Universidade Federal Fluminense
Centro Tecnológico
Escola de Engenharia
Curso de Engenharia de Telecomunicações

Algoritmos de Compressão de Imagens Baseados em Casamento de Padrões com Modelos Probabilisticos Condicionados Visando a Implementação em Plataformas de Processamento Paralelo.

Autor: Fabio Agricola de Miranda
Orientador: Prof. Murilo Bresciani de Carvalho

Niterói-RJ
Novembro / 2017

FABIO AGRICOLA DE MIRANDA

Algoritmos de Compressão de Imagens Baseados em Casamento de Padrões com Modelos Probabilísticos Condicionados Visando a Implementação em Plataformas de Processamento Paralelo.

Dissertação apresentada ao Curso de Mestrado em Engenharia de Telecomunicações da Universidade Federal Fluminense, como requisito parcial para obtenção do Grau de Mestre. Área de Concentração: Sistemas de Telecomunicações.

Orientador: Prof. Dr. MURILO BRESCIANI DE CARVALHO

Niterói-RJ/2017

FABIO AGRICOLA DE MIRANDA

Algoritmos de Compressão de Imagens Baseados em Casamento de Padrões com Modelos Probabilísticos Condicionados Visando a Implementação em Plataformas de Processamento Paralelo.

Dissertação apresentada ao Curso de Mestrado em Engenharia de Telecomunicações da Universidade Federal Fluminense, como requisito parcial para obtenção do Grau de Mestre. Área de Concentração: Sistemas de Telecomunicações.

Aprovada em novembro de 2017.

BANCA EXAMINADORA

Prof. Dr. MURILO BRESCIANI DE CARVALHO - Orientador, D.Sc., UFF

Prof. CARLA LIBERAL PAGLIARI, Ph.D., IME

Prof. TADEU NAGASHIMA FERREIRA, D.Sc., UFF

Niterói-RJ

2017

Dedico este trabalho à memória do meu irmão Auler, que partiu prematuramente deixando grande saudade.

Agradecimentos

Agradeço a Deus por permitir que mais essa etapa se concretize na minha vida.

Agradeço à minha esposa e filhos pelo apoio e compreensão durante a realização do curso.

Agradeço ao meu orientador, Murilo Brecianni Carvalho, pela sua paciência, dedicação e broncas durante a realização deste trabalho.

Agradeço à todos os professores da UFF que contribuíram de alguma forma para que eu crescesse em conhecimentos.

Agradeço ao meu ex-gerente, Ralf, por permitir que eu gozasse de algumas horas matinais livres para desenvolver meus trabalhos.

Agradeço a todos que, de alguma forma, apoiaram ou contribuíram para a realização deste sonho.

Lista de Figuras

2.1 Sistema de comunicação básico. 3
2.2 Histograma das imagens "Columns" e "Pepper". 14
2.3 Histograma da imagem da Lena para três diferentes valores do *pixel* antecessor 16

3.1 Histograma da imagem da Lena 18
3.2 Imagem do valor absoluto do resíduo da "Lena". 18
3.3 Função $y = b^{|x|}$, para b igual 0,99; 0,9; 0,8 e 0,5. 23

4.1 Árvore de segmentação de um vetor de entrada de dimensão $N = 4$. 25
4.2 Árvore de segmentação de um vetor de dimensão $N = 4$. 28
4.3 Árvore de segmentação de um vetor de dimensão $N = 4$ com os custos Lagrangeano. . . . 29
4.4 Fluxo do MMP. 31
4.5 Árvore de segmentação para um bloco 2×2. 33
4.6 Exemplo de cálculo da dimensão do Dicionário de Estado para um bloco de dimensões $4X4$, considerando $N_{s_{max}} = 4096$ e $Act_{max} = 200$. 38
4.7 Exemplo de cálculo da Rugosidade para um bloco do dicionário de dimensões $4X4$ cujos elementos do bloco do dicionário possuem valor igual a 100. 40
4.8 Imagens da Lena recuperadas após compressão usando o MMP-2D e MMP-2D-SM. 43
4.9 Gráfico comparativo entre o MMP-2D e MMP-2D-SM usando a imagem Baboon. 44
4.10 Gráfico comparativo entre o MMP-2D e MMP-2D-SM usando a imagem Barbara. 44
4.11 Gráfico comparativo entre o MMP-2D e MMP-2D-SM usando a imagem Columns. 45
4.12 Gráfico comparativo entre o MMP-2D e MMP-2D-SM usando a imagem Lena. 45
4.13 Gráfico comparativo entre o MMP-2D e MMP-2D-SM usando a imagem Pepper. 46

5.1 Organização de grade, blocos e *threads* no CUDA [12] 48
5.2 Fluxo do MMP-GPU, destacando o que roda na CPU e na GPU. 50
5.3 Exemplo de codificação de 3 blocos no MMP-2D-SM, sendo um da escla (0,0) e 2 da escala (1,0). 53
5.4 Exemplo de cálculo da Rugosidade para um bloco do dicionário de dimensões $4X4$. 54
5.5 Exemplo de cálculo da Rugosidade para um bloco do dicionário de dimensões $4X4$. 55
5.6 Gráfico comparativo entre o MMP-2D-SM e MMP-2D-SM-A usando a imagem Baboon. . 56
5.7 Gráfico comparativo entre o MMP-2D-SM e MMP-2D-SM-A usando a imagem Barbara. . 56

5.8 Gráfico comparativo entre o MMP-2D-SM e MMP-2D-SM-A usando a imagem Columns. 57
5.9 Gráfico comparativo entre o MMP-2D-SM e MMP-2D-SM-A usando a imagem Lena. . . . 57
5.10 Gráfico comparativo entre o MMP-2D-SM e MMP-2D-SM-A usando a imagem Pepper. . 58

6.1 Exemplo de cálculo da dimensão do Dicionário de Estado para um bloco de dimensões $4X4$, considerando $N_{s_{max}} = 4096$ e $VAR_{max} = 4000$. 61
6.2 Exemplo de cálculo da Distância da Vizinhança para um bloco do dicionário de dimensões $4X4$ cujos elementos do bloco do dicionário possuem valor igual a 100. 62
6.3 Imagens da Lena recuperadas após compressão usando o MMP-2D-SM e MMP-2D-ND. . 63
6.4 Gráfico comparativo entre o MMP-2D-ND e MMP-2D-SM usando a imagem Baboon . . . 64
6.5 Gráfico comparativo entre o MMP-2D-ND e MMP-2D-SM usando a imagem Barbara . . 64
6.6 Gráfico comparativo entre o MMP-2D-ND e MMP-2D-SM usando a imagem Columns . . 65
6.7 Gráfico comparativo entre o MMP-2D-ND e MMP-2D-SM usando a imagem Lena 65
6.8 Gráfico comparativo entre o MMP-2D-ND e MMP-2D-SM usando a imagem Pepper . . . 66
6.9 Gráfico comparativo entre o MMP-2D-ND e MMP-2D-ND-A usando a imagem Baboon. . 67
6.10 Gráfico comparativo entre o MMP-2D-ND e MMP-2D-ND-A usando a imagem Barbara. . 68
6.11 Gráfico comparativo entre o MMP-2D-ND e MMP-2D-ND-A usando a imagem Columns. 68
6.12 Gráfico comparativo entre o MMP-2D-ND e MMP-2D-ND-A usando a imagem Lena. . . . 69
6.13 Gráfico comparativo entre o MMP-2D-ND e MMP-2D-ND-A usando a imagem Pepper. . 69

A.1 baboon 73
A.2 Histograma da imagem Baboon. 73
A.3 barbara 74
A.4 Histograma da imagem Barbara. 75
A.5 columns 76
A.6 Histograma da imagem Columns. 77
A.7 lena 78
A.8 Histograma da imagem Lena. 79
A.9 pepper 80
A.10 Histograma da imagem Pepper. 81

Lista de Tabelas

2.1 Dados das imagens usadas nesta tese 4
2.2 Entropia de primeira ordem das imagens 6
2.3 Probabilidade dos símbolos da fonte hipotética. 9
2.4 Processo de codificação aritmética 9
2.5 Processo de codificação aritmética usando aritmética binária. Os intervalos representam o fragmento de L. 12
2.6 Resultados obtidos com a codificação aritmética. 13

3.1 Resultados obtidos com a Predição Linear 20
3.2 Resultados obtidos com a Predição Linear arbitrando o valor do preditor 20
3.3 Detalhes do cálculo da manipulação da frequência relativa. 22
3.4 Detalhes do cálculo da manipulação da frequência relativa. 22
3.5 Melhores resultados obtidos com o condicionamento estatístico 23

5.1 Comparação do tempo de execução entre o MMP-2D e o MMP-2D-GPU 51

A.1 Dados das imagens Baboon. 72
A.2 Dados das imagens Barbara. 74
A.3 Dados das imagens Columns. 76
A.4 Dados das imagens Lena. 78
A.5 Dados das imagens Pepper. 80

Sumário

Agradecimentos **v**

Lista de Figuras **vii**

Lista de Tabelas **viii**

Resumo **xi**

Abstract **xii**

1 INTRODUÇÃO **1**

2 COMPRESSÃO DE DADOS **3**

2.1 Sistema de comunicação 3

2.2 Teoria da Informação 4

2.3 Compressão de Dados 6

2.4 Codificação 8

2.4.1 Codificação Aritmética 9

3 PREDIÇÃO LINEAR E CONDICIONAMENTO ESTATÍSTICO **17**

3.1 Introdução 17

3.2 Predição Linear 17

3.2.1 Determinação analítica do valor do preditor ótimo 19

3.2.2 Resultados 19

3.3 Condicionamento Estatístico 20

4 ALGORITMO DE COMPRESSÃO MULTI-DIMENSIONAL MULTISCALE PARSER (MMP) **24**

4.1 Introdução 24

4.2 MMP 24

4.2.1 Dicionário 26

4.2.2 Modelo probabilístico do Codificador Aritmético 26

4.2.3 Critério de Fidelidade 27

4.2.4 Otimização da árvore de segmentação 27
4.2.5 Codificação dos índices 29
4.2.6 Atualização do Dicionário 29
4.2.7 Recuperando a representação do vetor de entrada no descompressor 30
4.3 MMP-2D 32
4.3.1 Árvore de Segmentação 2D 32
4.3.2 Otimização da Árvore de Segmentação 2D 34
4.4 MMP-2D SIDE-MATCH (MMP-2D-SM) 35
4.4.1 Determinação da Cardinalidade do Dicionário de Estado 36
4.4.2 Composição do Dicionário de Estado 39
4.4.3 Composição do Modelo Estatístico do Dicionário de Estado 41
4.4.4 Uso da imagem recuperada 41
4.4.5 Codificação 42
4.4.6 Tempo de codificação 42
4.4.7 Resultados das Simulações 42

5 Uso da GPU no MMP-2D e MMP-2D-SM **47**
5.1 Introdução 47
5.2 Organização do CUDA 47
5.3 Uso da GPU no MMP-2D 49
5.3.1 GPU no MMP-2D-SM 52

6 MMP-2D-ND **59**
6.1 Introdução 59
6.2 Determinação da Cardinalidade do Dicionário de Estado 60
6.3 Composição do Dicionário de Estado 61
6.4 Comparação com o MMP-2D-SM 62
6.5 Uso da imagem original 67
6.6 Tempo de processamento e implementação usando a GPU 70

7 CONCLUSÃO **71**

A IMAGENS USADAS **72**
A.1 Imagem Baboon 72
A.2 Imagem Barbara 74
A.3 Imagem Columns 76
A.4 Imagem Lena 78
A.5 Imagem Pepper 80

Resumo

Neste trabalho serão estudadas formas de melhorar o desempenho de algoritmos de compressão de dados baseados em casamento aproximado de padrões utilizando modelamento estatístico condicionado visando uma implementação em GPU (Graphics Processing Unit). Em trabalhos anteriores, o desempenho de um algoritmo deste tipo, o MMP (Multidimensional Multiscale Parser), foi melhorado tanto do ponto de vista da eficiência da compressão (Side-Match-MMP, MMP-Intra), quanto do ponto de vista da velocidade de processamento por meio de processamento paralelo, mas não ambas simultaneamente. A implementação dos algoritmos Side-Match-MMP e MMP-intra, de modo a aproveitar uma plataforma de processamento paralelo como a GPU, não é trivial. Este trabalho propõe alternativas ao Side-Match-MMP que são mais adequadas a este propósito.

Palavras-chave: Compressão de dados. Processamento Digital de imagem. GPU.

Abstract

In this work we propose to study new ways to improve the performance of data compression algorithm based on Multiscale Matching of Recurrent Patterns using conditioned statistical modelling, becoming possible its implementation in Graphics Processing Unit (GPU). In previous works, performance of these algorithms has improved, as well the processing speed using platform of parallel processing, but not both simultaneously. The implementation of Side-Match-MMP and MMP-Intra algorithms for platform of parallel processing is not trivial. This work proposes an alternative to implement the Side-Match-MMP algorithms that is more suitable to this goal.

Keywords: Data Compression. Digital Image Processing. GPU.

Capítulo 1

INTRODUÇÃO

Nas últimas décadas o mundo experimentou uma transformação na forma que as comunicações são realizadas. Saímos de uma era em que os sistemas eram analógicos e entramos numa, onde os sistemas são digitais. O fato é que quando transformamos sinais essencialmente analógicos, como o som e a imagem, em digitais, eles passam ocupar uma banda muito maior do que seus equivalentes analógicos, já que precisam atender o Teorema de Nyquist da amostragem e ser quantizados com um determinado número de níveis para manter o nível de distorção abaixo de um determinado limite. Por exemplo, no sistema de telefonia analógico a voz humana precisa de uma banda de 3,4 kHz para ser transmitida de forma inteligível. Quando esse sinal é digitalizado, é necessário amostrá-lo em uma taxa maior que o dobro da banda. Normalmente a taxa de amostragem é 8 kHz, e cada amostra é quantizada em 256 níveis, necessitando de 8 bits para representar cada amostra, o que leva a uma taxa de transmissão de 64 kbits por segundo. Caso esse sinal digitalizado fosse transmitido por um modem binário ocuparia uma banda de 32 kHz [1]. Logo, apesar dos sinais na forma digital serem mais confiáveis, o custo em banda para transmití-los em um canal de comunicação sem qualquer tratamento adicional é maior que os seus equivalentes analógicos, demandando modulações eficientes, compressão de dados, ou ambas as técnicas para compensar esse aumento, principalmente, se a banda for um recurso limitado. Como um sinal analógico, após digitalizado, normalmente não está em sua forma de representação mais compacta, surge a necessidade de compactar essa representação usando um algoritmo de compressão.

Algoritmos de compressão exploram duas características do sinal, que são a redundância e irrelevância. Algoritmos que exploram apenas a redundância são sem perda. Infelizmente compressão sem perda consegue um nível de compressão moderada. Quando exploramos a irrelevância, suprimindo alguma informação por ser considerada irrelevante, taxas de compressão maiores são conseguidas.

Uma imagem recuperada após ser comprimida por um algoritmo que explora a irrelevância não é uma cópia fiel da imagem original, ou seja, ela terá um grau de distorção. Logo, ao usar algoritmos que exploram a irrelevância, o compromisso entre o que pode ser considerando irrelevante e o grau de distorção que a supressão dessa informação causará deve ser considerado.

Programas como arj e 7zip exploram a redundância. Já programas que comprimem áudio, voz, vídeo, imagem, ou qualquer outro tipo de audiovisual exploram a irrelevância e a redundância. São exemplos desta classe o JPEG, JPEG2000, MPEG2, MPEG4, H.264, HEVC, MP3, CELP, etc.

Não há em geral um único processo de compressão que possa compactar qualquer representação digital de um sinal analógico na forma mais compacta possível. Por essa razão, vários algoritmos são desenvolvidos visando atender a um tipo de fonte especifica. Por exemplo, a voz limitada em banda de um sistema de telefonia pode ser processada usando-se a técnica de predição linear, que é usado em muitos codificadores de voz. Por outro lado, sinais de áudio são normalmente processados por bancos de filtros, enquanto sinais de imagens e vídeos são processados por transformadas discretas.

Nesta tese restringimos nossa atenção à forma de como manipular o modelo probabilístico, que chamamos de Condicionamento Estatístico, em um algoritmo de compressão que usa a técnica de busca de padrões chamado Multi-Dimensional Multiscale Parser (MMP). O MMP utiliza a estatística da imagem que está sendo comprimida para otimizar uma árvore de segmentação e então codificar aritmeticamente os símbolos que serão enviados para o descompressor. Esse algoritmo já possui uma variante que manipula o modelo probabilístico, melhorando o seu desempenho, mas aumentando muito a carga computacional, que já é alta no MMP. No nosso trabalho vamos analisar outra forma de manipular o modelo probabilístico, de forma que o processo possa ser paralelizado em uma unidade de processamento gráfico (GPU), reduzindo o tempo de compressão.

No capítulo 2 vamos apresentar os conceitos da teoria da informação e compressão de dados.

No capítulo 3 vamos apresentar um pequeno ensaio mostrando como o Condicionamento Estatístico trabalha e comparar os resultados com outra técnica mais difundida, que é a Predição.

No capítulo 4 vamos apresentar o algoritmo MMP, MMP-2D e sua variante, o MMP-2D-SM.

No capítulo 5 vamos apresentar o processamento paralelo utilizando uma unidade de processamento gráfico, e sua aplicação no MMP e o porquê da impossibilidade de usá-lo no MMP-SM.

No capítulo 6 apresentamos o conceito e os resultados do trabalho objeto desta tese.

No capitulo 7 apresentamos uma conclusão.

Capítulo 2

COMPRESSÃO DE DADOS

2.1 Sistema de comunicação

O principal objetivo de um sistema de comunicação é representar na saída do receptor, de forma idêntica ou aproximada, o símbolo recebido na entrada da transmissor. Um sistema de comunicação básico é composto de cinco elementos: fonte, transmissor, canal, receptor e destino, conforme mostrado na figura 2.1.

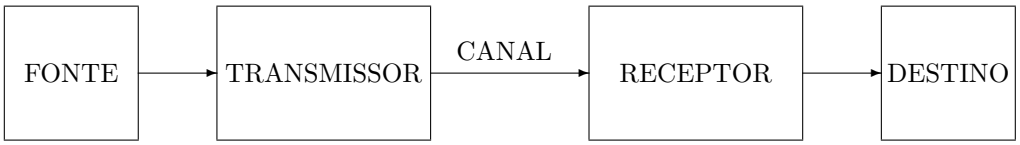


Figura 2.1: Sistema de comunicação básico.

O transmissor da figura 2.1 realiza diversas funções: inicialmente os símbolos emitidos pela fonte são convertidos em palavras código por um algoritmo de compressão que reduz a quantidade média de *bits* utilizados para representar a fonte. Em seguida essas palavras código são processadas por um segundo código cuja função é aumentar a robustez do sistema quanto aos inevitáveis erros que irão ocorrer no canal, e por último, a saída do segundo codificador é modulada segundo algum esquema de modulação digital para adaptá-la ao meio físico de comunicação. No receptor, implementam-se operações reversas de modo a restaurar os dados em seu formato aproximado ou original.

2.2 Teoria da Informação

Atualmente as fontes já entregam seus símbolos representados na forma digital para o transmissor. Mas ao converter seus símbolos para o formato digital, essa conversão não visa maximizar compactação, mas preservar a informação digitalizada da forma o mais fiel possível, ou se preferir, com a menor distorção, demandando uma grande banda para sua transmissão. Por exemplo, um áudio monoaural para ser convertido para o formato digital e amostrado numa frequência de 44,1 kHz, e então é quantizado em 2^{16} níveis. Se esse áudio precisasse ser transmitido em tempo real, seria necessário um canal de 1.411.200 *bits* por segundo, ou seja, 1,4 Mbps, que usando a modulação BPSK ocuparia um a banda de no mínimo 706 kHz. Como uma forma de comparação, uma estação de FM que transmite dois canais de áudio analógicos (estéreo) usa uma banda de 200 kHz. No caso de um sinal de vídeo digitalizado, a taxa bruta em *bits* por segundo é muito maior. Por exemplo, um sinal de uma câmera de vídeo digital 1080i tem um a taxa bruta de 1.080 linhas x 1.920 colunas x 30 quadros por segundo x 3 componentes por *pixel* x 8 *bits* por componente, totalizando 1.492.992.000 *bits*/s, ou seja 1.5 Gbits/s. Com a tecnologia de modens atual consegue-se uma ocupação de cerca de 20 a 30 Mbits/s quando esse sinal é modulado. A banda disponível do canal de TV analógica é de 6Mhz. Isto mostra claramente que sem a compressão de dados a radiodifusão de áudio e TV digitais nos mesmos canais usados pelos seus recíprocos analógicos seria inviável.

Nesta tese, as fontes utilizadas são imagens em escala de cinza cujos *pixels* já estão representados na forma digital. As imagens foram amostradas no domínio espacial com x amostras na horizontal e y amostras na vertical e o valor de cada amostra é quantizada linearmente em 256 níveis usando 8 *bits*. Logo uma imagem que foi amostrada com 512 amostras na horizontal e vertical possui $512 \times 512 \times 8 = 2.097.152$ *bits*, ou 262.144 bytes.

A tabela 2.1 apresenta os dados das imagens usadas, e que são apresentadas no anexo A. A primeira e segunda colunas contém o número de linhas e colunas da imagem respectivamente. A terceira coluna o número de níveis de quantização. A quarta coluna contém o número de *bits* usados para codificar cada nível. A quinta coluna contém o número de bytes de cada imagem.

Imagem	Número de linhas	Número de colunas	Níveis de quantização	*bits* por nível	Número de bytes
Baboon	512	512	256	8	262.144
Barbara	512	512	256	8	262.144
Columns	640	480	256	8	307.200
Lena	512	512	256	8	262.144
Pepper	256	256	256	8	65.536

Tabela 2.1: Dados das imagens usadas nesta tese

Medida da Informação e Entropia

Em 1948 C. E. Shannon apresentou em sua Teoria Matemática da Comunicação [11]. Segundo essa teoria quanto menos provável a ocorrência de um determinado símbolo de uma fonte, maior é a quantidade de informação contida nele. Shannon ainda definiu matematicamente a medida da informação como sendo

$$I_s = -\log_b p_s \tag{2.1}$$

onde

I_s é a quantidade de informação do símbolo;

p_s é a probabilidade de ocorrência do símbolo;

b número de elementos do alfabeto usado para representar o símbolo.

A palavra "informação" na teoria de Shannon não deve ser confundida com o sinônimo de "conhecimento" ou "significado", mas como a menor quantidade de elementos do alfabeto necessário para representar o símbolo. Por exemplo, num canal de comunicação binário, onde o alfabeto é composto de dígitos binário (*bits*) 0 e 1, logo $b = 2$, a equação 2.1 informa qual é a menor quantidade de *bits* que pode ser usada para representar o símbolo.

Se uma fonte discreta pode gerar M símbolos distintos, tendo cada símbolo uma probabilidade de ocorrência, p_s, a quantidade média de informação gerada por essa fonte é chamada de Entropia, e é definida como

$$H = -\sum_{s=1}^{M} p_s \log_b p_s \tag{2.2}$$

onde

H é a Entropia da fonte.

A Entropia de uma fonte é a menor quantidade média de elementos do alfabeto necessária para representar os símbolos de uma fonte, de modo que o receptor tenha como diferenciar entre as representações possíveis e apresentar o símbolo correto para o destino. A Entropia de uma fonte tende a zero se a probabilidade de qualquer um dos símbolos que a fonte gera tende a 1. E é máxima quando ocorrência dos símbolos é equiprovável, ou igual a $p = \frac{1}{M}$. Por esta razão a Entropia é chamada também de "Medida da Incerteza" de uma fonte, já que quanto maior a incerteza de qual símbolo a fonte vai gerar, maior a Entropia.

A tabela 2.2 apresenta o valor da Entropia de primeira ordem das imagens usadas e pode ser obtida utilizando-se a Equação 2.2 onde as probabilidades são estimadas medindo-se as frequências relativas dos valores dos *pixels* da imagem. A Entropia de primeira ordem é um a aproximação obtida considerando-se

que os *pixels* são independentes. Nesta tabela há uma coluna chamada "Entropia Máxima", que expressa a Entropia de ordem 0 de cada imagem, que é obtida considerando que os *pixels* são independentes e equiprováveis.

Imagem	Entropia (H)	Entropia Máxima (H_{max})
Baboon	7,3582	8
Barbara	7,4664	8
Columns	6.9266	8
Lena	7,4455	8
Pepper	7,5432	8

Tabela 2.2: Entropia de primeira ordem das imagens

Se uma fonte gera r símbolos por segundo, a taxa média com a qual a informação precisa ser enviada do transmissor para o receptor é

$$R = r \times H. \tag{2.3}$$

Na equação 2.3, R é a menor taxa (elementos do alfabeto por unidade de tempo) na qual o transmissor deve transmitir para o receptor para que o sistema de comunicação consiga enviar seus símbolos da fonte para o destino.

2.3 Compressão de Dados

Compressão de dados é um processo usado para representar os símbolos de uma fonte com o menor número de *bits* possível. Quando a imagem recuperada na descompreesão possui distorção igual a zero, o processo de compressão é chamado sem perdas. Caso contrário, com perdas.

Na compressão sem perdas, o único indicador de desempenho do algoritmo é a taxa de compressão, já que a imagem recuperada no processo de descompressão é igual a imagem original. Logo, quanto maior a taxa de compressão, melhor o desempenho do compressor. No caso dos algoritmos com perdas, temos que analisar o quanto a imagem recuperada difere da imagem original (distorção) para uma determinada taxa média de *bits* por *pixel* (taxa de compressão). Trataremos desse assunto na próxima subseção.

Desempenho do algoritmo de compressão de dados

Na compressão com perda, há dois indicadores do desempenho do algoritmo: a distorção causada pelo processo de compressão e descompressão para uma determinada taxa de compressão; e a própria taxa de compressão.

A taxa expressa o número médio de *bits* que será necessário para representar os símbolos de uma fonte após a compressão. No nosso caso, como estamos trabalhando com imagens, usamos o número de *bits* por *pixel* (*bits/pixel*).

A distorção mede o quão diferente é a imagem original da imagem recuperada pelo descompressor após codificada por um algoritmo com perdas. Para medir essa distorção, alguma função matemática que expressa essa diferença pode ser usada. No nosso caso usaremos a relação sinal ruído de pico (Peak Signal-to-Noise Ratio - PSNR).

O PSNR é uma expressão que relaciona a maior potência de um sinal e a potência média quadrática do ruído que distorce ou afeta esse sinal. Como muitos sinais possuem uma grande faixa dinâmica, ou seja, a relação entre o maior valor e o menor valor que pode assumir é grande, o PSNR é usaualmente expresso em termo de uma escala logarítmica.

O PSNR é definido como

$$PSNR = 10 \log_{10} \frac{max^2(sinal)}{MSE}, \tag{2.4}$$

onde:

$max^2(sinal)$ é a maior potência do sinal original, sem ser afetado pelo ruído;
MSE é o erro médio quadrático do sinal original menos o sinal afetado pelo ruído.

O MSE, na verdade, é a potência do ruído, já que o sinal afetado pelo ruído é o *sinal original* + *ruido*. Quando o sinal original é subtraído do sinal afetado pelo ruído resta apenas o ruído.

Uma imagem que é recuperada após ser comprimida por um algoritmo com perdas pode ser vista como a imagem original corrompida pelo ruído. No caso das imagens em escala de cinza usadas nas nossas simulações, o PSNR é definido como

$$PSNR = 10 \log_{10} \frac{255^2}{MSE}. \tag{2.5}$$

O MSE para a imagem é obtido pela equação 2.6.

$$MSE = \frac{1}{NM} \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} [I_{orig}(i,j) - I_{rec}(i,j)]^2 \tag{2.6}$$

onde

N é o número de linhas da imagem.

M é o número de colunas da imagem.

$I_{orig}(i,j)$ são os *pixel* da imagem original.
$I_{rec}(i,j)$ são os *pixel* da imagem recuperada.

Quanto maior for o PSNR mais as imagens comparadas se assemelham. Se uma imagem for igual a outra, o PSNR tende ao infinito, já que o MSE tende a zero.

Os métodos matemáticos de cálculo da distorção não são os únicos usados para medir a fidelidade dos resultados de um algoritmo de compressão. Na compressão de voz, áudio, imagem e vídeo uma avaliação perceptual muitas vezes é utilizada. Nesse método uma população selecionada é usada para comparar as versões original e a que passou pelo processo de compressão e descompressão usando os órgãos visuais ou auditivos para posteriormente expressar sua opiniões sobre a qualidade observada. Esse tipo de teste é subjetivo e os resultados podem variar dependendo das pessoas escolhidas para compor a população.

Teoria Taxa-Distorção e função $R(D)$

A teoria Taxa-Distorção estuda a compressão com perdas dos símbolos de uma fonte sujeita a algum critério de fidelidade. A relação entre a taxa de codificação e a distorção é conhecida como função $R(D)$. A função $R(D)$ de uma fonte especifica o menor número médio de *bits* necessários para representar uma saída produzida pela fonte com distorção média igual a D [2]. Como raramente sabemos qual a estatística de uma fonte a priori, a determinação da função $R(D)$ é um pouco complicada. Mas podemos, e é o que fazemos com os algoritmos de compressão com perda usados nesta tese, fazer o processo de codificação gerar mais ou menos distorção, calcular o PSNR e a taxa de *bits* por símbolo média e traçar curvas $R(D)$ para os diversos algoritmos usados para compará-los. Para uma determinada distorção, o algoritmo que codifica os símbolos de uma fonte com um menor número médio de *bits*, possui melhor desempenho.

2.4 Codificação

Normalmente uma fonte não gera símbolos equiprováveis. Assim também são as imagens naturais. Logo, como foi visto na seção 2.2, deve haver um esquema de codificação que, na média, usa menos *bits* para representar cada *pixel* do que quando consideramos os *pixels* equiprováveis. Um processo de compressão pode envolver as etapas de transformação, quantização e codificação de Entropia ou qualquer combinação dessas etapas. A proposta do estágio de transformação é converter os símbolos gerados pela fonte em outros símbolos, preferencialmente, não correlatados, que pertençam à outro domínio. A proposta do estágio de quantização é mapear os símbolos que ocupam uma faixa de valores X para outros símbolos que ocupam uma faixa reduzida de valores Y, de modo que se possa representa-los com menos *bits*. A quantização pode ser escalar, que é quando um símbolo da faixa X é mapeado para um outro símbolo que pertence à faixa Y, ou vetorial, que é quando um grupo símbolos é mapeado para um único

símbolo ou um grupo de símbolos que pertencem à faixa Y. O codificador de Entropia usa a estatística da fonte para representar os símbolos com o menor número possível de *bits* adequados para transmissão ou armazenamento.

Apesar da baixa taxa de compressão, um algoritmo pode compreender apenas da codificação de Entropia, não existindo a transformação ou quantização. Na próxima subseção vamos apresentar o codificador aritmético, que é o codificador de Entropia usado pelo MMP.

2.4.1 Codificação Aritmética

Para explicarmos como a codificação aritmética trabalha, vamos supor que uma fonte hipotética possa gerar três símbolos com probabilidades, conforme mostrado na tabela 2.3. Para entender o processo, vamos supor que uma mensagem composta pelos símbolos "b, b, b e EOF" seja enviada pela fonte para ser codificada.

Símbolo	Probabilidade
a	0,40
b	0,50
EOF	0,10

Tabela 2.3: Probabilidade dos símbolos da fonte hipotética.

A ideia da codificação aritmética é representar todos os símbolos de uma fonte em subintervalo de números reais no intervalo $[0; 1)$. A designação do subintervalo para cada símbolo pode seguir qualquer ordem desde que a ordem escolhida seja conhecida pelo decodificador. No momento em que um símbolo é selecionado, o intervalo para o qual ele foi designado é escalonado de acordo com a probabilidade dos símbolos da fonte, e assim sucessivamente até o último símbolo, conforme mostrado na tabela 2.4.

Intervalo	Entrada	Subintervalo		
		a	b	#
$[0; 1)$	b	$[0; 0, 4)$	$[0, 4; 0, 9)$	$[0, 9; 1)$
$[0, 4; 0, 9)$	b	$[0, 40; 0, 60)$	$[0, 60; 0, 85)$	$[0, 85; 0, 90)$
$[0, 60; 0, 85)$	b	$[0, 600; 0, 700)$	$[0, 700; 0, 825)$	$[0, 825; 0, 900)$
$[0, 700; 0, 825)$	EOF	$[0, 7000; 0, 750)$	$[0, 7050; 0, 8125)$	$[0, 8125; 0, 8250)$

Tabela 2.4: Processo de codificação aritmética

Na tabela 2.4 vemos que inicialmente a designação dos subintervalos foi feita da seguinte forma: o subintervalo $[0; 0, 4)$ para o símbolo 'a'; o subintervalo $[0, 4; 0, 9)$ para o símbolo 'b'; e o subintervalo $[0, 9; 1, 0)$ para o símbolo 'EOF'. Quando o codificador recebe o primeiro símbolo, que no nosso caso é o 'b', o intervalo designado para esse símbolo é selecionado e os subintervalos escalonado conforme a probabilidade de cada símbolo da fonte, da seguinte forma: o intervalo do símbolo 'b' inicia em $0, 4$ e termina em $0, 9$. Logo, o novo subintervalo para o símbolo 'a' inicia em $0, 4$ e termina em

$0,4 + (0,4 - 0) \times (0,9 - 0,4) = 0,4 + (0,4) \times (0,5) = 0,4 + 0,2 = 0,6.$

O novo subintervalo de 'b' inicia em $0,6$ e termina em

$0,6 + (0,9 - 0,4) \times (0,9 - 0,4) = 0,6 + (0,5) \times (0,5) = 0,6 + 0,25 = 0,85.$

E o novo subintervalo de 'EOF' inicia em $0,85$ e termina em

$0,85 + (1,0 - 0,9) \times (0,9 - 0,4) = 0,85 + (0,1) \times (0,5) = 0,85 + 0,05 = 0,90,$

conforme mostrado na segunda linha da tabela 2.4.

Quanto mais longa a mensagem se torna, menor o intervalo para representá-la, e maior o número de dígitos necessários para especificar o intervalo, conforme mostrado na tabela 2.4.

Quando o último símbolo é recebido, o intervalo $[0,8125; 0,8250)$ é designado. Qualquer valor dentro desse intervalo pode ser enviado para o decodificador. Por exemplo, vamos supor que o codificador envie o número, $0,812543$. Como o decodificador conhece a probabilidade dos símbolos e como os subintervalos foram designados no intervalo $[0; 1)$, o decodificador verifica que esse número recebido pertence ao subintervalo da letra 'b'. Como ocorre no codificador, o decodificador seleciona e escalona o subintervalo, conforme mostrado na tabela 2.4. Então é verificado que o número resultante também pertence ao intervalo da letra 'b'. Esse intervalo é selecionado e escalonado. E assim sucessivamente até o símbolo 'EOF' ser encontrado.

A função do símbolo 'EOF' é justamente indicar para o decodificador quando a mensagem termina. Se não há um símbolo designado para esta finalidade, o decodificador continuaria decodificando indefinidamente.

Essa implementação básica da codificação aritmética possui dois grandes problemas:

a) conforme o intervalo encolhe, um número maior de casas decimais é necessário para representar o intervalo, o que acarreta problema de precisão, já que nenhuma máquina possui precisão infinita;

b) nenhuma saída é produzida até toda a mensagem ter sido codificada. Se a mensagem for longa, poderia haver problema de armazenamento e de tempo para que a comunicação seja realizada.

Uma solução para resolver os dois problemas é liberar cada número logo que ele passe a ser conhecido e então multiplicar por 10 o valor dos extremos do intervalo de modo que ele só reflita a parte desconhecida do próprio intervalo.

Um número torna-se conhecido quando ele ocupa a casa decimal mais significativa dos valores extremos do intervalo. Por exemplo, quando o símbolo 'EOF' é recebido, a sub-faixa $[0,8125; 0,8250)$ é selecionada. Podemos notar que o número 8 ocupa a casa decimal mais significativa dos valores extremos do intervalo. Se o processo de codificação continuasse, o valor dessa casa decimal não seria mais alterado,

independentemente de quantos símbolos ainda seriam codificados. Logo, o codificador poderia enviar esse dígito para o decodificador, e multiplicar por 10 e desconsiderar a parte inteira dos extremos. O novo intervalo passaria a ser $[0,125; 0,250)$, e o processo continuaria como já explicado.

Outro ponto que devemos observar no nosso exemplo de codificação aritmética usando dígitos decimais é que para gerar o primeiro dígito, foram necessários receber quatro símbolos. Como podemos observar na tabela 2.3, o símbolo 'b' tem grande probabilidade, o que acaba causando esse comportamento, mostrando que ele codifica símbolos mais prováveis com menos, neste caso, dígitos.

Uma implementação prática do Codificador Aritmético em circuitos microprocessados usa aritmética inteira e binária [4], e não decimal como apresentamos. Um intervalo entre $[0; L)$, onde L é um inteiro cujo valor deve ser grande, é usado para representar o intervalo $[0; 1)$. Três pontos nesse intervalo devem ser conhecidos: a primeira quarta parte; a metade; e a última quarta parte.

Após um símbolo ser recebido, e o respectivo intervalo selecionado, cinco ações podem ser executadas:

a) Se o intervalo que representa o símbolo estiver entre $[\frac{1}{4}L; \frac{3}{4}L)$, significa que, em binário, os extremos do intervalo são $0,0\ldots$ e $0,1\ldots$. Como o *bit* mais significativo da parte decimal dos extremos não é o mesmo, não há uma definição de qual *bit* deve ser transmitido. Neste caso o algoritmo deve apenas incrementar um contador para saber quantas vezes esta situação aconteceu até que a transmissão de um *bit* aconteça, e dobrar o intervalo para ambos os lados.

b) Se o intervalo que representa o símbolo estiver entre $[0; \frac{1}{2}L)$, significa que, em binário, os extremos do intervalo são $0,0\ldots$ e $0,0\ldots$, ou seja, o *bit* mais significativo da parte decimal dos extremos é o mesmo e igual a 0. Neste caso um *bit* 0 seguido do número indicado pelo contador mencionado no item 'a' de *bit* 1 são transmitidos. Após a transmissão o contador deve ser zerado e o intervalo dobrado para a direita.

c) Se o intervalo que representa o símbolo estiver entre $[\frac{1}{2}L; 1)$, significa que, em binário, os extremos do intervalo são $0,1\ldots$ e $0,1\ldots$, ou seja, o *bit* mais significativo da parte decimal dos extremos é o mesmo e igual a 1. Neste caso um *bit* 1 seguido do número indicado pelo contador mencionado no item 'a' de *bit* 0 são transmitidos. Após a transmissão o contador deve ser zerado e o intervalo dobrado para a esquerda.

d) Se as situações do item 'a', 'b' e 'c' não ocorrem, o algoritmo simplesmente processa o próximo símbolo.

e) Se o símbolo de fim de mensagem for recebido, o algoritmo não lê outros símbolos. Neste caso

o algoritmo simplesmente executa os passo do item 'a', 'b', 'c' e 'd' até que os extremos do intervalo não esteja nos subintervalos $[0; \frac{1}{2}L)$, $[\frac{1}{4}L; \frac{3}{4}L)$ ou $[\frac{1}{2}L; 1)$.

O termo "Dobrar Intervalo" nos itens 'a', 'b' e 'c', significa fazer uma operação equivalente àquela mostrada no codificador aritmético usando dígitos decimais. Toda vez que um digito é transmitido os extremos do intervalo são mutiplicados po 10 e a parte inteira do resultado é ignorada, tornando intervalo 10 vezes maior. Quando usamos aritmética binária, os extremos devem ser multiplicado por 2 ou $(10)_2$ e a parte inteira do resultado descartada, dobrando o intervalo. Mas estamos usando um intervalo $[0; L)$, onde, L é um número inteiro para representar o intervalo $[0; 1)$. Neste caso, temos que "Dobrar Intervalo" para direita, esquerda ou ambos os lados para, além de dobrar o intervalo, mantê-lo dentro do intervalo $[0; L)$. Por exemplo, vamos supor que $L = 100$ e que o símbolo enviado pela fonte define o subintervalo $[80; 90)$. Se este subintervalo for simplesmente dobrado, o novo subintervalo será $[160; 180)$, que não pertence ao intervalo $[0; 100)$. Logo precisamos dobrar o subintervalo para esquerda que significa dobrar a diferença entre os extremos do subintervalo e o limite inferior dos intervalos que definem as situações 'a', 'b' ou 'c' $(0,25; 0,00; 0,50)$. Usando novamente o exemplo acima para dobrar o subintervalo de acordo com o critério apresentado, o subinterval $[80; 90)$ está de acordo com a situação 'c'. Para dobrar o intervalo, o cálculo deve ser $[2(80 - 50); 2(90 - 50)) = [60; 80)$. Como podemos notar, o subintervalo foi dobrado, e além de se manter dentro do intervalo $[0; 100)$, está a esquerda do intervalo anterior.

A tabela 2.5 demonstra como a codificação aritmética é realizada usando aritmética binária com a mesma sequência de símbolos do exemplo apresentado na tabela 2.4.

Passo	Intervalo	Entrada	Ação	Subintervalo		
				a	b	EOF
1	$[0; 1)$	b	Subdividir	$[0,00; 0,40)$	$[0,40; 0,90)$	$[0,90; 1,00)$
2	$[0,4; 0,9)$	b	Subdividir	$[0,40; 0,60)$	$[0,60; 0,85)$	$[0,85; 0,90)$
3	$[0,60; 0,85)$		Enviar um *bit* 1 Expande para a esquerda			
4	$[0,20; 0,70)$	b	subdividir	$[0,20; 0,40)$	$[0,40; 0,65)$	$[0,65; 0,70)$
5	$[0,40; 0,65)$		Está no subintervalo $[\frac{1}{4}L; \frac{3}{4}L)$ Incrementar contador Expandir para a ambos os lados			
6	$[0,30; 0,80)$	EOF	Subdividir	$[0,30; 0,50)$	$[0,50; 0,75)$	$[0,75; 0,80)$
7	$[0,75; 0,80)$		Enviar *bits* 10 Expande para a esquerda			
8	$[0,50; 0,60)$		Enviar um *bit* 1 Expande para a esquerda			
9	$[0,00; 0,20)$		Enviar um *bit* 0 Expande para a direita			
10	$[0,00; 0,40)$		Enviar um *bit* 0 Expande para a direita			
11	$[0,00; 0,80)$		Encerrar a codificação			

Tabela 2.5: Processo de codificação aritmética usando aritmética binária. Os intervalos representam o fragmento de L.

No passo 1 a ação é subdividir, selecionar o subintervalo correspondente ao símbolo que foi re-

cebido. No passo 2 são realizadas as mesmas ações do passo 1, considerando o símbolo recebido. No passo 3 o subintervalo selecionado está entre $\frac{1}{2}L$ e L. Nesse caso o codificador envia um *bit* 1, e dobra o subintervalo para a esquerda. No passo 4 são realizadas as mesmas ações do passo 1, considerando o símbolo recebido. No passo 5 o subintervalo selecionado está entre $\frac{1}{4}L$ e $\frac{3}{4}L$. O codificador deve incrementar o contador e dobrar o subintervalo para ambos os lados. No passo 6 é recebido o carácter de fim de mensagem. O codificador realiza as mesmas ações do passo 1 e não recebe mais qualquer símbolo. No passo 7 o subintervalo está entre $\frac{1}{2}L$ e L. Nesse caso o codificador envia um *bit* 1 seguido de um *bit* 0 porque desde o último *bit* enviado, o subintervalo esteve entre $\frac{1}{4}L$ e $\frac{3}{4}L$ uma vez. Quando isso acontece o *bit* de saída deve ser seguido pelo seu oposto o número de vezes que o subintervalo esteve entre entre $\frac{1}{4}L$ e $\frac{3}{4}L$. O subintervalo deve ser dobrado para a esquerda. No passo 8 o subintervalo selecionado está entre $\frac{1}{2}L$ e L. Nesse caso o codificador envia um *bit* 1, e dobra o subintervalo para a esquerda. No passo 9 o subintervalo selecionado está entre 0 e $\frac{1}{2}L$. Nesse caso o codificador envia um *bit* 0, e dobra o subintervalo para a direita. No passo 10 o subintervalo selecionado está entre 0 e $\frac{1}{2}L$. Nesse caso o codificador envia um *bit* 0, e dobra o subintervalo para a direita. No passo 11 o subintervalo não atende a nenhuma das situações para continuar e o processo de codificação é encerrado.

A sequência de *bits* enviada pelo codificador é 1101000. Se considerarmos essa sequência como a parte decimal de um número binário, temos:

$$0,1101 = 2^{-1} + 2^{-2} + 2^{-4} = 0,5 + 0,25 + 0,0625 = 0,1825,$$

que é o número do extremo inferior do intervalo obtido no exemplo da tabela 2.4.

A figura 2.2 apresenta um gráfico em barras que ilustra o número de ocorrência de cada *pixel* nas imagens "Columns" e "Pepper". Este tipo de gráfico é chamado de histograma.

Imagem	Taxa ($bits/pixel$)	Taxa de compressão (%)	Entropia de Primeira Ordem (H)
Baboon	7,35	8,15	7,3582
Barbara	7,45	6,94	7,4664
Columns	6,85	14,32	6.9266
Lena	7,42	7,20	7,4455
Pepper	7,56	5,48	7,5432

Tabela 2.6: Resultados obtidos com a codificação aritmética.

A tabela 2.6 apresenta os resultados da codificação aritmética aplicadas as imagens usadas nesta tese. A imagem que apresentou a maior taxa de compressão foi "Columns" porque possui uma grande concentração de *pixels* próximo ao valor máximo, fazendo com que os *pixels* possuam grande frequência relativa nesta região do histograma e um conjunto grande de *pixels* com baixa frequência relativa, o que contribui para um melhor desempenho do codificador aritmético. A imagem que apresentou o pior de-

sempenho foi a ”Pepper”, já que, entre as imagens usadas, é a que possui o histograma com a frequência relativa dos *pixels* mais equitativamente distribuída, como mostrado na figura 2.2.

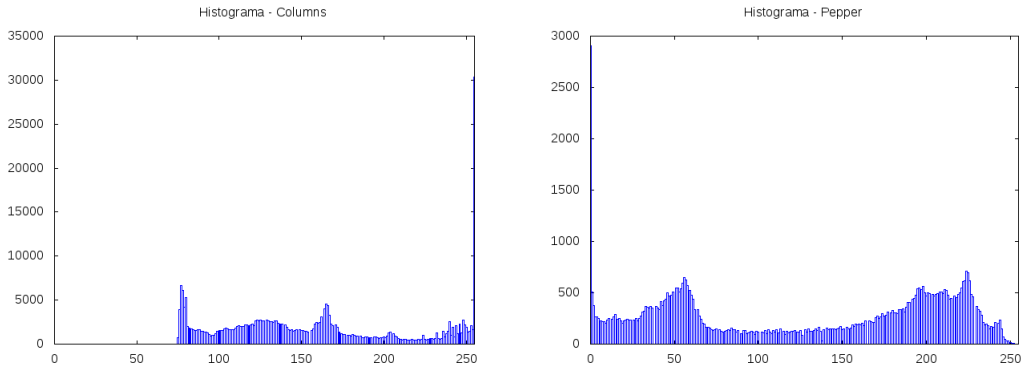


Figura 2.2: Histograma das imagens ”Columns” e ”Pepper”.

Se comparamos o valor de *bits* por *pixel* na tabela 2.6 com o valor da Entropia das imagens na tabela 2.1, vemos que os valores apresentados na primeira são menores do que na segunda tabela, exceto para a imagem Pepper. Essa diferença é causada pelo fato do algoritmo não conhecer a frequência relativa dos *pixels* da imagem a priori, e não que a teoria de Shannon esteja errada.

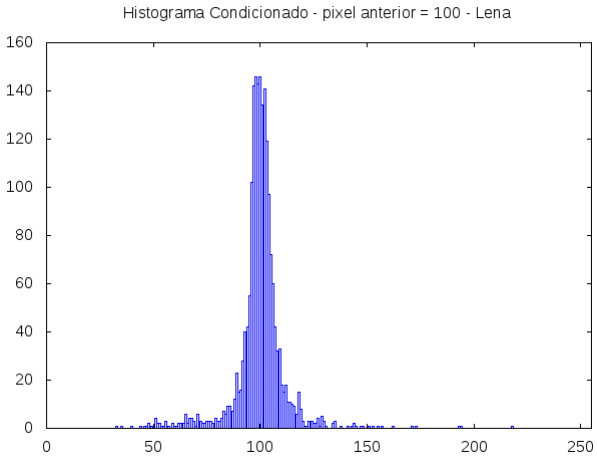
O codificador aritmético usado trabalha com um modelo probabilístico adaptativo, ou seja, a estatística da imagem é obtida conforme os *pixels* são codificados. Inicialmente todos os *pixels* são considerados equiprováveis, logo, é esperado, dependendo do valor dos *pixels* lidos inicialmente da imagem que as frequências relativas do modelo estatísco do codificador não retrate as mesmas frequências relativas da imagem. O modelo estatístico do codificador aritmético necessita codificar alguns *pixels* para que as frequências relativas do seu modelo comece a se aproximar das frequências relativas da imagem. Esse período de convergência acaba resultando nesta pequena distorção entre o número de $bit/pixel$ da imagem codificada e sua Entropia. A imagem ”Pepper” foi a única que apresentou uma taxa de $bits/pixel$ menor do que a Entropia porque dentre as imagens testadas é a que possui um histograma mais equitativamente distribuído, o que pode ter contribuído para uma convergência mais rápida das frequências relativas do modelo probabilístico do codificador aritmético durante o processo de codificação.

No anexo A são apresentadas as imagens usadas nas simulações realizadas nesta tese. Para cada imagem há uma tabela com dados da respectiva imagem. Um desses dados é a Entropia de segunda ordem, que é tomada considerando o *pixel* antecessor, considerando que a imagem é lida da esquerda para direita e de cima para baixo. Como pode ser notado a Entropia de segunda ordem é menor do que a Entropia de primeira ordem para todas as imagens, mostrando que há um certo grau de dependência entre os *pixels* de uma imagem natural. Se os *pixels* da imagem fossem independentes, a Entropia de segunda ordem seria igual a de primeira ordem. Logo, quanto menor a diferença entre esses dois parâmetros, menor a dependência entre os *pixels* de uma imagem.

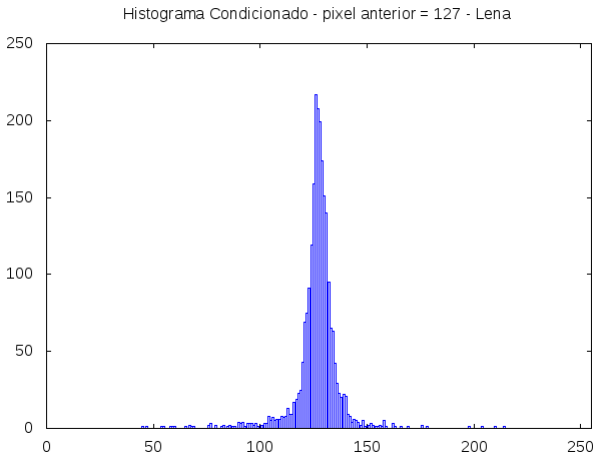
A independência entre dois eventos A e B indica que a ocorrência de um deles não altera a probabilidade de ocorrência do outro, ou seja, $P(A|B) = P(A)$. Se tomarmos a frequência relativa de um *pixel* de valor p e tomarmos a frequência relativa desse mesmo *pixel* quando o *pixel* anterior tem valor q, vamos verificar que, para as imagens com as quais trabalhamos, há diferença entre essas duas frequências relativas, indicando uma dependência entre os *pixels* da imagem.

Na figura 2.3 são apresentados três histogramas da imagem da Lena tomados quando o *pixel* antecessor possui o valor, 100, 127 e 200 respectivamente. Como pode ser observado, o envelope desses histogramas é totalmente diferente do envelope do histograma original da imagem da Lena (vide anexo A). Outro ponto interessante é que a concentração das maiores frequências relativas sempre está no em torno do valor do *pixel* antecessor, indicando uma dependência entre esses *pixels*.

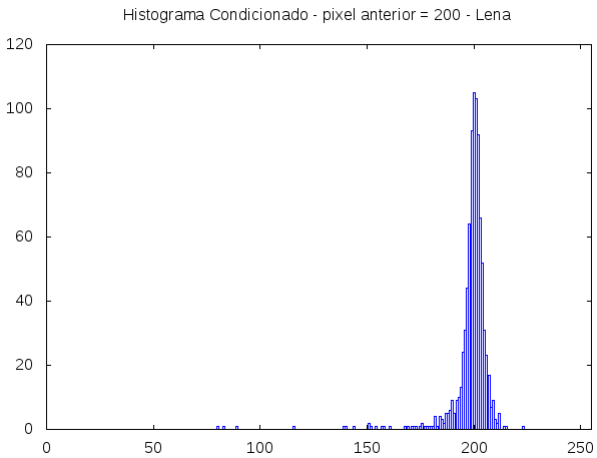
O codificador aritmético é considerado um codificador perfeito, no sentido de que permite que símbolos sejam representados com qualquer número arbitrário de *bits* mediante a escolha adequada do modelo probabilístico. Contudo, a escolha do modelo deve ser feita à parte e por isso diz-se que ele permite separar a codificação em si da modelagem estatística. Por exemplo, se usarmos um modelo estatístico fixo calculado a partir das frequências relativas da fonte, o desempenho será limitado pela Entropia de primeira ordem. Contudo, um modelo mais sofisticado, que varie símbolo a símbolo, levando em consideração as probabilidades dos símbolos condicionadas aos símbolos anteriores, pode apresentar um desempenho muito superior.



(a) *pixel* anterior = 100



(b) *pixel* anterior = 127



(c) *pixel* anterior = 200

Figura 2.3: Histograma da imagem da Lena para três diferentes valores do *pixel* antecessor

Capítulo 3

PREDIÇÃO LINEAR E CONDICIONAMENTO ESTATÍSTICO

3.1 Introdução

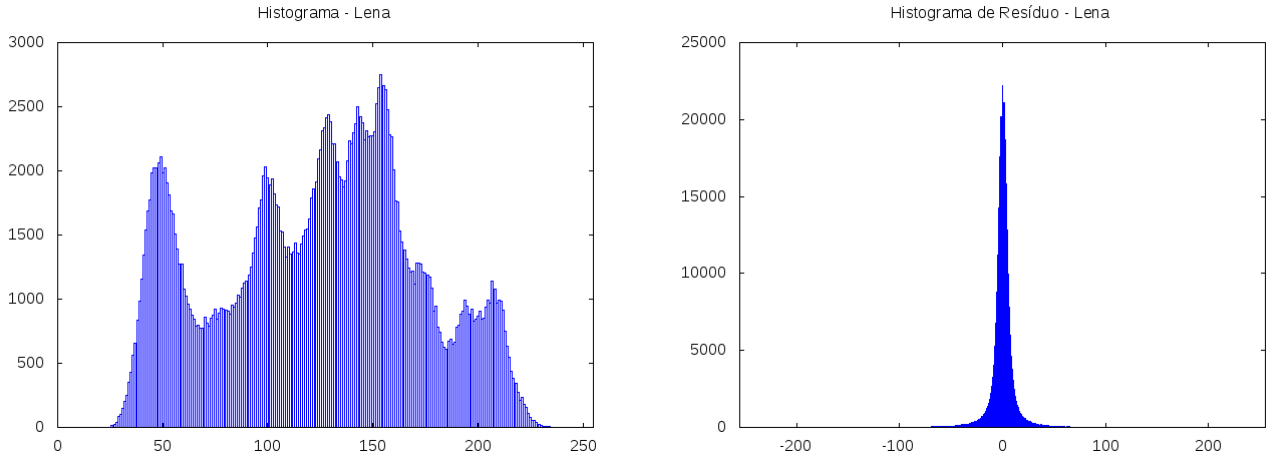
A taxa de compressão do codficador aritmético pode ser melhorada mediante a escolha adequada do modelo probabilístico ou transformando o conjunto de símbolos de uma fonte em um outro conjunto com menor Entropia. Neste capítulo vamos apresentar duas técnicas simples, onde cada uma explora uma das duas características. Ambas as técnicas não exploram a irrelevância e por esta razão são algoritmos de compressão sem perdas. Vamos apresentar os resultados de simulações e comparar com os já obtidos no capítulo 2.

A técnica que usamos para transformar o conjunto de *pixels* em um conjunto de símbolos com menor Entropia é chamada de "Predição Linear". A outra, que manipula o modelo probabilístico do codificador aritmético de acordo com símbolo que será codificado, é chamada de "Condicionamento Estatístico".

3.2 Predição Linear

Na técnica de Predição Linear, ao invés de codificar o *pixel*, a diferença do *pixel* pela sua estimativa, que chamamos de "resíduo", é codificada. A estimativa do valor do *pixel* é uma combinação linear dos *pixels* vizinho já codificados (causal). Como as imagens naturais possuem áreas onde há correlação entre os *pixels*, é esperado que a estimativa do valor do *pixel* seja bem próximo do seu valor real em grande parte de uma imagem, só possuindo valores muito diferentes nas regiões de bordas dos objetos que compõem a imagem.

A figura 3.1 apresenta o histograma da imagem e do resíduo da imagem da Lena, considerando que a estimativa do valor dos *pixels* seja o próprio valor do *pixel* antecessor. A Entropia do novo conjunto de símbolos resultante dessa operação é 5,065, que é menor que a Entropia da imagem original.



(a) : Original (b) : Resíduo

Figura 3.1: Histograma da imagem da Lena

Como pode ser observado na figura 3.1 o histograma do resíduo da imagem da Lena contém uma pequena faixa cujos valores possui uma grande frequência relativa. Devido a essa característica, a codificação aritmética do resíduo resulta em uma taxa de compressão maior que a codificação direta dos *pixels*.

A figura 3.2 contém a imagem do valor absoluto do resíduo da Lena. Como pode ser observado, os *pixels* com maior nível de energia, e que acabam se destacando, são os de algumas bordas das distintas áreas da imagem.

Figura 3.2: Imagem do valor absoluto do resíduo da "Lena".

Na nossa simulação o cálculo da estimativa do valor dos *pixels* é obtido apenas multiplicando o

pixel anterior pelo preditor, que é representado pela letra grega α. O objetivo deste fator é obter, na média, a melhor estimativa do valor do *pixel*, já que, como dito, os *pixels* de uma região são correlatados, mas não iguais.

A equação 3.1 representa matematicamente o cálculo do valor estimado do *pixel*.

$$Pixel_{estimado} = \alpha \; Pixel_{anterior}. \tag{3.1}$$

3.2.1 Determinação analítica do valor do preditor ótimo

A determinação analítica do valor do preditor ótimo α pode ser feita do seguinte modo:

Seja $x(n)$ um processo estocástico e $\hat{x}(n) = \alpha x(n-1)$ o resultado da predição. O resíduo é nesse caso dado por:

$$e(n) = x(n) - \alpha x(n-1) \tag{3.2}$$

Deseja-se minimizar a potência média do resíduo, dada por:

$$E\left\{e^2(n)\right\} = E\left\{\left[x(n) - \alpha x(n-1)\right]^2\right\} = E\left\{x^2(n)\right\} - 2\alpha E\left\{x(n)\,x(n-1)\right\} + \alpha^2 E\left\{x^2(n-1)\right\} \tag{3.3}$$

Se assumirmos que $x(n)$ é um processo estocástico estacionário, ou seja, $E\left\{x^2(n-1)\right\} = E\left\{x^2(n)\right\}$, temos:

$$E\left\{e^2(n)\right\} = \sigma_x^2 - 2\alpha\rho + \alpha^2\sigma_x^2, \tag{3.4}$$

onde:
σ_x^2 é a energia do sinal;
e ρ é a correlação entre o valor de cada *pixel* e o seu antecessor.

O coeficiente ótimo é obtido derivando-se a Equação (3.4) e igualando-se a zero, o que fornece:

$$\alpha = \frac{\rho}{\sigma_x^2} \tag{3.5}$$

3.2.2 Resultados

A tabela 3.1 apresenta os resultados obtidos nas simulações. O fator α ótimo foi determinado usando a equação 3.5.

Simulações também foram realizadas arbitrando o valor do preditor até que o melhor desempenho fosse encontrado. Os resultados obtidos para essas simulações são apresentados na tabela 3.2.

Imagem	Preditor (α)	*bits/pixel*	Taxa de compressão
Baboon	0,987	6,51	18,69%
Barbara	0,972	6,06	24,31%
Columns	0.999	3,27	59,16%
Lena	0,996	5,08	36,54%
Pepper	0,990	5,47	31,57%

Tabela 3.1: Resultados obtidos com a Predição Linear

Comparando os resultados da tabelas 3.1 e 3.2, observamos que a taxa de compressão ficaram bem próximas. As pequenas diferenças que ocorreram podem ter sua origem devido ao fato de termos considerado as imagens um processo estocástico estacionário para determinar o valor do preditor analiticamente, o que pôde resultar numa equação para o cálculo de um preditor quase ótimo.

Imagem	Preditor (α)	*bits/pixel*	Taxa de compressão
Baboon	0,94	6,49	18,86%
Barbara	0,99	6,04	24,48%
Columns	1,00	3,27	59,16%
Lena	0,99	5,08	36,50%
Pepper	1,00	5,45	31,92%

Tabela 3.2: Resultados obtidos com a Predição Linear arbitrando o valor do preditor

3.3 Condicionamento Estatístico

Uma outra forma de melhorar o desempenho do codificador aritmético é manipular o seu modelo probabilístico. Essa técnica é chamada de "Condicionamento Estatístico". Para cada símbolo (*pixel*) que vai ser codificado, a manipulação do modelo probabilístico é feita de tal forma que o símbolo que será codificado apresente uma probabilidade maior que sua probabilidade real, buscando reduzir assim o número de *bits* necessário para representa-lo após a codificação. Para o emprego dessa técnica, dois pontos devem ser observados:

1) A forma como a manipulação é realizada deve ser bem definida e de conhecimento do codificador e decodificador.

2) O decodificador não conhece o símbolo que será decodificado antes de decodifica-lo. Logo a manipulação do modelo probabilístico não pode ocorrer considerando o símbolo que será codificado, já que o decodificador não pode manipular o seu modelo probabilístico considerando um símbolo que ele não conhece. Por esta razão, e considerando a correlação existente em muitas regiões de uma imagem, a manipulação do modelo probabilístico é realizada considerando o símbolo codificado anteriormente.

Na nossa simulação, para mostrar o desempenho do Condicionamento Estatístico, a manipulação do modelo probabilístico é realizada multiplicando a frequência relativa por uma função exponencial, cujo

valor da base deve estar entre 0 e 1 e o expoente é um vetor que expressa a distância euclidiana entre o valor do *pixel* e todos os valores que o *pixel* pode assumir, conforme apresentado na equação 3.6.

$$Freq_{temp} = \lfloor 1 + Freq_{rel}\ b^{D[n]} \rfloor \tag{3.6}$$

onde:

$\lfloor . \rfloor$ significa que apenas a parte inteira do resultado será considerada;

$D[n]$ é um vetor contendo a distância euclidiana entre o valor do símbolo e todos os valores que um símbolo pode assumir;

b é base da função exponencial;

$Freq_{rel}$ é a frequência relativa do modelo probabilístico do codificador aritmético;

$Freq_{temp}$ é a frequência relativa manipulada que chamamos de frequência relativa temporária.

O modelo estatístico usado pelo codificador aritmético inicia a frequência relativa de todos os símbolos com o valor 1. Isto acontece porque o codificador aritmético usado determina a probabilidade de cada símbolo conforme a equação 3.7, onde: $P[simbolo]$ é a probabilidade do símbolo; $Freq_{rel}[simbolo]$ é a frequência relativa do símbolo que se deseja determinar a probabilidade; e $\sum Freq_{rel}$ é o somatório da frequência relativa de todos os símbolos. Se a frequência relativa dos símbolos for iniciada com o valor zero, a probabilidade dos símbolos seria zero, e deste modo não conseguiria particionar o intervalo e realizar a codificação. Como o menor valor que a parte $Freq_{rel}\ b^{D[n]}$ da equação 3.6 pode assumir é zero, e para garantir que a frequência relativa temporária não assuma valores menor que 1, tornando o codificador aritmético incapaz de realiza a codificação, o valor 1 é somado a $Freq_{rel}\ b^{D[n]}$ na equação 3.6.

$$P[simbolo] = \frac{Freq_{rel}[simbolo]}{\sum Freq_{rel}}, \tag{3.7}$$

Por exemplo, vamos supor que uma fonte pode gerar um conjunto de 10 símbolos (0, 1, 2, 3, 4, 5, 6, 7, 8 e 9). Inicialmente o vetor das frequências relativas contém o valor 1 para todos os símbolos. A probabilidade de ocorrência de cada símbolo é 1/10. Vamos supor que o primeiro símbolo gerado seja o número ”5”. Após manipular a frequência relativa usando a equação 3.6 (vide tabela 3.3), o símbolo ”5” passa a ter uma probabilidade de 18%, uma melhora de 8 pontos percentuais em relação ao modelo original.

No exemplo acima, a manipulação do modelo estatístico usou o próprio símbolo que seria codificado. Como já foi dito, este não é o procedimento que deve ser adotado, já que o decodificador não teria como fazer a manipulação do modelo probabilístico, sem conhecer o símbolo. Logo, o algoritmo deteremina o modelo probabilístico temporário usando o símbolo codificado anteriormente. Como um exemplo vamos supor que a fonte já gerou a seguinte sequência de símbolos: $[5; 1; 2; 5; 6; 8; 2; 8]$. O novo

Símbolo	$freq_{rel}$	$D[n]$	b	$b^{D[n]}$	$\lfloor 1 + freq_{rel} * b^{D[n]} \rfloor$	Probabilidade
0	1	25	0,9	0,071790	1	1/11
1	1	16	0,9	0,185302	1	1/11
2	1	9	0,9	0,387420	1	1/11
3	1	4	0,9	0,656100	1	1/11
4	1	1	0,9	0,900000	1	1/11
5	**1**	**0**	**0,9**	**1,000000**	**2**	**2/11**
6	1	1	0,9	0,900000	1	1/11
7	1	4	0,9	0,656100	1	1/11
8	1	9	0,9	0,387420	1	1/11
9	1	16	0,9	0,185302	1	1/11

Tabela 3.3: Detalhes do cálculo da manipulação da frequência relativa.

símbolo gerado pela fonte é o digito 9. Neste caso o símbolo 8 seria usado para deteremina o modelo probabilístico temporário. Como mostrado na tabela 3.4. Mesmo manipulando o modelo estatístico usando o símbolo enviado anteriormente, a probabildade do símbolo "9" foi alterada de 1/18 para 1/11, ou seja, a probabilidade do símbolo "9", aumentou de 5,8% para 6,2%.

Símbolo	$freq_{rel}$	$D[n]$	b	$b^{D[n]}$	$\lfloor 1 + freq_{rel} * b^{D[n]} \rfloor$	Probabilidade
0	1	81	0,9	0,001179	1	1/11
1	2	64	0,9	0,005726	1	1/11
2	3	49	0,9	0,022528	1	1/11
3	1	36	0,9	0,071790	1	1/11
4	1	25	0,9	0,185302	1	1/11
5	3	16	0,9	0,387420	1	1/11
6	2	9	0,9	0,656100	1	1/11
7	1	4	0,9	0,900000	1	1/11
8	3	1	0,9	1,000000	2	3/11
9	**1**	**0**	**0,9**	**0,900000**	**1**	**1/11**

Tabela 3.4: Detalhes do cálculo da manipulação da frequência relativa.

A principal função da manipulação do modelo probabilístico apresentada é a redução das frequências relativas dos símbolos conforme a distância euclidiana entre o símbolo tomado como a referência e os outros símbolos do conjunto gerado pela fonte aumenta. Para símbolos com distância euclidiana pequenas, a tendência é que a frequência relativa se mantenha a mesma. O único símbolo cuja frequência relativa aumenta em uma unidade é o símbolo tomado como a referência para o cálculo da distância euclidiana, que, normalmente, não é o que será codificado. Logo, se o processo reduz a frequência relativa de alguns símbolos é de se esperar que $\sum Freq_{rel}$ na equação 3.6 diminua e consequentemente a probabilidade do símbolos cuja frequência relativa não tenha grande alteração, aumente. O número de símbolos no em torno do símbolo tomado como referência para o cálculo da distância euclidiana cuja frequência relativa não altera ou altera pouco é regulado pela base do expoente na equação 3.6. Logo para imagens cujo histograma de resíduo e mais espalhado, a base tende a ter um valor ótimo mais próximo da unidade, já que, como pode ser observado na figura 3.3, conforme a base tende a 1, a parte central da curva $y = b^{|x|}$ fica mais larga. Isso mantem ou reduz pouco a probabilidade de *pixels* cujo valor está mais afastado do centro, mas em contrapartida aumenta a soma das frequências relativas resultando em um valor menor da probabilidade do *pixel* de interesse. Como uma consequência deste fato, imagens cuja diferença do

pixel e seu antecessor está mais conentrada no em torno de zero possui um ganho de compressão maior quando usamos este tipo de Condicionamento Estatístico.

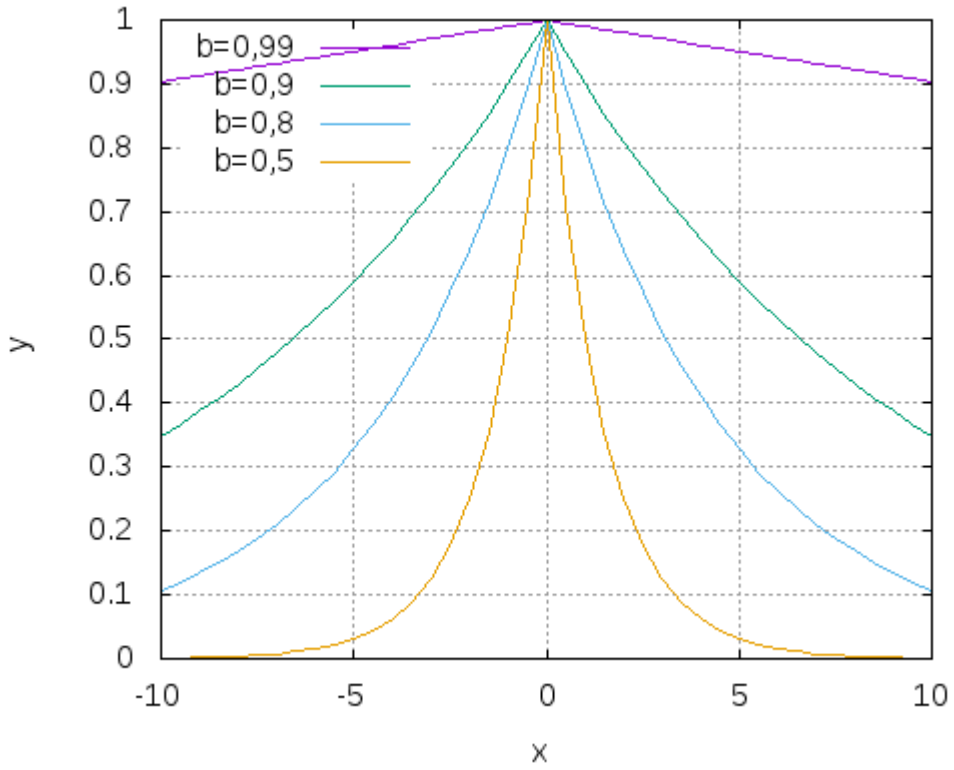


Figura 3.3: Função $y = b^{|x|}$, para b igual 0,99; 0,9; 0,8 e 0,5.

A tabela 3.5 apresenta os melhores resultados obtidos com o Condicionamento Estatístico, destacando o valor da base usada. Se compararmos as tabelas 3.1 e 3.5, vemos que o desempenho do Condicionamento Estatístico foi um pouco abaixo que o desempenho da predição, mas apresentou resultados satisfatórios para ser considerado como uma alternativa para melhorar o desempenho da codificação aritmética. O desafio é como manipular os dados disponíveis para criar o modelo estatístico temporários que maximize o ganho da compressão.

Imagem	Base (b)	*bits/pixel*	Taxa de compressão
Baboon	0,9980	6,50	18,79%
Barbara	0,9980	6,51	18,61%
Columns	0,9900	4,08	48,97%
Lena	0,9930	5,30	33,71%
Pepper	0,9930	5,37	32,90%

Tabela 3.5: Melhores resultados obtidos com o condicionamento estatístico

Capítulo 4

ALGORITMO DE COMPRESSÃO MULTI-DIMENSIONAL MULTISCALE PARSER (MMP)

4.1 Introdução

O MMP é um esquema de compressão com perdas baseado na busca de padrões que explora a irrelevância e redundância de imagens para obter alta taxa de compressão. A irrelevância pode ser observada no algoritmo quando a informação original é representada por cópias aproximadas, e a redundância quando as cópias aproximadas são representadas por índices que são codificados com menor número de *bits*, se comparados a imagem original [2].

4.2 MMP

O MMP possui, basicamente, dois estágios: transformação; e codificação de entropia. Na transformação, um vetor composto por símbolos gerados pela fonte é recebido como entrada. Esse vetor é comparado com os elementos de um dicionário, e o índice que identifica o elemento do dicionário que melhor o representa é enviado para o codificador aritmético para ser codificado, conforme apresentado no capítulo 2. A busca é feita para o vetor de entrada, bem como para suas segmentações. A segmentação é realizada de modo que sempre resulte em 2 subvetores de mesma dimensão. Todos os elementos do vetor que são segmentados devem pertencer somente a um dos dois subvetores. Esses subvetores devem ser segmentados em outros subvetores até que subvetores de uma única dimensão sejam alcançados. Para que a segmentação atenda as condições descritas, a dimensão do vetor de entrada deve ser uma potência de 2.

A figura 4.1 apresenta graficamente a segmentação de um vetor de 4 elementos. Essa representação é chamada de "Arvore de Segmentação". Cada vetor é chamado de "nó". Cada conjunto de vetores de mesma dimensão é chamado de "Escala". E cada vetor de uma escala é chamado de "Partição da Escala". Na árvore de segmentação, cada vetor ou nó é representado por V_l^k, onde, o índice superior, k, especifica a escala, e o índice inferior, l, especifica a partição ou vetor da escala.

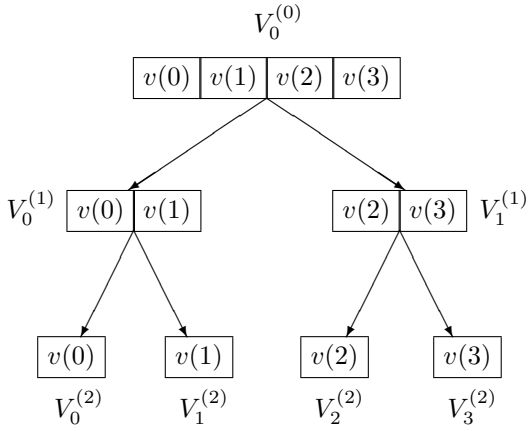


Figura 4.1: Árvore de segmentação de um vetor de entrada de dimensão $N = 4$.

O número de escalas que uma árvore de segmentação possui depende da dimensão do vetor de entrada, e pode ser determinado pela equação 4.1.

$$K = 1 + \log_2(N), \tag{4.1}$$

onde:
K é o número de escalas que uma árvore de segmentação;
N é a dimensão do vetor de entrada.

O número de partições que uma determinada escala possui é determinado pela equação 4.2.

$$P = 2^k, \tag{4.2}$$

onde:
P é o número de partições em uma escala;
k é o parâmetro que identifica a escala;

O número de elementos de cada vetor é determinado pela equação 4.3.

$$E = \frac{N}{2^k}, \tag{4.3}$$

onde:
E é o número de elementos dos vetores de uma determinada escala;

N é o número de elementos do vetor de entrada;
k é o parâmetro que identifica a escala.

Os elementos de um vetor de uma escala, pode ser obtido pela equação 4.4.

$$\mathbf{V}_l^{(k)} = \left[v(\frac{Nl}{2^k}), v(\frac{Nl}{2^k}+1),\ \ldots\ , v(\frac{Nl}{2^k}+m) \right], \tag{4.4}$$

onde:
N é o número de elementos do vetor de entrada;
k é o parâmetro que identifica a escala (pode assumir valores de 0 a $\log_2(N)$);
l é o parâmetro que identifica a partição (pode assumir valores de 0 a $2^k - 1$);
m pode assumir valores de 0 a $\frac{N}{2^k} - 1$.

4.2.1 Dicionário

Para cada nó da árvore de segmentação deve ser encontrado um elemento do dicionário que melhor o represente. Logo o dicionário deve ser composto de $1 + \log_2(N)$ subdicionários, um para cada escalas que a árvore de segmentação possua.

Para um dicionário conter todos vetores que contemplam todas as combinações possíveis de símbolos gerados pela fonte, seria necessário um grande recurso de armazenamento. Por exemplo, se o vetor de entrada é composto de quatro elementos, e se cada elemento pudesse assumir valores entre 0 e 255, seriam necessários, aproximadamente, 17 gigabytes de memória para armazenar somente os vetores para escala 0. Por esta razão, o MMP trabalha com um dicionário de dimensão reduzida, que não tem como representar de forma fiel todos os possíveis vetores de entrada. Para selecionar o elemento do dicionário que melhor representa um nó da árvore de segmentação, um critério de fidelidade deve ser usado.

Inicialmente a quantidade de elementos em cada escala do dicionário é igual ao numero de valores que um *pixel* pode assumir, sendo que todos os componentes de cada elemento do dicionário possuem um dos valores que os *pixels* podem assumir. Conforme os vetores de entrada vão sendo codificados, novos elementos são inseridos nas escalas do dicionário, aumentando a probabilidade de que elementos das escalas mais baixas encontrem bons candidatos que atendam ao critério de fidelidade e tenham seus índices transmitidos, aumentando a taxa de compressão.

4.2.2 Modelo probabilístico do Codificador Aritmético

Para codificar os índices dos elementos do dicionário que representam os segmentos do vetor de entrada, o codificador aritmético deve manter um modelo probabilístico para cada escala do dicionário. O modelo probabilístico é adaptativo e inicialmente todos os elementos de cada escala possuem a mesma probabilidade que vai sendo atualizada sempre que um índice é codificado.

4.2.3 Critério de Fidelidade

Para decidir quais elementos e de quais escalas do dicionário serão selecionados para representar o vetor de entrada, um critério de fidelidade deve ser estabelecido. No MMP, o critério de fidelidade é o "Custo Lagrangeano", que calcula o nível de distorção que a escolha de um elemento do dicionário causa, levando em consideração a quantidade de *bits* necessárias para codificar o índice que o identifica no dicionário (taxa-distorção). A seleção dos elementos do dicionário, bem como a escolha dos segmentos que representam o vetor de entrada, é decidido considerando o custo Lagrangeano. Isto significa que a escolha de um elemento de uma escala do dicionário pode causar mais distorção em detrimento do uso de menos *bits* para codificar o índice que o representa.

O custo Lagrangeano é determinado somando a distância euclidiana entre vetor da árvore de segmentação e um elemento da respectiva escala do dicionário com o número de *bits* (taxa) necessário para codificar o índices que o identifica no dicionário, conforme equação 4.5. A taxa é determinada pela equação 2.1, $-\log_b p_s$. A probabilidade p_s é obtida dos dados do modelo probabilistico do codificador aritmético. A taxa é poderada por uma constante λ, cujo valor deve ser definido no início do algoritmo e sua função é determinar o quanto de distorção pode ser aceitável em detrimento de uma menor taxa para codificar o índice do dicionário. Quanto maior o valor de λ, maior a distorção da imagem recuperada após ser comprimida. Se λ for igual a 0, a imagem é comprimida sem qualquer distorção, ou seja, sem erros. Em [2] é demonstrado que o custo Lagrangeano minimiza a taxa-distorção.

$$J = D + \lambda R \tag{4.5}$$

onde:
J é o custo lagrangeano;
D a distância euclidiana entre o vetor e o elemento do dicionário;
R número teórico de *bits* necessário para codificar o índice do dicionário (taxa).

4.2.4 Otimização da árvore de segmentação

Após ser definido quais elementos do dicionário que melhor representa os nós da árvore de segmentação, o MMP determina quais segmentações (partição) do vetor de entrada terão os índices do elemento do diconário que melhor os representam codificados e consequentemente transmitidos. Essa atividade é chamada de "Otimização da Árvore de Segmentação".

Para que o descompressor consiga determinar de qual partição e escala um determinado índice recebido representa, informação adicional deve ser enviada. Essa informação adicional é chamada de "*flag* de segmentação". O alfabeto do *flag* é composto de apenas dois símbolos: "S" (segmenta); e "M" (*matching*). O primeiro indica que a partição foi segmentada. O segundo indica que a partição não foi segmentada e o índice recebido referencia a própria partição. Como acontece com os índices do dicionário,

existe um modelo probabilistico para os *flags*, e o seus custos Lagrangeanos são considerados na decisão de otimização da árvore de segmentação, como veremos a seguir.

A figura 4.2 apresenta uma árvore de segmentação mostrando o índice e o respectivo valor do custo Lagrangeano para os cada nó, bem como e o custo Lagrangeano de cada *flag* que indica que a partição será segmentada. Os $I_l^{(k)}$ e $J_l^{(k)}$ são os índices e custo Lagrangeano, respectivamente, do elemento do dicionário que melhor representa o vetor da escala k e partição o l. Os $J_{F_{Sl}}^{(k)}$ são os custo Lagrangeano dos *flags* para indicar a segmentação do vetor da escala k e partição l.

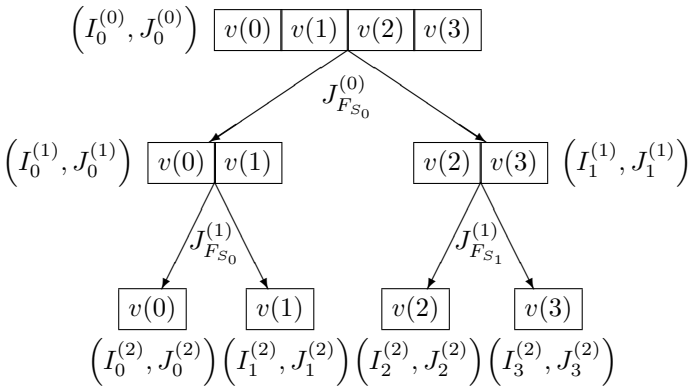


Figura 4.2: Árvore de segmentação de um vetor de dimensão $N = 4$.

Já que os vetores da última escala não podem ser segmentados, o processo de otimização da árvore de segmentação, para definir quais índices serão enviados, começa na penúltima escala. Os vetores dessa escala são analisados, e assim regressivamente até a primeira escala ou escala 0.

Para decidir se um dado índice $I_l^{(k)}$ é o que vai ser codificado, o custo Lagrangeano do vetor representado pelo índice deve ser comparado com o custo Lagrangeano de sua segmentação. Para fazer essa comparação, o custo Lagrangeano da segmentação do vetor é obtido somando os custos Lagrangeano do (flag) de segmentação e dos dois subvetores resultantes da segmentação, conforme a equação 4.6.

$$J_l^{(k)'} = J_{F_l}^{(k)} + J_{2l}^{(k+1)} + J_{2l+1}^{(k+1)}, \tag{4.6}$$

onde,

$J_l^{(k)'}$ é o custo total para codificar o *flag* de segmentação e os vetores resultantes da segmentação;
$J_{F_l}^{(k)}$ é o custo para codificar o *flag* de segmentação;
$J_{2l}^{(k+1)}$ e $J_{2l+1}^{(k+1)}$ são os custos para codificar cada um dos subvetores resultantes da segmentação.

Após determinar $J_l^{(k)'}$, ele é comparado à soma do custo para codificar o *flag* que indica que o vetor não será segmentado, $J_{M_l}^{(k)}$, com o custo para codificar o índice do elemento do dicionário que representa o vetor $J_l^{(k)}$. Se $J_l^{(k)'} < J_{F_l}^{(k)} + J_{M_l}^{(k)}$, o algoritmo conclui que é melhor segmentar, caso contrário, não

segmenta. Se a segmentação for a melhor opção, o custo Lagrangeano do vetor que está sendo analisado passa a ser $J_l^{(k)'}$, caso contrário, o algoritmo poda a árvore de segmentação a partir do vetor que está sendo analisado. Após encerrar a análise do vetor, o algoritmo segue para analisar um outro vetor da mesma escala. Caso não haja mais vetor na escala para ser analisado, o algoritmo começa a analisar os vetores da escala anterior. Esse processo segue até que o vetor de entrada seja analisado.

A figura 4.3 mostra um exemplo da árvore de segmentação da figura 4.6 otimizada.

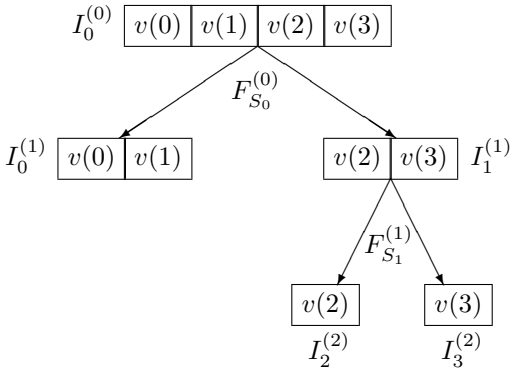


Figura 4.3: Árvore de segmentação de um vetor de dimensão $N = 4$ com os custos Lagrangeano.

4.2.5 Codificação dos índices

Após terminar a otimização da árvore de segmentação, os índices e *flags* são enviados para o codificador aritmético no sentido crescente das escalas e partições. Vamos supor que após a otimizção a configuração da árvore seja a mostrada na figura 4.3. A sequência de *flag* e índices enviado para o codificador aritmético é $F_{S_0}^{(0)}$, $F_{M_0}^{(1)}$, $I_0^{(1)}$, $F_{S_1}^{(1)}$, $F_{M_2}^{(2)}$, $I_2^{(2)}$, $F_{M_3}^{(2)}$, e $I_3^{(2)}$. Relembrando, $F_{S_l}^{(k)}$ é o *flag* indicando que o vetor da escala k e partição l deve ser segmentado. $F_{M_l}^{(k)}$ é o *flag* indicando que o vetor da escala k e partição l não deve ser segmentado, e o índice do elemento do dicionário que representa esse vetor deve ser enviado. $I_l^{(k)}$ é o índice do dicionário que representa o vetor da escala k e partição l.

Conforme os *flags* e índices são codificados, os respectivos modelos probabilísticos do codificador aritmético são atualizados.

4.2.6 Atualização do Dicionário

A atualização do dicionário é realizada com a concatenação dos elementos do próprio dicionário cujos índices foram codificados e transmitidos. Então uma versão expandida e outra contraída dessas concatenações é computada. Por exemplo, na figura 4.3, os elementos do dicionário que representam os vetores $V_2^{(2)}$ e $V_3^{(2)}$ são concatenados para formar um vetor da segunda partição da primeira escala, que será chamado de $V_1^{(1)^*}$. O elemento do dicionário que representa o vetor $V_0^{(1)}$ é concatenado com o vetor $V_1^{(1)^*}$ para formar o vetor da escala zero, $V_0^{(0)^*}$. Então uma versão expandida e contraida dos

vetores $V_0^{(0)^*}$ e $V_1^{(1)^*}$ é computada usando interpolação e decimação respectivamente. É lógico que no exemplo simplista da figura 4.3, o vetor $V_0^{(0)^*}$ só será contraído e o vetor $V_1^{(1)^*}$ expandido, já que não há necessidade de computar versões para os vetores da última escala porque essa já possui vetores com todas as combinações dos símbolos gerados pela fonte.

Os elementos do dicionário cujos os índices foram codificados são usados no lugar dos segmentos do vetor de entrada porque o dicionário do descompressor precisa estar sincronizado com o dicionário do compressor para realizar a recuperação do vetor corretamente. Como o MMP é um algoritmo com perdas, o descompressor não conhece o real conteúdo do vetor de entrada, e desse modo, não poderia realizar a atualização de forma correta se o compressor usasse essa informação para atualizar o dicionário, impossibilitando a sincronização do dicionário do descompressor com o dicionário do compressor.

Após gerar as versões expandidas e contraídas dos blocos concatenados, conforme descrito, é verificado nas respectivas escalas do dicionário se já existem elementos iguais, e caso não haja, eles são inseridos no dicionário, bem como os índices que os identificam são, também, inseridos nas respectivas escalas do modelo probabilístico do codificador aritmético.

A figura 4.4 apresenta o fluxograma do MMP para codificar cada vetor de entrada.

4.2.7 Recuperando a representação do vetor de entrada no descompressor

Quando o descompressor recebe a sequência $F_{S_0}^{(0)}$, $F_{M_0}^{(1)}$, $I_0^{(1)}$, $F_{S_1}^{(1)}$, $F_{M_2}^{(2)}$, $I_2^{(2)}$, $F_{M_3}^{(2)}$, e $I_3^{(2)}$, conhecendo a dimensão do vetor de entrada e a ordem que os índices e *flags* são codificados, o trabalho do descompressor resume em selecionar o elemento na escala correta do seu dicionário e montar a representação do vetor de entrada. Ao receber o *flag* $F_{S_0}^{(0)}$, ele toma conhecimento que o vetor de entrada foi segmentado. Logo em seguida é recebido o *flag* $F_{M_0}^{(1)}$, que indica que o próximo símbolo recebido é o índice do elemento do dicionário que representa a primeira partição da primeira escala do vetor de entrada. Ao receber o índice $I_0^{(1)}$, ele vai na respectiva escala do seu dicionário e lê o elemento identificado pelo índice recebido e insere na respectiva posição da representação do vetor de entrada que está sendo recuperado. Ao receber o *flag* $F_{S_1}^{(1)}$, o descompressor toma conhecimento que o vetor da segunda partição da primeira escala também foi segmentado. E assim ele vai compondo a representação do vetor de entrada até que o processo seja concluído.

Conhecendo os elementos do dicionário que representam os segmentos do vetor de entrada, o descompressor atualiza o seu dicionário da mesma forma que o compressor.

O descompressor sabe quando a seguimentação atinge a última escala, logo os *flags* $F_{M_2}^{(2)}$ e $F_{M_3}^{(2)}$ não precisam ser enviados, indicando que os vetores $V_2^{(2)}$ e $V_3^{(2)}$ não serão segmentados, já que eles não podem ser segmentados. Desse modo os *flags* e índices que realmente necessitam ser enviados são $F_{S_0}^{(0)}$, $F_{M_0}^{(1)}$, $I_0^{(1)}$, $F_{S_1}^{(1)}$, $I_2^{(2)}$, e $I_3^{(2)}$.

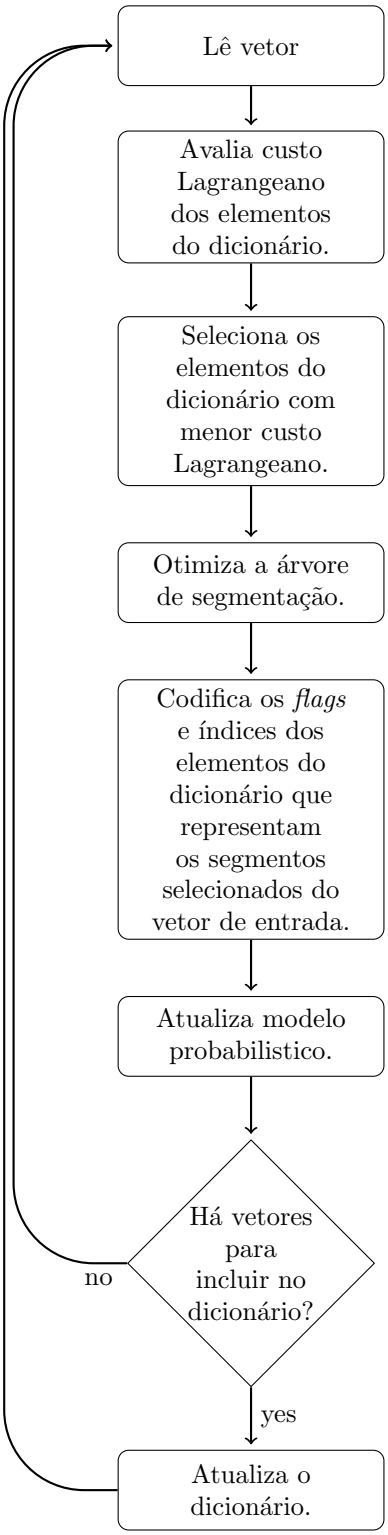


Figura 4.4: Fluxo do MMP.

4.3 MMP-2D

O MMP usado nas simulações realizadas com as imagens é uma versão em duas dimensões, chamada de MMP-2D. O MMP-2D trabalha do mesmo modo que o MMP, mas recebe como entrada um bloco de dimensões N x M, e não um vetor. Como a entrada é um bloco, a segmentação ocorre em duas dimensões, o que acaba tendo um impacto na forma como a árvore de segmentação é estabelecida e otimizada. As outras atividades do algoritmo são realizadas como no MMP, mas devemos sempre ter em mente que a entrada é um bloco, e não um vetor. Logo, os elementos do dicionário são blocos. E a expansão e contração para a atualização do dicionário são realizadas nas duas dimensões após concatenar dois segmentos de um bloco da árvore de segmentação cujos índices foram enviados para o codificador aritmético.

Para a imagem ser segmentada atendendo aos critérios apresentados para o MMP, as dimensões do bloco, N e M, devem ser uma potência de 2.

A imagem poderia ser lida como vetor pelo MMP, mas como já foi dito, uma imagem natural normalmente possui áreas cujos *pixels* são correlatos. Logo, realizando a leitura em blocos, uma melhor taxa de compressão é conseguida, já que a probabilidade de representar uma quantidade maior de *pixels* com apenas um ínidice é maior do que se a imagem fosse lida como vetor.

4.3.1 Árvore de Segmentação 2D

A grande diferença no processo de compressão do MMP-2D em relação ao MMP, é forma que a árvore de segmentação é estabelecida e otimizada. A figura 4.5 apresenta uma árvore de segmentação para um bloco de entrada de dimensões 2×2. Como pode ser observado, as escalas e partições são definidas por duas variáveis. Outro ponto é que os blocos da escala $(1, 1)$ são alcançados duas vezes no processo de segmentação. Para blocos de entrada de dimensões maiores, onde $N = M$, isto pode acontecer em outras escalas da árvore de segmentação. Para reduzir o tempo de processamento o algoritmo deve possuir um mecanismo para verificar se os blocos de uma determinada escala já foram visitados durante a otimização da árvore de segmentação.

O número de escalas que a árvore de segmentação possui depende da dimensão do bloco de entrada, e pode ser determinada pela equação 4.7.

$$K = [1 + \log_2(N)] \times [1 + \log_2(M)], \tag{4.7}$$

onde:

K é o número de escalas que uma árvore de segmentação possui;

N é o número de linhas do bloco de entrada;

M é o número de colunas do bloco de entrada.

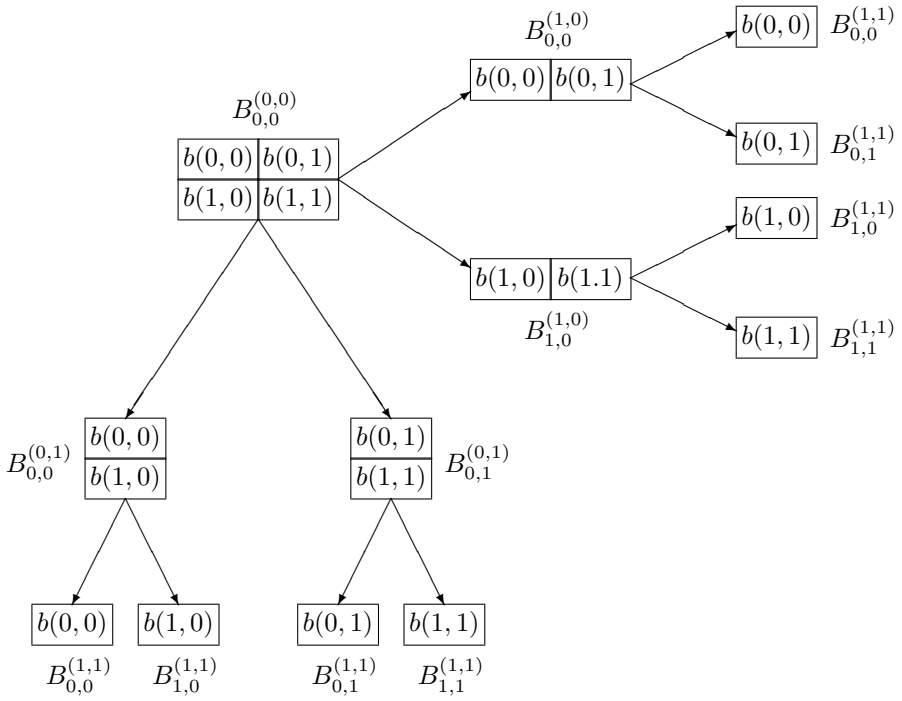


Figura 4.5: Árvore de segmentação para um bloco 2×2.

O número de partições que uma determinada escala possui é determinado pela equação 4.8.

$$P = 2^p \times 2^q \tag{4.8}$$

onde:

P é o número de partições de uma escala;

p é o parâmetro que identifica a escala vertical;

q é o parâmetro que identifica a escala horizontal.

O número de linhas e colunas que um bloco de uma determinada escala possui é determinado pelas equações 4.9 e 4.10 respectivamente.

$$L = \frac{N}{2^p}, \tag{4.9}$$

$$C = \frac{M}{2^q}, \tag{4.10}$$

onde:

L é o número de linhas do bloco;

C é o número de colunas do bloco;

N é o número de linhas do bloco de entrada;

M é o número de colunas do bloco de entrada.

O número de elementos de um bloco da escala (p, q) é igual a $L \times C$.

Os elementos de um bloco de uma escala, pode ser obtido pela equação 4.11.

$$\mathbf{B}_{k,l}^{(p,q)} = \begin{bmatrix} b(\frac{Nk}{2^p}+0;\frac{Ml}{2^q}+0) & b(\frac{Nk}{2^p}+0;\frac{Ml}{2^q}+1) & \dots & b(\frac{Nk}{2^p}+0;\frac{Ml}{2^q}+\frac{M}{2^q}-1) \\ b(\frac{Nk}{2^p}+1;\frac{Ml}{2^q}+0) & b(\frac{Nk}{2^p}+1;\frac{Ml}{2^q}+1) & \dots & b(\frac{Nk}{2^p}+1;\frac{Ml}{2^q}+\frac{M}{2^q}-1) \\ \vdots & \vdots & & \vdots \\ b(\frac{Nk}{2^p}+\frac{N}{2^p}-1;\frac{Ml}{2^q}+0) & b(\frac{Nk}{2^p}+\frac{N}{2^p}-1;\frac{Ml}{2^q}+1) & \dots & b(\frac{Nk}{2^p}+\frac{N}{2^p}-1;\frac{Ml}{2^q}+\frac{M}{2^q}-1) \end{bmatrix} \tag{4.11}$$

onde:

N é o número de linhas do bloco de entrada;

M é o número de colunas do bloco de entrada;

p é o parâmetro que identifica a escala vertical. p pode assumir valores de 0 a $\log_2(N) - 1$;

q é o parâmetro que identifica a escala horizontal. q pode assumir valores de 0 a $\log_2(M) - 1$;

k é o parâmetro que identifica a partição quando a segmentação ocorre na vertical. k pode assumir valores de 0 a $2^p - 1$;

l é o parâmetro que identifica a partição quando a segmentação ocorre na horizontal. l pode assumir valores de 0 a $2^q - 1$.

Por exemplo, se desejamos conhecer os elementos do bloco $\mathbf{B}_{1,0}^{(1,0)}$ da árvore de segmentação na figura 4.5, primeiro devemos avaliar o valor de $\frac{N}{2^p}$ e $\frac{M}{2^q}$, para determinar o número de linha e colunas que o bloco possui e então substituir os valores na equação 4.11. Logo, temos

$$\frac{N}{2^p} = \frac{2}{2^1} = 1, \tag{4.12}$$

$$\frac{M}{2^q} = \frac{2}{2^0} = 2, \tag{4.13}$$

$$\mathbf{B}_{1,0}^{(1,0)} = \begin{bmatrix} b\left(\frac{1\times2}{2^1}+0, \frac{0\times2}{2^0}+0\right) & b\left(\frac{1\times2}{2^1}+0, \frac{0\times2}{2^0}+1\right) \end{bmatrix} = \begin{bmatrix} b(1,0) & b(1,1) \end{bmatrix}. \tag{4.14}$$

4.3.2 Otimização da Árvore de Segmentação 2D

Como no MMP, para decidir se um dado índice $I_{k,l}^{(p,q)}$ vai ser codificado, o custo Lagrangeano do bloco identificado pelo índice deve ser comparado com o custo Lagrangeano de sua segmentações. Se o bloco possui o número de linhas ou de colunas igual a 1, o processo de tomada de decisão é igual a do MMP, ou seja em apenas uma dimensão. Mas se o bloco possui número de linhas e de colunas maior que 1, ele pode ser segmentado tanto na vertical como na horizontal, e, neste caso, essas duas segmentações devem ser consideradas para a tomada de decisão.

Após determinar $J_{k,l}^{(p,q)}$, ele é comparado à soma do custo para codificar o *flag* de segmentação com os custos para codificar os índices que identificam os elementos do dicionário que representam os dois blocos resultantes dessa segmentação, conforme equação 4.15.

$$J_{V_{k,l}}^{(p,q)'} = J_{2k,l}^{(p+1,q)} + J_{2k+1,l}^{(p+1,q)} + J_{F_V}^{(p,q)} \tag{4.15}$$

O mesmo acontece para a segmentação na horizontal, conforme equação 4.16.

$$J_{H_{k,l}}^{(p,q)'} = J_{k,2l}^{(p,q+1)} + J_{k,2l+1}^{(p,q+1)} + J_{F_H}^{(p,q)} \tag{4.16}$$

O processo para determinar se o que será enviado é o *flag* indicando que o bloco não foi segmentado acompanhado do índice do elemento dicionário que o representa ou os resultantes das segmentações é o seguinte:

a) O algoritmo compara os custos para segmentar na vertical e horizontal. Logo se $J_{H_{k,l}}^{(p,q)'} < J_{V_{k,l}}^{(p,q)}$ o algoritmo considera que se for necessário segmentar, a segmentação será realizada na horizontal. Caso contrário, na vertical.

b) O custo para não segmentar é comparado com o custo para segmentar escolhido no passo 'a'. Logo se $J_{H_{k,l}}^{(p,q)'} < J_{F_M}^{(p,q)} + J_{k,l}^{(p,q)}$ ou $J_{V_{k,l}}^{(p,q)'} < J_{F_M}^{(p,q)} + J_{k,l}^{(p,q)}$, dependendo da escolha realizada no passo 'a', o algoritmo considera que segmentar é a melhor opção. Caso contrário, não segmenta.

$J_{F_M}^{(p,q)}$ é o custo para codificar o *flag* que indica que o bloco não será segmentado (matching).

O fluxograma para codificar cada bloco é o mesmo apresentado na figura 4.4 para o MMP.

4.4 MMP-2D SIDE-MATCH (MMP-2D-SM)

O MMP-2D seleciona o elemento do dicionário que representa o bloco lido da imagem ou qualquer um dos seus segmentos sem considerar sua vizinhança. Isto pode causar descontinuidades na imagem recuperada após o processo de compressão, como pode ser observado na imagem recuperada da Lena na figura 4.8. Um modo de resolver esse problema foi apresentado em [5]. Nesse algoritmo, o processo de escolha do elemento do dicionário que vai representar um determinado bloco ou um dos seus segmentos considera a sua vizinhança. Como uma consequência do modo que algoritmo trabalha, ele acaba realizando um condiconamento estatísco, que melhora o desempenho do codificador artmético, tornando taxa-distorção resultante melhor que a do MMP-2D. Em contrapartida, o tempo de processamento aumenta consideravelmente. Esse algoritmo foi denominado MMP-2D-SM (Side Match).

No MMP-2D-SM, para cada bloco de entrada, bem como cada segmento do bloco de entrada, que é processado durante a otimização da árvore de segmentação, um novo dicionário, chamado de "Dicionário de Estado", é composto com alguns elementos do dicionário. Para cada Dicionário de Estado que é composto, um modelo probabilístico associado deve ser composto também, já que o elemento que representa o bloco e seus segmentos para a otimização da árvore de segmentação é obtido do Dicionário de Estado. O modo como o modelo probabilístico do Dicionário de Estado é composto, confere uma maior probabilidade aos elementos que o compõem, que refletem numa melhora do desempenho da codificação, já que é esse modelo usado pelo codificador aritmético.

Como aconteceu na comparação entre o MMP-2D e o MMP, a principal diferença do MMP-2D-SM e o MMP-2D é a forma que a árvore de segmentação é otimizada. E são somente essas diferenças que serão apresentada nesta seção, já que outras atividades como as atualizações do dicionário e modelo estatísco acontecem como no MMP-2D.

A composição do Dicionário de Estado é realizada em duas partes:
1) Determinação da cardinalidade (número de blocos) do Dicionário de Estado;
2) A seleção dos elementos do dicionário que vão compor o Dicionário de Estado.

4.4.1 Determinação da Cardinalidade do Dicionário de Estado

A cardinalidade do Dicionário de Estado é uma função da dispersão do valor dos *pixels* do bloco. Nosso sentimento diz que um bloco onde os valores dos *pixels* são poucos dispersos devem encontrar menos blocos no dicionário que atendam algum critério de seleção e blocos em que o valores dos *pixels* estão muito dispersos devem encontrar mais blocos do dicionário que atendam a esse mesmo critério de seleção. Baseado nesta premissa, precisamos ter um indicador que traduza de alguma forma se esses valores estão mais ou menos dispersos em um bloco que está sendo processado. No caso do MMP-2D-SM o indicador usado, que foi escolhido heuristicamente, é chamado de "Atividade da Vizinhança". A palavra "vizinhança" no nome do indicador é devido ao fato desse indicador ser obtido dos blocos vizinhos causal (considerando a varredura da imagem da esquerda para direita e de cima para baixo, os blocos são o superior e lateral esquerdo) ao que está sendo processado, já que não pode ser obtida do bloco que está sendo processado porque o descompressor teria que conhecer os valores dos *pixels* desse bloco para realizar a operação inversa, antes de realmente conhecê-los. Como já foi dito, uma imagem natural possui regiões em que os *pixels* são correlatos. Devido a este fato, usar a vizinhança para obter esse indicador, na média, é uma boa aproximação.

A dimensão do Dicionário de Estado é obtida pela equação 4.17.

$$N_s = \left\lfloor N_{s_{max}} \frac{Act_u + Act_l}{2Act_{max}} \right\rfloor \tag{4.17}$$

Na equação 4.17 Act_u e Act_l são as atividades nas duas vizinhanças superior e lateral esquerda, respectivamente, do bloco que está sendo processado. Act_{max} é a maior "Atividade" obtida dos blocos que são lidos da imagem. Act_{max} deve ser computada e transmitida antes da compressão da imagem iniciar, já que o descompressor necessita dessa informação para o operação de descompressão. E $N_{s_{max}}$ é um parâmetro do algoritmo que determina a maior dimensão que o Dicionário de Estado pode ter, já que a fração na equação 4.17 pode assumir valores entre 0 e 1 inclusive. O símbolo $\lfloor . \rfloor$ significa que somente a parte inteira do resultado é considerada. Nas simulações executadas o valor atribuídos a $N_{s_{max}}$ foi igual a 4096. O algoritmo também limita a dimensão mínima que o Dicionário de Estado pode assumir, que nas nossas simulações foi atribuido o valor igual a 16.

Act_u e Act_l é o maior valor, entre dois valores obtidos, conforme apresentados nas equações 4.18 e 4.19 respectivamente.

$$Act_u = max\left[AU_n, AU_m\right] \tag{4.18}$$

$$Act_l = max\left[AL_n, AL_m\right] \tag{4.19}$$

AU_n e AU_m, bem como AL_n e AL_m, é o maior valor da soma do valor absoluto da diferença dos *pixels* adjacentes em cada linha e cada coluna dos blocos vizinhos superior e lateral esquerdo ao bloco que está sendo processado. As equações 4.20, 4.21, 4.22 e 4.23 traduzem matematicamente a forma como AU_n, AU_m, AL_n e AL_m são obtidos. $p_u(n,m)$ e $p_l(n,m)$ são os *pixels* dos blocos vizinhos superior e lateral esquerdo respectivamente. N e M são os números de linhas e colunas respectivamente dos blocos vizinhos.

$$Au_n = max\left[\sum_{m=0}^{M-2}\left|p_u(0,m+1) - p_u(0,m)\right|, \ \ldots \ , \sum_{m=0}^{M-2}\left|p_u(N-1,m+1) - p_u(N-1,m)\right|\right] \tag{4.20}$$

$$Au_m = max\left[\sum_{n=0}^{N-2}\left|p_l(n+1,0) - p_l(n,0)\right|, \ \ldots \ , \sum_{n=0}^{N-2}\left|p_l(n+1,M-1) - p_l(n,M-1)\right|\right] \tag{4.21}$$

$$Al_n = max\left[\sum_{m=0}^{M-2}\left|p_l(0,m+1) - p_l(0,m)\right|, \ \ldots \ , \sum_{m=0}^{M-2}\left|p_l(N-1,m+1) - p_l(N-1,m)\right|\right] \tag{4.22}$$

$$Al_m = max\left[\sum_{n=0}^{N-2}\left|p_l(n+1,0) - p_l(n,0)\right|, \ \ldots \ , \sum_{n=0}^{N-2}\left|p_l(n+1,M-1) - p_l(n,M-1)\right|\right] \tag{4.23}$$

A figura 4.6 apresenta um exemplo de como a dimensão do Dicionário de Estado, N_s, é calculada para um bloco 4×4. $A_{u_{m_0}}$, por exemplo, é obtido somano o valor absoluto do resultado da subtração dos *pixels* da primeira coluna a esquerda ($|189-50|+|50-51|+|51-49| = 142$). Para determinar $A_{u_{n_0}}$, os *pixels* usados são os da linha superior do bloco ($|200-50|+|50-48|+|48-49| = 153$). Os outros A_{u_m} e A_{u_n}, bem como A_{l_m} e A_{l_n} são obtidos de forma semelhante.

Após determinar o dimensão, o algoritmo seleciona os elementos do dicionário que vão compor o Dicionário de Estado.

$Au_n = max\,[142; 152; 156; 26] = 156$
$Au_m = max\,[153; 147; 149; 10] = 153$
$Al_n = max\,[10; 10; 7; 13] = 13$
$Al_m = max\,[23; 19; 36; 40] = 40$

$Act_u = max\,[156, 153] = 153$
$Act_l = max\,[13, 40] = 40$

$N_s = \left\lfloor 4096 \frac{153+40}{2\times 200} \right\rfloor = 1976$

$Au_{m_3} = 29$
$Au_{m_2} = 156$
$Au_{m_1} = 152$
$Au_{m_0} = 142$

$Au_{n_0} = 153$	49	48	50	200
$Au_{n_1} = 147$	51	52	196	194
$Au_{n_2} = 149$	50	195	197	199
$Au_{n_3} = 10$	189	190	188	181

Bloco vizinho superior

$Al_{m_3} = 40$
$Al_{m_2} = 36$
$Al_{m_1} = 19$
$Al_{m_0} = 23$

$Al_{n_0} = 10$	116	117	111	108
$Al_{n_1} = 10$	110	118	118	120
$Al_{n_2} = 7$	112	115	112	111
$Al_{n_3} = 13$	127	130	135	130

Bloco vizinho lateral

Bloco sendo processado

Figura 4.6: Exemplo de cálculo da dimensão do Dicionário de Estado para um bloco de dimensões $4X4$, considerando $N_{s_{max}} = 4096$ e $Act_{max} = 200$.

4.4.2 Composição do Dicionário de Estado

Os elementos que compõem o Dicionário de Estado são escolhidos segundo um outro parâmetro, selecionado heuristicamente, que é chamado de "Rugosidade". A Rugosidade foi concebida para que o algoritmo preserve a continuidade entre os blocos. A Rugosidade é uma aproximação por diferenças finitas das derivadas de primeira e segunda ordens avaliadas na fronteira entre dois blocos e mede o grau de descontinuidade que há entre esses blocos. Para selecionar os elementos do dicionário que vão compor o Dicionário de Estado, o algoritmo calcula a rugosidade para todos os elementos do dicionário que pertencem a escala do bloco que está sendo processado. Os N_s elementos que apresentam o menor valor para a Rugosidade (descontinuidade) são selecionados para compor o Dicionário de Estado. Como ocorre com o cálculo da Atividade da Vizinhança, a Rugosidade é calculada para os blocos vizinhos causais superior e lateral esquerdo.

A Rugosidade é definida pela equação 4.24.

$$R = R_l + R_u, \tag{4.24}$$

onde R_l e R_u são as parcelas da Rugosidade referentes aos blocos vizinhos lateral esquerdo e superior respectivamente e são definidas na equação 4.25 e 4.26.

$$R_l = \sum_n \left\lfloor \left| \frac{p_l(n, M-1) - p_l(n, M) + p(n, 0) - p(n, 1)}{2} + p(n, 0) - p_l(n, M) \right| \right\rfloor \tag{4.25}$$

$$R_u = \sum_m \left\lfloor \left| \frac{p_u(N-1, m) - p_u(N, m) + p(0, m) - p(1, m)}{2} + p(0, m) - p_u(N, m) \right| \right\rfloor \tag{4.26}$$

Nas equações 4.25 e 4.26, $p(n, m)$, $p_l(n, m)$ e $p_u(n, m)$ são os *pixels* do bloco do dicionário, bloco vizinho lateral esquerdo e bloco vizinho superior respectivamente, e o símbolo $\lfloor . \rfloor$ significa que somente a parte inteira do resultado é considerada. N e M são o número de linhas e colunas respectivamente do bloco que está sendo analisado.

Para cada nó da árvore de segmentação um Dicionário de Estado distinto é composto. Após a composição do Dicionário de Estado, o algoritmo determina qual elemento desse dicionário possui o menor custo Lagrangeano, memorizando tanto o custo como o índice do dicionário que identifica o elemento, como acontece no MMP-2D.

A grande diferença dos elemento do dicionário do MMP-2D e do Dicionário de Estado do MMP-2D-SM é que o último contém elementos que minimizam a descontinuidade entre os blocos, fazendo com que o efeito de blocagem seja menor no MMP-2D-SM do que no MMP-2D.

A figura 4.7 apresenta um exemplo de cálculo da Rugosidade para um elemento do dicionário

candidato a compor o Dicionário de Estado.

$$R_l = \left\lfloor \left| \frac{3}{2} - 8 \right| \right\rfloor + \left\lfloor \left| \frac{-2}{2} - 20 \right| \right\rfloor + \left\lfloor \left| \frac{1}{2} - 11 \right| \right\rfloor + \left\lfloor \left| \frac{5}{2} - 30 \right| \right\rfloor = 67$$

$$R_u = \left\lfloor \left| \frac{-139}{2} - 89 \right| \right\rfloor + \left\lfloor \left| \frac{5}{2} - 90 \right| \right\rfloor + \left\lfloor \left| \frac{9}{2} - 88 \right| \right\rfloor + \left\lfloor \left| \frac{18}{2} - 81 \right| \right\rfloor = 400$$

$$R = R_l + R_u = 67 + 400 = 467$$

R_l é a parcela da Rugosidade referente ao bloco vizinho lateral esquerdo.

R_u é a parcela da Rugosidade referente ao bloco vizinho superior.

49	48	50	200
51	52	196	194
50	195	197	199
189	190	188	181

Bloco vizinho superior

116	117	111	108
110	118	118	120
112	115	112	111
127	130	135	130

Bloco vizinho lateral

100	100	100	100
100	100	100	100
100	100	100	100
100	100	100	100

Bloco do dicionário sendo analisado

Figura 4.7: Exemplo de cálculo da Rugosidade para um bloco do dicionário de dimensões $4X4$ cujos elementos do bloco do dicionário possuem valor igual a 100.

4.4.3 Composição do Modelo Estatístico do Dicionário de Estado

Para cada elemento selecionado do dicionário para compor o Dicionário de Estado, a respectiva frequência relativa do índice que identifica o elemento é tomada do modelo probabilístico para compor o Modelo Probabilístico do Dicionário de Estado. Quando esse processo termina, a soma das frequências relativas do Modelo Probabilístico do Dicionário de Estado é calculada para o algoritmo determinar a probabilidade dos índices. O condicionamento estatístico ocorre porque essa soma acaba sendo menor que a do modelo probabilístico original, já que o Dicionário de Estado tem apenas uma fração dos elementos do dicionário. Por exemplo, vamos supor que eu tenho um dicionário com 100 elementos e a frequência relativa dos índices que identificam os elementos seja igual a 1. Neste caso, a probabilidade de cada índice é $0,01$. Vamos supor que foram selecionados 10 elementos dos 100 para compor o Dicionário de Estado. Ao tomar a frequência relativa desses índices e somá-los, o valor obtido será igual a 10 e, consequentemente, a probabilidade dos índices do Dicionário de Estado serão iguais a $0,1$, de acordo com a equação 3.7. O Dicionário de Estado, bem como o seu modelo probabilístico são usados tanto para determinar o índice do dicionário que vai representar o bloco ou os seus segmentos e para realizar a codificação aritmética. Como a probabilidade dos índices que serão codificados aumenta, o codificador aritmético acaba codificando usando menos *bits*, ocorrendo desta forma o Condicionamento Estatístico.

4.4.4 Uso da imagem recuperada

Como foi dito, para recuperar a imagem comprimida, o descompressor tem que ter acesso as mesmas informações que o compressor usou no processo de compressão. No MMP-2D, o compressor, bem como o descompressor, realizam a atualização do modelo probabilístico e dicionário utilizando os índices codificados e os elementos do dicionário que esses índices identificam respectivamente. Isto é suficiente para que a recuperação da imagem seja realizada com sucesso. No MMP-2D-SM, além do modelo probabilístico e do dicionário, o descompressor necessita manter sincronizado com o compressor a informação dos blocos vizinhos usados para compor o Dicionário de Estado. Mas o descompressor não pode ter acesso as informações dos blocos que pertencem a imagem original, já que o algoritmo é com perdas e os *pixels* da imagem recuperada é uma cópia aproximada, mas não fiel da imagem original. Logo, se o compressor usar a imagem original para compor os Dicionários de Estados durante o processo de compressão da imagem e o descompressor usar a parte da imagem recuperada durante o processo de descompressão, quando a imagem estiver totalmente recuperada pode conter mais distorção que o tolerado, já que o erro se propagaria em cascata ao longo do processo de descompressão. Para resolver este problema durante o processo de compressão, o compressor vai montando uma imagem com os blocos que são identificados pelos índices que são enviados para o descompressor. Essa imagem é chamada de "Imagem Recuperada" e é usada no processo de composição do Dicionário de Estado pelo compressor. Deste modo, o descompressor tem acesso as mesmas informações usadas pelo compressor na fase de recuperação da imagem.

4.4.5 Codificação

Como no MMP-2D, após a otimização da árvore de segmentação, os índices e *flags* de segmentação são enviados para o codificador aritmético. A diferença do processo é que o codificador usa o Modelo Estatístico do Dicionário de Estado para codificação desses índíces. Como o dicionário de Estado, bem como o seu modelo estatístico é determinado e mantido somente durante o período necessário para calcular o custo Lagrangeano e selecionar o melhor candidato do Dicionário de Estado que vai representar o bloco, quando os índices são codificados é necessário determinar novamente o Dicionário de Estado, bem como o seu modelo estatístico, para que o codificador aritmético possa realizar a codificação.

4.4.6 Tempo de codificação

A complexidade computacional do MMP-2D-SM se concentra na otimização da árvore de segmentação. Aproximadamente, 98,5% do tempo de execução do algoritmo é gasto nessa etapa devido, principalmente, à composição dos Dicionários de Estado. 1,3% é gasto no processo de codificação. E 0,2% na execução de outras etapas.

Apesar de durante a codificação ser necessário a composição de Dicionários de Estado, esses só são compostos para os nós da árvore de segmentação cujos índices são codificados. Por exemplo, se a dimensão do bloco lido da imagem for 16×16, no máximo, seriam necessários compor 256 Dicionários de Estado, que quando comparado aos 31.381 [6] que são necessários compor durante a otimização da árvore de segmentação, mostra o porquê, apesar de ser necessário compor o Dicionário de Estado, na codificação o tempo gasto é bem menor.

4.4.7 Resultados das Simulações

A figura 4.8 mostra duas imagens da Lena recuperadas após serem codificada pelo MMP-2D e MMP-2D-SM. Podemos observar que o efeito de blocagem é atenuado na imagem recuperada quando comprimida usando o MMP-2D-SM.

As figuras de 4.9 a 4.13 apresentam os gráficos que comparam odesempenho do MMP-2D-SM com o MMP-2D. Podemos observar que o ganho do MMP2D-SM sobre o MMP2D varia bastante dependendo do conteúdo estatístico da imagem, mas é, em geral, superior ou equivalente (perde por menos de poucos décimos de dB quando comparado ao MMP2D para a imagem da Barbara em taxas inferiores à 0.8 bpp).

MMP-2D

MMP-2D-SM

Figura 4.8: Imagens da Lena recuperadas após compressão usando o MMP-2D e MMP-2D-SM.

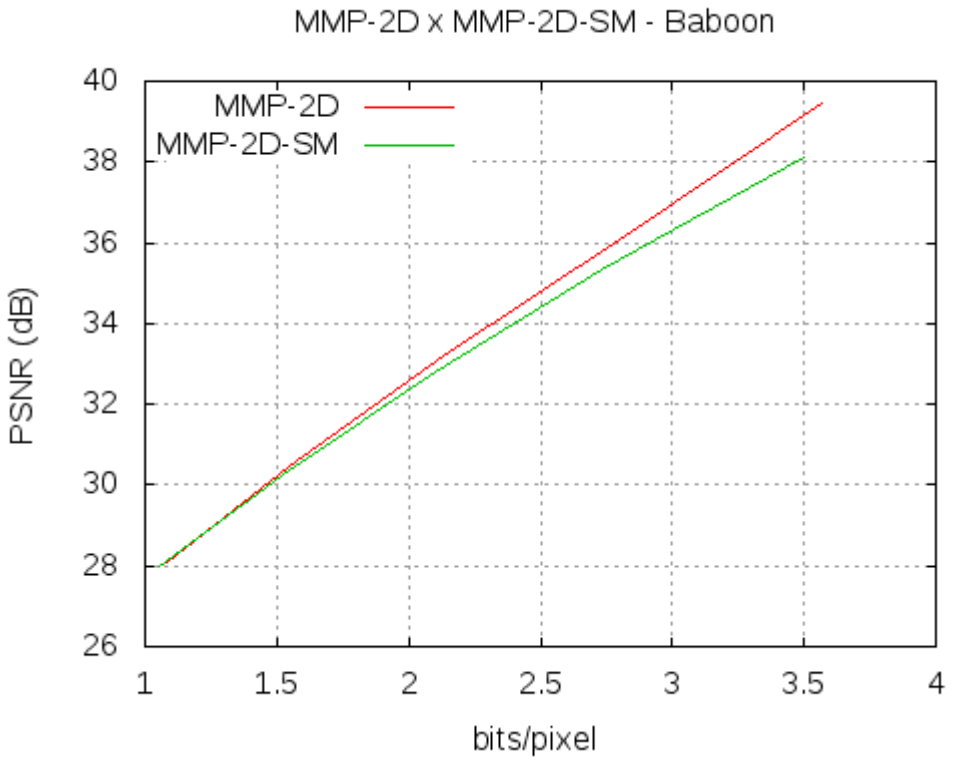


Figura 4.9: Gráfico comparativo entre o MMP-2D e MMP-2D-SM usando a imagem Baboon.

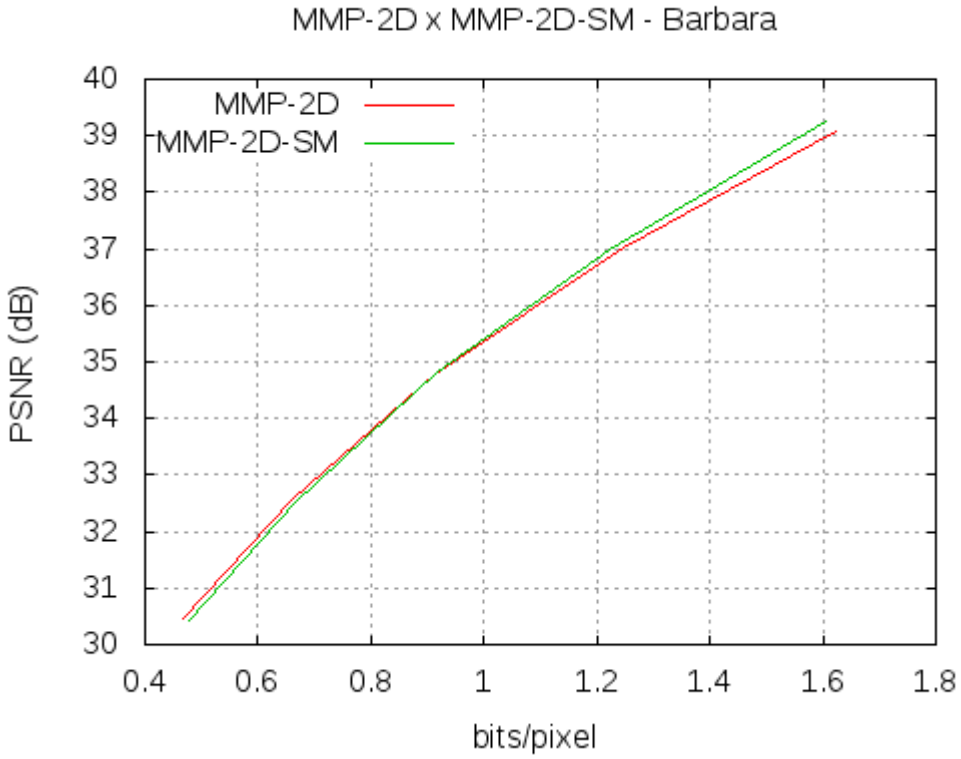


Figura 4.10: Gráfico comparativo entre o MMP-2D e MMP-2D-SM usando a imagem Barbara.

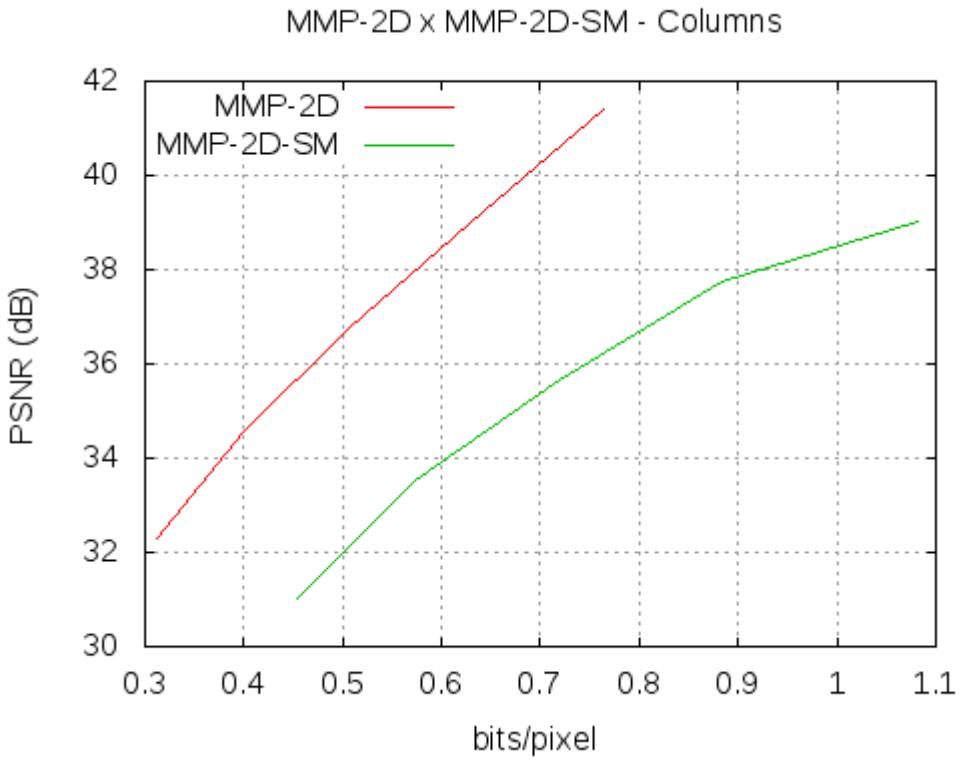


Figura 4.11: Gráfico comparativo entre o MMP-2D e MMP-2D-SM usando a imagem Columns.

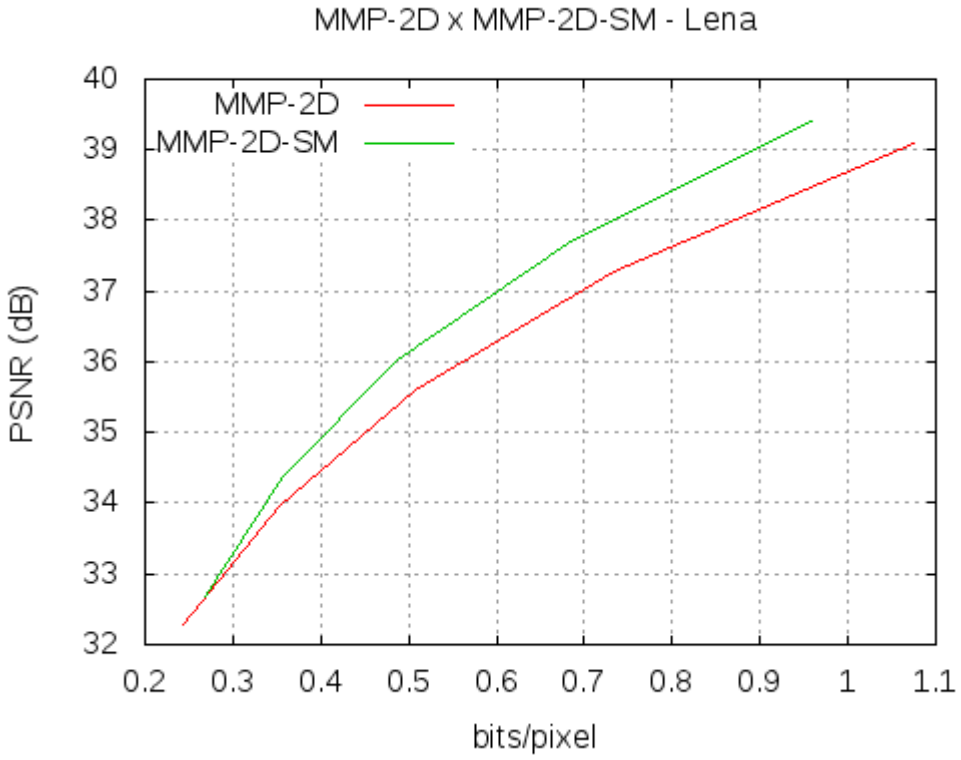


Figura 4.12: Gráfico comparativo entre o MMP-2D e MMP-2D-SM usando a imagem Lena.

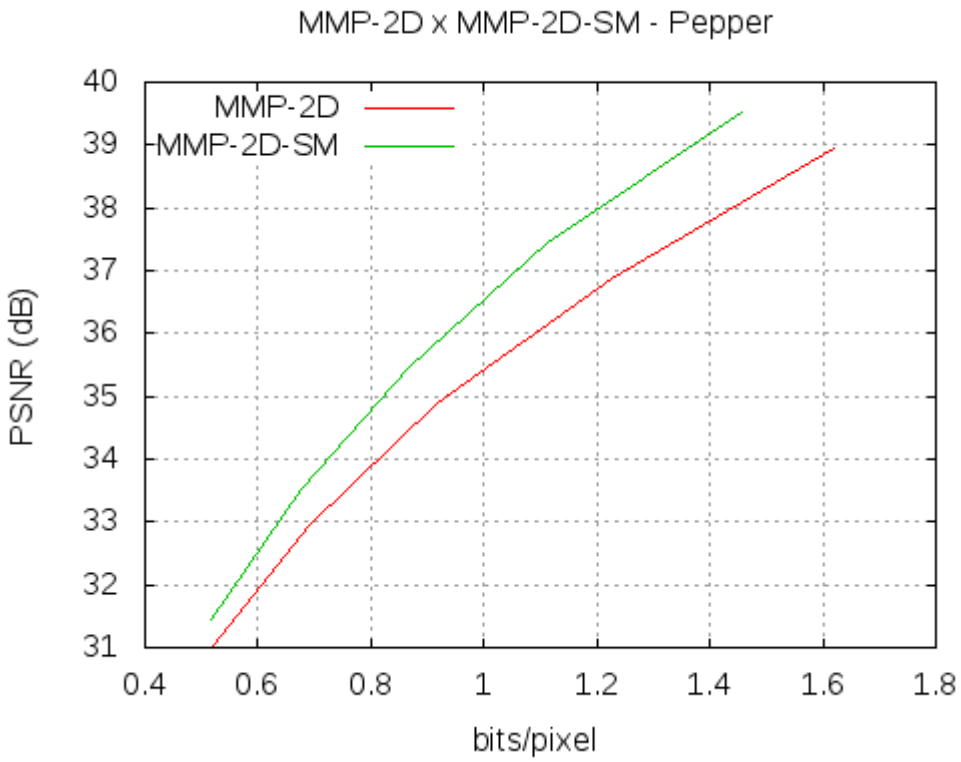


Figura 4.13: Gráfico comparativo entre o MMP-2D e MMP-2D-SM usando a imagem Pepper.

Capítulo 5

Uso da GPU no MMP-2D e MMP-2D-SM

5.1 Introdução

Desde o início deste século, a capacidade de paralelização dos processadores e a capacidade de armazenamento da memória interna das Unidades de Processamento Gráfico (GPU) vêm aumentando com o objetivo de criar imagens mais realistas para atender, principalmente, ao mercado de aplicativos gráficos e de jogos. Essa melhora no hardware das GPU começou a ser explorada para a realização de processamento paralelo. Em Novembro de 2006, observando essa tendência, a NVIDIA lançou o *Compute Unified Device Architecture* (CUDA), uma plataforma de computação paralela de proposta geral e um modelo de programação que alavancou a computação paralela nas GPU's, resolvendo muitos problemas computacionais complexos de um modo mais eficiente. A interface C para o CUDA estende a capacidade de programação, permitindo o programador definir funções, denominadas *kernels*, que, quando chamadas são executadas em paralelo por vários *threads*, ao contrário do que acontece em uma função regular em C.

5.2 Organização do CUDA

O CUDA organiza a execução de cada *thread* em blocos (*blocks*), que são organizados em grades (*grids*). Para identificar cada cópia do *kernel* que roda simultaneamente, CUDA provê três variáveis interna para esse controle que são: $GridDim$, $BlockID$, $BlockDim$ e $ThreadID$. Cada variável possui três componentes que são identificadas pelas extensão $.x$, $.y$ e $.z$.

A figura 5.1 apresenta de uma forma amigável como as grades, blocos e *thread* são organizados, destacando o significado de algumas variáveis internas. O paralelepípedo maior é uma grade que está dividido em paralelepípedo menores que são os blocos. O cubo menor a esquerda é o detalhamento de um bloco que está dividido em cubos menores que são os *thread*. Tanto os blocos como os *thread* são

identificados como se fossem um sistema tridimensional. Apesar das três dimensões disponíveis, os blocos e *thread* podem ser definidos somente em uma ou duas dimensões.

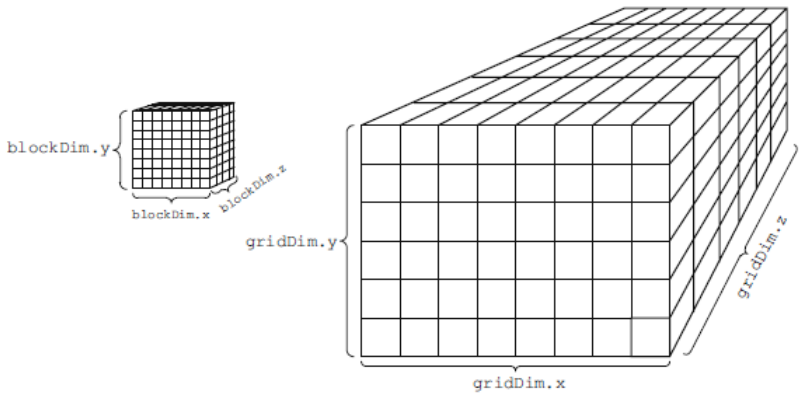


Figura 5.1: Organização de grade, blocos e *threads* no CUDA [12]

No momento que um *kernel* é chamado, é necessário definir como será a estrutura das grades e blocos. Normalmente, essa definição deve ser feita de forma que facilite a obtenção dos dados retornado pela função e maximize o desempenho da GPU.

Os processadores da GPU não possuem acesso direto a memória do computador onde estão instalados, bem como o processador do computador também não tem acesso direto a memória da GPU. Para transferir dados da memória do computador para a memória da GPU, o CUDA disponibiliza funções para realizar essa tarefa. A transferência de dados entre a memória do computador e da GPU, e da GPU para o computador despende algum tempo que deve ser considerado quando se deseja usar a GPU, já que, dependendo da aplicação, pode resultar numa redução inexpressiva de tempo, se não for bem avaliada.

A paralelização também não pode ser realizada, se uma função que é executada várias vezes para realizar um cálculo depende de resultados anteriores dela própria.

Nós veremos nas duas próximas seções que o cálculo do custo Lagrangeano no MMP-2D pode ser paralelizado com a GPU, mas a forma que a composição do Dicionário de Estado é realizada no MMP-2D-SM não permite que possamos realizar a paralelização e com isto reduzir o tempo de compressão de uma imagem usando esse algoritmo.

5.3 Uso da GPU no MMP-2D

O MMP possui uma complexidade computacional muito grande. Numa máquina com processador AMD Athlon II X2 250 com 2 núcleos, relógio de 3,0 GHz e memória de 4 Giga Bytes, o tempo para codificar a imagem da Lena é de 443 segundos. 98,4% desse tempo é gasto na execução do cálculo do custo Lagrangeano durante a otimização da árvore de segmentação [6]. Como foi dito, o custo Lagrangeano é calculado para todos os elementos do dicionário da respectiva escala cujo bloco de um determinado nó pertença. Isto significa que para um dicionário inicial cujo número de elementos de cada escala é igual 256, 246.016 cálculos do custo Lagrangeano são realizados quando um bloco de 16×16 é lido da imagem, considerando que nós visitados não são revisitados na árvore de segmentação. Como o número de elementos de cada escala do dicionário tende a aumentar a cada novo bloco da imagem que é processado, a quantidade de cálculos do custo Lagrangeano aumenta também.

Para calcular o custo Lagrangeano para cada elemento de uma determinada escala do dicionário, os únicos dados necessários são os valores dos *pixels* do bloco do nó da árvore de segmentação, os valores do elemento do dicionário e a probabilidade do seu índice, e o valor de λ. Como no início da otimização da árvore de segmentação todos esses dados já estão disponíveis, a paralelização usando a GPU é possível.

Uma versão do MMP-2D que usa a GPU, chamada MMP-2D-GPU, foi desenvolvida e é apresentada em [6]. A figura 5.2 apresenta uma fluxograma do MMP-2D-GPU destacando o que é executado na GPU e no computador onde a GPU está instalada.

Como o cálculo do custo envolve os dados do dicionário e do modelo probabilístico de seu elementos, uma cópia do dicionário tem que ser mantida na memória da GPU, bem como a estatística de ocorrência dos índices que identifica os seus elementos. Por esta razão que no fluxograma da figura 5.2 há algumas funções, além da realização do cálculo custo Lagrangeano propriamente dito, realizados na GPU. Essas funções tem o objetivo de atualizar tanto a cópia do dicionário, bem como do respectivo modelo probabilístico residente na GPU. Outra função que é realizada na GPU é a busca pelo elemento do dicionário em cada escala que possui o menor custo Lagrangeano.

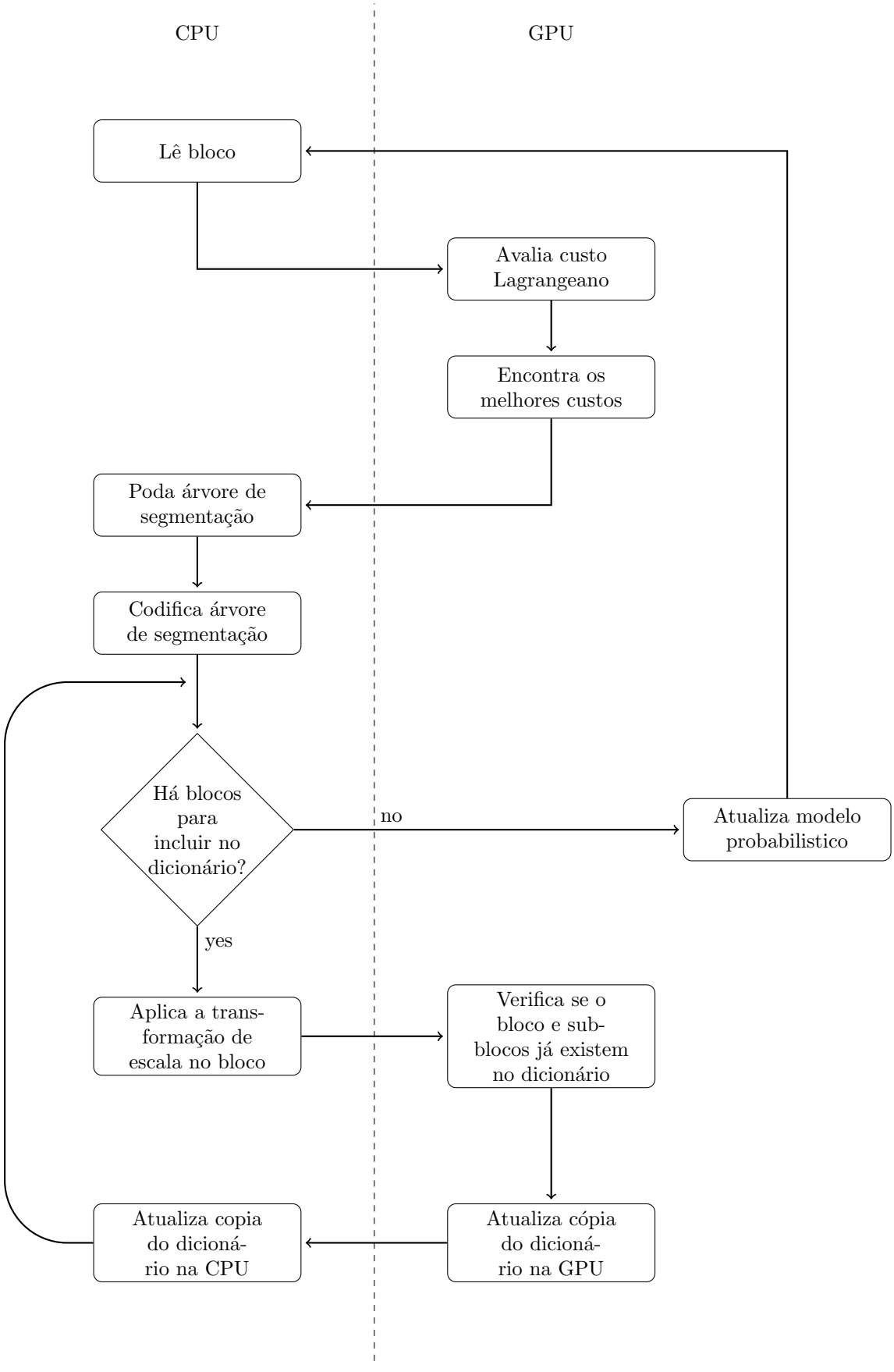


Figura 5.2: Fluxo do MMP-GPU, destacando o que roda na CPU e na GPU.

A cópia do dicionário na GPU é organizada como uma matriz de 16384 colunas por 6400 linhas, sendo que em cada coluna contém as 25 escalas de mesmo índice do dicionário. Cada escala possui 256 elementos, independentemente de sua dimensão. A razão para manter o mesmo número de elementos para cada escala do dicionário, apesar de consumir mais memória, é a melhora no desempenho [6].

Para realizar o cálculo do custo Lagrangeano, o MMP-2D-GPU conta com cinco *kernels*. O primeiro *kernel* é responsável pela a avaliação do custo Lagrangeano dos 961 blocos (supondo que o bloco lido da imagem possui dimensão 16×16) que é executado em uma grade de 2048 x 1 blocos, cada bloco endereçando 8 elementos do dicionário. Os *thread* de cada bloco avalia o custo Lagrangeano.

O segundo *kernel* encontra para cada um dos 961 blocos o elemento do dicionário que possui o menor custo Lagrangeano. Esse *kernel* é executado em uma grade de 31 x 31 blocos, e cada bloco é organizado em 128 x 1 *threads*.

O terceiro *kernel* faz parte do processo de atualização do dicionário. Ele verifica se um determinado bloco já faz parte do dicionário. Após sua conclusão ele retorna 25 *flags* para indicar se os candidatos podem ou não serem inclusos. A dimensão dessa grade é a mesma do primeiro *kernel*.

O quarto *kernel* é usado para atualizar o dicionário da GPU. Ele é executado em uma grade de 5 x 5 blocos com cada bloco rodando 16 x 16 *threads*. Nesta grade, cada bloco é associado a um dos 25 elementos a serem incluídos no dicionário, um para cada escala, e cada *thread* é associado a cada *pixel*.

O quinto *kernel* é usado para atualizar o vetor de frequências relativas e de frequências acumuladas que é mantido na GPU para o cálculo do custo Lagrangeano, que é realizado pelo primeiro *kernel*.

A tabela 5.1 apresenta os tempos de execução para codificar as imagens apresentadas no anexo A, excetuando a imagem Baboon, com o MMP-2D e o MMP-2D-GPU. O algoritmo rodou em um computador com processador AMD Athlon II X2 250 com 2 núcleos, relógio de 3,0 GHz, memória de 4 Giga Bytes, e GPU GeForce GT 730 da NVidia.

Imagem	Tempo (seg) - MMP-2D	Tempo (seg) - MMP-2D-GPU	Redução (%)
Barbara	677	98	85,52
Columns	735	80	89,12
Lena	443	45	89,84
Pepper	166	11	93,37

Tabela 5.1: Comparação do tempo de execução entre o MMP-2D e o MMP-2D-GPU

A imagem Baboon não entrou na comparação porque durante o processamento o tamanho dicionário cresceu além do que a memória da GPU usada pôde suportar.

5.3.1 GPU no MMP-2D-SM

Como foi dito no capítulo anterior, 98,5% do tempo usado pelo o algoritmo MMP-2D-SM é despendido com a otimização da árvore de segmentação. Grande parte desse tempo é devido ao número de dicionários de Estado que necessitam ser compostos para, então, realizar o cálculo do custo Lagrangeano. Logo a implementação de uma versão do MMP-2D-SM que usasse o recurso de paralelização da GPU, poderia reduzir o tempo de processamento do algoritmo.

Para selecionar o melhor bloco do Dicionário de Estado que representará um dos segmentos do bloco lido da imagem, o MMP-2D-SM tem que compor o Dicionário de Estado, e então calcular o custo Lagrangeano para todos os elementos desse dicionário. Mas como vimos, a composição do dicionário de Estado usa os *pixels* dos blocos vizinhos lateral esquerdo e superior de uma imagem recuperada, ou seja, esses blocos têm que ser obtidos pelo mesmo processo que bloco que está sendo processado. Isso cria uma dependência no cálculo do Dicionário de Estado que o impede de ser paralelizado na GPU. Para entender melhor, a figura 5.3 mostra uma sequencia de três blocos de dimensão 4×4: o que está sendo processado; o vizinho lateral esquerdo; e o vizinho superior. Os quadrados com um 'L' escrito no seu interior representam os *pixels* usados para o cálculo da Rugosidade do bloco vizinho lateral esquerdo. Os quadrados com um 'U' escrito no seu interior representam os *pixels* do bloco usado no cálculo da Rugosidade do vizinho superior. E os quadrados com um 'B' escrito no seu interior representam os *pixels* do bloco cuja Rugosidade está sendo calculada. Na escala $(0, 0)$ os *pixels* do blocos vizinhos da imagem recuperada já estão disponíveis porque foram determinados durante o processamento de blocos lido da imagem anteriormente. O mesmo acontece com a primeira partição da escala $(1, 0)$. Entretanto, quando o algoritmo vai computar o Dicionário de Estado para a segunda partição da escala $(1, 0)$, o seu vizinho superior é a primeira partição da mesma escala, que foi computado um pouco antes. Na figura 5.3 é mostrado a dependência de apenas uma partição de uma escala, mas essa situação ocorre em outras partições de outras escalas durante a otimização da árvore de segmentação.

Para resolver o problema da inviabilidade do uso da GPU pelo MMP-2D-SM, uma modificação no algoritmo foi proposta: usar a imagem original para a determinação do Dicionário de Estado no lugar da imagem recuperada. Como a imagem já está disponível desde o inicio, as vizinhanças usadas para o cálculo da Rugosidade estariam disponíveis para qualquer partição de qualquer escala de qualquer bloco lido da imagem. Com isso a paralelização na GPU seria possível.

Uma versão do MMP-2D-SM usando a imagem original, chamada MMP-2D-SM-A (Aproximado), foi desenvolvida para rodar sem usar o recurso da GPU. O objetivo foi verificar como seria o desempenho com a mudança proposta, já que a imagem recuperada, apesar de aproximada, difere da imagem original, pode causar distorções na imagem recuperada pelo descompressor, conforme dito no capítulo anterior. Os resultados mostraram que o desempenho do MMP-2D-SM-A ficou muito abaixo do MMP-2D-SM, chegando a apresentar uma queda de 10 dB no PSNR na imagem Columns para uma taxa de 0,7 *pixels* por *bit*. A razão para essa degradação é a forma como a Rugosidade é calculada. A Rugosidade é

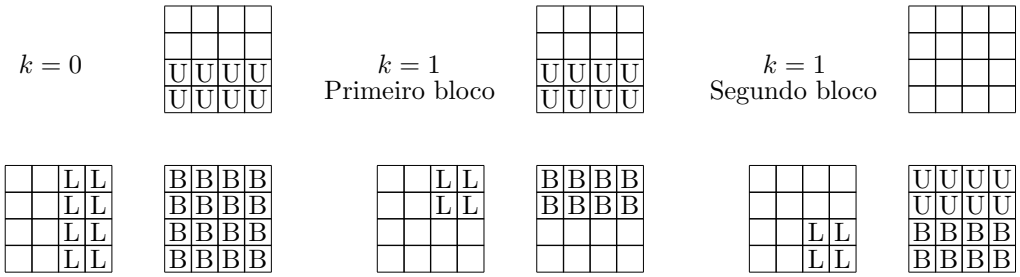


Figura 5.3: Exemplo de codificação de 3 blocos no MMP-2D-SM, sendo um da escla (0,0) e 2 da escala (1,0).

calculada de acordo com as equações 4.24, 4.25 e 4.26 e é sensível ao sentido que a variação dos *pixels* ocorre. Como um exemplo, as figuras 5.4 e 5.5 apresentam o cálculo da Rugosidade em duas situações. Podemos observar que na figura 5.5 a variação dos *pixel* envolvidos na fronteira dos blocos é menor do que o da figura 5.4, mas o valor dos *pixels* dos blocos vizinhos estão diminuindo no sentido do bloco do elemento do dicionário. No bloco do elemento do dicionário os *pixels* diminuem também no mesmo sentido. Mas na fronteira entre os blocos do elemento do dicionário e os vizinhos, o valor do *pixel* diminui no sentido contrário. Já na figura 5.4, estão todos diminuindo no mesmo sentido, resultando num valor da Rugosidade menor do que na figura 5.5. Essa sensibilidade no sentido da variação pode estar levando ao baixo desempenho quando usamos a imagem original, já que a tendência é que imagem recuperada seja formada por blocos cujo sentido de variação do valor dos *pixels* estejam de acordo com o sentido da variação computada pelo Rugosidade. No caso da imagem original, isto pode não estar ocorrendo.

Os gráficos das figuras 5.6 a 5.10 mostram os resultados usando o MMP-2D, MMP-2D-SM e MMP-2D-SM-A obtidos para as imagens relacionadas no anexo A. Nestes gráficos ficam claro a degradação do PSNR quando MMP-2D-SM-A é usado na compressão dessas imagens.

$$R_l = \left\lfloor \left| \frac{1+0}{2} - 1 \right| \right\rfloor + \left\lfloor \left| \frac{1+1}{2} - 1 \right| \right\rfloor + \left\lfloor \left| \frac{1+1}{2} - 1 \right| \right\rfloor + \left\lfloor \left| \frac{1+1}{2} - 1 \right| \right\rfloor = 0$$

$$R_u = \left\lfloor \left| \frac{1+0}{2} - 1 \right| \right\rfloor + \left\lfloor \left| \frac{1+1}{2} - 1 \right| \right\rfloor + \left\lfloor \left| \frac{1+1}{2} - 1 \right| \right\rfloor + \left\lfloor \left| \frac{1+1}{2} - 1 \right| \right\rfloor = 0$$

$$R = R_l + R_u = 0 + 0 = 0$$

R_l é a parcela da Rugosidade referente ao bloco vizinho lateral esquerdo.

R_u é a parcela da Rugosidade referente ao bloco vizinho superior.

X	X	X	X
X	X	X	X
103	103	103	103
102	102	102	102

Bloco vizinho superior

X	X	103	102
X	X	103	102
X	X	103	102
X	X	103	102

Bloco vizinho lateral

101	101	101	101
101	100	100	100
101	100	X	X
101	100	X	X

Bloco do dicionário sendo analisado

Figura 5.4: Exemplo de cálculo da Rugosidade para um bloco do dicionário de dimensões $4X4$.

$$R_l = \left\lfloor \left| \frac{1+0}{2} + 1 \right| \right\rfloor + \left\lfloor \left| \frac{1+1}{2} + 1 \right| \right\rfloor + \left\lfloor \left| \frac{1+1}{2} + 1 \right| \right\rfloor + \left\lfloor \left| \frac{1+1}{2} + 1 \right| \right\rfloor = 7$$

$$R_u = \left\lfloor \left| \frac{1+0}{2} + 1 \right| \right\rfloor + \left\lfloor \left| \frac{1+1}{2} + 1 \right| \right\rfloor + \left\lfloor \left| \frac{1+1}{2} + 1 \right| \right\rfloor + \left\lfloor \left| \frac{1+1}{2} + 1 \right| \right\rfloor = 7$$

$$R = R_l + R_u = 7 + 7 = 14$$

R_l é a parcela da Rugosidade referente ao bloco vizinho lateral esquerdo.

R_u é a parcela da Rugosidade referente ao bloco vizinho superior.

X	X	X	X
X	X	X	X
101	101	101	101
100	100	100	100

Bloco vizinho superior

X	X	101	100
X	X	101	100
X	X	101	100
X	X	101	100

Bloco vizinho lateral

101	101	101	101
101	100	100	100
101	100	X	X
101	100	X	X

Bloco do dicionário sendo analisado

Figura 5.5: Exemplo de cálculo da Rugosidade para um bloco do dicionário de dimensões $4X4$.

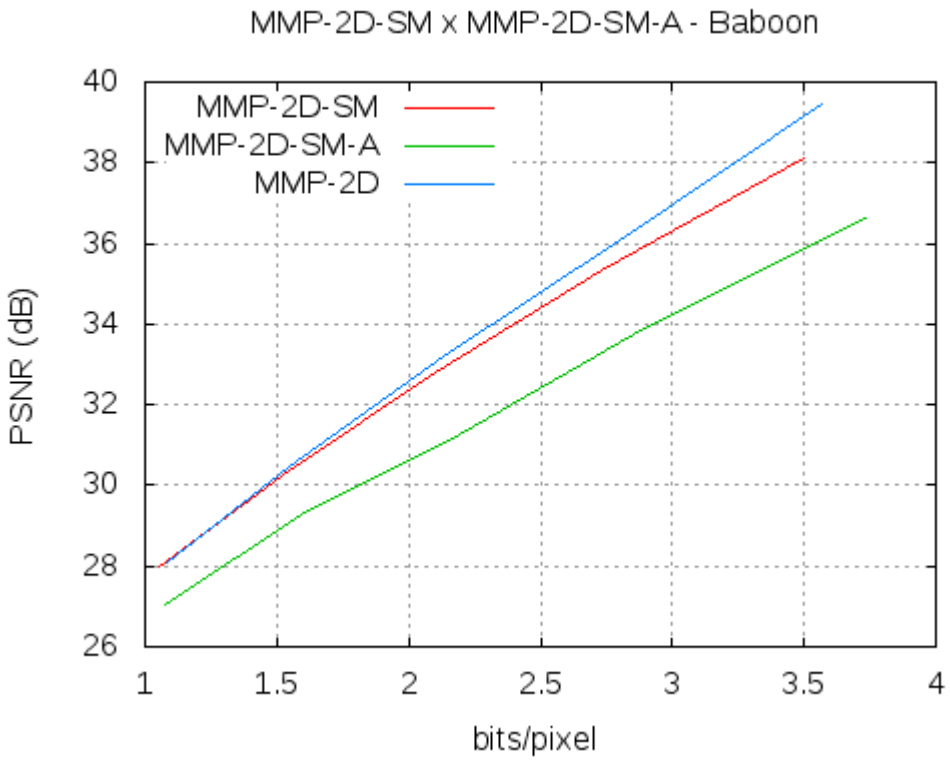


Figura 5.6: Gráfico comparativo entre o MMP-2D-SM e MMP-2D-SM-A usando a imagem Baboon.

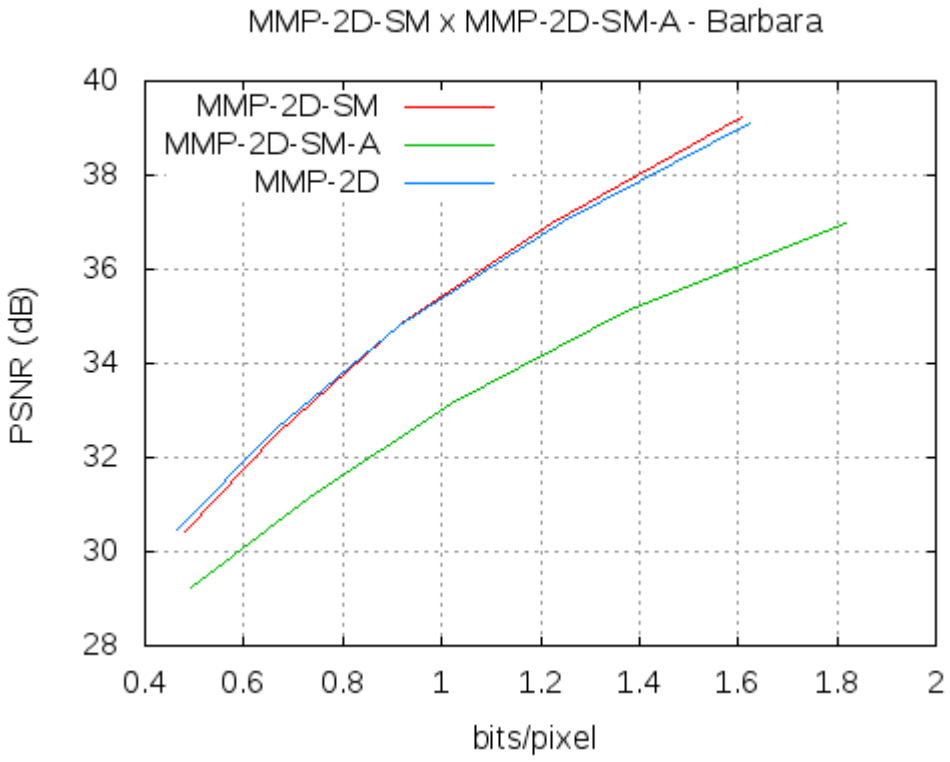


Figura 5.7: Gráfico comparativo entre o MMP-2D-SM e MMP-2D-SM-A usando a imagem Barbara.

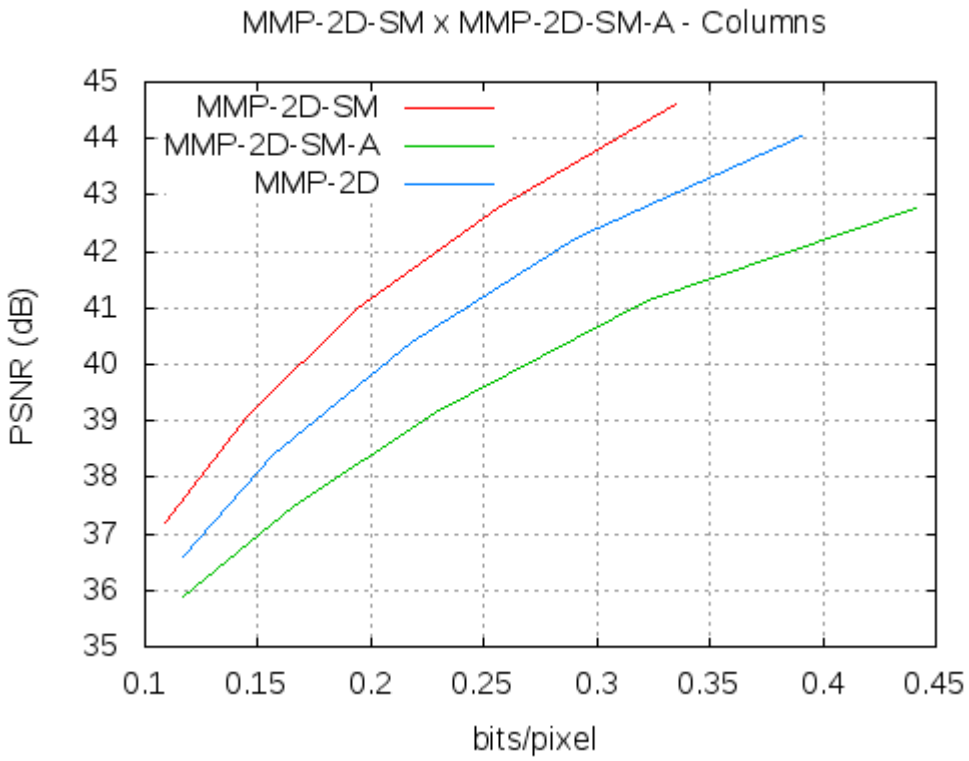


Figura 5.8: Gráfico comparativo entre o MMP-2D-SM e MMP-2D-SM-A usando a imagem Columns.

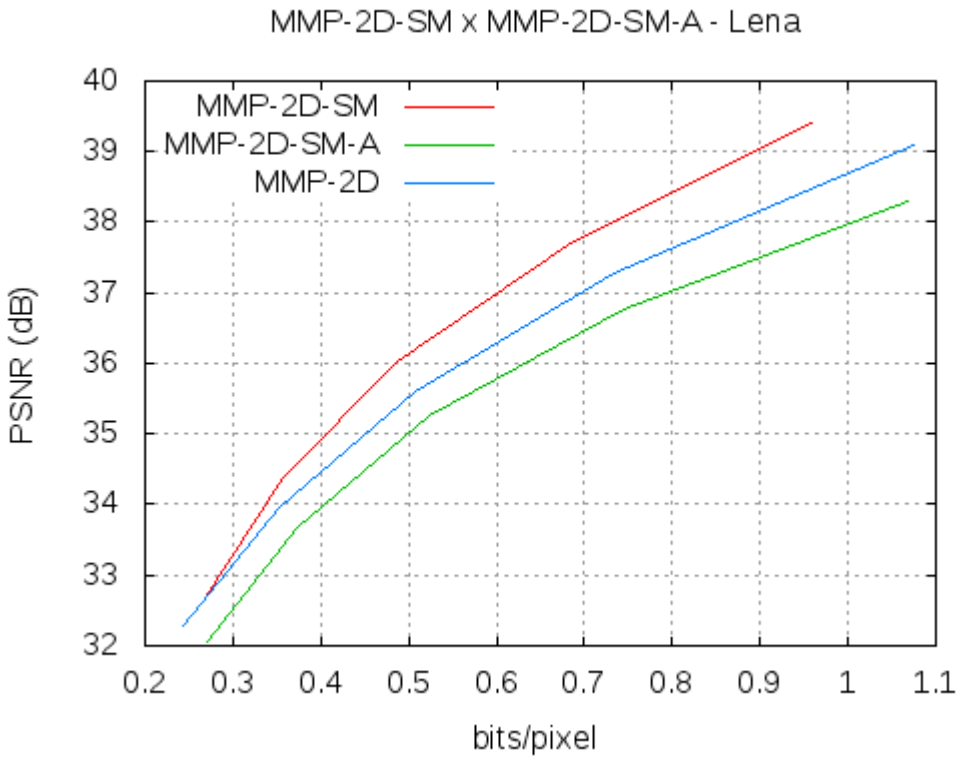


Figura 5.9: Gráfico comparativo entre o MMP-2D-SM e MMP-2D-SM-A usando a imagem Lena.

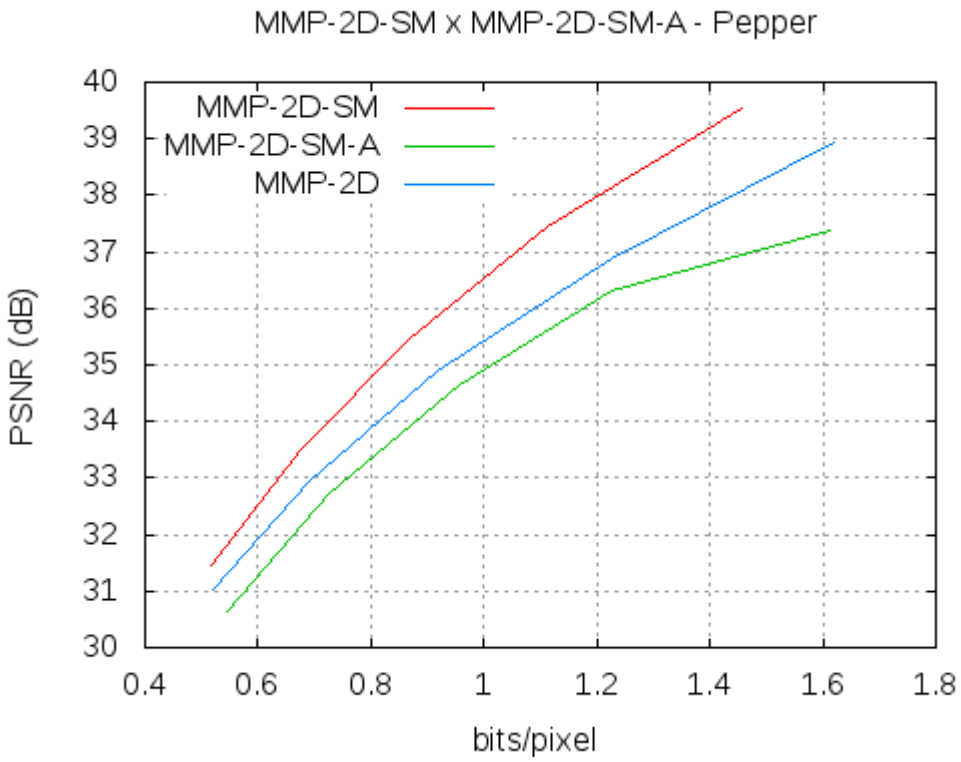


Figura 5.10: Gráfico comparativo entre o MMP-2D-SM e MMP-2D-SM-A usando a imagem Pepper.

Capítulo 6

MMP-2D-ND

6.1 Introdução

No capítulo anterior vimos que o MMP-2D-SM-A não foi uma boa alternativa para solucionar o problema de implementação do MMP-2D-SM usando a GPU. Apresentamos que a provável razão para o baixo desempenho desse algoritmo quando usamos a imagem original no lugar da recuperada é a sensibilidade do cálculo da Rugosidade ao sentido da variação dos *pixels* na fronteira entre os blocos da vizinhança. Neste capítulo vamos propor um nova métrica para determinar a dimensão e os elementos que vão compor o Dicionário de Estado, apresentando os resultados usando a imagem recuperada e original, e demonstrando sua viabilidade para ser implementado usando a GPU. Devido à característica da métrica usada, o algoritmo foi denominado de MMP-2D-ND, onde ND significa *Neighborhood Distance*.

O MMP-2D-ND é idêntico ao MMP-2D-SM, exceto na métrica usada para dimensionar e compor o Dicionário de Estado. Para dimensionar e compor o Dicionário de Estado uma métrica baseada na Variância e distância euclidiana são usadas respectivamente. A escolha dessas métricas foi realizada de forma heurística buscando alcançar o resultado esperado, que é um algoritmo com um desempenho igual ou melhor ao MMP-2D-SM, que ao usar a imagem original no lugar da recuperada não degradasse de forma acentuada seu desempenho, permitindo, desse modo, sua implementação usando a GPU.

Como o MMP-2D-ND é o MMP-2D-SM com métricas diferenciadas para a composição do Dicionário de Estado, este capítulo se resume na explicação do cálculo dessas métricas, da comparação dos resultados com o MMP-2D-SM, e dos resultados quando a imagem original é usada no lugar da imagem recuperada. Qualquer detalhe sobre o algoritmo já foi amplamente apresentado nos capítulos 4 e 5, e que devem ser consultados, caso seja necessário.

6.2 Determinação da Cardinalidade do Dicionário de Estado

Como dito, a dimensão do Dicionário de Estado é uma função de quão disperso é o valor dos *pixels* de um bloco. Entre as medidas testadas, a que a que apresentou o melhor desempenho para a métrica que será usada para selecionar os elementos do Dicionário de Estado foi a baseada na Variância do valor dos *pixels* dos blocos.

A dimensão do Dicionário de Estado é determinado pela equação 6.1. Como pode ser observado a equação usa o desvio padrão no lugar da variância porque o seu uso resultou num melhor desempenho do algoritmo.

$$N_s = \left\lfloor N_{s_{max}} \times \frac{\lfloor \sqrt{VAR_l + VAR_u} \rfloor}{\lfloor \sqrt{2VAR_{max}} \rfloor} \right\rfloor, \tag{6.1}$$

onde:
N_s é o número de elementos do dicionário de estado;
$N_{s_{max}}$ é o número máximo de elementos do dicionário de estado;
VAR_l é a variância do bloco vizinho lateral esquerdo;
VAR_u é a variância do bloco vizinho superior;
VAR_{max} é a variância máxima;
$\lfloor . \rfloor$ significa que somente a parte inteira do resultado deve ser considerada.

Na equação 6.1, $N_{s_{max}}$ é a maior dimensão que o Dicionário de Estado pode assumir. É um parâmetro configurado no algoritmo, e nas simulações realizadas o valor atribuído foi 4096. VAR_{max} é maior variância obtida dos blocos que são lidos da imagem, usando a equação 6.2. VAR_{max} deve ser computada e seu valor transmitido para o descompressor antes do início da compressão da imagem.

A variância é determinada pela bem conhecida equação $E\left[X^2\right] - E^2\left[X\right]$, que quando aplicada a um bloco da imagem torna-se a equação 6.2.

$$VAR = \left\lfloor \frac{\sum_{n=0}^{N-1} \sum_{m}^{M-1} p^2(n,m)}{N \times M} \right\rfloor - \left\lfloor \left(\frac{\sum_{n=0}^{N-1} \sum_{m}^{M-1} p(n,m)}{N \times M} \right)^2 \right\rfloor \tag{6.2}$$

Na equação 6.2, $\lfloor . \rfloor$ significa que somente a parte inteira do resultado da equação é considerada. N e M são os números de linhas e colunas do bloco respectivamente. $p(n,m)$ é o valor dos *pixels* do bloco.

Como no MMP-2D-SM a dimensão mínima que o Dicionário de Estado pode assumir é configurado no algoritmo e o valor usado nas simulações foi 16.

A figura 6.1 apresenta um exemplo de cálculo da dimensão do Dicionário de Estado.

$$VAR = \left\lfloor \frac{\sum_{n=0}^{3}\sum_{m}^{3} p^2(n,m)}{16} \right\rfloor - \left\lfloor \left(\frac{\sum_{n=0}^{3}\sum_{m}^{3} p(n,m)}{16} \right)^2 \right\rfloor$$

$$VAR_l = \left\lfloor \frac{224266}{16} \right\rfloor - \left\lfloor \left(\frac{1890}{16} \right)^2 \right\rfloor = 63$$

$$VAR_u = \left\lfloor \frac{387423}{16} \right\rfloor - \left\lfloor \left(\frac{2229}{16} \right)^2 \right\rfloor = 4806$$

$$N_s = \left\lfloor 4096 \times \frac{\lfloor \sqrt{63+4806} \rfloor}{\lfloor \sqrt{2 \times 4000} \rfloor} \right\rfloor = \lfloor 4096 \times \frac{69}{89} \rfloor = 3175$$

49	48	50	200
51	52	196	194
50	195	197	199
189	190	188	181

Bloco vizinho superior

116	117	111	108
110	118	118	120
112	115	112	111
127	130	135	130

Bloco vizinho lateral

Bloco sendo processado

Figura 6.1: Exemplo de cálculo da dimensão do Dicionário de Estado para um bloco de dimensões $4X4$, considerando $N_{s_{max}} = 4096$ e $VAR_{max} = 4000$.

6.3 Composição do Dicionário de Estado

A seleção dos elementos do Dicionário de Estado é realizada do mesmo modo que no MMP-2D-SM, exceto que neste algoritmo usamos outra métrica, que é baseada na distância euclidiana, e denominamos de "Distância da Vizinhança". A "Distância da Vizinhança" é definida nas equações 6.3, 6.4 e 6.5.

$$D = D_l + D_u \tag{6.3}$$

$$D_l = \sum_{n} \left[p_l(n, M) - p_d(n, 0) \right]^2 \tag{6.4}$$

$$D_l = \sum_{m} \left[p_u(N, m) - pd(0, m) \right]^2 \tag{6.5}$$

Na equação 6.3, D é a Distância da Vizinhança. D_l e D_u são as parcelas da Distância da Vizinhança referentes ao bloco lateral esquerdo e superior respectivamente. Nas equações 6.4 e 6.5, $p_l(n, m)$,

$p_u(n,m)$ e $p_d(n,m)$ são os *pixels* do bloco vizinho lateral esquerdo, vizinho superior, e do elemento do dicionário respectivamente. N e M são os números de linhas e colunas dos blocos respectivamente.

Como pode ser observado nas equações acima, somente os *pixels* da fronteira são usados. A figura 6.2 apresenta o cálculo da Distância da Vizinhança para um elemento do dicionário, candidato à seleção ao Dicionário de Estado.

$D_l = (108-100)^2 + (120-100)^2 + (111-100)^2 + (130-100)^2$
$D_l = 1485$

$D_u = (189-100)^2 + (190-100)^2 + (198-100)^2 + (181-100)^2$
$D_u = 30326$

$D = D_l + D_u = 1485 + 30326 = 31811$

D_l é a parcela da Distância da Vizinhança referente ao bloco vizinho lateral esquerdo.

D_u é a parcela da Distância da Vizinhança referente ao bloco vizinho superior.

49	48	50	200
51	52	196	194
50	195	197	199
189	190	188	181

Bloco vizinho superior

116	117	111	108
110	118	118	120
112	115	112	111
127	130	135	130

Bloco vizinho lateral

100	100	100	100
100	100	100	100
100	100	100	100
100	100	100	100

Bloco do dicionário sendo analisado

Figura 6.2: Exemplo de cálculo da Distância da Vizinhança para um bloco do dicionário de dimensões $4X4$ cujos elementos do bloco do dicionário possuem valor igual a 100.

6.4 Comparação com o MMP-2D-SM

A figura 6.3 apresenta duas imagens da lena. Uma recuperada após ser comprimida com o MMP-2D-SM e outra com o MMP-2D-ND. Podemos notar que, perceptualmente, as imagens são semelhante. Ambas apresentam o efeito de blocagem atenuado. Mas, apenas 30% dos *pixels* possuem o mesmo valor.

As figura 6.4 a 6.8 apresentam os gráficos da função $R(D)$ com curvas do MMP-2D-SM e MMP-

MMP-2D-SM

MMP-2D-ND

Figura 6.3: Imagens da Lena recuperadas após compressão usando o MMP-2D-SM e MMP-2D-ND.

2D-ND para as imagens apresentadas no anexo A. Em todas, excetuando a imagem da Lena, o MMP-2D-ND apresenta um desempenho inferior ao MMP-2D-SM, mas essa degradação não é maior que 1,5 dB para um $\lambda = 64$, que é um valor típico para o algoritmo. No gráfico da imagem da Lena, o MMP-2D-ND apresenta um desempenho um pouco melhor que o MMP-2D-SM nas taxas abaixo de $0,34$ *bits/pixel*. Acima desta taxa, o desempenho é inferior, não sendo inferior a $0,7$ dB do PSNR do MMP-2D-SM. Para $\lambda = 64$, o desempenho é o mesmo.

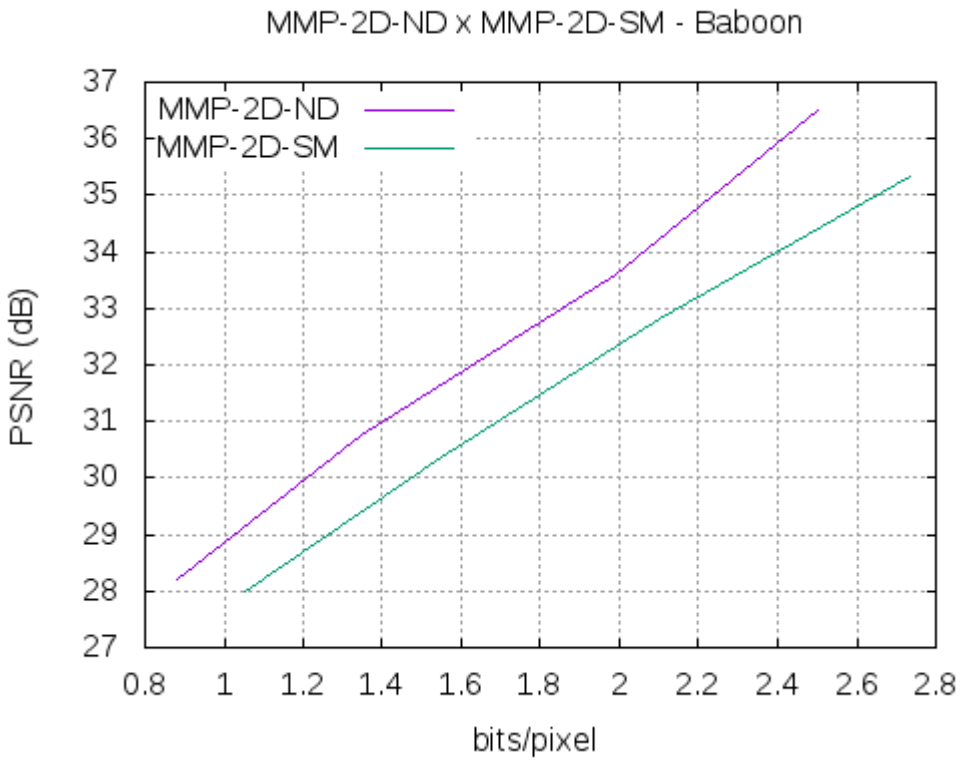


Figura 6.4: Gráfico comparativo entre o MMP-2D-ND e MMP-2D-SM usando a imagem Baboon

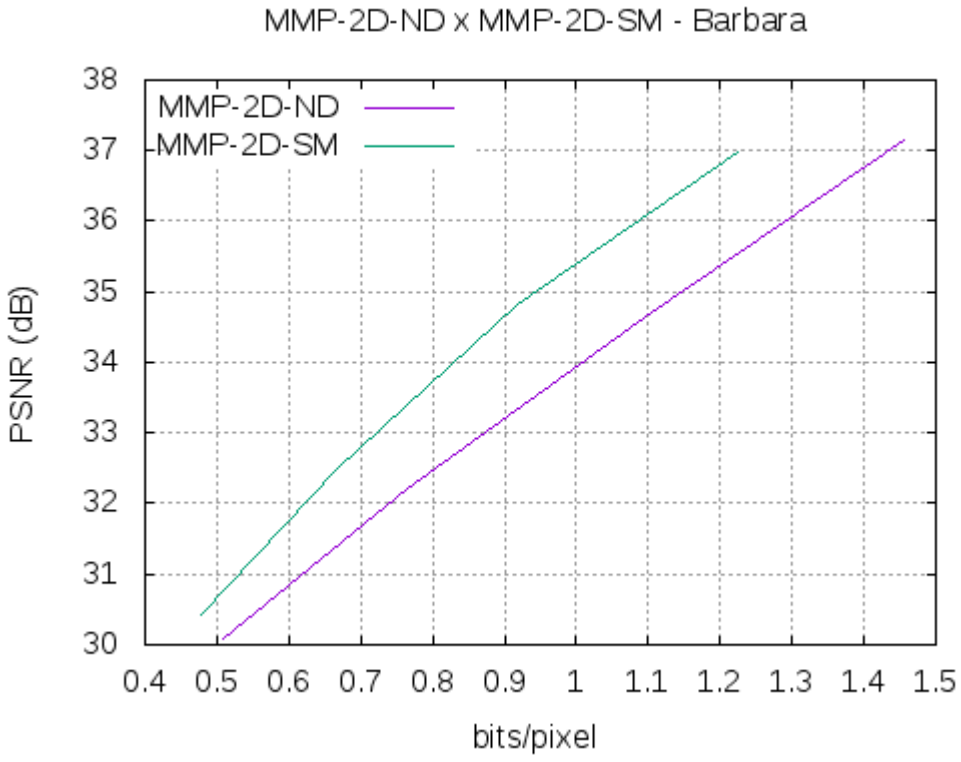


Figura 6.5: Gráfico comparativo entre o MMP-2D-ND e MMP-2D-SM usando a imagem Barbara

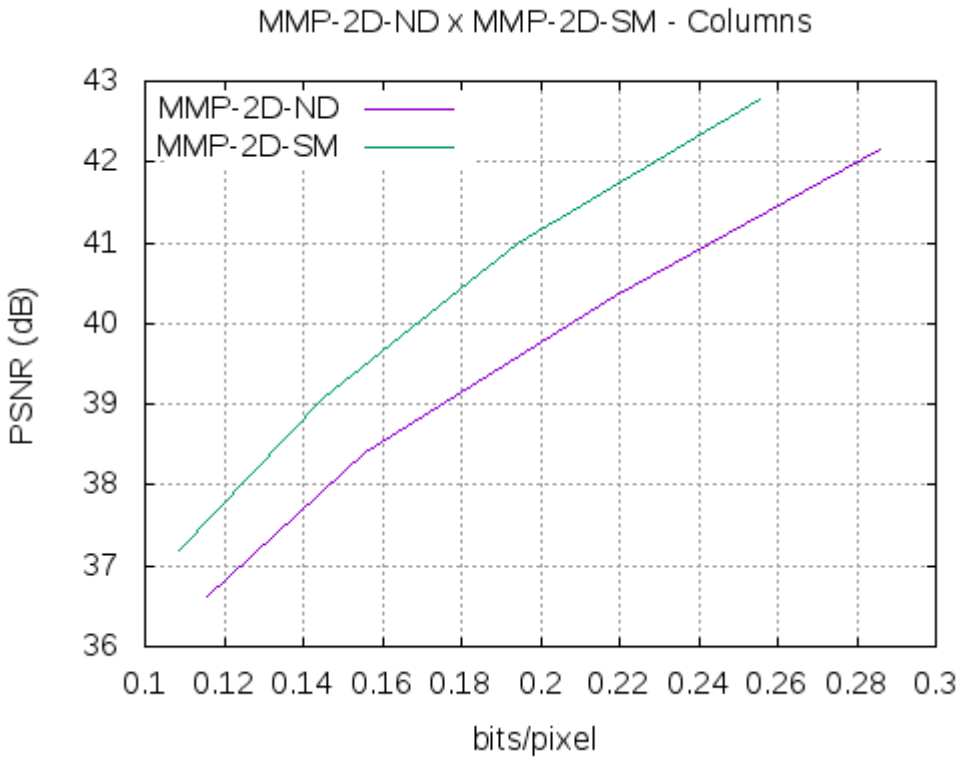


Figura 6.6: Gráfico comparativo entre o MMP-2D-ND e MMP-2D-SM usando a imagem Columns

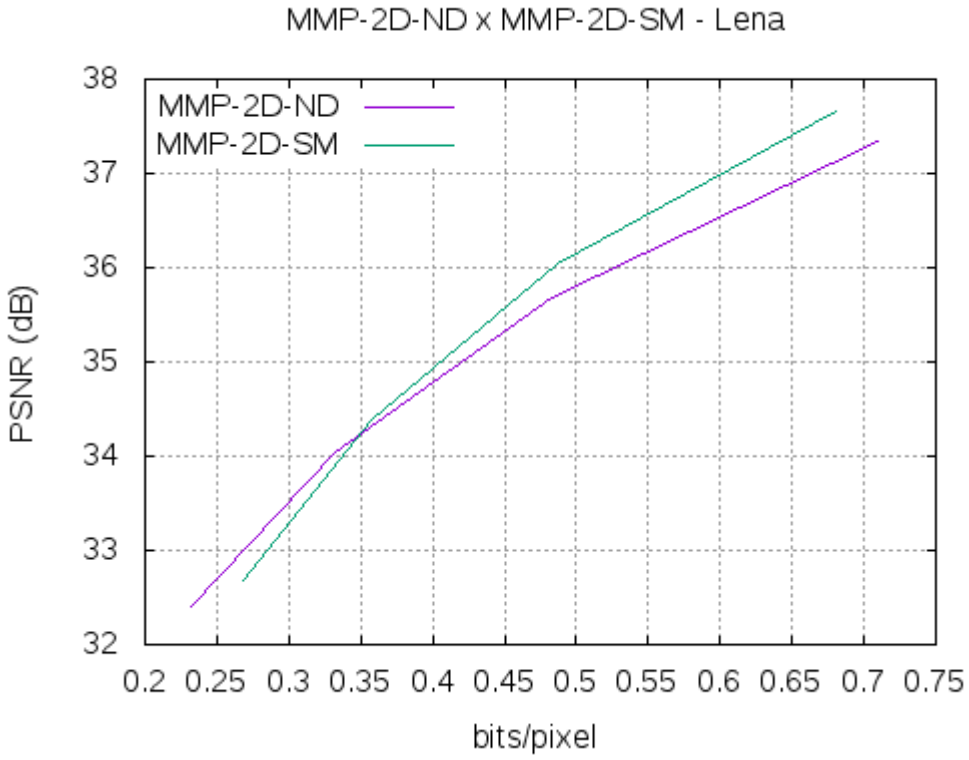


Figura 6.7: Gráfico comparativo entre o MMP-2D-ND e MMP-2D-SM usando a imagem Lena

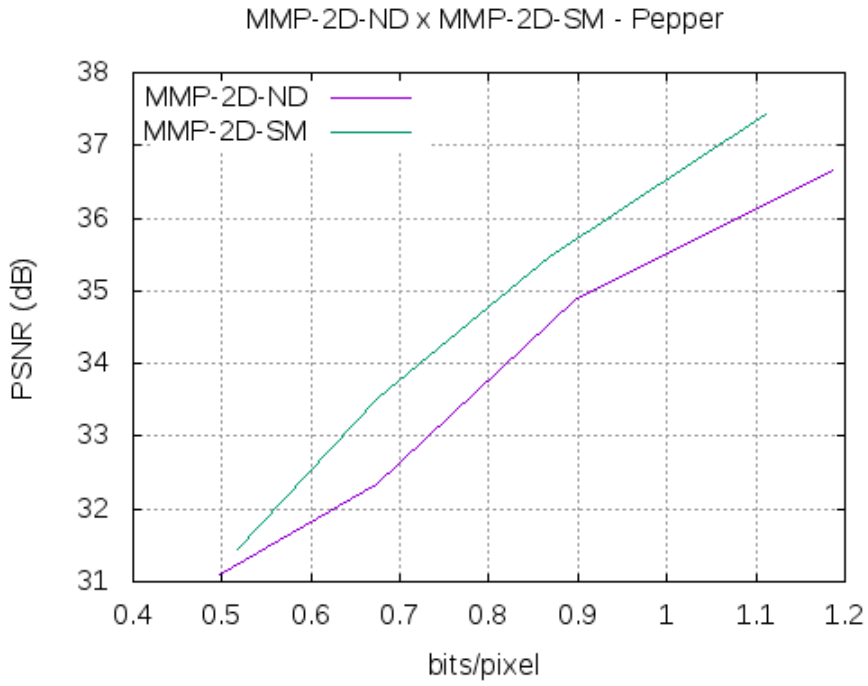


Figura 6.8: Gráfico comparativo entre o MMP-2D-ND e MMP-2D-SM usando a imagem Pepper

6.5 Uso da imagem original

Como o MMP-2D-SM, a versão alterada do MMP-2D-ND, que usa a imagem original no lugar da imagem recuperada na composição do Dicionário de Estado foi implementada e testada. As figuras 6.10 a 6.14 apresentam os gráficos da função $R(D)$ com as curvas para o MMP-2D, MMP-2D-ND e MMP-2D-ND-A. Como pode ser observado, o uso da imagem original no lugar da recuperada não degradou o desempenho do MMP-2D-ND, tornado esse algoritmo mais adequados para ser implementado na GPU do que a versão alterada do MMP-2D-SM. Em 3 imagens apresentou um desempenho igual ou melhor que o MMP-2D. O desempenho foi inferior na imagem Barbara e na imagem Pepper para taxas entre $0,58$ e $0,85$ *bits/pixel*.

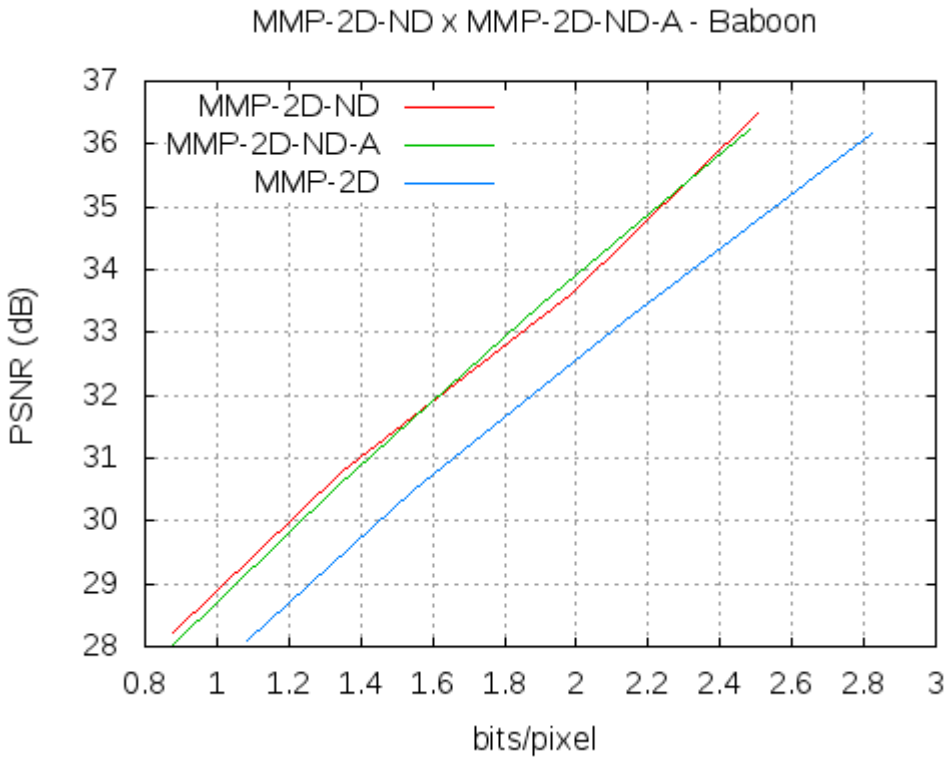


Figura 6.9: Gráfico comparativo entre o MMP-2D-ND e MMP-2D-ND-A usando a imagem Baboon.

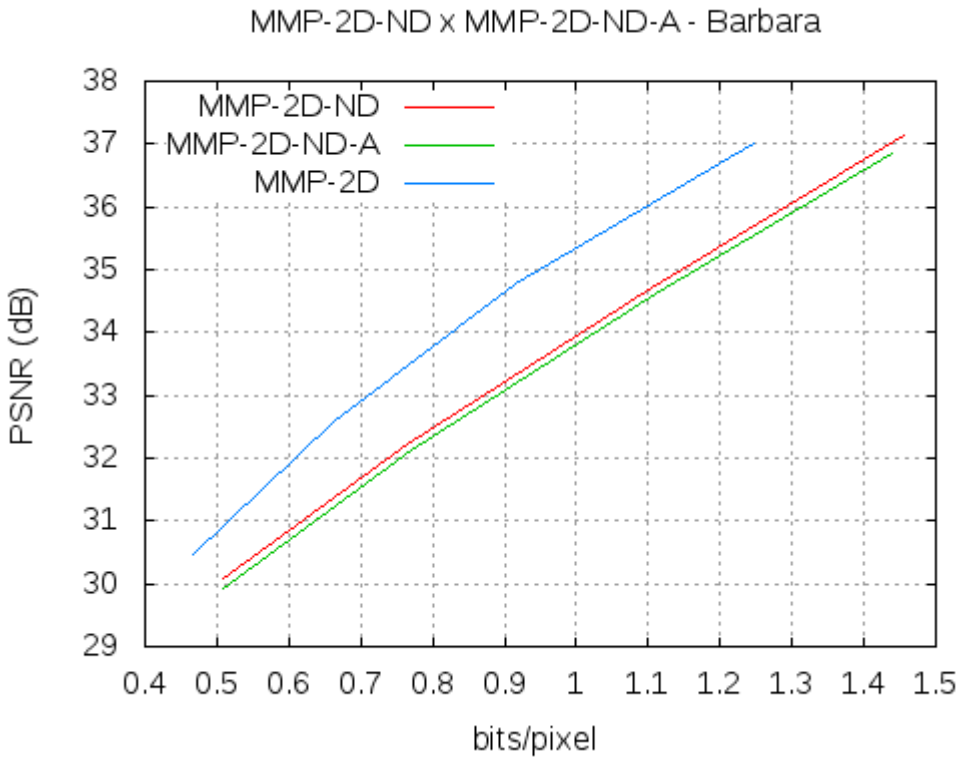


Figura 6.10: Gráfico comparativo entre o MMP-2D-ND e MMP-2D-ND-A usando a imagem Barbara.

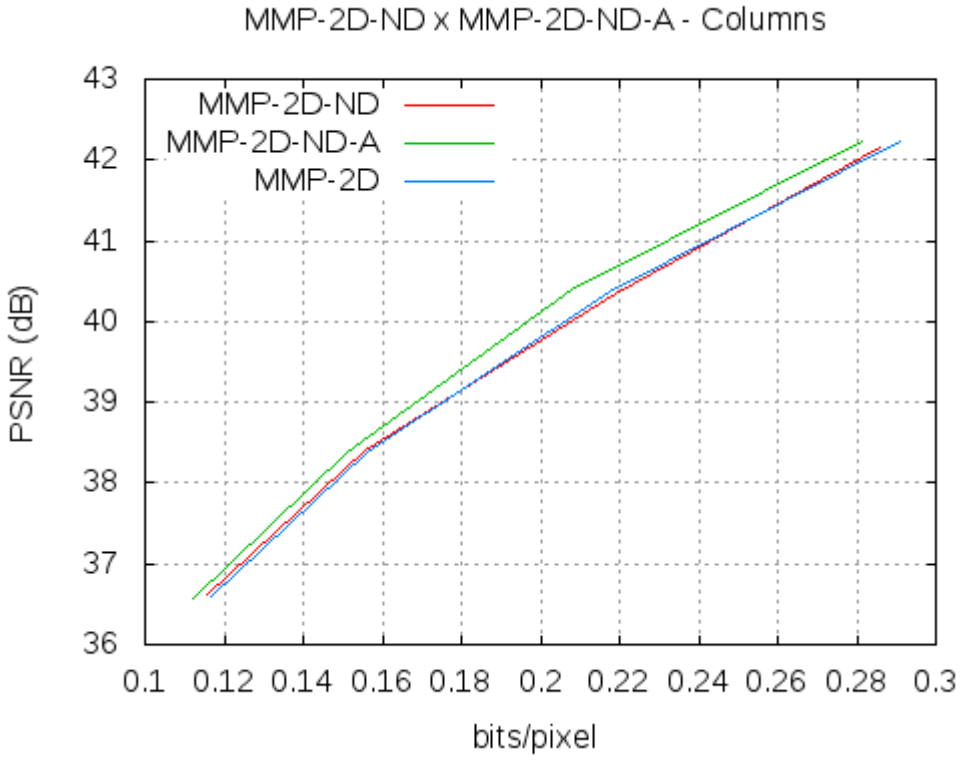


Figura 6.11: Gráfico comparativo entre o MMP-2D-ND e MMP-2D-ND-A usando a imagem Columns.

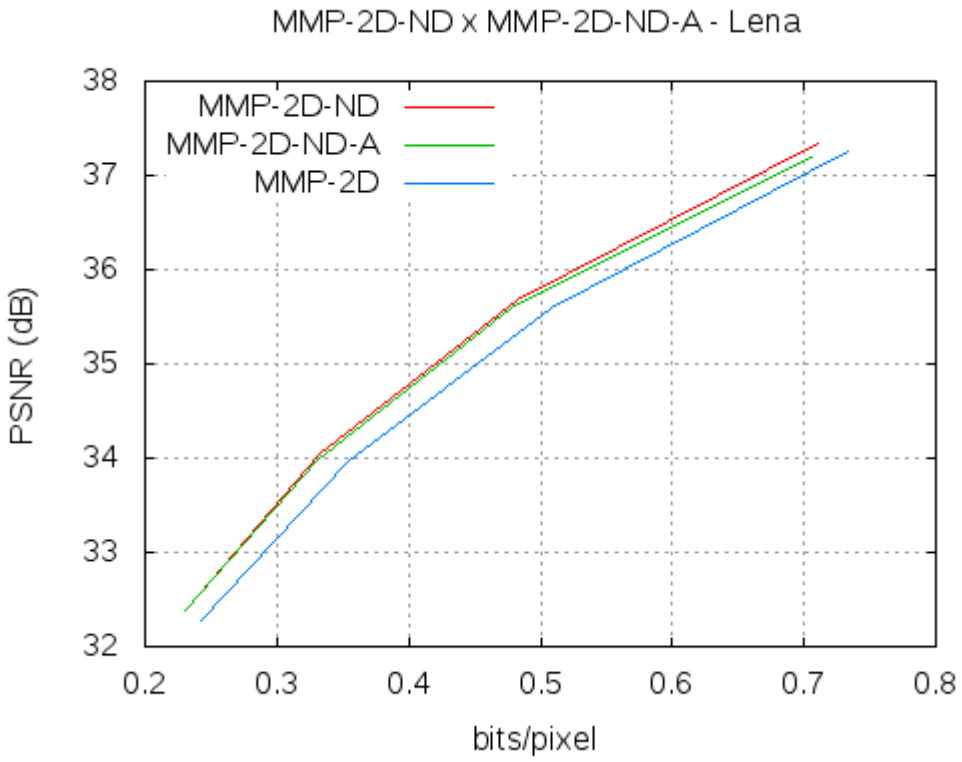


Figura 6.12: Gráfico comparativo entre o MMP-2D-ND e MMP-2D-ND-A usando a imagem Lena.

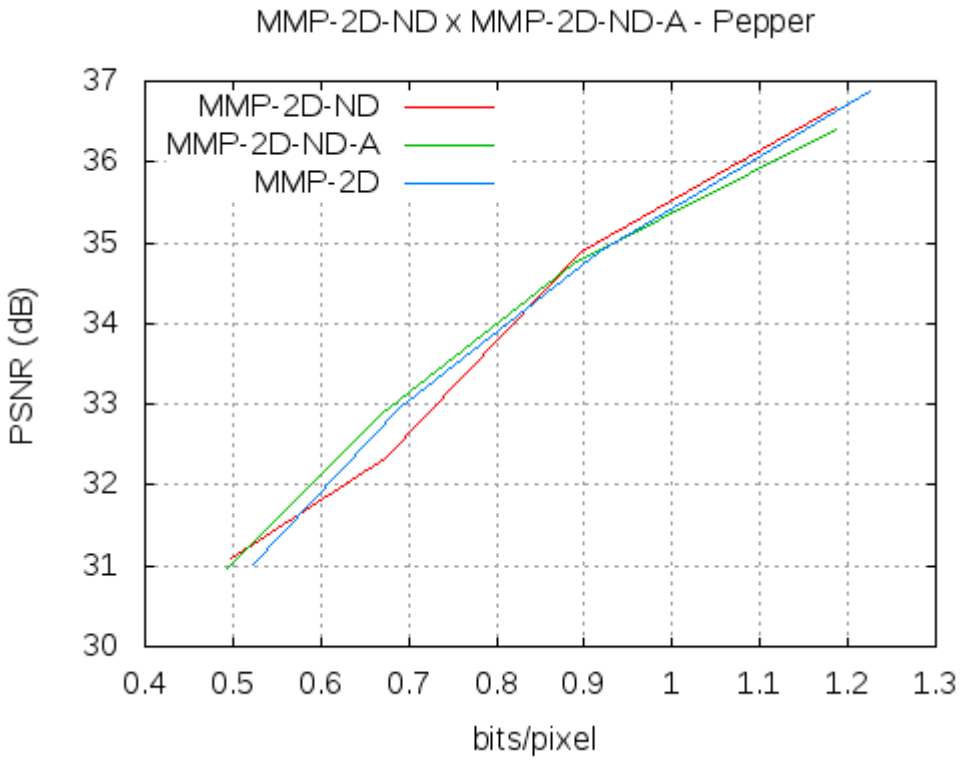


Figura 6.13: Gráfico comparativo entre o MMP-2D-ND e MMP-2D-ND-A usando a imagem Pepper.

6.6 Tempo de processamento e implementação usando a GPU

O tempo de processamento do MMP-2D-ND é de aproximadamente 6 horas. Esse tempo pode variar dependendo do tamanho e complexidade da imagem, bem como do valor de λ. Mas, de um modo geral, é longo. A execução do MMP-2D na GPU descrita no capítulo 4 reduziu o tempo, na média, em 87% com a configuração de *hardware* descrita. Se na implementação do MMP-2D-ND usando a GPU for conseguida a mesma redução, o tempo de processamento cairia para, aproximadamente, 43 minutos, que ainda é um tempo longo. Para encurtar essa duração, algumas otimizações no código ainda podem ser realizadas e o uso de GPU's com multiprocessadores melhores e capacidade de memória maior devem ser testados, buscando alcançar uma redução que torne o tempo de processamento curto o suficiente para aplicações em tempo real.

Capítulo 7

CONCLUSÃO

Apesar da versão do MMP-2D-ND que usa a imagem original não ter degradado a imagem recuperada pelo descompressor quando comparado com a versão que usa a imagem recuperada, seu desempenho em relação ao MMP-2D-SM foi um pouco pior. Isso pode decorrer do fato de que o MMP-2D-SM consegue compor o Dicionário de Estado com elementos que causam menor distorção do que o MMP-2D-ND. Como a rugosidade leva em conta a derivada, e portanto a inclinação dos níveis de cinza na fronteira, a tendência é que os elementos selecionados do dicionário reflitam o mesmo comportamento, conforme os *pixels* se afastem das fronteiras verificadas, se aproximando mais da variação do nínel de cinza do bloco lido da imagem ou os seus segmentos, que estão sendo processado, devido à correlação existente nas diversas áreas de uma imagem natural. Já a Distância da Vizinhança, mede apenas a diferença dos *pixels* na fronteria, o que não é suficiente, ou menos eficiente, para refletir o comportamento da variação do valor da escala de cinza conforme os *pixels* se afastam da fronteira. Mas a forma como a Rugosidade é calculada pode ser a causa da degradação do desempenho observada na solução usando a imagem original para permitir o uso de processamento paralelo. Logo, uma sugestão para trabalho futuro é manter a investigação de medidas baseada na distância euclidiana, que mostram-se mais adequada para o uso de processamento paralelo, porém envolvendo a vizinhança de uma forma diferente para tentar capturar outras características (comportamento) dos blocos envolvidos, e não apenas a sua diferença na fronteira. Por exemplo, seria interessante uma medida que, de algum modo, explorasse a direcionalidade na vizinhança, como os modos de predição que o codificador HEVC realiza, bem como a sua implementação em uma platarforma de processamento paralelo, como, por exemplo, a GPU.

Apêndice A

IMAGENS USADAS

Cinco imagens foram escolhidas para serem usadas nas simulações realizadas. As imagens foram obtidas em "http://people.sc.fsu.edu/ jburkardt/data/pgmb/pgmb.html", que pertence à Universidade do Sul da Flórida (Florida South University). Todas as imagens são em escala de cinza, sendo cada *pixel* quantizado em 256 níveis (8 *bits*), e os arquivos estão no formato "pgm" binário. As cenas contidas em cada uma são naturais.

Abaixo é apresntado cada uma das imagens utilizadas.

A.1 Imagem Baboon

Baboon é uma imagem que contém uma grande quantidade de detalhes, bem como a menor dependência entre os *pixels* adjacente, o que pode ser verificado comparando a Entropia de primeira e segunda ordem. Um outro ponto que pode ser avaliado com esta imagem foi o comportamento da expansão do dicionário do MMP, principalmente na GPU devido ao limite de mémória desses dispositivos.

A tabela A.1 contém os dados da imagem.

Parâmetros da Imagem	Valor
Número de coluna	512
Número de linhas	512
Número de *bits* por *pixel*	8
Valor médio	142,65
Variância	2213,24
Desvio padrão	47,05
Entropia de primeira ordem	7,3582
Entropia de segunda ordem	6,1473

Tabela A.1: Dados das imagens Baboon.

A figura A.1 contém a imagem Baboon e a figura A.2 o respectivo histograma.

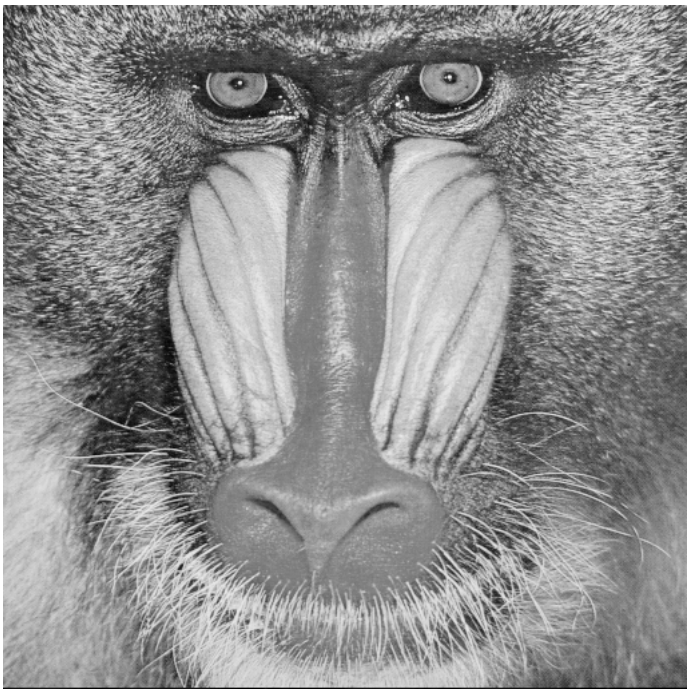


Figura A.1: baboon

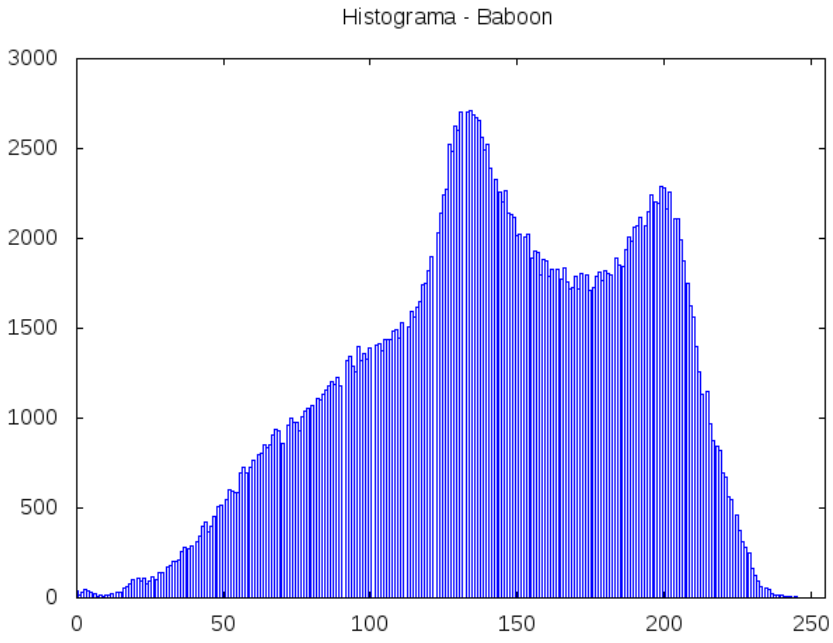


Figura A.2: Histograma da imagem Baboon.

A.2 Imagem Barbara

Barbara é uma imagem que contém algumas regiões com detalhes e outras com uma variação bem suave na intensidade dos *pixels*. O histograma mostra que a imagem não possui *pixels* na região de maior intensidade da escala de cinza. E foi escolhida apenas por mesclar áreas com características distintas.

A tabela A.2 contém os dados da imagem.

Parâmetros da Imagem	Valor
Número de coluna	512
Número de linhas	512
Número de *bits* por *pixel*	8
Valor médio	95,45
Variância	2231,65
Desvio padrão	47,24
Entropia de primeira ordem	7,4664
Entropia de segunda ordem	5,7761

Tabela A.2: Dados das imagens Barbara.

A figura A.3 contém a imagem Barbara e a figura A.4 o respectivo histograma.

Figura A.3: barbara

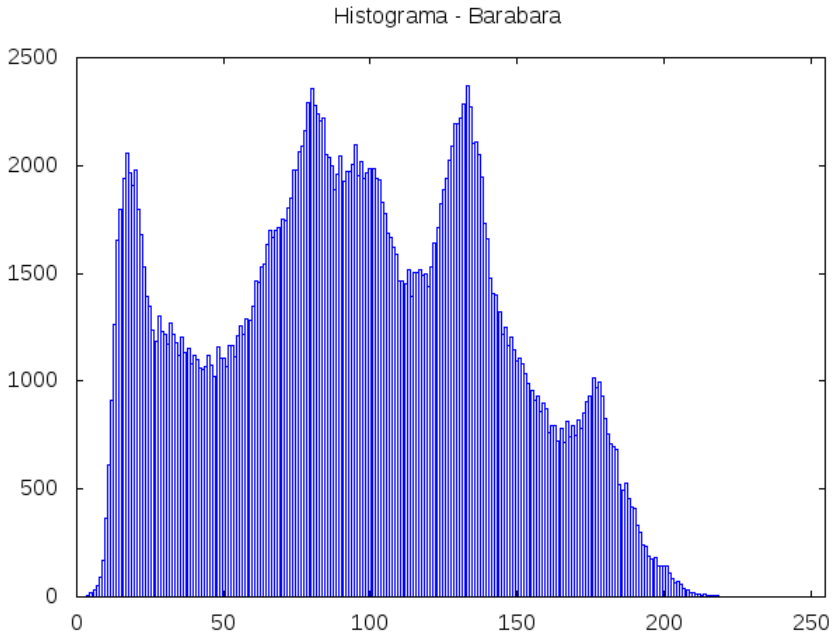


Figura A.4: Histograma da imagem Barbara.

A.3 Imagem Columns

Columns é uma imagem que contém poucas regiões com detalhes. Possui regiões com variação bem suave na intensidade dos *pixels*. Não possui *pixels* na região de baixa intensidade da escala de cinza. O valor dos *pixels* possui pouca variação na região acima de 75 e uma grande concentração de *pixels* de valor de maior intensidade da escala de cinza (255), o que ajudou a demonstrar o comportamento do Codificador Aritmético quando símbolos com alta probabilidade são codificados.

A tabela A.3 contém os dados da imagem.

Parâmetros da Imagem	Valor
Número de coluna	640
Número de linhas	480
Número de *bits* por *pixel*	8
Valor médio	159,86
Variância	3442,75
Desvio padrão	58,68
Entropia de primeira ordem	6,9266
Entropia de segunda ordem	2,8521

Tabela A.3: Dados das imagens Columns.

A figura A.5 contém a imagem Columns e a figura A.6 o respectivo histograma.

Figura A.5: columns

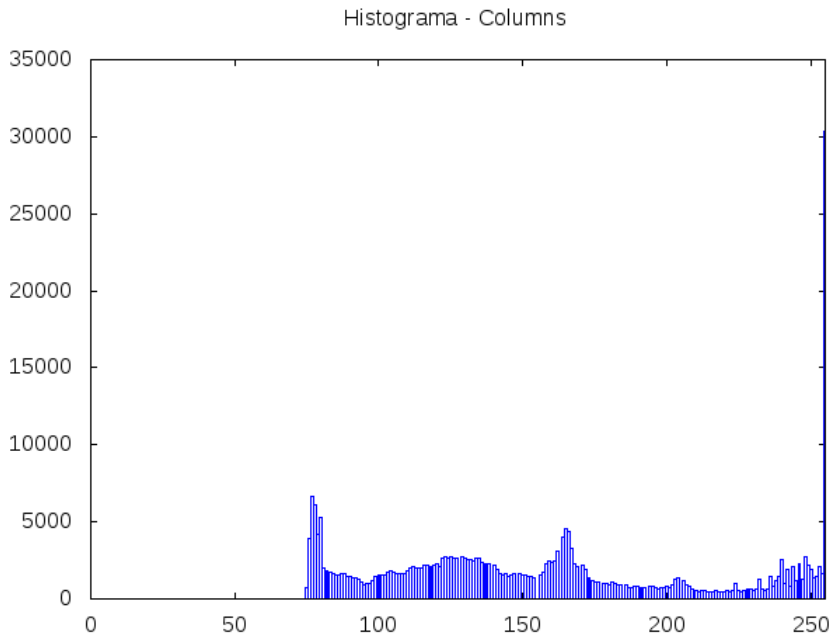


Figura A.6: Histograma da imagem Columns.

A.4 Imagem Lena

Lena é uma das imagens mais populares para quem trabalha com processamento digital de imagem. Por esta razão muitas vezes são encontrados resultados de trabalhos em que essa imagem é usada. A sua escolha é devido a essa grande popularidade, o que pode facilitar a comparação com resultados de outros trabalhos. A imagem mescla áreas com variações suaves do valor dos *pixels*, bem como a transição dessas áreas com outras que estão em outro plano. O histograma mostra que a imagem contém uma faixa grande de valores para os *pixels* concentrados na região central da faixa de escala de cinza.

A tabela A.4 contém os dados da imagem.

Parâmetros da Imagem	Valor
Número de coluna	512
Número de linhas	512
Número de *bits* por *pixel*	8
Valor médio	123,53
Variância	2295,72
Desvio padrão	47,91
Entropia de primeira ordem	7,4449
Entropia de segunda ordem	4,8862

Tabela A.4: Dados das imagens Lena.

Figura A.7: lena

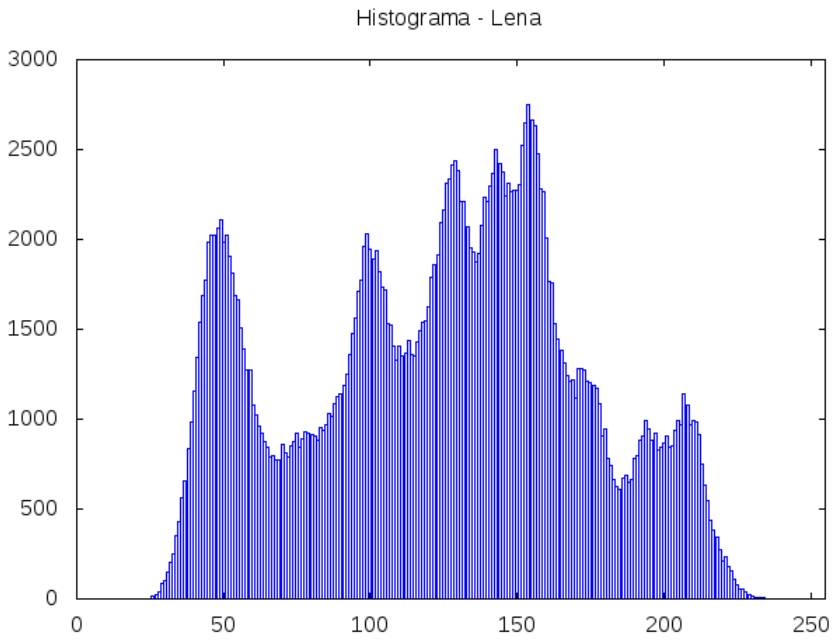


Figura A.8: Histograma da imagem Lena.

A.5 Imagem Pepper

Pepper é a imagem que possui a distribuição mais plana, bem como cobre quase toda a faixa de escala de cinza. Isto pode ser observado quando verificamos o valor de sua variância, que é o maior entre as imagens selecionadas. Devido a distribuição dos *pixels* quase plana, ajudou também a demonstrar o comportamento do Codificador Aritmético para uma fonte com essa característica.

Parâmetros da Imagem	Valor
Número de coluna	256
Número de linhas	256
Número de *bits* por *pixel*	8
Valor médio	124,96
Variância	6546,71
Desvio padrão	80,91
Entropia de primeira ordem	7,5432
Entropia de segunda ordem	4,6648

Tabela A.5: Dados das imagens Pepper.

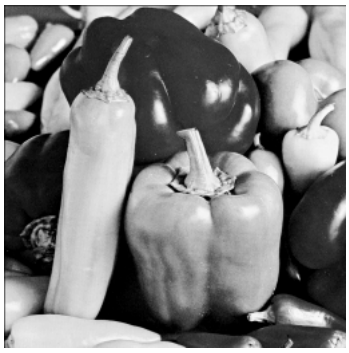


Figura A.9: pepper

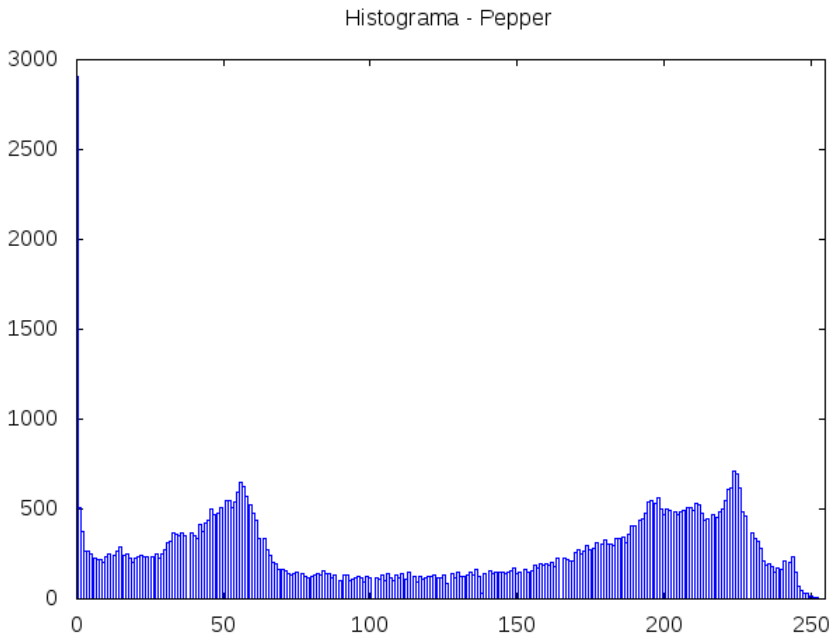


Figura A.10: Histograma da imagem Pepper.

Referências Bibliográficas

[1] Castellani V Bennedeto S., Biglieri E. *Digital Transmission Theory.* Prentice Hall, 1987.

[2] Murilo Bresciani de Carvalho. *Compressão de Sinais Multi-Dimensionais Usando Recorrência de Padrões Multiescalas.* PhD thesis, 2001.

[3] Janson Sanders e Edward Kandrot. *CUDA by Example.* Addison Wesley, 2010.

[4] Paul G. Howard e Jeffrey Scott Vitter. Pratical implementation of arithmetic coding, 1992.

[5] Tiago Ribeiro Eduardo A. B. da Silva Murilo B. Carvalho Frederico Silva Pinage Eddie Batista de Lima Filho, João Silva. Universal image compression using multiscale recurrent patterns with adaptive probability model. *IEEE Transactions on Image Processing*, 4(17):512–527, 2008.

[6] Tiago Ribeiro Murilo B. Carvalho Nuno M. M. Rodrigues Patricio Domingues Sergio M.M. de Faria Erica S. da Costa, João Silva. Gpu accelerated multiscale pattern matching algorithm. 2014.

[7] Anil K. Jain. *Fundamentals of Digital Image processing.* Prentice Hall, 1989.

[8] Tom Lookabaugh Dave Lindbergh Richard L. Baker Jerry D. Gibson, Toby Berger. *Digital Compression for Multimedia.* Morgan Kaufmann, 1998.

[9] Alberto Leon-Garcia. *Probability and Random Process for Electrical Engineering.* Addison Wesley, 1994.

[10] Jae S. Lim. *Two-Dimensional Signal and Image Processing.* Prentice Hall, 1990.

[11] Claude E. Shannon. A mathematical theory of communication. 1948.

[12] Nicholas Wilt. *The CUDA Handbook.* Addison Wesley, 2013.