0;
    }

```

```

for(i=disp[eu];i<dispf;i++)
    {
        if (i<(y-7))
            {
                N[i-disp[eu]][i+7]=-0.3;

                if(i>6)
                    N[i-disp[eu]][i-7]=-0.3;

            }

    }

```

```

for (i=0; i<nu_elem[eu]; i++)
    {
        for (j=0; j<hMatrixLen; j++)
            {
                soma[i]+=matrixB[i][j];
            }
    }

```

//Atribuicao de valores quaisquer

```

for (i=0;i<y;i++)
    {

```

```

        x_antigo[i]=0.000000;
        x_novo[i]=0.755000;
    }
    stop=0;

//printf("Vou entrar no loop\n");

    stop=0;

    inicio=MPI_Wtime();
    iteracoes=0;

    while((iteracoes<500)&&(stop==0))
    { //03

        //iteracoes++;

        for (i=0;i<y;i++)
        {
            x_antigo[i]=x_novo[i];
            x_novo[i]=0;
        }

        for (j=0;j<nu_elem[eu];j++)
            aux[j]=0;        //aux=0

        for (i=disp[eu];i<dispf;i++)
        {
            for (j=0;j<y;j++)

```

```

    aux[i-disp[eu]]+=N[i-disp[eu]][j]*x_antigo[j];

    aux[i-disp[eu]]+=h[i-disp[eu]];
}

```

```

/*****/

```

```

MPI_Allgather(aux,nu_elem[eu],MPI_FLOAT,x_novo,nu_elem,disp,MPI_FLOAT,MPI_COMM_WORLD);

```

```

//{04

```

```

/*****/

```

```

    for (j=0;j<nu_elem[eu];j++)
        aux[j]=0;    //aux2=0

```

```

/*****/

```

```

    for(i=disp[eu];i<dispf;i++)
    {

        aux[i-disp[eu]]=fabsf(x_novo[i]-x_antigo[i]);

    }

```



```

        max[0]=0;

for(i=0;i<nu_elem[eu];i++)

    if(aux[i]>max[0])
        max[0]=aux[i];    //max:norma infinita parcial

    for(i=0;i<np;i++)
        erros[i]=0.0;

MPI_Gather(max,1,MPI_FLOAT,erros,1,MPI_FLOAT,0,MPI_COMM_WORLD);

    if(eu==0) //calculando erro
    {
        max[0]=0;
        for(i=0;i<np;i++)
        {

            if(erros[i]>max[0])
                max[0]=erros[i]; //max:norma infinita total

        }
    }
    MPI_Bcast(&max[0],1,MPI_FLOAT,0,MPI_COMM_WORLD);

    if (max[0]<0.01)
    { //05
        for (j=0;j<nu_elem[eu];j++)

```

```

        aux[j]=0;

        for(i=disp[eu];i<dispf;i++)
        {
            for(j=0;j<y;j++)
                aux[i-disp[eu]]=fabsf((-N[i][j]+matrixD[i][i])*
x_novo[j])-h[i-disp[eu]]);
        }

        max1[0]=0.0;
        for(i=0;i<nu_elem[eu];i++)

            if(aux[i]>max1[0])
                max1[0]=aux[i];

MPI_Gather(max1,1,MPI_FLOAT,residuos,1,MPI_FLOAT,0,MPI_COMM_WORLD);

        for(i=0;i<np;i++)
        {
            erros[i]=0.0;
        }

        if(eu==0) //calculando residuo
        {
            max1[0]=0;

            for(i=0;i<np;i++)
            {

                if(erros[i]>max1[0])

```

```

max1[0]=erros[i]; //max:norma infinita total

    }

    }
if (max1[0]<0.01)
    stop=1;

// fim calculo erro

MPI_Bcast(&stop,1,MPI_INT,0,MPI_COMM_WORLD); //todo mundo
recebe stop
} //!05

iteracoes++;

} //!03
fim=MPI_Wtime();
//-----

// printf("iteracoes:%d\n",iteracoes);
if(eu==0)
{ //!05

printf("erro= %2.15f, iteracoes= %d, tempo= %.4f\n\n",max[0],iteracoes,fim-
inicio);

if ((output=fopen("imageFile","w"))==NULL)

```

```

        {
            printf("Arquivo imageFile não foi criado.\n");
            exit(1);
        }

for(i=0;i<nu_elem[eu];i++)
x_novo[i]=- (1.0/soma[i])*x_novo[i];

//-----rearranjo da matriz f2-----
//printf("f2[4900]=%f\n",f2[4899]);

for (index01=0;index01<x;index01++)
{
    for (index02=0;index02<x;index02++)
    {
        fprintf(output,"%f ", x_novo[index01*x+index02]);
    }
    fprintf(output,"\n");
}

fclose(output);

fclose(input);

} //!05

//if(eu==0)
//{
//for(i=0;i<y;i++)
// for(j=0;j<y;j++)

```

```
//printf("b[i][j]=%f\n", N[i][j]);
// printf("sou: %d, tenho: %d, %d, %d\n",eu,nu_elem[eu],disp[eu],dispf);
//}
free(disp);
free(nu_elem);
// free(g);
free(h);
free(x_antigo);
free(x_novo);
free(aux);
free(N);

//free(D);
free(errores);
MPI_Finalize();
exit(0);
} /*main*/
```

