**UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE FÍSICA DE SÃO CARLOS
DEPARTAMENTO DE FÍSICA E INFORMÁTICA**

Arquiteturas para Dilatação Exata

Maximiliam Luppe

Tese apresentada ao Instituto de Física de
São Carlos, da Universidade de São
Paulo, para obtenção do título de Doutor
em Física Aplicada

Orientador: *Prof. Dr. Valentin Obac Roda*

SÃO CARLOS

2003

Luppe, Maximilian

“Arquiteturas para Dilatação Exata”

Maximilian Luppe – São Carlos, 2003

Tese (Doutorado) – Área de Física da Universidade de São Paulo,
2003 - Páginas: 84

Orientador: Prof. Dr. Valentin Obac Roda

1. Dilatação Exata.

I. Título

IFSC**UNIVERSIDADE
DE SÃO PAULO****Instituto de Física de São Carlos**

Av. Trabalhador São-carlense, 400
Caixa Postal 369
13560-970 - São Carlos - SP - Brasil

**MEMBROS DA COMISSÃO JULGADORA DA TESE DE
DOUTORADO DE MAXIMILIAM LUPPE, APRESENTADA AO
INSTITUTO DE FÍSICA DE SÃO CARLOS, UNIVERSIDADE DE
SÃO PAULO, EM 14-03-2003.**

COMISSÃO JULGADORA:

Prof. Dr. Valentim Obac Roda (Presidente) – EESC-USP

Prof. Dr. Gonzalo Travieso – IFSC-USP

Prof. Dr. Eduardo Marques – ICMC-USP

Prof. Dr. Roberto Marcondes César Júnior – IME-USP

Profa. Dra. Agma Juci Machado Traina – ICMC-USP

Dedico este trabalho a todos os que, de
alguma forma, colaboraram para a minha
formação profissional, intelectual e
pessoal.

Agradecimentos

- Agradeço, inicialmente, ao caríssimo Prof. Dr. Valentin Obac Roda, pela sua orientação, paciência, carinho e profissionalismo com que tem dedicado a este trabalho e a quem devo todo respeito;
- Ao Prof. Dr. Luciano da Fontoura Costa, coordenador do Grupo de Visão Cibernética do Instituto de Física de São Carlos, pelo seu incentivo, idéias e co-orientação, além de ter acreditado neste trabalho;
- À Banca de Qualificação, formada pelos Profs. Drs. Eduardo Marques, Luciano da Fontoura Costa e Adilson Gonzaga, que acreditaram no meu potencial e me incentivaram a prosseguir;
- A todos que me acompanharam nesta longa jornada: Isaura, Pott, Leonardo, Marcelo, Emerson, Fabriciu, estagiários e alunos de iniciação científica;
- À Fundação Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pela ajuda financeira ao longo do desenvolvimento deste projeto;
- E ao apoio do projeto temático da FAPESP: “Desenvolvimento e avaliação de métodos originais e precisos em análise de formas e imagens e visão computacional”, proc. 1999/12765-2.

Sumário

LISTA DE FIGURAS	iii
LISTA DE TABELAS	iv
RESUMO	v
ABSTRACT	vi
1. Introdução	1
1.1. O Algoritmo	2
1.2. Contribuições	2
1.3. Organização dos Capítulos	3
2. Conceitos Básicos	5
2.1. Transformada de Distância	5
2.1.1. Representação de Distâncias Euclidianas	10
2.2. Dimensão Fractal	12
2.3. Geração de Esqueletos	15
2.3.1. Esqueletos por Zona de Influência (SKIZ)	17
2.4. Algoritmo para Dilatação Exata	17
2.5. Dispositivos Reconfiguráveis	21
2.6. Conclusão	22
3. Revisão Bibliográfica	23
3.1 Transformada de Distância - Algoritmos e Arquiteturas	23
3.1.1 Algoritmos Paralelos	23
3.1.2 Implementações em <i>Hardware</i>	25
3.2 Geração de Esqueletos - Algoritmos e Arquiteturas	26
3.3 Conclusão	29
4. Arquiteturas Propostas	31
4.1. Propostas Iniciais	32
4.1.1 Máscaras	32
4.1.2 <i>Mesh</i> de Processadores	34
4.2. Arquitetura para TDE	35
4.2.1 Algoritmo para Dilatação Exata Modificado	35
4.2.2 Implementação em <i>Hardware</i> do Algoritmo Modificado	37
4.3. Arquitetura para Geração de Esqueletos	40

4.3.1 Algoritmo para Propagação de Rótulos Modificado	40
4.3.2 Implementação em <i>Hardware</i> do Algoritmo Modificado	41
4.4. Algoritmo utilizando MPI	42
4.5. Conclusão	44
5. Implementação das Arquiteturas	45
5.1. Dilatação Exata	45
5.1.1 Célula de Processamento	48
5.1.2 Célula Central	51
5.1.3 Célula de Deslocamento	53
5.1.4 Célula de Deslocamento FIFO	53
5.1.5 Arquitetura Final	54
5.2 Propagação de Rótulos	56
5.2.1 Célula de Processamento	57
5.2.2 Célula Central	59
5.2.3 Célula de Deslocamento	60
5.2.4 Célula de Deslocamento FIFO	60
5.2.5 Arquitetura Final	61
5.2.6 Geração de Esqueletos	63
5.3. MPI	66
6. Conclusões	71
6.1. Trabalhos Futuros	73
7. Bibliografia	75
Apêndice	80

Lista de Figuras

Figura 2.1 - Gráficos mostrando as métricas <i>Chessboard</i> , <i>City Block</i> , <i>Chamfer Ótima</i> e <i>Quasi-</i>	7
Figura 2.2 - Módulo da diferença entre a distância Euclidiana calculada a partir de um ponto	8
Figura 2.3 - Módulo do desvio entre o valor esperado (distância Euclidiana) e o valor obtido	9
Figura 2.4 - Módulo do desvio entre o valor esperado (distância Euclidiana) e o valor obtido	10
Figura 2.5- Representação da estrutura SEDR	11
Figura 2.6 - Exemplo de <i>Structured Walking</i>	13
Figura 2.7 - Exemplo de Mapa de Distância	14
Figura 2.8 - Exemplo de Contagem de Caixas	14
Figura 2.9 - Exemplo de SKIZ	17
Figura 2.10 - Resultados da Dilatação Exata	18
Figura 2.11 - Exemplos da Transformada de Distância	19
Figura 2.12 - Exemplo de Propagação de Rótulos	19
Figura 2.13- Cálculo do esqueleto utilizando pixels 4 conectados	20
Figura 2.14 - Esqueletos obtidos pela limiarização do esqueleto original (a), juntamente com as	21
Figura 3.1 - Diagrama do Elemento de Processamento da arquitetura PPA	26
Figura 3.2 - Exemplo de problema para remoção em paralelo	28
Figura 4.1 - Matriz de distâncias Euclidianas (esquerda) e de respectivos índices (direita)	37
Figura 4.2 - Célula de Processamento Baseada no Algoritmo de Dilatação Exata Modificado	38
Figura 4.3 - Diagrama da Arquitetura para implementar o algoritmo de Dilatação Exata	39
Figura 4.4 - Exemplo de Propagação de Rótulos	41
Figura 4.5 - Célula de Processamento para Dilatação Exata e Propagação de Rótulos	42
Figura 5.1 - Gráfico da Representação da Distância	46
Figura 5.2 - Célula de Processamento PE_RASTER	50
Figura 5.3 - Célula Central PE0_RASTER	52
Figura 5.4 - Célula de Deslocamento	53
Figura 5.5 - Célula de Deslocamento FIFO	54
Figura 5.6 - Esquema geral da arquitetura para Dilatação Exata	55
Figura 5.7 - Célula de Processamento para Propagação de Rótulos	58
Figura 5.8 - Célula Central para Propagação de Rótulos	59
Figura 5.9 - Célula de Deslocamento para Propagação de Rótulos	60
Figura 5.10 - Célula de Deslocamento FIFO para Propagação de Rótulos	61
Figura 5.11 - Esquema geral da arquitetura para Dilatação Exata e Propagação de Rótulos	63
Figura 5.12 - Esquema geral do Módulo para Geração de Esqueleto	64
Figura 5.13 - Exemplo de Dilatação Exata: a) imagem original, b) mapa de distância,	64
Figura 5.14 - Exemplo de SKIZ 1: a) imagem original com as regiões rotuladas,	65
Figura 5.15 - Outro exemplo de SKIZ: a) imagem original com pontos aleatórios já rotulados	65
Figura 5.16 - Exemplo de esqueleto: a) imagem original de um neurônio, b) imagem do contorn	66
Figura 5.17 - Resultado do processamento paralelo utilizando MPI para o cálculo do Algoritmo	66
Figura 5.18 - Resultado do processamento paralelo utilizando MPI para o cálculo do Algoritmo	67

Lista de Tabelas

Tabela 2.1 - Lista de Índice das Possíveis Distâncias Distintas	
12	
Tabela 2.2 - Lista Modificada de Índice das Possíveis Distâncias Distintas	12
Tabela 5.1 - Distância Máxima por número de Bits para cada representação	45
Tabela 5.2 - Relação de Número de Células de Processamento (Ncproc) e Número de Células	47

Resumo

A Transformada Distância é uma importante ferramenta para o processamento de imagens digitais. A partir dela podemos calcular a dimensão fractal e obter o esqueleto de um objeto. Estas operações são muito importantes na análise de formas e no reconhecimento de padrões. Porém poucas são as implementações em hardware específico para este processamento. Neste trabalho apresentamos a implementação de arquiteturas paralelas para a determinação da Transformada Distância e para a geração de esqueletos baseados no algoritmo de Dilatação Exata e Propagação de Rótulos. Propomos também uma implementação do algoritmo utilizando a biblioteca MPI para o processamento paralelo.

Abstract

Distance Transform is an important tool for digital imaging processing. Using the Distance Transform it is possible to evaluate the fractal dimension and skeletons of objects. Fractal dimension and skeletons are very important in shape analysis and pattern recognition operations, few are the hardware implementation for these operations. In this work we present a parallel implementation based on the exact dilation and label propagation for the evaluation of the Distance Transform and skeletons generation. We also propose an algorithm implementation using the MPI library for parallel processing.

1. Introdução

Soluções simples e práticas para problemas difíceis e complexos. Esta deve ser a linha de pensamento de todo trabalho que envolve o desenvolvimento de projetos na área de alta tecnologia. É claro que podem existir outras soluções (mais complexas, é claro) para os mesmos problemas, porém as próprias soluções podem acabar se tornando mais um problema a ser resolvido. Tomemos como exemplo pelo absurdo a solução da multiplicação por meio de logaritmos. Sabemos que o logaritmo de um produto é igual à soma dos logaritmos dos fatores. É claro que é bem mais fácil calcular uma soma do que um produto, mas a solução adotada é extremamente complexa – o uso de logaritmos.

Na área de Processamento Digital de Imagens (PDI), o Reconhecimento de Padrões e a Análise de Formas são áreas complexas e que carecem de soluções práticas, principalmente no que diz respeito a implementações em *hardware* de algoritmos desta área. De forma mais específica, o cálculo da dimensão fractal e a esqueletização, que têm sido amplamente utilizados nestas áreas do PDI, tanto por fornecer informações sobre a complexidade da forma – no caso da dimensão fractal, quanto por permitir uma representação mais estruturada e mais simples de uma imagem – no caso da esqueletização, carecem de implementações práticas e eficientes em *hardware* paralelo.

Desta forma, este trabalho descreve a implementação em *hardware* de um algoritmo elegante e bastante funcional que envolve vários conceitos na área de PDI, em especial, na área de análise de formas. As arquiteturas propostas se baseiam no que há de mais comum e utilizado no PDI: uma máscara de processadores que

realizam operações locais, e que podem ser perfeitamente classificadas como arquiteturas tipo CLIP (*Cellular Logical Image Processor*) [1], ideais para o processamento inicial de imagens (diga-se grande quantidade de dados).

1.1. O Algoritmo

A idéia da implementação em *hardware* de um algoritmo para PDI é a principal motivação para este trabalho, porém a escolha do Algoritmo para Dilatação Exata e Propagação de Rótulos [4] surgiu de uma proposta feita pelo Prof. Dr. Luciano da Fontoura Costa, coordenador do Grupo de Pesquisa em Visão Cibernética (GPVC-IFSC-USP) e autor do algoritmo.

Este algoritmo tem servido de base para diversos trabalhos desenvolvidos pelo GPVC, sendo extremamente versátil e tendo demonstrado grande potencial de aplicação nas mais diversas áreas do conhecimento que envolva a análise de formas. Contudo, o seu tempo processamento é lento e consome grande parte do tempo total de processamento das aplicações nos quais ele é implementado. Assim surgiu a proposta de se estudar a viabilidade da sua implementação em *hardware* dedicado e a posterior construção de uma placa PCI para ser conectada a um PC e servir como co-processador para as aplicações em que este algoritmo for utilizado, minimizando o seu tempo de processamento.

1.2. Contribuições

Os resultados deste trabalho contribuem para o estudo e desenvolvimento de novas arquiteturas para o Processamento Digital de Imagens, mais especificamente na área de Análise de Formas. As arquiteturas criadas mostram uma maneira prática e rápida de se obter processamento paralelo por meio do uso de máscara, possibilitando um processamento em tempo real (na mesma taxa de aquisição dos dados) de imagens. Além do mais, o algoritmo implementado possibilita a extração de várias informações da imagem, o que reforça a contribuição deste trabalho. Outra

importante contribuição deste trabalho para a área de conhecimento foi o artigo “*Parallel implementation of exact dilations and multi-scale skeletonization*” [49] a ser publicado, que contempla os principais resultados obtidos neste trabalho.

Este trabalho possibilitou, ainda, a integração do Grupo de Pesquisa em Visão Cibernética do IFSC/USP, coordenado pelo Prof. Dr. Luciano da Fontoura Costa, com o Grupo de Visão Computacional Rápida do SEL/EESC-USP, coordenado pelo Prof. Dr. Valentin Obac Roda, por meio do projeto temático da FAPESP: “Desenvolvimento e avaliação de métodos originais e precisos em análise de formas e imagens e visão computacional”, proc. 1999/12765-2.

1.3. Organização dos Capítulos

Como o algoritmo utilizado para o desenvolvimento das arquiteturas possibilita a extração de vários dados de uma dada imagem, um capítulo inicial que trate de forma resumida os conceitos envolvidos se faz necessário. Neste capítulo (Capítulo 2) também faremos a apresentação do algoritmo utilizado, mostrando as suas particularidades.

No terceiro capítulo faremos a revisão bibliográfica na área de arquiteturas para o PDI, em especial daquelas que tratam da Transformada de Distância (TD) e da Geração de Esqueletos. É certo que existem poucas arquiteturas nesta área e as existentes não extraem todas as informações que o algoritmo utilizado neste trabalho permite extrair das imagens. Muito do que se tem feito na área de processamento paralelo se concentra na paralelização de algoritmos lineares, ou mesmo no desenvolvimento de algoritmos paralelos, que são baseados em arquiteturas paralelas “convencionais” (Hipercube) ou em modelos computacionais de arquiteturas paralelas (EREW-PRAM – Exclusive Read Exclusive Write - Parallel Random Access Memory). Assim este capítulo tratará de implementações em *hardware* de algoritmos paralelos.

No quarto capítulo faremos uma análise do algoritmo base deste trabalho e veremos como é possível a sua paralelização para uma possível implementação em *hardware* paralelo. Apresentaremos a arquitetura proposta e analisaremos uma possível implementação deste algoritmo em paralelo com o uso da biblioteca MPI.

No quinto capítulo apresentaremos os resultados obtidos na implementação, assim como alguns exemplos de imagens processadas e as possibilidades de uso destas arquiteturas.

No sexto e último capítulo faremos as considerações finais e apresentaremos as conclusões, assim como a exposição das contribuições deste trabalho e as propostas de trabalhos futuros.

Vale a pena salientar que todo o trabalho foi desenvolvido utilizando ferramentas para a implementação em dispositivos reconfiguráveis, que apresentam uma excelente plataforma na criação, desenvolvimento e simulação de arquiteturas para o Processamento Digital de Imagens.

Neste trabalho, as ferramentas utilizadas foram os aplicativos Altera Quartus II (versões 2.0 Full e 2.1 Web Edition), para síntese e simulação das arquiteturas, e Borland Delphi 6.0 Personal, para geração e visualização dos arquivos de simulação.

2. Conceitos Básicos

Apresentaremos neste capítulo o algoritmo que serve de base para este trabalho, o Algoritmo de Dilatação Exata e de Propagação de Rótulos. Porém, antes de apresentarmos o algoritmo, faremos uma breve descrição dos conceitos envolvidos nele, ou seja, das informações que podemos extrair de uma imagem por meio do seu uso, e de suas aplicações. Como veremos, este algoritmo é simples, mas bastante elegante e versátil, possibilitando que sejam extraídas várias informações, todas elas baseadas principalmente na Transformada de Distância Euclidiana (TDE).

2.1. Transformada de Distância

A Transformada de Distância (TD) foi inicialmente introduzida por Rosenfeld e Pfaltz [2]. É uma operação que, a partir de uma imagem binária contendo objetos e não-objetos (fundo), fornece outra imagem, em níveis de cinza, cujo valor de cada pixel, que pode ser tanto do objeto como do fundo, representa a sua distância à fronteira mais próxima. Normalmente a distância é calculada a partir de cada pixel pertencente ao objeto ao pixel mais próximo pertencente ao fundo. As aplicações da TD incluem a segmentação de imagens, a obtenção de esqueletos, a análise de *clusters*, etc.

Dada uma imagem I com $N \times N$ elementos binários, consideramos o elemento $I(i, j) = 0$, da linha i e da coluna j , pertencente ao fundo e o elemento $I(i, j) = 1$ pertencente ao objeto. Um Mapa de Distância (MD) é uma imagem onde o valor de cada elemento $MD[i, j]$ representa a distância de um pixel pertencente ao

fundo ao pixel mais próximo pertencente a um objeto existente na imagem. A Transformada de Distância é um procedimento (ou algoritmo) que calcula este mapa de distância a partir de uma imagem binária que contém o objeto (ou pelo menos o seu contorno).

Para o cálculo da TD é necessário especificar a métrica que será utilizada no cálculo da distância. A métrica mais conhecida é a que define a distância Euclidiana, definida por: $d(P_1, P_2) = \sqrt{(i_2 - i_1)^2 + (j_2 - j_1)^2}$, onde $P_1 = (i_1, j_1)$ e $P_2 = (i_2, j_2)$ representam dois pixels da imagem. Esta métrica define a Transformada de Distância Euclidiana (TDE). Porém, o custo computacional da TDE é muito alto, por envolver operações de multiplicação e raiz quadrada e, desta forma, tentativas de aproximar a distância Euclidiana utilizando métodos computacionais mais eficientes têm sido implementadas.

Aproximações da distância euclidiana podem ser obtidas a partir da distância com peso $d_w(P_1, P_2)$, entre os pixels $P_1 = (i_1, j_1)$ e $P_2 = (i_2, j_2)$, que é definida pela seguinte expressão:

$$d_w(P_1, P_2) = \begin{cases} w_0|i_1 - i_2| + w_1|j_1 - j_2| & \text{se } |i_1 - i_2| \geq |j_1 - j_2|, \\ w_1|i_1 - i_2| + w_0|j_1 - j_2| & \text{caso contrário,} \end{cases}$$

onde w_0 e w_1 são constantes não negativas. A distância com peso pode ser considerada como uma generalização de várias métricas. Por exemplo, d_w é:

- uma distância *City Block* se $(w_0, w_1) = (1, 1)$,
- uma distância *Chessboard* se $(w_0, w_1) = (0, 1)$,
- uma distância *Chamfer Ótima* se $(w_0, w_1) = (1, 1/\sqrt{2} - 1 + \sqrt{\sqrt{2} - 1})$,
- uma distância Quasi-Euclidiana se $(w_0, w_1) = (1, \sqrt{2} - 1)$,
- etc.

Apesar destas distâncias serem bem menos custosas computacionalmente do que a distância euclidiana, existe um compromisso entre o tempo de execução com os erros decorrentes destas aproximações. Um importante aspecto que deve também ser levado em consideração é a propriedade de invariância com relação à rotação, muito importante nas aplicações de reconhecimento e análise de formas. Em [3], Consularo realiza um estudo, tanto da TD como da invariância com relação à rotação, de várias métricas e mostra que, para pequenas distâncias, a métrica que melhor substitui a Euclidiana é a *Chamfer*, surgindo erros maiores para distâncias maiores.

Na Figura 2.1 temos as métricas *Chessboard*, *City Block*, *Chamfer Ótima* e Quasi-Euclidiana calculadas a partir de um ponto centrado em (20,20).

Figura 2.1 - Gráficos mostrando as métricas *Chessboard*, *City Block*, *Chamfer Ótima* e Quasi-Euclidiana calculadas em torno de um ponto centrado nas coordenadas (20,20)

Na Figura 2.2 temos o módulo da diferença entre as distâncias calculadas acima e a distância Euclidiana.

Figura 2.2 – Módulo da diferença entre a distância Euclidiana calculada a partir de um ponto centrado em (20,20) e as métricas *Chessboard*, *City Block*, *Chamfer Ótima* e *Quasi-Euclidiana* calculadas sobre este mesmo ponto

Nas Figuras 2.3 e 2.4 temos o erro, calculado pelo módulo da porcentagem de desvio entre o valor da distância Euclidiana e o valor obtido pelas métricas *Chessboard*, *City Block*, *Chamfer Ótima* e *Quasi-Euclidiana*, para distâncias até 20, de um ponto centrado em (20,20) (Figura 2.3), e até 100, de um ponto centrado em (100,100) (Figura 2.4).

Conforme podemos verificar, a métrica *Chamfer Ótima* é a que apresenta o menor erro com relação a distância Euclidiana entre todas as distâncias apresentadas, enquanto que a *City Block* apresenta o maior erro. Contudo, tanto a distância *Chamfer Ótimo* quanto a distância *Quase-Euclidiana* são baseadas em operações com

números reais, enquanto que as distâncias *Chessboard* e *City Block* trabalham apenas com números inteiros, o que torna a distância *Chessboard* mais adequada para implementações em *hardware*. Assim, a métrica mais adequada para implementações em *hardware* é a *Chessboard* que, apesar de apresentar um erro quase cinco vezes maior que as métricas *Chamfer Ótimo* e Quase-Euclidiana, este é menor que a *City Block*.

Figura 2.3 – Módulo do desvio entre o valor esperado (distância Euclidiana) e o valor obtido pelas métricas *Chessboard*, *City Block*, *Chamfer Ótimo* e Quasi-Euclidiana para distâncias até 20

Figura 2.4 – Módulo do desvio entre o valor esperado (distância Euclidiana) e o valor obtido pelas métricas *Chessboard*, *City Block*, *Chamfer Ótima* e *Quasi-Euclidiana* para distâncias até 100

Os métodos para se calcular a TD podem ser realizados de duas formas distintas: ou toma-se o fundo e calcula a distância de seus pontos até o ponto mais próximo do objeto, ou toma-se o objeto e calcula a distância dos pontos de sua borda aos pontos do fundo.

2.1.1. Representação de Distâncias Euclidianas

Um problema relacionado com o cálculo da TD está ligado com a representação da distância, principalmente no caso da distância Euclidiana, uma vez que seria necessário utilizar uma representação numérica em ponto flutuante, uma vez que os valores das distâncias são valores reais. Tal representação representa um grande custo sob o ponto de vista de implementação em *hardware*, visto que as operações aritméticas com esta representação necessitam de circuitos complexos. Uma primeira solução seria trabalhar com o quadrado da distância, o que

proporcionaria valores inteiros. Porém esta não é a melhor solução, visto que nem todos os valores inteiros seriam utilizados para representar as distâncias que podem ser calculadas entre pixels de uma imagem. A melhor opção é ordená-las em ordem crescente num vetor e utilizar o índice deste vetor como medida de distância.

Costa [4] apresenta um método para o cálculo da TD baseado numa estrutura de dados chamada SEDR (*Sorted Exact Distances Representation*). A estrutura SEDR armazena as distâncias Euclidianas e suas posições relativas dentro de uma matriz quadrada em ordem crescente, como é descrita abaixo. A estes valores damos o nome de Possíveis Distâncias Distintas.

Figura 2.5– Representação da estrutura SEDR

Esta estrutura é baseada numa lista indexada onde cada elemento armazena a distância Euclidiana d , o número total de elementos equidistantes e_d que possuem a mesma distância d a partir da origem, e um ponteiro para uma sub-lista que armazena todas as coordenadas relativas destes elementos à origem. O primeiro elemento desta lista (índice 0) será a distância 0, o segundo elemento (índice 1) será a distância 1, o terceiro elemento (índice 2) será a distância $\sqrt{2}$, e assim sucessivamente, como é mostrado no seguinte exemplo para um disco de raio 3. A utilização desta representação torna possível trabalhar com o índice (valor inteiro) ao invés da distância d (valor real), simplificando o processamento. Desta forma, após o processamento, a conversão do índice para a distância d pode ser realizada através de uma *Look Up Table* (LUT).

Tabela 2.1 – Lista de Índice das Possíveis Distâncias Distintas

Distância	0	1	$\sqrt{2}$	2	$\sqrt{5}$	$\sqrt{8}$	3	$\sqrt{10}$	$\sqrt{13}$	4
Índice	0	1	2	3	4	5	6	7	8	9

Considerando que os elementos equidistantes e_d possuem simetria com respeito ao eixo Cartesiano, é possível representá-los pelo formato (i, j) , onde i e j seriam valores inteiros que representam as coordenadas relativas à origem. Assim, os elementos equidistantes seriam representados pelas combinações $(+i, +j)$, $(+i, -j)$, $(-i, -j)$ e $(-i, +j)$. Contudo, algumas distâncias têm um número maior de elementos equidistantes (múltiplos de quatro) e, nestes casos, mais elementos teriam que ser introduzidos na estrutura SEDR, repetindo alguns valores da distância:

Tabela 2.2 – Lista Modificada de Índice das Possíveis Distâncias Distintas

Distância	0	1	$\sqrt{2}$	2	$\sqrt{5}_1$	$\sqrt{5}_2$	$\sqrt{8}$	3	$\sqrt{10}_1$	$\sqrt{10}_2$	$\sqrt{13}_1$	$\sqrt{13}_2$
Índice	0	1	2	3	4	5	6	7	8	9	10	11
(i, j)	(0,0)	(1,0)	(1,1)	(2,0)	(2,1)	(1,2)	(2,2)	(3,0)	(3,1)	(1,3)	(3,2)	(2,3)

Outro método seria considerar as combinações extras $(+j, +i)$, $(+j, -i)$, $(-j, -i)$ e $(-j, +i)$, além das quatro combinações descritas acima. É necessário levar em consideração a conveniência da implementação e do ganho real obtido antes de modificar a representação da estrutura SEDR.

2.2. Dimensão Fractal

O termo fractal surgiu no trabalho de Mandelbrot [5], que considerava que a natureza não poderia ser descrita por meio da geometria convencional, dita Euclidiana, ou seja, “As nuvens não são esferas, as montanhas não são cones, as linhas costeiras não são círculos e nem o relâmpago viaja em linha reta”. Para tanto ele propôs um novo conceito de geometria, chamada de geometria fractal. A principal característica do fractal está relacionada à sua dimensão, com o qual é possível determinar o grau de complexidade de uma linha ou a rugosidade de uma

superfície, ou, de acordo com Russ [6], a dimensão fractal é a taxa com a qual o perímetro (ou área de uma superfície) de um objeto aumenta quando a escala de medida é reduzida.

De acordo com Allen et al. [7], as estratégias de análise para medir a dimensão fractal podem ser divididas em dois grupos: métodos baseados em vetores e métodos baseados em matrizes. Dentre os métodos baseados em vetores, temos o método *Structured Walking*, que será explicado a seguir.

Imagine que queiramos medir o comprimento de uma linha costeira com um compasso de ponta seca que possui uma abertura inicial ε , conforme a Figura 2.6. O número de passos realizados com o compasso multiplicado por ε nos fornece um comprimento aproximado de $L(\varepsilon)$. Utilizando uma nova abertura ε' menor que a anterior, obteremos um novo valor de $L(\varepsilon')$. À medida que diminuimos o valor de ε , imaginamos que valor de $L(\varepsilon)$ irá convergir para o verdadeiro comprimento da linha costeira. Mas isto não ocorre e observamos que o valor de $L(\varepsilon)$ tende a aumentar sem limites. Isto se deve ao fato de que quanto menor a escala (valor de ε), mais detalhes aparecem. Desta forma podemos estabelecer a relação $L(\varepsilon) = M \varepsilon^{(1-D)}$, onde M é uma constante e D é definida como a dimensão fractal. Podemos observar que, a partir de um gráfico entre o logaritmo do perímetro (valor de $L(\varepsilon)$) obtido em cada escala e o logaritmo da escala (valor de ε), podemos obter o valor da dimensão fractal. Esta dimensão é mais conhecida por Dimensão de Richardson.

Figura 2.6 – Exemplo de *Structured Walking*

Dentre os métodos baseados em matrizes, temos os métodos de Contagem de Caixas (*Box Counting*) e de Mapa de Distância. O método de Mapa de Distância baseia-se no uso de um processo conhecido por salsicha de Minkowski. Neste processo, cada ponto que pertence ao contorno do objeto é dilatado ou sobreposto

por círculos de raio ε , formando assim faixas, ou bandas, mais conhecidas por salsichas de Minkowski, que possuem uma área $A(\varepsilon) \propto \varepsilon^{(2-D)}$ [9]. Limiarizando um mapa de distâncias (criado a partir dos pontos de contorno do objeto) em diferentes níveis de cinza, criamos faixas similares àsquelas obtidas pelo processo da salsicha de Minkowski. Assim como no caso do método *Structured Walking*, também podemos obter a dimensão fractal por meio de um gráfico do logaritmo do perímetro $L(\varepsilon)$ pelo logaritmo do raio ε , sendo o perímetro $L(\varepsilon)$ obtido por meio da divisão da área $A(\varepsilon)$ pela largura média da salsicha, igual ao valor de limiar. Porém, é mais interessante utilizar um gráfico do logaritmo da área $A(\varepsilon)$ pelo logaritmo do raio ε e, ao invés de limiarizar o mapa de distância em diferentes valores de níveis de cinza para obter a área da faixa, utilizar o histograma da imagem para calcular a área da faixa para cada nível de cinza. A dimensão fractal obtida por este método é mais conhecida por Dimensão Minkowski.

Figura 2.7 – Exemplo de Mapa de Distância

O método de Contagem de Caixas consiste em dividir a imagem, que contém apenas a borda do objeto, em quadrados de largura ε (caixas) e contar o número n de caixas que são necessárias para cobrir toda a borda. As caixas que se sobrepõem à borda formam uma “faixa” cuja área é definida por $A(\varepsilon) = n\varepsilon^2$ e o seu comprimento será igual à área $A(\varepsilon)$ dividida pela largura média ε , ou seja $L(\varepsilon) = n\varepsilon$. Por meio de um gráfico do logaritmo do perímetro $L(\varepsilon)$ pelo logaritmo da largura ε podemos determinar a dimensão fractal, que é mais conhecida por Dimensão Kolmogorov.

Figura 2.8 – Exemplo de Contagem de Caixas

Além destes, outros métodos para o cálculo da dimensão fractal foram criados [8] e avaliados [9-11], sendo que de acordo com Bérubé e Jébrak [9] e Allen et al. [7], o método de Mapa de Distância é o melhor método, dentre os métodos baseados em matrizes, por ser mais exato e menos sensível a variações de tamanho e resolução. As aplicações da dimensão fractal estão relacionadas à análise de formas, tal como a análise de formas de neurônios [10], estudo de infiltração em solos [11], análise de rugosidade [12], entre outras.

2.3. Geração de Esqueletos

Não existe ainda uma definição clara e completa a respeito de esqueletos. Discute-se muito mais a respeito de suas características de conectividade (o esqueleto possui o mesmo número de componentes que o objeto), largura unitária e centralidade (o objeto pode ser reconstruído a partir do esqueleto) do que o que realmente é um esqueleto e qual é a melhor forma de obtê-lo.

A primeira definição do que seria um esqueleto foi dada por Blum¹, numa analogia a um incêndio em um gramado. Nesta analogia, o esqueleto (ou eixo médio, conforme definiu Blum) seria formado pelos pontos onde duas frentes de fogo se encontram, tendo estas partido de diferentes pontos da borda do gramado ao mesmo tempo e a velocidade constante.

Desde então, vários métodos têm sido criados para se obter esqueletos baseados nesta idéia de eixo médio. Não existe uma regra geral para classificar os métodos, ou algoritmos, para geração de esqueletos, mas eles podem ser divididos em dois grandes grupos [13 e 14]. O primeiro grupo refere-se aos métodos que geram esqueletos a partir de sucessivas remoções de pixels da borda do objeto, de acordo com alguma regra de exclusão que define se um determinado pixel será ou não removido. A regra mais geral está relacionada à condição de que a remoção de um pixel não poderá dividir o objeto em duas ou mais partes (conectividade). Estes

métodos são chamados de *peeling* ou *thinning*. Algoritmos sequenciais [15-16] e paralelos [17] têm sido desenvolvidos utilizando esta estratégia. Em [13], Lam et al. realizam um estudo de diversos algoritmos sequenciais e paralelos. Dentro desta classe de algoritmos também temos aqueles que são baseados em Morfologia Matemática [18], onde o esqueleto é obtido através de sucessivas erosões controladas.

Outro grupo refere-se aos métodos que se baseiam no conceito de “máximos discos”: esqueleto é o conjunto dos centros dos máximos discos que podem ser inscritos num objeto e que toca em pelo menos dois pontos da borda deste (um disco é dito máximo se ele não está incluso em nenhum outro disco contido no objeto). Este método, também conhecido por MAT (*Medial Axis Transform*) também pode ser estendido para outras formas geométricas, como quadrados e retângulos [19]. Muitos autores que utilizam este método também utilizam a TD para se certificarem de que os pontos obtidos realmente são centros dos discos máximos e para obter as distâncias destes pontos à borda e, assim, poder reconstruir o objeto sobrepondo discos, com raios iguais a estas distâncias, aos pontos do esqueleto. Vários algoritmos foram implementados utilizando diversas métricas, tais como: *Chamfer 2-3* [20], *City Block*, *Chessboard*, *Chamfer 3-4* e *Chamfer 5-7-11* [21], distância Euclidiana obtida por Morfologia Matemática [22] e distância Euclidiana [23]. Salientamos que todos estes métodos foram desenvolvidos para objetos 2D, mas existem trabalhos para geração de esqueletos a partir de objetos 3D, utilizando tanto *thinning* [24], como TD [25].

Vale a pena ressaltar que esta divisão dos métodos para se gerar esqueletos em dois grupos não é única e nem definitiva, uma vez que não existe ainda um consenso geral sobre o que é esqueleto e qual a melhor forma de obtê-los. Para exemplificar, Lam et al. [13] definem o método de geração de esqueletos desenvolvido por Arcelli e di Baja [15] como sendo *thinning*, sendo que os autores o consideram baseados na TD [16].

¹ Malandain, G.; Vidal, S. F., “Euclidean skeletons”, *Image and Vision Computing*, v.16, p.317-327, 1998

A principal aplicação da geração de esqueletos está na área de reconhecimento de padrões e na análise de formas, porque ela permite uma representação mais simples e mais estruturada da imagem, e tem sido utilizada em neuromorfometria [26], inspeção de circuitos impressos [27], detecção de face [28] e análise de dimensão fractal [29], entre outras.

2.3.1. Esqueletos por Zona de Influência (SKIZ)

Da mesma forma que os esqueletos dos objetos de uma imagem podem ser obtidos, também é possível gerar esqueletos do fundo. O esqueleto do fundo, também conhecido por Esqueleto por Área de Influência, ou SKIZ (*Skeleton by Influence Zone*), possibilita a separação de todos os objetos de uma imagem.

Na Figura abaixo temos um exemplo de SKIZ. A imagem a esquerda possui vários pontos e, após o processamento, apresentam as suas áreas de influência (figura central) e à direita temos a imagem original sobreposta ao SKIZ.

Figura 2.9 – Exemplo de SKIZ

Aplicações de SKIZ incluem a análise de *clusters* [30], relações espaciais entre objetos numa imagem [31], entre outras.

2.4. Algoritmo para Dilatação Exata

Apresentamos agora o algoritmo utilizado como base para o desenvolvimento da arquitetura que serve de base para este trabalho: o Algoritmo de Dilatação Exata e

Propagação de Rótulos [4]. Este método fornece vários resultados, além da TD, conforme é mostrado na figura abaixo.

Figura 2.10 – Resultados da Dilatação Exata

Uma vez que a estrutura SEDR foi definida, é possível determinar a Dilatação Exata seguindo todo o contorno do objeto de forma sequencial para cada conjunto e_d de cada posição relativa para cada distância Euclidiana. Para cada passo, isto é, para cada distância d e para cada pixel p que pertence ao contorno, os pixels correspondentes às posições relativas associadas à distância d são analisados ao redor do pixel p e o valor **1** é associado ao pixel que ainda não foi associado.

para cada distância d da SEDR **faça**
para cada pixel p do contorno **faça**
para cada posição relativa e_d **faça**
 se o pixel $p + e_d$ está vazio
então coloque 1 neste pixel

Algoritmo 1 – Algoritmo de Dilatação Exata

Se, ao invés de associarmos o valor **1** aos pixels, nós associarmos o valor d , então nós obteremos a TD. Na Figura 2.10 podemos ver a imagem original (esquerda), o mapa de distância nas distâncias 7, 20 e 45, respectivamente. Lembramos que a partir deste mapa é possível obter a dimensão fractal do objeto.

Figura 2.11 – Exemplos da Transformada de Distância

Este processo, também chamado por Transformada de Distância Exata, também pode ser utilizado para a propagação de rótulos. Para isto, inicialmente enumeramos em ordem crescente cada um dos pixels que fazem parte do contorno do objeto, utilizando um algoritmo de busca de contorno e, durante o processo de propagação de distância, o valor do rótulo do pixel é associado ao pixel que estiver vazio, ao invés da sua distância d . Na Figura 2.12 podemos ver um exemplo de propagação de rótulos.

Figura 2.12 – Exemplo de Propagação de Rótulos

A partir do mapa de rótulos é possível se obter o esqueleto S do objeto, realizando uma operação de máxima diferença entre um pixel e seus 4-conectados, como é mostrado na Figura 2.13. Esta operação produz uma família de esqueletos multi-escala com as seguintes características:

- esqueletos com largura de 1 pixel e 8 conectados;
- preservação da topologia;
- permite reconstituição em escala especial sem deslocamento de borda.

Figura 2.13– Cálculo do esqueleto utilizando pixels 4 conectados

Uma família de esqueletos multi-escalas pode ser facilmente obtida por meio da limiarização da imagem S em diferentes valores inteiros, o qual define a escala espacial. Na figura 2.14 abaixo temos um exemplo de limiarização e reconstrução do objeto original a partir dos esqueletos. A primeira figura é o esqueleto obtido, juntamente com o seu mapa de distância. As imagens seguintes representam os esqueletos obtidos com os limiares 2, 17, 32 e 47 e a respectiva reconstrução do objeto a partir deste.

Figura 2.14 – Esqueletos obtidos pela limiarização do esqueleto original (a), juntamente com as informações do mapa de distância (b). Os pares seguintes são os esqueletos obtidos através do limiares de 2 (c), 17 (e), 32 (g) e 47 (i), com as respectivas reconstruções (d), (f), (h) e (j)

Se, ao invés de enumerarmos o contorno dos objetos, nós enumerarmos os objetos dentro da imagem e aplicarmos o algoritmo de Propagação de Rótulos, nós obteremos uma imagem com as áreas de influência de cada objeto, ou seja, o SKIZ (*Skeleton by Influence Zone*).

2.5. Dispositivos Reconfiguráveis

Os dispositivos reconfiguráveis têm crescido muito nos últimos anos, não só em quantidade de famílias de dispositivos disponíveis no mercado, como também no número de elementos reconfiguráveis por dispositivo, que hoje já ultrapassa o 1 milhão de portas lógicas. Seu uso tem sido uma alternativa prática e barata aos dispositivos tipo ASIC (*Application Specific Integrated Circuit* – Circuito Integrado de Aplicação Específica), que são muito caros e apresentam um tempo de desenvolvimento bem mais longo. Juntamente com os dispositivos reconfiguráveis, as ferramentas de desenvolvimento também tem acompanhado esta evolução, tornando-se fator importante e decisivo no sucesso de um fabricante de dispositivos reconfiguráveis.

Como foi dito anteriormente, o uso dos dispositivos reconfiguráveis neste trabalho se deve ao fato de que as ferramentas de desenvolvimento para este tipo de dispositivo têm se apresentado como uma prática plataforma no desenvolvimento de arquiteturas para o PDI, possibilitando a sua rápida implementação e simulação.

2.6. Conclusão

Podemos ver que este simples algoritmo é extremamente sofisticado por fornecer várias informações sobre a imagem. A única desvantagem deste algoritmo é que ele é extremamente caro, sob o ponto de vista computacional, por envolver três laços de iteração. Este foi o grande desafio deste trabalho, tornar o algoritmo viável por sua implementação em uma arquitetura paralela para o processamento em tempo real.

Notamos também a importância da TD nas áreas de Análise de Formas. Vimos que através dela podemos obter Esqueletos, SKIZ, Dimensão Fractal e realizarmos a Dilatação de objetos, e, assim, utilizá-la em diversos campos do PDI. Isto nos desafia a criarmos uma arquitetura unicamente para obter a TDE para que ela possa ser utilizada no processamento de imagens em tempo real, além da arquitetura para calcular o Algoritmo de Dilatação Exata e Propagação de Rótulos.

3. Revisão Bibliográfica

3.1 Transformada de Distância – Algoritmos e Arquiteturas

3.1.1 Algoritmos Paralelos

Conforme foi citado anteriormente, os métodos para se calcular a Transformada de Distância (TD) podem ser realizados de duas maneiras distintas: ou toma-se o fundo e calcula a distância de seus pontos até o ponto mais próximo do objeto, ou toma-se o objeto e calcula a menor distância entre seus pontos e os pontos do fundo. A segunda maneira, muitas vezes, é implementada através de ondas para evitar que pontos do fundo sejam analisados várias vezes, ou seja, todo pixel que já tiver sua distância calculada pode ser utilizado para calcular a distância de seus vizinhos.

Implementações de algoritmos em paralelo da Transformada de Distância Euclidiana (TDE) têm sido propostas com o intuito de obter resultados mais precisos, de tal maneira que o tempo necessário para os cálculos seja próximo, ou menor, que o tempo necessário para computar a Transformada de Distância com as métricas aproximadas.

A primeira implementação paralela da TDE foi proposta por Yamada apud Takala e Viitanen¹. Este algoritmo é da ordem de $O(N)$ com relação ao tempo, para

¹ Yamada, H. “Complete Euclidean distance transformation by parallel operation.” Proceedings of 7th International Conference on Pattern Recognition, Montreal (Canadá), pp.69-71, 1984. apud Takala, J. H.; Viitanen, J. O. “Distance Transform Algorithm for Bit-Serial SIMD Architectures.” Computer Vision and Image Understanding, v.74, n.2, pp. 150-161, 1999

um arranjo de $N \times N$ processadores 8-conectados. Quando este algoritmo é convertido para a forma sequencial sobre uma imagem de $N \times N$ elementos, sua complexidade se torna da ordem de $O(N^3)$ com relação ao tempo.

Hirata [32] propõe um algoritmo unificado para computar mapas de distância. O algoritmo, cuja complexidade com relação ao tempo de execução é da ordem $O(N^2)$ para uma imagem binária de $N \times N$ elementos, foi inicialmente desenvolvido para o cálculo da TDE. Hirata mostra que este algoritmo pode ser implementado para uma variedade de distâncias. A versão paralela deste algoritmo possui uma complexidade da ordem $O(N^2/p)$ com $1 \leq p \leq N$ processadores e se baseia no trabalho de Chen e Chuang [33], onde a TDE é dividida em dois passos. No primeiro passo, as distâncias são calculadas entre os pixels de uma mesma coluna. No segundo passo, o cálculo das distâncias são realizados nas colunas. Isto permite o paralelismo do primeiro passo. Aliás, grande parte das implementações em paralelo para o cálculo da TD são baseados nesta divisão em dois passos, um na vertical e outro na horizontal.

Lee et al. [34] realizam um estudo da implementação de um algoritmo para computar a TDE em duas arquiteturas paralelas: o modelo EREW-PRAM (*Exclusive Read Exclusive Write - Parallel Random Access Memory*) e uma arquitetura tipo Hipercubo. O modelo PRAM consiste de processadores autônomos, sendo que cada processador pode acessar aleatoriamente uma memória comum. Em cada ciclo, cada processador realiza a mesma instrução. Para o modelo EREW-PRAM, uma posição de memória não pode ser simultaneamente acessada por mais de um processador.

Este algoritmo também se baseia no cálculo da TDE inicialmente em cada coluna e, depois, em cada linha, contudo com um melhorias que levam a um algoritmo com complexidade da ordem de $O(\log^2 N)$ para N^2 processadores, tanto para o modelo EREW-PRAM, como para o Hipercubo. Lee e Horng [36] também apresentam uma implementação paralela para a TD baseada na métrica *Chessboard* utilizando os mesmos conceitos.

Fujiwara et al. [36] utilizam estes mesmos modelos de arquitetura EREW-PRAM e Hipercubo para implementar um algoritmo paralelo para calcular a TD, mas baseado nas métricas com peso descritas anteriormente no Capítulo 2. Os algoritmos são da ordem de $O(\log N)$ no tempo utilizando $N^2/\log N$ processadores no modelo EREW-PRAM e $O(N^2/p^2 + (N \log p)/p)$ no Hipercubo com p^2 processadores (para $1 \leq p \leq N$).

Kolountzakis e Kutulakos [37] também apresentam uma implementação em paralelo do cálculo da TD para o modelo EREW-PRAM com p processadores e que possui uma complexidade da ordem de $O(N^2 \log N)$. Finalmente, Pan e Li [38] apresentam um algoritmo paralelo na ordem de $O(1)$ para um arranjo de $p^2 \times p^2$ processadores.

Todos estes algoritmos em paralelo baseiam suas estimativas de complexidade sob o ponto de vista de tempo de execução em implementações no modelo de arquitetura PRAM e em arquiteturas caras tipo MIMD (*Multiple Instruction Multiple Data*) e Hipercubo. Estas implementações não levam em conta questões importantes como a sobrecarga de comunicação entre os diversos processadores e o tempo necessário para carregar a imagem a ser processada na memória das arquiteturas.

3.1.2 Implementações em *Hardware*

Takala e Viitanen [39] consideram que as arquiteturas SIMD (*Single Instruction Multiple Data*) são as mais adequadas para o processamento de imagem, principalmente aquelas que possuem granularidade fina, por serem mais baratas do que as arquiteturas MIMD e por possuírem um controle mais fácil. Como forma de diminuir ainda mais os custos de implementação, Takala e Viitanen propõem que o processamento seja feito bit a bit, o que aumentaria a eficiência em área dos processadores. Além do mais, as arquiteturas seriais (processamento bit a bit), torna mais fácil a integração do sensor de imagem (tipicamente um CCD – *Charge Coupled Device*) com a arquitetura em uma única pastilha.

Baglieto et al. [40] apresentam um algoritmo para calcular a TDE baseado na arquitetura PPA (*Polymorphic Processor Array*), que é um arranjo de $N \times N$ elementos de processamento (PE), onde cada PE possui uma chave de comunicação com os demais PEs e uma memória. A direção de movimento de dados entre os PEs é realizada ao mesmo tempo e em uma única direção. Ou seja, em um determinado tempo, todos os PEs enviam dados na mesma direção (Norte, Leste, Sul, Oeste). Cada PE pode definir se a sua conexão é Fechada (os dados passam direto), ou Aberta (o processador envia dados).

Figura 3.1 – Diagrama do Elemento de Processamento da arquitetura PPA

Pela sua estrutura, é possível associar cada PE a um pixel. O algoritmo para o cálculo da TDE é dividido em 2 passos, similarmente ao que é proposto por Chen e Chuang [33] e Hirata [32]. No primeiro passo cada pixel (PE) procura qual é o pixel do objeto mais próximo dentro da coluna a qual ele pertence e armazena este valor temporariamente. No segundo passo ele realiza uma série de operações de busca-e-comparação pela linhas e determina a distância do pixel mais próximo num processo de triangulação com outros dois pixels pertencentes à mesma linha. Esta operação é repetida até que o valor do pixel mais próximo não mude mais. De acordo com Baglieto, o algoritmo possui uma complexidade da ordem de $O(N)$.

3.2 Geração de Esqueletos – Algoritmos e Arquiteturas Paralelos

Assim como no caso da Transformada de Distância, muitas das implementações em paralelo são baseadas em arquiteturas complexas, como o Hipercubo e o modelo PRAM, onde a maior preocupação está em obter algoritmos que sejam da menor ordem de complexidade. Para exemplificar, Jenq e Sahni [19] apresentam um algoritmo baseado no conceito de eixo médio dado por Blum,

utilizando máximos quadrados. Este algoritmo possui uma complexidade da ordem de $O(n^2)$ na sua forma seqüencial, mas quando implementado em um Hipercubo, sua complexidade passa a ser de $O(\log^2 N)$ e, numa implementação em CREW-PRAM (*Concurrent Read Exclusive Write – Parallel Random Access Memory*), de $O(\log N)$, ambas as implementações utilizando $O(N^2)$ processadores.

Ferreira e Ubéda [41] apresentam dois algoritmos baseados no eixo médio, um utilizando máximos discos e outro utilizando máximos quadrados, ambos baseados nas métricas *Chessboard* e *City Block*. Tais algoritmos possuem tempo de processamento da ordem de $O(\log N)$, utilizando $N^2/\log N$ processadores, tanto numa implementação em Hipercubo, como em EREW PRAM. Eles também observam que um disco na métrica *Chessboard* é na verdade um quadrado com lados de tamanho ímpar. Assim poderia se pensar que o algoritmo para o cálculo dos máximos discos na métrica *Chessboard* também poderia ser utilizado para implementar o algoritmo para máximos quadrados, porém esta idéia não é válida pois o algoritmo implementado para o cálculo de máximos quadrados se baseia em quadrados de qualquer tamanho.

Na área de *hardware* para geração de esqueletos, Ranganathan e Doreswamy [42] desenvolveram uma arquitetura baseando-se no trabalho de Arcelli e di Baja [15], utilizando N processadores em *pipeline* para processar imagens de $N \times N$ pixels. Vale a pena lembrar que em [15], o algoritmo desenvolvido baseia-se na distância *City Block* de cada pixel para definir se este será, ou não, removido. Cada processador realiza as comparações de distância entre os pixels, de acordo com as regras de remoção estabelecidas. Como estas regras se baseiam na diferença de valor da distância de cada pixel e, na métrica *City Block* esta diferença é de ± 1 , o processo de comparação fica bastante simplificado e, conseqüentemente, o processador. A imagem a ser processada é enviada através da arquitetura, o que demora $6N + N^2$ ciclos ($6N$ para preencher a arquitetura e N^2 para processar toda a imagem). Porém a imagem resultante deve ser novamente enviada através da arquitetura para completar o processamento, o que acaba dobrando o tempo de execução. De acordo com os autores, a velocidade máxima de processamento é de 200MHz e que o processamento de uma imagem de 512×512 pixels demora 2,59ms e de uma

imagem de 256 x 256 demora 0,327ms. Porém, considerando que ela leva $2x(6N+N^2)$ ciclos para processar uma imagem, isto levaria a um tempo de 2,65ms para uma imagem de 512 x 512 pixels e 0,671ms para uma imagem de 256 x 256 pixels.

Considerando *Thinning*, Bourbakis et al. [43] apresentam uma arquitetura baseada em uma matriz de processadores, onde cada processador está interconectado com seus 8 vizinhos. De acordo com os autores, um grande problema que surge no processamento paralelo para a remoção de pixels está diretamente relacionada com a figura abaixo. Num típico algoritmo de remoção, o pixel P iria se considerar parte do esqueleto e removeria o seu vizinho Q. Ao mesmo tempo, Q também acharia que pertence ao esqueleto e removeria P, resultando na quebra do esqueleto. Duas seriam as forma de se resolver isso. A primeira seria dividir o processamento em dois passos paralelos consecutivos. Um processaria os pixels de cima para baixo (ou da direita para a esquerda) e o outro no sentido contrário. Assim, um dos pixels seria removido no primeiro passo e o outro seria preservado no segundo passo. Esta solução para o problema acima dobraria o tempo de execução. Outra solução seria aumentar o tamanho da máscara. Isto permitiria ao pixel P, ou Q, verificar se o outro pixel é uma borda, ou não. Esta solução teria como consequência um aumento significativo na complexidade do circuito. Os autores utilizam as duas soluções para implementar a arquitetura, desenvolvendo dois algoritmos paralelos. O primeiro, utilizando a primeira solução, mais sensível a variações do contorno dos objetos, e o segundo, utilizando uma máscara maior, mais robusto e totalmente paralelo.

Figura 3.2 – Exemplo de problema para remoção em paralelo

Várias arquiteturas também já foram implementadas utilizando Morfologia Matemática para a geração de esqueletos através de *thinning* [44-47]. Todas são

totalmente paralelas e se baseiam no uso de máscaras para realizar a convolução dos elementos estruturantes com a imagem. Os processadores são extremamente simples por trabalharem apenas com informações binárias. Atualmente estamos desenvolvendo uma arquitetura para operações morfológicas em tempo real em nosso grupo de pesquisa.

Além destas implementações “convencionais” para a geração de esqueletos, muitas outras tem sido desenvolvidas, como o uso de máquinas de estados finitos [47], autômatos [14], e até mesmo lógica neural [48]. Curiosamente, apesar de estas não serem implementações comuns para geração de esqueletos, o uso de autômatos e lógica neural são os que melhor se aproximam da definição inicial de esqueletos dada por Blum (“gramado em chamas”), pois as informações se propagam na forma de ondas entre os elementos e partem de pontos específicos (os pixels dos objetos). Em [14], as informações se propagam pela matriz de autômatos numa onda em *métrica City Block* e utiliza o índice de iteração para definir a distância. Quando um autômato recebe informações de duas ou mais frentes de onda, ele sabe que é parte do esqueleto. De acordo com as simulações realizadas, a arquitetura possui uma frequência máxima de operação de 60MHz e o tempo total de processamento vai depender do tamanho do objeto a ser processado

3.3 Conclusão

Conforme foi visto anteriormente, existem duas formas básicas para se obter esqueletos: *Thinning* (onde inclui-se a Morfologia Matemática) ou Centro de Discos Máximos (associado à Transformada de Distância). A geração de esqueletos por *Thinning* consiste em uma operação de erosões controladas sucessivas, onde é verificado se um pixel deve ou não ser removido, de acordo com alguma regra de exclusão, até que mais nenhum pixel possa ser removido. A desvantagem deste método com relação ao uso da Transformada de Distância é que não é possível reconstruir o objeto inicial a partir do esqueleto (se bem que em alguns casos utiliza-se o índice da iteração para se definir uma distância) e os esqueletos gerados nem sempre são de boa qualidade (aqui, qualidade ainda é um termo subjetivo). Por isso a

sua grande aplicação consiste em afinar algo que já estava relativamente estreito para um melhor processamento.

A grande vantagem do *Thinning* está no fato que a sua implementação é bem mais fácil do que a Transformada de Distância, uma vez que uma simples operação de convolução local, aplicada várias vezes sobre a imagem, é suficiente. Por isso é bem mais extensa a bibliografia de trabalhos publicados a respeito das implementações deste método. Com relação à implementação de arquiteturas para esqueletização, novamente, grande parte dos trabalhos publicados são baseados em *Thinning*, ou Morfologia Matemática.

Outra forma de se obter esqueletos, é por meio da Transformada de Distância, quer seja utilizada a métrica Euclidiana, ou não. Verificamos que são muito poucas as arquitetura que realizam a Transformada de Distância ou geram esqueletos baseados na métrica Euclidiana. Desta forma, este ainda é um campo aberto para o desenvolvimento e isto serve de motivação para o desenvolvimento de arquiteturas nesta área.

4. Arquiteturas Propostas

No Capítulo 2, pudemos observar que a Transformada de Distância (TD), mais especificamente a Transformada de Distância Euclidiana (TDE), possui uma grande importância no Processamento Digital de Imagens, mais especificamente na área de Análise de Formas. Ela é um instrumento importante nesta área e várias são as suas aplicações, conforme pode ser verificado em [6]. Vimos também que a Geração de Esqueletos (Esqueletização), que pode ser obtida através da TD, também é uma ferramenta importante para a análise e classificação de formas.

Ainda no Capítulo 2, apresentamos o algoritmo para Dilatação Exata e Propagação de Rótulos [4], que demonstrou ser bastante versátil. Apesar de utilizar a estrutura SEDR, evitando o uso de operações aritméticas complexas, como o cálculo de raiz quadrada, o método de Dilatação Exata e Propagação de Rótulos não possui um bom desempenho no que diz respeito à velocidade de execução.

Vimos, no Capítulo 3, que diversos algoritmos (seqüenciais e paralelos) foram e têm sido desenvolvidos para se obter tanto a TD como Esqueletos. Contudo, quase não existem arquiteturas implementadas para a obtenção da TD, e a maioria das que existem são baseadas em métricas diferentes da Euclidiana. Da mesma forma, as arquiteturas para Geração de Esqueletos são, na sua grande maioria, baseadas em *peeling* (*thinning*) ou, quando baseadas em discos máximos, utilizam métricas não Euclidianas, o que faz com que os Esqueletos gerados nem sempre possuam as propriedades esperadas de conectividade, largura unitária e centralidade.

Assim, cria-se o desafio de se desenvolver arquiteturas para a TDE e a Geração de Esqueletos que possam trabalhar em tempo real. Para tanto, este trabalho baseou-se no algoritmo de Dilatação Exata e Propagação de Rótulos [4], simples porém eficiente, que calcula tanto a TDE, como também permite a Geração de Esqueletos. Para o processamento em tempo real, duas são as características que as arquiteturas devem possuir: i) devem ser capazes de trabalhar com sinais de vídeo, ou seja, a imagem a ser processada está em formato *raster*, ii) devem ter a capacidade de processar uma grande quantidade de dados ao mesmo tempo. Desta forma, a melhor escolha é que elas sejam baseadas num arranjo de processadores, um para cada pixel (ii), similar à arquitetura CLIP [1], interconectados em *pipeline* (i), conforme as arquiteturas para Geração de Esqueletos baseadas em *thinning* [43-47].

A seguir, apresentaremos as propostas iniciais para a implementação das arquiteturas. Após, apresentaremos como o algoritmo utilizado pode ser modificado de tal forma que ele possa ser processado em paralelo e utilizado para implementar a arquitetura para o cálculo da TDE. Depois, utilizando como base a arquitetura para TDE, juntamente com o algoritmo modificado, implementaremos a arquitetura para a Geração de Esqueletos. Finalmente apresentaremos uma proposta de implementação do algoritmo utilizando a biblioteca MPI para processamento paralelo.

4.1. Propostas Iniciais

4.1.1 Máscaras

Apesar de ser um dos métodos mais antigos de processamento de imagens, o método das máscaras pode ser utilizado para implementar a TDE. Esta implementação pode ser abordada de duas formas: *Raster-scan* e *Contour-scan*.

a) Máscara *Raster-scan*

Esta abordagem se baseia na forma mais clássica de implementação para processamento de imagens. Uma máscara contendo as distâncias Euclidianas indicadas pela SEDR seria sobreposta à imagem e toda vez que o pixel central desta máscara encontrasse um pixel pertencente ao contorno, os pixels da imagem sobrepostos pela máscara seriam atualizados, contanto que os valores dos pixels da imagem fossem maiores do que os da máscara. A implementação clássica de máscara consiste de um conjunto de células de deslocamento/processamento horizontal, associadas a um conjunto de memórias FIFO que realizariam o deslocamento vertical da imagem. Desta forma a largura da imagem a ser processada seria igual à largura da máscara mais o tamanho (profundidade) das memórias FIFO, enquanto que a altura não teria limites.

Uma desvantagem desta implementação é que o tamanho da máscara seria relativamente grande, comparada com as implementações convencionais de máscara. Por exemplo, para se calcular uma distância Euclidiana de apenas 10 unidades seria necessária uma máscara de $21 \times 21 = 441$ elementos de deslocamento/processamento. Outra desvantagem desta implementação é que todos os pixels da imagem seriam analisados para se determinar qual pertence ao contorno e qual não pertence. A principal e mais interessante vantagem desta implementação é que a imagem não necessitaria ser armazenada em memória, podendo ser processada em tempo-real (na mesma taxa de aquisição da imagem). Estima-se que a complexidade de execução desta implementação seja da ordem de $O(N^2)$.

b) Máscara *Countor-scan*

A idéia seria fazer a máscara percorrer apenas os pixels pertencentes ao contorno, ao invés de percorrer todos os pixels da imagem. A implementação mais plausível para esta proposta seria criar um *mesh* de células de deslocamento (que também realizariam o processamento) com a possibilidade de deslocar a imagem nas quatro direções (Norte, Sul, Leste, Oeste), fazendo com que o ponto central “perseguisse” o contorno do objeto. Apesar de ser um processamento bem mais rápido que a abordagem anterior, o seu controle se torna bem mais complexo e

exigiria que o número de células de deslocamento/processamento fosse o mesmo que o número de pixels da imagem a ser processada.

4.1.2 *Mesh* de Processadores

A idéia seria associar a cada pixel da imagem uma célula de processamento, e esta célula teria a capacidade de se conectar às demais células de processamento de acordo com os deslocamentos relativos de cada distância Euclidiana armazenada na estrutura SEDR.

Esta abordagem se basearia em uma implementação em duas camadas: a primeira camada seria a camada de processamento, onde teríamos as células de processamento propriamente ditas e onde estaria armazenada a imagem (uma célula para cada pixel). A segunda camada seria a “camada de interconexão” entre as células de processamento. A interconexão entre as células seria estabelecida em função da estrutura SEDR modificada (formada somente pelos pares i,j), a partir de cada célula que possuísse um pixel do objeto, e seria dinamicamente reconfigurável.

Desta forma, a cada reconfiguração as distâncias entre os pixel do objeto e do fundo seriam estabelecidas e os respectivos valores seriam atualizados. Esta abordagem demonstra ser extremamente complexa e necessita que o número de células de processamento seja igual ao número de pixels da imagem.

Tendo em vista que, tanto para a implementação da TDE utilizando a Máscara *Countor-scan*, quanto utilizando o *Mesh* de Processadores, o tamanho da arquitetura teria que ser igual ao tamanho da imagem a ser processada, o que demandaria um grande consumo de células lógicas, deu-se preferência e prioridade para a implementação utilizando a Máscara *Raster-scan*, conforme será visto a seguir.

4.2. Arquitetura para TDE

4.2.1 Algoritmo para Dilatação Exata Modificado

Para compreendermos melhor como o algoritmo pode ser utilizado para implementar a arquitetura para TDE, iremos analisar, inicialmente, uma pequena modificação do algoritmo apresentado no Capítulo 2, destinado ao cálculo da TDE. Diferentemente do Algoritmo 2.1, o Algoritmo 4.1 abaixo atribui ao pixel vazio o valor d .

para cada distância d da SEDR **faça**
para cada pixel p do contorno **faça**
para cada posição relativa e_d **faça**
se o pixel $p + e_d$ está vazio
então coloque d neste pixel

Algoritmo 4.1 – Algoritmo de Dilatação Exata para TD

Verificamos que, para cada distância d existente na estrutura SEDR, avaliamos os pixels ao redor dos pixels de contorno e atribuímos aos pixels vazios o valor d (pixels vazios são pixels que ainda não receberam nenhum valor). Verificamos também que os valores de distância serão avaliados em ordem crescente. Isto garante que a distância armazenada em cada pixel seja a menor distância do contorno.

Podemos incluir no Algoritmo 4.1 uma pequena comparação para garantir que o valor de distância armazenado no pixel seja o menor possível, conforme pode ser visto no Algoritmo 4.2 (considere que os pixels que não pertence ao contorno possuem o máximo valor inteiro e o pixels que pertencem ao contorno possuem valor 0). Como o Algoritmo 4.1 garante que a distância atribuída aos pixels será realmente a menor distância do contorno, esta comparação se faz, aparentemente, desnecessária.

para cada distância d da SEDR **faça**
para cada pixel p do contorno **faça**
para cada posição relativa e_d **faça**
se a distância do pixel $p + e_d$ é maior que d
então coloque d neste pixel

Algoritmo 4.2 – Algoritmo de Dilatação Exata para TD modificado

Porém, com esta modificação, podemos alterar a ordem de execução dos laços, conforme pode ser verificado no Algoritmo 4.3. Isto quer dizer que, para cada pixel do contorno, avaliamos todas as distâncias ao redor deste. Desta forma, os pixels da imagem podem ser avaliados pelos pixels de contorno concorrentemente, o que viabiliza a sua execução em paralelo.

para cada pixel p do contorno **faça**
para cada distância d da SEDR **faça**
para cada posição relativa e_d **faça**
se a distância do pixel $p + e_d$ é maior que d
então coloque d neste pixel

Algoritmo 4.3 – Alteração da ordem de execução dos laços do Algoritmo modificado

Uma última alteração que podemos realizar é converter a estrutura SEDR, originalmente armazenada numa lista, em uma matriz, onde cada elemento desta matriz armazena a sua distância Euclidiana (ou o índice, de acordo com a Tabela 2.1) do elemento central desta. Para tanto, a matriz deve ser de $(2R+1) \times (2R+1)$, sendo R a máxima distância, em pixels, da SEDR, cujo valor seja inteiro. A escolha de uma matriz quadrada é necessária para se garantir a simetria com relação à rotação, além de somente os elementos que possuírem distância menor ou igual a R devem receber os valores de d , ficando os demais com o valor máximo (máximo inteiro). A Figura 4.1 abaixo mostra esta matriz para R igual a 4.

				4													
			$\sqrt{13}$	$\sqrt{10}$	3	$\sqrt{10}$	$\sqrt{13}$										
		$\sqrt{13}$	$\sqrt{8}$	$\sqrt{5}$	2	$\sqrt{5}$	$\sqrt{8}$	$\sqrt{13}$									
		$\sqrt{10}$	$\sqrt{5}$	$\sqrt{2}$	1	$\sqrt{2}$	$\sqrt{5}$	$\sqrt{10}$									
4	3	2	1	0	1	2	3	4									
		$\sqrt{10}$	$\sqrt{5}$	$\sqrt{2}$	1	$\sqrt{2}$	$\sqrt{5}$	$\sqrt{10}$									
		$\sqrt{13}$	$\sqrt{8}$	$\sqrt{5}$	2	$\sqrt{5}$	$\sqrt{8}$	$\sqrt{13}$									
			$\sqrt{13}$	$\sqrt{10}$	3	$\sqrt{10}$	$\sqrt{13}$										
				4													

				9													
			8	7	6	7	8										
		8	5	4	3	4	5	8									
		7	4	2	1	2	4	7									
9	6	3	1	0	1	3	6	9									
		7	4	2	1	2	4	7									
		8	5	4	3	4	5	8									
			8	7	6	7	8										
				9													

Figura 4.1 – Matriz de distâncias Euclidianas (esquerda) e de respectivos índices (direita)

A matriz da esquerda armazena em seus elementos a distância Euclidiana, enquanto que a matriz da direita faz uso do índice das distâncias Euclidianas, de acordo com a Tabela 2.1. Os elementos em branco possuem o valor máximo, que pode ser $R+1$, para a matriz de distância Euclidianas, como MAXINT, para a matriz de índices.

Assim, percorrendo a imagem com esta matriz, toda vez que o elemento central se sobrepõe a um pixel de contorno, todos os pixels que estiverem sob a matriz terão seus valores atualizados pelo menor valor entre o valor que possui e o valor armazenado no elemento da matriz imediatamente acima dele.

4.2.2 Implementação em *Hardware* do Algoritmo Modificado

Como vimos, esta abordagem se baseia na forma mais clássica de processamento de imagens: o uso máscaras. Só que ao invés de realizarmos uma convolução da máscara com a imagem, iremos utilizar um princípio de “comparar e substituir”. A principal desvantagem desta máscara é o seu tamanho. Como vimos, para se calcular a distância Euclidiana até 4, é necessária uma máscara de 9×9 elementos. A principal vantagem desta implementação é que a imagem não necessita estar armazenada em memória, sendo processada em tempo-real.

A célula de processamento, baseada no Algoritmo 4.3, é apresentada na Figura 4.2. Como podemos observar, é uma célula bastante simples, composta apenas de um comparador e um multiplexador. A célula de processamento recebe dados dois dados: um proveniente da imagem (D_{in}), e outro de uma pequena célula de memória (registrador) que armazena a sua posição (distância ou índice) relativa à célula central (D_{local}).

Figura 4.2 – Célula de Processamento Baseada no Algoritmo de Dilatação Exata Modificado

A célula de processamento também recebe o sinal **modify** da célula central, indicando que a célula deve, ou não, realizar a comparação entre os dados de entrada e fornecer, na saída D_{out} , o menor valor. Se o sinal **modify** estiver ativado, na saída D_{out} teremos o menor valor entre D_{in} e D_{local} . Caso contrário, teremos o valor D_{in} . O sinal **modify** é ativado toda vez que a célula central estiver sobre um pixel do contorno. A célula central apenas verifica se o pixel é um pixel de contorno e envia o sinal **modify** para a célula de processamento.

A arquitetura final para TD será, então, um arranjo de processadores, conforme mostra a Figura 4.3. As células de deslocamento servem para processarmos imagens maiores do que o tamanho da máscara. Sem as células de deslocamento, o tamanho da imagem a ser processada seria limitada pela largura da máscara ($2R + 1$). Utilizando as células de deslocamento, podemos processar imagens mais largas. Com relação à altura da imagem, não existe limite.

4.3. Arquitetura para Geração de Esqueletos

4.3.1 Algoritmo para Propagação de Rótulos Modificado

Analisando o algoritmo completo para Dilatação Exata e Propagação de Rótulos (Algoritmo 4.4), onde atribuímos ao pixel o valor de distância d e do rótulo p do pixel do contorno mais próximo deste pixel, é possível observar que cada pixel será testado inicialmente pela sua distância e depois pelo seu rótulo. Em outras palavras, o pixel terá sempre a menor distância e o menor rótulo.

para cada distância d da SEDR **faça**
para cada pixel p do contorno **faça**
para cada posição relativa e_d **faça**
se o pixel $p + e_d$ está vazio
então coloque d e p neste pixel

Algoritmo 4.4 – Algoritmo de Dilatação Exata e Propagação de Rótulos

Podemos verificar esta propriedade pela seguinte figura (Figura 4.4). Nesta figura, onde os pixels do contorno estão em cinza escuro e possuem o seu valor de rótulo e distância (0), está sendo avaliada a distância 1 ao redor dos pixels de contorno. Na primeira imagem, são avaliados os pixels que distam 1 do pixel com rótulo p , sendo a eles atribuídos os valores p e 1. Na imagem seguinte, são avaliados os pixels ao redor do pixel com rótulo $p+1$. Podemos verificar que dois destes pixels já haviam sido avaliados no passo anterior, e receberam o valor p de rótulo e 1 de distância. O mesmo ocorre com o pixel com rótulo $p+2$. Podemos observar que o pixel abaixo do pixel Q possui o rótulo $p+1$ e a distância 1. Isto significa que, apesar dele estar equidistante dos pixels com rótulo $p+1$, $p+2$ e $p+3$, foi atribuído a ele o valor $p+1$. Podemos concluir que o pixel Q receberá o valor de rótulo $p+1$ (está equidistante de $p+1$ e $p+3$) e de distância $\sqrt{2}$ (ou índice 2).

Figura 4.4 – Exemplo de Propagação de Rótulos

Esta propriedade nos permite reestruturar o Algoritmo 4.4, com a inclusão de duas novas comparações:

para cada pixel p do contorno **faça**
para cada distância d da SEDR **faça**
para cada posição relativa e_d **faça**
 se (a distância associada a $p + e_d$ é maior do que d)
 ou (a distância associada a $p + e_d$ é igual a d
 e o rótulo associado a $p + e_d$ é maior do que p)
 então associe d e p ao pixel $p + e_d$

Algoritmo 4.5 – Algoritmo de Dilatação Exata e Propagação de Rótulos Modificado

Novamente, da mesma forma que o algoritmo modificado para Dilatação Exata (Algoritmo 4.3), podemos quebrar a ordem de execução dos laços, possibilitando sua implementação em paralelo, conforme veremos a seguir.

4.3.2 Implementação em *Hardware* do Algoritmo Modificado

Da mesma forma como foi feito com o algoritmo para Dilatação Exata Modificado para a criação da arquitetura para TD, também podemos criar uma arquitetura que realize, além da Dilatação Exata, a Propagação de Rótulos. A figura abaixo (Figura 4.5) descreve a célula básica de processamento para esta arquitetura, baseada no Algoritmo 4.5.

Figura 4.5 – Célula de Processamento para Dilatação Exata e Propagação de Rótulos

Além de comparar as distâncias (D_{in} e D_{local}), esta célula de processamento realiza a comparação entre o rótulo do pixel da imagem (L_{in}) com o rótulo do pixel que está sob a célula de processamento central ($L_{central}$). Toda vez que o sinal **modify** for ativado, a célula irá selecionar a menor distância (e seu respectivo rótulo) e, caso elas sejam iguais, irá selecionar aquela que estiver associada ao menor rótulo. Vale lembrar que a comparação de rótulos é entre aquele já associado os pixels da imagem e aquele pertencente ao pixel do contorno, que irá ativar a comparação.

A arquitetura final é semelhante à arquitetura apresentada na Figura 4.3, onde a célula de processamento é substituída pela célula de processamento da Figura 4.5. Ao passar a imagem, contendo os pixels de contorno já rotulados, pela arquitetura, na saída teremos tanto o mapa de distância, quanto o mapa de rótulos.

4.4. Algoritmo utilizando MPI

Uma possível forma de paralelização seria separar o contorno de cada objeto, já devidamente enumerado (rotulado), e dividi-lo em diversos sub-contornos, que seriam processados em separado e de forma independente, cada um gerando um mapa de distâncias. Ao final deste processo, os mapas de distância seriam intercalados, sendo que cada pixel do mapa final receberia o pixel correspondente de cada um dos mapas dos sub-contornos que possuísse o menor valor de distância. Caso dois ou mais pixels possuíssem o mesmo valor de distância, seria utilizado o

que estivesse associado ao pixel do contorno que possuísse o menor rótulo, tudo isso de acordo com o Algoritmo 4.5.

Apesar de que, em tese, o tempo de processamento diminuiria de forma proporcional ao número de sub-contornos, a principal desvantagem deste método é que um único pixel seria avaliado diversas vezes. Mas isto possibilita visualizarmos que esta divisão poderia ser reduzida até um subconjunto de um pixel, ou seja, um processador para cada pixel.

O interesse em utilizar a biblioteca MPI está no fato de que ela prevê uma operação que realiza exatamente o que é proposto pelo Algoritmo 4.5, ou seja, a redução de todos os valores para o menor valor (distância), juntamente com um índice associado a este (rótulo). A operação de redução é a MPI-REDUCE utilizando o operador MPI_MINLOC.

O operando MPI-REDUCE combina os elementos na entrada `sendbuf` de cada processo, e retorna o valor combinado ao processo `root` em `recvbuf`. Existem várias operações `op` pré definidas incluindo `MPI_MAX`, `MPI_MIN`, `MPI_SUM`, e `MI_PROD`.

```
MPI_REDUCE( sendbuf, recvbuf, count, datatype, op, root, comm)
[ IN sendbuf] address of send buffer (choice)
[ OUT recvbuf] address of receive buffer (choice)
[ IN count] number of elements in send buffer (integer)
[ IN datatype] data type of elements of send buffer (handle)
[ IN op] reduce operation (handle)
[ IN root] rank of root process (integer)
[ IN comm] communicator (handle)
```

A operação que define MINLOC é descrita por:

$$\begin{pmatrix} u \\ i \end{pmatrix} \circ \begin{pmatrix} v \\ j \end{pmatrix} = \begin{pmatrix} w \\ k \end{pmatrix}, \text{ onde } w = \min(u, v) \text{ e } k = \begin{cases} i & u < v \\ \min(i, j) & \text{se } u = v \\ j & u > v \end{cases}$$

ou seja, dado dois vetores, o resultado da primeira linha será o menor valor entre os elementos da primeira linha dos dois vetores e a segunda linha irá receber o índice do menor valor da primeira linha e, se forem iguais, o menor índice.

4.5. Conclusão

Apresentamos as modificações que podem ser realizadas no Algoritmo de Dilatação Exata e Propagação de Rótulos e as propostas de implementação de arquiteturas para a Transformada de Distância e a Geração de Esqueletos baseadas no algoritmo modificado. Vimos que é possível a sua implementação por meio de células de processamento básicas, que podem ser arranjadas de forma matricial. A simplicidade das células de processamento permite que elas sejam facilmente implementadas em *pipeline*.

Verificamos que o algoritmo modificado descreve o princípio de operação de um operador da biblioteca MPI. Isto torna atrativo o seu uso para implementar o algoritmo de Dilatação Exata e Propagação de Rótulos.

No próximo capítulo iremos apresentar a implementação das arquiteturas e os resultados obtidos com elas. Apresentaremos também o algoritmo implementado utilizando a biblioteca MPI, comprovando apenas o seu funcionamento.

5. Implementação das Arquiteturas

5.1. Dilatação Exata

Para a implementação da arquitetura, a primeira coisa a ser observada é qual deve ser a largura do barramento de dados. A largura do barramento de dados irá definir qual será a maior distância que poderá ser calculada, e, desta forma, também definirá o tamanho da máscara, ou seja, o tamanho da arquitetura. No caso da arquitetura para Dilatação Exata, o barramento de dados deve conter pelo menos um bit para representar a imagem binária que será processada, e que também será utilizado pela célula central para identificar o pixel pertencente ao objeto. Devemos então escolher quantos bits serão necessários para representar a distância.

Abaixo temos a Tabela 5.1 que descreve qual é a máxima distância (R) que pode ser representada, tanto utilizando R^2 , quanto a estrutura SEDR. Podemos observar que se utilizarmos 7 bits para representar a distância, o valor máximo de R^2 será de 125, o que resulta numa distância máxima de $\sqrt{125} \cong 11,180$. Enquanto que, ao utilizarmos a estrutura SEDR, existirão 128 Possíveis Distâncias Distintas (N_{dist}) e a maior distância possível será de aproximadamente 18,601.

Tabela 5.1 – Distância Máxima por número de Bits para cada representação

Bits	R^2	R	N_{dist}	SEDR
1	1	1,000	2	1,000
2	2	1,414	4	2,000
3	5	2,236	8	3,162
4	13	3,606	16	5,385

5	29	5,385	32	8,246
6	61	7,810	64	12,369
7	125	11,180	128	18,601
8	250	15,811	256	27,459

Podemos ver que, conforme havíamos comentado no Capítulo 2, a melhor escolha para representar as distâncias é pela estrutura SEDR, já que, para um mesmo número de bits, podemos representar um maior número de distâncias e, conseqüentemente, um maior valor de distância. O gráfico apresentado na Figura 5.1 mostra a máxima distância representada por número de bits para cada um das formas de representação.

Figura 5.1 – Gráfico da Representação da Distância

Como é conveniente que o barramento de dados tenha largura igual a um valor, em bits, que seja múltiplo de 8, decidimos por representar as distâncias utilizando 7 bits. O bit restante, conforme foi mencionado, será utilizado para propagar a imagem binária pela arquitetura.

Assim, poderá ser implementado um arranjo com $(2R+1) \times (2R+1)$ elementos de processamento, ou seja $(2 \times 18 + 1) \times (2 \times 18 + 1) = 37 \times 37 = 1369$ elementos. Porém,

nem todas as células serão de processamento. Temos a célula central, que controla toda a arquitetura, e as células de deslocamento, responsáveis pela largura da imagem que pode ser processada. Vale a pena ressaltar que algumas das células de processamento irão apenas deslocar os dados, sem processá-los, conforme pode ser visto na Figura 4.1 (bolas verdes sem valor). Desta forma, é conveniente substituí-las por células de deslocamento, que são mais simples e consumirão menos circuitos lógicos na sua implementação, conforme veremos a seguir.

A Tabela 5.3 abaixo mostra a relação de células de deslocamento (excluindo as responsáveis pela largura da imagem) e de células de processamento em função da máxima distância R . Esta tabela foi montada considerando apenas os elementos que tenham distância menor ou igual a R , descartando, assim, alguns valores que ainda poderiam ser utilizados. Por exemplo, no caso da escolha de se utilizar 7 bits para representar a distância, têm-se 128 Possíveis Distintas Distâncias. Porém, sete destas (18,028; 18,111; 18,248; 18,358; 18,385; 18,439 e 18,601) não são consideradas.

Tabela 5.2 – Relação de Número de Células de Processamento (N_{cproc}) e Número de Células de Deslocamento (N_{cdesl}) em função do valor de R

R	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
N_{cproc}	4	12	28	48	80	112	148	196	252	316	376	440	528	612	708	796	900	1008
N_{cdesl}	4	12	20	32	40	56	76	92	108	124	152	184	200	228	252	292	324	360

A seguir apresentamos o resultado da implementação da arquitetura utilizando a ferramenta Quartus II (versões 2.0 Full Edition SP2 e 2.1 Web Edition SP1), da Altera. Optamos por utilizar os dispositivos da família Stratix, também da Altera, por serem estes os mais densos.

A compilação das arquiteturas foi feita utilizando a configuração padrão do Quartus II, sendo somente verificado se a otimização estava configurada para área, e não velocidade. A escolha dos dispositivos Stratix foi configurada para AUTO para dar liberdade ao aplicativo na escolha do menor dispositivo em que coubesse a arquitetura.

Os dados de simulação foram entrados no formato TBL, próprio do Quartus II, e convertido para VWL (que é um formato mais compacto) utilizando o próprio Quartus II. Para a criação dos arquivos de simulação foi utilizado o Delphi para converter imagens no formato BMP para o formato TBL, e vice-versa para visualização dos resultados. Abaixo temos um cabeçalho de um arquivo TBL.

```
%
Copyright (C) 1991-2000 Altera Corporation
Any megafunction design, and related net list (encrypted or decrypted),
support information, device programming or simulation file, and any other
associated documentation or information provided by Altera or a partner
under Altera's Megafunction Partnership Program may be used only to
program PLD devices (but not masked PLD devices) from Altera. Any other
use of such megafunction design, net list, support information, device
programming or simulation file, or any other related documentation or
information is prohibited for any other purpose, including, but not
limited to modification, reverse engineering, de-compiling, or use with
any other silicon devices, unless such use is explicitly licensed under
a separate agreement with Altera or a megafunction partner. Title to
the intellectual property, including patents, copyrights, trademarks,
trade secrets, or maskworks, embodied in any such megafunction design,
net list, support information, device programming or simulation file, or
any other related documentation or information provided by Altera or a
megafunction partner, remains with Altera, the megafunction partner, or
their respective licensors. No other licenses, including any licenses
needed under any third party's intellectual property, are provided herein.
%

GROUP CREATE |Distance = |Distance[5] |Distance[4] |Distance[3] |Distance[2]
|Distance[1] |Distance[0] ;
GROUP CREATE |Label = |Label[8] |Label[7] |Label[6] |Label[5] |Label[4] |Label[3]
|Label[2] |Label[1] |Label[0] ;
GROUP CREATE |Dist_out = |Dist_out[5] |Dist_out[4] |Dist_out[3] |Dist_out[2]
|Dist_out[1] |Dist_out[0] ;
GROUP CREATE |Labe_out = |Labe_out[8] |Labe_out[7] |Labe_out[6] |Labe_out[5]
|Labe_out[4] |Labe_out[3] |Labe_out[2] |Labe_out[1] |Labe_out[0] ;
INPUTS |clock |clear |Image |Distance |Label;
OUTPUTS |Imag_out |Dist_out |Labe_out;
UNIT ns;
RADIX HEX;
PATTERN
    0.0> 0 1 0 3F 1FF = 0 00 000
    50.0> 1 1 0 3F 1FF = 0 00 000
    100.0> 0 0 0 3F 1FF = 0 00 000
    150.0> 1 0 0 3F 1FF = 0 00 000
    ...
    2356100.0> 0 0 0 3F 1FF = 0 3F 1FF
    2356150.0> 1 0 0 3F 1FF = 0 3F 1FF
    2356200.0> 0 0 0 3F 1FF = 0 3F 1FF
    2356250.0> 1 0 0 3F 1FF = 0 3F 1FF
    2356300.0> X X X X X = X X X
;
```

5.1.1 Célula de Processamento

A célula de processamento, cujo esquema está representado na Figura 4.2, foi implementada em VHDL. A escolha do uso de uma linguagem de programação em *hardware*, ao invés de esquemáticos, deve-se ao fato que isto facilita a sua

portabilidade e aprimoramento. Ela realiza a operação já descrita anteriormente, ou seja, compara dois valores (um local – `Dist_loc` – e outro proveniente de outra célula de processamento – `Dist_in`) e apresenta o menor valor (`Dist_out`) quando `modify` estiver ativo. Caso `modify` não esteja ativo, ela apresenta na saída o valor da entrada `Dist_in`. A célula também faz o deslocamento da imagem binária, por meio de `Image_in` e `Image_out`. Toda a operação da célula de processamento é coordenada pela transição positiva do sinal de `clock`. A célula ainda possui a entrada `reset`, cuja função limpar o conteúdo da célula.

Abaixo temos a listagem do código VHDL e o respectivo componente gerado (Figura 5.2).

```

library IEEE;
use IEEE.std_logic_1164.all;

entity proc_01 is
    generic(DistW : integer := 7);
    port (
        Dist_in: in STD_LOGIC_VECTOR (DistW-1 downto 0);
        Dist_loc: in STD_LOGIC_VECTOR (DistW-1 downto 0);
        Image_in: in STD_LOGIC;
        modify: in STD_LOGIC;
        clock: in STD_LOGIC;
        reset: in STD_LOGIC;
        Dist_out: out STD_LOGIC_VECTOR (DistW-1 downto 0);
        Image_out: out STD_LOGIC );
end proc_01;

architecture proc_01_arch of proc_01 is
begin
    process (clock, reset)
    begin
        if clock'event and clock='1' then
            if reset='1' then
                Dist_out <= (others => '1');
                Image_out <= '0';
            else
                if (Dist_in > Dist_loc and modify = '1') then
                    Dist_out <= Dist_loc;
                else
                    Dist_out <= Dist_in;
                end if;
                Image_out <= Image_in;
            end if;
        end if;
    end process;
end proc_01_arch;

```

Listagem do código VHDL da Célula de Processamento

Figura 5.2 – Célula de Processamento PE_RASTER

5.1.2 Célula Central

A célula central também foi implementada em VHDL. Sua função é identificar o pixel que pertence ao objeto e enviar o sinal `modify` para as células de processamento. Durante a transição positiva do `clock`, a célula central avalia o sinal de `reset` e, se não estiver ativado, transfere os dados de entrada `Image_in` e `Dist_i` para as respectivas saídas `Image_out` e `Dist_o`. Caso `reset` esteja ativado, é enviado zero para a saída `Image_out` e o valor máximo para `Dist_o`. Durante a transição negativa do `clock`, a célula central avalia o estado da entrada `Image_in` e, de acordo com o estado desta, ativa a saída `modify`. Se a entrada `Image_in` for igual a 1, isto significa que a célula central está sobre um pixel do objeto e, portanto, deve ativar as células de processamento. O fato de a célula central trabalhar tanto com a transição positiva, quanto com a transição negativa do `clock`, serve para estabilizar os dados nas entradas das células de processamento, antes de ativá-las.

Abaixo temos a listagem do código VHDL e o respectivo componente gerado (Figura 5.3).

```

library IEEE;
use IEEE.std_logic_1164.all;

entity proc0_01 is
    generic(DistW : integer := 7);
    port (
        Image_in: in STD_LOGIC;
        Dist_i: in STD_LOGIC_VECTOR (DistW-1 downto 0);
        reset: in STD_LOGIC;
        clock: in STD_LOGIC;
        Image_out: out STD_LOGIC;
        Dist_o: out STD_LOGIC_VECTOR (DistW-1 downto 0);
        modify: out STD_LOGIC );
end proc0_01;

architecture proc0_01_arch of proc0_01 is
begin
    process (clock, reset)
    begin
        if (clock'EVENT AND clock = '0') then
            if Image_in = '1' then
                modify <= '1';
            else
                modify <= '0';
            end if;
        end if;

        if (clock'EVENT AND clock = '1') then
            if reset='1' then
                Dist_o <= (others => '1');
                Image_out <= '0';
                modify <= '0';
            else
                Image_out <= Image_in;
                Dist_o <= Dist_i;
            end if;
        end if;
    end process;
end proc0_01_arch;

```

Listagem do código VHDL da Célula Central

Figura 5.3 – Célula Central PE0_RASTER

5.1.3 Célula de Deslocamento

A célula de deslocamento é composta apenas de um registrador de 7 bits, para deslocar a distância, e um FF tipo D, para deslocar a imagem binária. Uma possível modificação que poderia ser realizada, sem que tivesse algum efeito funcional, é a integração do FF ao registrador, formando ambos um só registrador de 8 bits.

Figura 5.4 – Célula de Deslocamento

5.1.4 Célula de Deslocamento FIFO

Ao invés de utilizarmos as células de deslocamento descritas acima para definir o tamanho da imagem a ser processada, o que resultaria num consumo excessivo de LE, optou-se por substituí-las por memórias tipo FIFO, aproveitando a disponibilidade de memória que os dispositivos Stratix possuem. Desta forma, a célula de deslocamento FIFO é composta de duas memórias FIFO, uma de 1 bit para deslocar a imagem binária e outra de 7 bits para deslocar a distância, configurada para trabalharem sempre cheias. Isto é feito ligando a saída *full* (que indica quando a memória está cheia) na entrada *rdreq*, que habilita a leitura dos dados. Com isso, ela só irá permitir a leitura dos dados quando estiver cheia. Da mesma forma que no caso

da célula de deslocamento acima, seria interessante unir ambas as memórias FIFO em uma só.

A memória FIFO foi configurada com um tamanho de 128 palavras que, juntamente com os da máscara de processadores, definem a largura máxima da imagem que pode ser processada.

Figura 5.5 – Célula de Deslocamento FIFO

5.1.5 Arquitetura Final

Abaixo, na Figura 5.6, temos o diagrama final da arquitetura implementada no Quartus II para a execução do Algoritmo de Dilatação Exata. Apesar de terem sido utilizados 7 bits para representar a distância, a arquitetura foi implementada apenas para a distância máxima de 15,811. Isto resulta em um valor de R igual a 15. Assim, a arquitetura foi implementada com 792 células de processamento, 168 células de deslocamento e 1 célula central, resultando em 961 células, além de 31 células de deslocamento FIFO.

Figura 5.6 – Esquema geral da arquitetura para Dilatação Exata

Abaixo temos o sumário do processamento. Esta arquitetura pode processar imagens a uma velocidade de ~80MHz.

SUMMARY

Processing status	Fitting Successful - Sun Jan 26 15:18:40 2003
Timing requirements/analysis status	No requirements
Chip name	raster15
Device for compilation	EP1C20F324C6
Total logic elements	10,990 / 20,060 (54 %)
Total pins	18 / 233 (7 %)

Total memory bits	31,744 / 294,912 (10 %)
PLLs	0 / 2 (0 %)
Device for timing analysis	EP1C20F324C6

RESOURCE USAGE SUMMARY

Logic cells	10,990 / 20,060 (54 %)
Flipflops	9,116 / 20,759 (43 %)
I/O pins	18 / 233 (7 %)
Clock pins	2
Dedicated input pins	0
Global signals	2
M4Ks	62 / 64 (96 %)
Total memory bits	31,744 / 294,912 (10 %)
Total RAM block bits	285,696 / 294,912 (96 %)
PLLs	0 / 2 (0 %)
Global clocks	2 / 8 (25 %)
Maximum fan-out node	clock
Maximum fan-out	9178
Total fan-out	44441
Average fan-out	4.01

5.2 Propagação de Rótulos

Novamente, assim como no caso da implementação da arquitetura para Dilatação Exata, a primeira coisa a ser observada é a largura do barramento de dados. Neste caso, o barramento de dados é dividido em três partes: um bit destina-se ao envio da imagem binária através da arquitetura, 9 bits destinam-se ao envio do valor do rótulo de cada pixel pertencente ao contorno do objeto e 6 bits destinam-se ao envio do valor da distância calculada para cada pixel da imagem, totalizando um total de 16 bits (como desejado, um número múltiplo de 8).

A escolha destes valores foi feita de tal forma a estabelecer um equilíbrio entre a máxima distância a ser calculada e o número máximo de pontos de contorno que o objeto deve ter. Com 9 bits é possível rotular 512 pixels de contorno, o que é

um número razoável, porém pequeno. Assim, restam 6 bits para representar a distância. Com 6 bits é possível representar 64 Possíveis Distâncias Distintas utilizando a SEDR, o que resulta numa distância máxima de 12,396. Este valor também é pequeno, porém representa o melhor equilíbrio entre estes dois valores. Se aumentarmos o número de bits para representar a distância para 7 bits, iremos aumentar a distância máxima para 18,601, ou seja, um pouco mais de 5 pixels, porém perderemos metade dos pontos de contorno, ou seja, só poderemos processar 256 rótulos.

Assim, estes valores (512 pontos de contorno e distância 12,396) representam o melhor equilíbrio num barramento de 16 bits. É claro que podemos aumentar o número de bits do barramento para 17, 18, ou até mesmo 32 bits, porém isto pode inviabilizar a implementação da arquitetura, conforme veremos a seguir.

Uma vez definido o tamanho do barramento de dados, iremos descrever os elementos da arquitetura para Propagação de Rótulos. Diferentemente da arquitetura para Dilatação Exata, esta foi implementada utilizando-se somente diagramas esquemáticos, pois foi a primeira arquitetura implementada. Mesmo que a outra arquitetura tenha sido implementada inicialmente utilizando-se diagramas esquemáticos e depois tenha sido convertido para VHDL, o mesmo não ocorreu com esta devido a um fato que será explicado mais adiante.

5.2.1 Célula de Processamento

A célula de processamento (Figura 5.7), além dos elementos presentes na Figura 4.5, possui um conjunto de registradores e flip-flops para permitir o seu funcionamento em um *pipeline* de 2 estágios. No primeiro estágio, os dados vindos da célula à esquerda (imagem binária, distância e rótulo), são armazenados pelo primeiro flip-flop (DFF) e pelos dois registradores (LPM_DFF). A saída dos registradores vai para os comparadores (LPM_COMP) e, de acordo com o algoritmo para propagação de rótulos, a saída destes irá controlar os multiplexadores (LPM_MUX), juntamente com o sinal `modify`. No segundo estágio, o resultado é

enviado para a saída dos multiplexadores (que podem ser configurados para trabalharem como um circuito *pipeline* de 1 ou mais estágios) e do flip-flop.

Para que todo este processamento seja realizado em apenas um ciclo de clock, optou-se por trabalhar tanto com a borda de subida (primeiro estágio), quanto com a borda de descida (segundo estágio) do sinal de clock. Curiosamente, só foi possível realizar esta operação em VHDL com versões mais novas que a versão 2.0 do Quartus II. A versão 2.0, mesmo com as correções SP1 e SP2, indicava que o sinal de clock era muito complexo para ser implementado e não realizava a síntese do projeto. Assim, a versão em VHDL ficou para segundo plano e só foi utilizada na arquitetura para Dilatação Exata, que foi implementada posteriormente.

Figura 5.7 – Célula de Processamento para Propagação de Rótulos

5.2.3 Célula de Deslocamento

A célula de deslocamento possui a metade da lógica da célula central e trabalha apenas com a borda de descida do sinal de clock, para trabalhar de forma síncrona com as demais células.

Figura 5.9 – Célula de Deslocamento para Propagação de Rótulos

5.2.4 Célula de Deslocamento FIFO

A célula de deslocamento FIFO possui apenas uma memória FIFO com profundidade de 128 palavras de dados e barramento de dados de 16 bits, integrando num só barramento os barramentos de imagem binária, distância e rótulo. O seu funcionamento é idêntico ao da célula de deslocamento FIFO utilizada na arquitetura para Dilatação Exata.

Figura 5.10 – Célula de Deslocamento FIFO para Propagação de Rótulos

5.2.5 Arquitetura Final

A arquitetura final é apresentada na Figura 5.11. Ela processa distâncias de até 12 pixels e realiza a propagação dos rótulos. Sua velocidade máxima de operação é de 47.78MHz, de acordo com a compilação realizada na ferramenta Quartus II. Ela ocupou 63% do dispositivo EP1S30B956C6, num total de 20.746 Elementos Lógicos (18.396 flip-flops). Abaixo temos um sumário da compilação.

SUMARRY:

Processing status	Fitting Successful - Mon Jan 27 13:03:24 2003
Timing requirements/analysis status	Requirements met
Chip name	raster12
Device name	EP1S30B956C6
Total logic elements	20,746 / 32,470 (63 %)
Total pins	52 / 691 (7 %)
Total memory bits	54,400 / 3,317,184 (1 %)
DSP block 9-bit elements	0 / 96 (0 %)

RESOURCE USAGE SUMMARY:

Logic cells	20,746 / 32,470 (63 %)
Flipflops	18,396 / 34,543 (53 %)
I/O pins	52 / 691 (7 %)
Clock pins	2
Dedicated input pins	0
Global signals	16
M512s	0 / 295 (0 %)
M4Ks	27 / 171 (15 %)
MegaRAMs	0 / 4 (0 %)
Total memory bits	54,400 / 3,317,184 (1 %)
DSP block 9-bit elements	0 / 96 (0 %)
PLLs	0 / 10 (0 %)
Global clocks	16 / 16 (100 %)
Regional clocks	0 / 16 (0 %)
Fast regional clocks	0 / 16 (0 %)
DIFFIOCLKs	0 / 32 (0 %)
SERDES transmitters	0 / 82 (0 %)
SERDES receivers	0 / 82 (0 %)
Maximum fan-out node	clock
Maximum fan-out	18423
Total fan-out	84334
Average fan-out	4,

Figura 5.11 – Esquema geral da arquitetura para Dilatação Exata e Propagação de Rótulos

5.2.6 Geração de Esqueletos

Para a geração de esqueletos a partir do mapa de rótulos, foi implementado um módulo de processamento que realiza a operação de máxima diferença descrita na Figura 2.13. Este módulo realiza as quatro subtrações e as compara duas a duas, selecionando sempre a maior diferença. Na saída deste módulo temos, então, o esqueleto do objeto.

Figura 5.14 – Exemplo de SKIZ 1: a) imagem original com as regiões rotuladas, b) mapa de distância, c) mapa de rótulo, d) SKIZ obtido por um processamento em *software* gráfico da imagem c), e e) OU lógico OR entre as figuras a) e d)

Figura 5.15 – Outro exemplo de SKIZ: a) imagem original com pontos aleatórios já rotulados, b) mapa de distância, c) mapa de rótulo, d) Esqueleto obtido pela arquitetura, e) SKIZ obtido depois de um limiar, e f) OU lógico entre imagens a) e e)

Figura 5.16 – Exemplo de esqueleto: a) imagem original de um neurônio, b) imagem do contorno rotulado, c) mapa de distância, d) mapa de rótulos, e) esqueleto obtido pela arquitetura, e f) esqueleto obtido pelo processamento da imagem d) por *software*

5.3. MPI

Abaixo temos o resultado da implementação utilizando MPI para o processamento do Algoritmo de Dilatação Exata e Propagação de Rótulos para dois pontos P1(64,56) e P2(64,72). Os resultados obtidos foram os mesmos obtidos pela versão sequencial.

Figura 5.17 – Resultado do processamento paralelo utilizando MPI para o cálculo do Algoritmo de Dilatação Exata e Propagação de Rótulos – Distâncias

Figura 5.18 – Resultado do processamento paralelo utilizando MPI para o cálculo do Algoritmo de Dilatação Exata e Propagação de Rótulos – Rótulos

Abaixo temos a listagem da implementação em paralelo, utilizando MPI, do Algoritmo de Dilatação Exata e Propagação de Rótulos.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <mpi.h>

#define NoError 0
#define ArgError 2
#define FileError 3

struct sPixel{
    float Dist;
    int Label;
};

struct sPonto{
    int X;
    int Y;
    int Label;
};

int main(int argc, char **argv){

    int i, j, k, l, Nd, X, Y, H, W, Rmax;
    FILE *arquivo;
    struct sPonto *Pontos;
    struct sPixel *Image;
    float dist, time;

    int versorX[4]={+1,-1,-1,+1};
    int versorY[4]={+1,+1,-1,-1};
```

```

//Dados paralelos
int Np, rank, n, q, r;
struct sPonto Ponto, *Local;
int *conts, *desls, desloc;
struct sPixel *ImageFinal;

MPI_Datatype tipoPonto[3] = {MPI_INT, MPI_INT, MPI_INT};
int compPonto[3] = {1, 1, 1};
MPI_Aint deslPonto[3];
MPI_Datatype MPI_Ponto;

/*****
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &Np);

//Criacao do novo tipo
MPI_Address(&Ponto.X, &deslPonto[0]);
MPI_Address(&Ponto.Y, &deslPonto[1]);
MPI_Address(&Ponto.Label, &deslPonto[2]);

deslPonto[2] -= deslPonto[0]; //2*sizeof(MPI_INT);
deslPonto[1] -= deslPonto[0]; //1*sizeof(MPI_INT);
deslPonto[0] = 0;

MPI_Type_struct(3, compPonto, deslPonto, tipoPonto, &MPI_Ponto);
MPI_Type_commit(&MPI_Ponto);
printf("Novo tipo criado (%d).\n", rank);

if(rank == 0){
    //Verifica se forneceu nome do arquivo de dados
    if(argc != 5){
        fprintf(stderr, "Entre com o nome do arquivo, a dimensao e o raio.\n");
        MPI_Finalize();
        exit(ArgError);
    }

    //Abre arquivo de dados //e cria os vetores de envio de dados
    arquivo = fopen(argv[1], "r");
    if(arquivo == NULL){
        fprintf(stderr, "Arquivo %s nao pode ser aberto.\n", argv[1]);
        MPI_Finalize();
        exit(FileError);
    }
    else{
        fscanf(arquivo, "%d", &Nd);
        printf("Nd=%d (%d)\n", Nd, rank);
        Pontos = (struct sPonto *)calloc(Nd, sizeof(struct sPonto));

        for(i = 0; i < Nd; i++){
            fscanf(arquivo, "%d", &Pontos[i].X);
            fscanf(arquivo, "%d", &Pontos[i].Y);
            fscanf(arquivo, "%d", &Pontos[i].Label);
        }
        fclose(arquivo);
    }
}

MPI_Bcast(&Nd, 1, MPI_INT, 0, MPI_COMM_WORLD);

Rmax = atoi(argv[4]);
H = atoi(argv[3]);
W = atoi(argv[2]);

Image = (struct sPixel *)calloc(H*W, sizeof(struct sPixel));
ImageFinal = (struct sPixel *)calloc(H*W, sizeof(struct sPixel));
for(i=0;i<W*H;i++){
    Image[i].Label = Nd+1;
    Image[i].Dist = Rmax;
}

printf("Matrizes criadas (%d).\n", rank);

//Distribuicao dos dados

```

```

conts = (int *)calloc(Np, sizeof(int));
desls = (int *)calloc(Np, sizeof(int));

n = ceil((float)Nd / Np);
Local = (struct sPonto *)calloc(n, sizeof(struct sPonto));

q = Nd / Np;
r = Nd % Np;

desloc = 0;
for(i = 0; i < Np; i++){
    if(i<r)
        conts[i] = q+1;
    else
        conts[i] = q;
    desls[i] = desloc;
    desloc += conts[i];
}

MPI_Scatterv(Pontos, conts, desls, MPI_Ponto,
             Local, n, MPI_Ponto, 0, MPI_COMM_WORLD);

printf("Dados distribuidos (%d).\n", rank);

printf("Comecando a processar(%d).\n", rank);
//Comeca o processamento
time = MPI_Wtime();
for(k = 0; k < Rmax; k++){
    //printf("k=%d (%d)\n", k, rank);
    for(l = 0; l <= k; l++){
        for(j = 0; j < conts[rank]; j++){
            for(i = 0; i < 4; i++){
                X = Local[j].X+versorX[i]*k;
                Y = Local[j].Y+versorY[i]*l;
                dist = sqrt(k*k+l*l);
                if ((X>=0) && (Y>=0) && (X<W) && (Y<H)){
                    if((Image[X+W*Y].Dist > dist) ||
                       ((Image[X+W*Y].Dist == dist) &&
                        (Image[X+W*Y].Label > Local[j].Label))){
                        Image[X+W*Y].Dist = dist;
                        Image[X+W*Y].Label = Local[j].Label;
                    }
                }
            }
        }
        if((k != 0) && (l != k)){
            for(i = 0; i < 4; i++){
                X = Local[j].X+versorX[i]*l;
                Y = Local[j].Y+versorY[i]*k;
                if ((X>=0) && (Y>=0) && (X<W) && (Y<H)){
                    if((Image[X+W*Y].Dist > dist) ||
                       ((Image[X+W*Y].Dist == dist) &&
                        (Image[X+W*Y].Label > Local[j].Label))){
                        Image[X+W*Y].Dist = dist;
                        Image[X+W*Y].Label = Local[j].Label;
                    }
                }
            }
        }
    }
}

time = MPI_Wtime() - time;
printf("Processing time:%fs (%d)\n", time, rank);

time = MPI_Wtime();
MPI_Reduce(Image, ImageFinal, H*W,
            MPI_FLOAT_INT, MPI_MINLOC, 0, MPI_COMM_WORLD);
time = MPI_Wtime() - time;
printf("Reducing time:%fs (%d)\n", time, rank);

if (rank == 0){
    arquivo = fopen("LabelParaleloMINLOC.txt", "wt");
    for(j=0;j<H;j++){
        for(i=0;i<W;i++){

```



```

        fprintf(arquivo, "%d\t", ImageFinal[i+W*j].Label);
        fprintf(arquivo, "\n");
    }
    fclose(arquivo);

    arquivo = fopen("DistParaleloMINLOC.txt", "wt");
    for(j=0; j<H; j++){
        for(i=0; i<W; i++){
            fprintf(arquivo, "%f\t", ImageFinal[i+W*j].Dist);
            fprintf(arquivo, "\n");
        }
        fclose(arquivo);
    }

    if (rank == 0){
        free(Pontos);
    }
    free(ImageFinal);
    free(Image);
    free(conds);
    free(desls);
    free(Local);
    MPI_Finalize();
    return NoError;
}

```

6. Conclusões

Neste trabalho apresentamos o desenvolvimento de arquiteturas para o processamento de imagens, mais especificamente, para o cálculo da Transformada de Distância Euclidiana e a Geração de Esqueletos. Tais operações são muito importantes na análise de imagens e no reconhecimento de padrões, possuindo uma ampla área de atuação.

As arquiteturas foram baseadas na modificação de um algoritmo já existente para estas operações, e que se demonstrou bastante prático e eficiente. Tal praticidade e eficiência se refletem nas arquiteturas, uma vez que estas são baseadas em máscaras de processadores que realizam as mesmas operações realizadas pelo algoritmo.

Uma deficiência que a arquitetura para Propagação de Rótulos possui é que ela necessita de que a imagem já tenha sido previamente processada (ele deve ter os pixels de borda rotulados). Por isso se faz extremamente necessário o desenvolvimento de um sistema para rotulagem que possa operar em tempo real e gere as informações necessárias para a arquitetura.

O uso de dispositivos reconfiguráveis, aliados às suas ferramentas de desenvolvimento, foi de grande valia, sem os quais boa parte do trabalho ficaria somente na concepção. Tais ferramentas computacionais permitem que o tempo de desenvolvimento, análise e implementação seja drasticamente reduzido.

As arquiteturas implementadas apresentaram-se eficientes para o que foi proposto, permitir o cálculo da Transformada de Distância e a Geração de Esqueletos em tempo real. Vale a pena salientar que tempo real, neste caso, está relacionado com a taxa de aquisição de um sinal de vídeo. A arquitetura para Transformada de Distância, baseada no algoritmo de Dilatação Exata, por ser menos complexa que a baseada no algoritmo de Propagação de Rótulos, possui uma frequência de operação quase duas vezes maior. Estas frequências de operação, porém, não são a máxima em que as arquiteturas podem operar, visto que durante a compilação (síntese) destas não foram impostas configurações para se extrair o máximo em velocidade.

Verificamos que as arquiteturas, apesar de serem básicas na sua concepção, ou seja, são baseadas em máscaras, acabam consumindo uma quantidade muito grande de elementos lógicos disponíveis nos dispositivos reconfiguráveis. Desta forma, as demais arquiteturas inicialmente propostas provavelmente irão consumir bem mais recursos lógicos, tornando a sua implementação difícil. Só para comparar, foi implementada uma arquitetura, baseada na proposta de Máscara *Contour-scan*, de 27x27 elementos (o que possibilita o processamento de imagens de apenas 27x27 pixels!!!). Foi verificado que o consumo de recursos lógicos por célula de processamento é quase 4 vezes maior do que a implementação da Máscara *Raster-scan*, principalmente devido aos elementos adicionados à Célula de Processamento para que a imagem possa se deslocar nas quatro direções (vide Apêndice).

Da mesma forma, a necessidade de uma grande quantidade de elementos lógicos faz com que só seja possível implementar as arquiteturas em dispositivos reconfiguráveis de alta densidade, que são disponíveis apenas em encapsulamentos tipo BGA (*Ball Grid Array*) com quase 1000 pinos. Isto inviabilizou a construção de um protótipo de placa PCI para servir de co-processador para o processamento das imagens. Neste caso fica muito mais interessante a compra de uma placa PCI de desenvolvimento, que custa cerca de US\$7.000,00.

Uma opção extremamente interessante é que estas arquiteturas sejam implementadas em chip (VLSI), para que possam operar em frequências bem mais altas e calculem distâncias maiores do que 12 ou 15 pixels de raio.

6.1 Trabalhos Futuros

Dentre os trabalhos futuros podemos propor, inicialmente, a criação e desenvolvimento de novos módulos que possam ser anexados às arquiteturas, como por exemplo um módulo para o cálculo da dimensão fractal em tempo real. Além disso existe a necessidade de se definir e implementar uma interface entre as arquiteturas e o mundo real, quer ela opere sozinha ou anexada a um *host*.

Devido à sua estrutura regular e uniforme, é extremamente interessante que estas arquiteturas possam ser implementadas em VLSI para que se possa obter o máximo desempenho com relação à frequência máxima de operação, como possibilitar o processamento de imagens maiores do que as atuais podem processar. Para tanto é necessário que ela seja totalmente especificada em uma linguagem de hardware, o permitiria a sua total portabilidade.

Dentre os trabalhos futuros também podemos propor a análise e o estudo da implementação do algoritmo para a extração de esqueleto de imagens tridimensionais [4]. A primeira idéia é de que isto poderia ser facilmente feito, utilizando uma versão tridimensional das arquiteturas implementadas, onde cada arquitetura implementada processaria uma fatia da imagem 3D. Contudo se faz necessário um estudo mais detalhado, começando desde as modificações do algoritmo para o processamento de imagens tridimensionais, passando pela possibilidade de paralelização deste e provável implementação em *hardware*.

A implementação em MPI foi apenas um ensaio de um trabalho mais aprofundado que pode ser realizado sobre o uso de bibliotecas paralelas para a implementação deste algoritmo. Pudemos verificar que a existência de um operador de redução que se encaixa perfeitamente na definição do algoritmo modificado pode ser facilmente utilizado nesta implementação. O que é reconhecido de fato é que a maior parte do processamento será devida ao tráfego de informações, principalmente entre o processo *master* e os demais. Esta dependência com o tráfego de dados torna o algoritmo interessante para o estudo do melhor tamanho de pacote de dados que

deve ser utilizado para reduzir esta perda. Outros trabalhos que poderiam ser desenvolvidos seriam novas propostas de paralelização e implementação em *software* paralelo do Algoritmo de Dilatação Exata e Propagação de Rótulos.

7. Bibliografia

- [1] Fountain, T. J; Matthews, K. N., Duff, M. J. B., “The CLIP7A Image Processor” IEEE Transactions on PAMI, v.10, n.3, pp.310319-61, 1988
- [2] Rosenfeld, A.; Pfaltz, J. L., “Distance Functions on Digital Pictures” Pattern Recognition, 1, pp.33-61, 1968
- [3] Consularo, L. A., “Técnicas de Mineração de Dados para Análise de Imagens”. São Carlos, 2000. Np. Tese (Doutorado) – Instituto de Física de São Carlos, Universidade de São Paulo.
- [4] Costa, L. F., “Robust Skeletonization through Exact Euclidean Distance Transform and its Applications to Neuromorphometry”, Real-Time Imaging, v.6, n.6, pp. 415-431, 2000
- [5] Mandelbrot, B. B., “The Fractal Geometry of Nature”, W. H. Freeman, New York, 1983
- [6] Russ, J. C. “The Image Processing Handbook”. 2nd ed., CRC Press, 1995
- [7] Allen, M., Brown, G. J., Miles, N. J., “Measurement of boundary fractal dimensions: review of current techniques”, Powder Technology, v.84, n.1, pp. 1-14, 1995
- [8] Asvestas, P., Matsopoulos, G. K., Nikita, K. S., “Estimation of fractal dimension of images using a fixed mass approach” Pattern Recognition Letters, v.20, pp. 347-354, 1999
- [9] Béburé, D., Jébrak, M., “High precision boundary fractal analysis for shape characterization”, Computer & Geosciences, v.25, pp. 1059-1071, 1999
- [10] Jelinek, H. F., Fernandez, E., “Neurons and fractals: how reliable and useful are calculations of fractal dimensions?”, Journal of Neuroscience Methods, v.81, pp. 9-18, 1998

- [11] Ogawa, S., Baveye, P., Boast, C. W., Parlange, J. Y., Steenhuis, T., "Surface fractal characteristics of preferential flow pattern in field soils: evaluating and effect of image processing", *Geoderma*, v.88, pp. 109-136, 1999
- [12] Hyslip, J. P., Vallejo, L. E., "Fractal analysis of the roughness and size distribution of granular materials", *Engineering Geology*, v.48, pp. 231-244, 1997
- [13] Lam, L., Lee, S. W. & Suen, C. Y., "Thinning Methodologies – A Comprehensive Survey"; *IEEE Transactions on PAMI*, v.14, n.9, pp. 869-885, 1992
- [14] Tzionas, P, Tsalides, Ph., Thanailakis, ^a, "A Parallel Skeletonization Algorithm Based on Two-Dimensional Cellular Automata and its VLSI Implementation", *Real-Time Imaging*, v.1, pp.105-117, 1995
- [15] Arcelli, C., Baja, G. S., "A Width-Independent Fast Thinning Algorithm", *IEEE Transactions on PAMI*, v.7, n.4, pp.463-474, 1985
- [16] Arcelli, C., Baja, G. S., "A One-Pass Two-Operation Process to Detect the Skeletal Pixels on the 4-Distance Transform", *IEEE Transactions on PAMI*, v.11, n.4, pp.463-474, 1989
- [17] Guo, Z. & Hall, R. W., "Fast Fully Parallel Thinning Algorithms"; *CVGIP: Image Understanding*, v.55, n.3, pp. 317-328, 1992
- [18] Jang, B.K. & Chin, R. T., "Analysis of Thinning Algorithms Using Mathematical Morphology"; *IEEE Transactions on PAMI*, v.12, n.6, pp. 541-551, 1990
- [19] Jenq, J. F. & Sahni, S., "Serial and Parallel Algorithms for the Medial Axis Transform"; *IEEE Transactions on PAMI*, v.14, n.12, pp. 1218-1224, 1992
- [20] Niblack, C. W.; Gibbons, P. B.; Capson, D. W.; "Generating Skeletons and Centerlines from the Distance Transform" *CVGIP: Graphical Models and Image Processing*, v.54, n.5, pp. 420-437, 1992
- [21] Baja, G. S.; Thiel, E., "Skeletonization algorithm running on path-based distance maps"; *Image and Vision Computing*, v.14, pp. 47-57, 1996
- [22] Shih, F. Y; Pu, C., C.; "A Skeletonization Algorithm by Maxima Tracking on Euclidean Distance Transform"; *Pattern Recognition*, v.28, n.3, pp.331-341, 1995

- [23]Ge, Y. & Fitzpatrick, J. M., "On the Generation of Skeletons from Discrete Euclidean Distance Maps"; IEEE Transactions on PAMI, v.18, n.11, pp. 1055-1066, 1996
- [24]Manzanera, A.; Bernard, T. M.; Prêteux, F.; Longuet, B., "n-dimensional skeletonization: a unified mathematical framework", Journal of Electronic Imaging, v.11, n.1, pp. 25-37, 2002
- [25]Malandain, G.; Vidal, S. F., "Euclidean Skeletons", Image and Vision Computing, v.16, pp. 317-327, 1998
- [26]Costa, L. da F.; Campos, A. G.; Estrozi, L. F.; Rios, L. G.; Bosco, A. "A Biologically-Motivated Approach to Image Representation and Its Applications to Neuromorphology." Lectures Notes on Computer Science, v.1811, pp.407-416, 2000
- [27]Paglieroni, D. W., "Distance Transforms: Properties and Machine Vision Applications", CVGIP: Graphical Models and Image Processing, v.54, n.1, pp. 56-74, 1992
- [28]Wang, J. G. & Sung, E., "Frontal-view face detection and facial feature extraction using color and morphological operations", Pattern Recognition Letters v.20, pp.1053-1068, 1999
- [29]Sagar, B. S. Daya, "Fractal Relation of a Morphological Skeleton", Chaos, Solitons & Fractals, v.7, n.11, pp. 1871-1879, 1996
- [30]Herbin, M., Bonnet, N., Vautrot, P. "A clustering method based on the estimation of the probability density function and on the skeleton by influence zones. Application to image processing", "; Pattern Recognition Letters, v.17, pp.1141-1150, 1996
- [31]Jan, S. -R; Hsueh, Y. -C, "Primitive spatial relations based on SKIZ", Image and Vision Computing, v.18, pp. 597-605, 2000
- [32]Hirata, T. "A unified linear-time algorithm for computing distance maps." Information Processing Letters, v.58, n.3, pp.129-133, 1996
- [33]Chen, L.; Chuang, H. Y. H. "A fast algorithm for Euclidean distance maps of a 2-D binary image." Information Processing Letters, v.51, pp.25-29, 1994
- [34]Lee, Y. H.; Horng, S. J.; Kao, T. W.; Jaung, F. S.; Chen, Y. J.; Tsai, H. R. "Parallel computation of exact Euclidean distance transform." Parallel Computing, v.22, n.2, pp. 311-325, 1996

- [35]Lee, Y. H.; Horng, S. J. "Optimal Computing the Chessboard Distance Transform on Parallel Processing Systems." *Computer Vision and Image Understanding*, v.73, n.3, pp. 374–390, 1999
- [36]Fujiwara, A.; Inoue, M.; Masuzawa, T.; Fujiwara, H. "A cost optimal parallel algorithm for weighted distance transforms." *Parallel Computing*, v.25, n.4, pp. 405-416, 1999
- [37]Kolountzakis, M. N.; Kutulakos, K. N., "Fast computation of the Euclidean distance maps for binary images", *Information Processing Letters*, v.43, pp.181-184, 1992
- [38]Pan, Y.; Li, K., "Constant-time algorithm for computing the Euclidean distance maps of binary images on 2D meshes with reconfigurable buse" *Information Sciences*, v.129, pp.209-221, 1999
- [39]Takala, J. H.; Viitanen, J. O. "Distance Transform Algorithm for Bit-Serial SIMD Architectures." *Computer Vision and Image Understanding*, v.74, n.2, pp. 150-161, 1999
- [40]Baglietto, P.; Maresca, M.; Migliardi, M. "Euclidean Distance Transform on Polymorphic Processor Array." *Proceedings of International Conference on Computer Architectures for Machine Perception, Como (Itália)*, pp.18-20, 1995
- [41]Ferreira, A. & Ubéda, S., "Computing the Medial Axis Transform in Parallel With Eight Scan Operations", *IEEE Transactions on PAMI*, v.21, n.3, pp.277-282, 1999
- [42]Ranganathan, N. & Doreswamy, K.B., "A VLSI Chip for Computing the Medial Axis Transform of an Image", *Proceedings of CAMP '95*, 1995
- [43]Bourbakis, N., Steffensen, N. & Saha, B., "Design of an Array Processor for Parallel Skeletonization of Images"; *IEEE Transactions on Circuits and Systems – II: Analog and Digital Signal Processing*, v.44, n.4, pp. 284-298, 1997
- [44]Abbott, L., Haralick, R. M. & Zhuang, X., "Pipeline Architectures for Morphologic Image Analysis"; *Machine Vision and Applications*, v.1, pp. 23-40, 1988
- [45]Chaudhuri, A. S., Cheung, P. Y. K. & Luk, W., "A Reconfigurable Data-Localized Array for Morphological Algorithms"; *LNCS v.1304*, pp. 344-353, 1997

- [46] Jonker, P. P. & Komen, E. R., Kraaijveld, M. A., “A scalable, real-time, image processing pipeline”, *Machine Vision and Applications*, v.8, pp. 110-121, 1995
- [47] Hujanen, A. A.; Waltz, F. M., “Pipelined Implementation of Binary Skeletonization Using Finite-State Machines”, *SPIE Conference on Machine Vision Applications in Industrial Inspection*, San Jose, Feb. 1995 - *SPIE Proc.* v.2423, n.2.
- [48] Kalmár, Z.; Marczell, Z.; Szepesvári, C.; Lörincz, A., “Parallel and robust skeletonization built on self-organizing elements”, *Neural Networks*, v.12, pp.163-172, 1999
- [49] Luppe, M.; Costa, L. da F.; Roda, V. O.; “Parallel implementation of exact dilations and multiscale skeletonization”, *Real-Time Imaging*, aceito.

Apêndice

Apresentação do resultado da implementação da proposta de arquitetura tipo Máscara *Contour-Scan*, para servir de comparação com a arquitetura *Raster-scan* implementada neste trabalho.

Processador XY(1) – Utilizando Barramento tri-state para o direcionamento dos dados

Processador XY(2) – Utilizando Multiplexadores para o direcionamento dos dados

Visão Geral da Arquitetura utilizando o Processador XY(1)

Detalhe da Arquitetura, mostrando as conexões nas quatro direções

Resultado da Compilação

Logic cells	43.765 / 57.120 (76 %)
Flipflops	11.664 / 59.193 (19 %)
I/O pins	30 / 691 (4 %)
Clock pins	1
Dedicated input pins	0
Global signals	1
M512s	0 / 574 (0 %)
M4Ks	0 / 292 (0 %)
MegaRAMs	0 / 6 (0 %)
Total memory bits	0 / 5.215.104 (0 %)
DSP block 9-bit elements	0 / 144 (0 %)
PLLs	0 / 12 (0 %)
Global clocks	1 / 16 (6 %)
Regional clocks	0 / 16 (0 %)
Fast regional clocks	0 / 16 (0 %)
DIFFIOCLKs	0 / 32 (0 %)
SERDES transmitters	0 / 116 (0 %)
SERDES receivers	0 / 116 (0 %)
Maximum fan-out node	dir[1]
Maximum fan-out	22616
Total fan-out	176602
Average fan-out	4.

EP1S60B956C6