TESE DE DOUTORADO

ARQUITETURAS DE HARDWARE PARA ACELERAÇÃO DE ALGORITMOS DE RECONSTRUÇÃO MORFOLÓGICA

Oscar Eduardo Anacona Mosquera

Brasília, Março de 2020

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia

TESE DE DOUTORADO

ARQUITETURAS DE HARDWARE PARA ACELERAÇÃO DE ALGORITMOS DE RECONSTRUÇÃO MORFOLÓGICA

Oscar Eduardo Anacona Mosquera

Tese de Doutorado submetida ao Departamento de Engenharia Mecânica como requisito parcial para obtenção do grau de Doutor em Sistemas Mecatrônicos

Banca Examinadora

Prof. Carlos H. Llanos Quintero, Dr., FT/UnB _____
Orientador

Profa. Alba Cristina M. A. de Melo, Dra., CIC/UnB _____
Examinador externo

Prof. George Luiz Medeiros Teodoro, Dr., DCC/UFMG _____
Examinador externo

Prof. Ricardo Pezzuol Jacobi, Dr., CIC/UnB _____
Co-orientador

FICHA CATALOGRÁFICA

ANACONA, OSCAR EDUARDO
ARQUITETURAS DE HARDWARE PARA ACELERAÇÃODE ALGORITMOS DE RECONSTRUÇÃO MORFOLÓGICA [Distrito Federal] 2020.
xvi, 103 p., 210 x 297 mm (ENM/FT/UnB, Doutor, Engenharia Mecânica, 2020).
Tese de Doutorado - Universidade de Brasília, Faculdade de Tecnologia.
Departamento de Engenharia Mecânica

1. Reconstrução morfológica	2. FPGA
3. Máquinas de vetores de suporte	4. Algoritmos bio-inspirados
I. ENM/FT/UnB	II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

ANACONA, O. (2020). *ARQUITETURAS DE HARDWARE PARA ACELERAÇÃODE ALGORITMOS DE RECONSTRUÇÃO MORFOLÓGICA*. Tese de Doutorado, Departamento de Engenharia Mecânica, Universidade de Brasília, Brasília, DF, 103 p.

CESSÃO DE DIREITOS

AUTOR: Oscar Eduardo Anacona Mosquera
TÍTULO: ARQUITETURAS DE HARDWARE PARA ACELERAÇÃODE ALGORITMOS DE RECONSTRUÇÃO MORFOLÓGICA.
GRAU: Doutor em Sistemas Mecatrônicos ANO: 2020

Oscar Eduardo Anacona Mosquera
Depto. de Engenharia Mecânica (ENM) - FT
Universidade de Brasília (UnB)
Campus Darcy Ribeiro
CEP 70919-970 - Brasília - DF - Brasil

Dedicatória

À minha mãe querida e meu querido pai.

Oscar Eduardo Anacona Mosquera

Agradecimentos

Ao meu orientador, o professor Carlos Humberto Llanos pela amizade, apoio, confiança, dedicação, paciência e orientação. Por ter me permitido trabalhar ao seu lado, aprendendo um pouco mais a cada dia.

Ao meu co-orientador, o professor Ricardo Pezzuol Jacobi, pela valiosa contribuição, paciência e valerosos ensinamentos durante todo o projeto.

Aos meus colegas e amigos: Renato, carlos Eduardo, Mario, Jones, Willian Molano, William Cuellar, Sérgio Cruz, Eder, e a tantos outros que participaram do LEIA - Laboratório de Sistemas Embarcados e Aplicações de Circuitos Integrados da Universidade de Brasília.

À CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) e o PPMEC (Programa de Pós-graduação em Sistemas Mecatrônicos) pelo suporte financeiro a este trabalho. Ao Grupo de Automação e Controle (GRACO/UnB) e a todos os meus professores pelo suporte e formação acadêmica.

Agradeço imensamente todas as pessoas que me permitiram chegar onde cheguei. É muito bom poder colher os frutos de um trabalho bem feito, com muito esforço e dedicação.

Oscar Eduardo Anacona Mosquera

RESUMO

Este trabalho apresenta um estudo da implementação de algoritmos para a reconstrução morfológica de imagens bio-medicas em FPGAs (*Field Programmable Gate Arrays*). As arquiteturas foram baseadas nos algoritmos *Sequential Reconstruction* (SR) e *Fast Hybrid* (FH) usando linguagem de descrição de hardware VHDL (*Very High Description Language*). A metodologia para avaliar a plataforma consistiu em verificar a arquitetura projetada no QuestaSim, fornecendo como dados de entrada as imagens a ser reconstruídas. Adicionalmente, a validação dos resultados da arquitetura foi feita usando linguagem C ou Matlab (usando a função *imreconstruct*). Além disso, um estudo consumo de recursos de *hardware* para diferentes tamanhos e conteúdos de imagens foram realizados com o intuito de verificar a aplicabilidade dos algoritmos em arquiteturas reconfiguráveis. Neste trabalho, para a aceleração do processo de reconstrução da imagem foi proposta uma arquitetura reconfigurável baseada no algoritmo FH junto com um algoritmo de aprendizagem de máquina, especificamente uma máquina de vetores de suporte (SVM). Para o treinamento da SVM foi usada uma metodologia de verificação/validação obtendo aproximadamente 20.000 dados de treinamento. Finalmente, foi implementada uma arquitetura que particiona a imagem original em quatro unidades de processamento, processando cada unidade em paralelo. O sistema final implementado fornece um pixel processado por cada ciclo de relógio, depois de um tempo de latência, sendo aproximadamente 8 vezes mais rápida que sua versão não particionada. Adicionalmente, foram feitas comparações rodando os algoritmos de reconstrução morfológica em um processador ARM embarcado dentro do FPGA.

ABSTRACT

This work presents a study of the implementation of algorithms for the morphological reconstruction of bio-medical images in FPGAs (Field Programmable Gate Arrays). The architectures were based on Sequential Reconstruction (SR) algorithms and Fast Hybrid (FH), using VHDL (Very High Description Language). The methodology for the evaluation of the platform consisted of verifying the architecture designed in QuestaSim, providing the images to be reconstructed as input data. Additionally, the validation of the results of the architecture was made using C or Matlab languages (using the imreconstruct function).

Additionally, a study of hardware resource consumption for different sizes and content of images was conducted, in order to verify the applicability of the algorithms in reconfigurable architectures. In this work, in order to accelerate the image reconstruction process, a reconfigurable architecture based on the FH algorithm is proposed together with machine learning, specifically a support vector machine (SVM). For the SVM training a verification/validation methodology was used, obtaining approximately 20,000 training data. Finally, an architecture was implemented that partitions the original image in four processing units, processing each unit in parallel. The final system implemented provides one pixel processed for each clock cycle, after a latency time, being approximately 8 times faster than its unpartitioned version. Lastly, comparisons were made by running the morphological reconstruction algorithms in an ARM processor embedded within the FPGA.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

1 INTRODUÇÃO

No processamento de imagens a Matemática Morfológica (MM) fornece um conjunto abrangente de ferramentas que permite se extrair de uma imagem informações dos objetos presentes na mesma, por exemplo formas e estruturas (Serra 1984). A análise morfológica pode ser usada para pré-processamento, segmentação e classificação, descrição quantitativa de objetos, reconhecimento de padrões, análise de texturas, etc. (Soille 1998). A utilização destes algoritmos gerou vários trabalhos de pesquisa na área da visão computacional, principalmente em aplicações para biomedicina, microscopia, biologia, mineralogia, petrografia, angiografia, compressão de vídeo, estereologia, monitoramento remoto, reconhecimento óptico de caracteres (*Optical Character Recognition* - OCR), entre outros (Serra 1984, Soille 1998, Heijmans e Ronse 1990, Bankhead et al. 2017).

Conhecendo-se previamente as características dos objetos a serem analisados, a matemática morfológica pode ser usada para extrair duas partes de uma imagem: (a) os objetos em estudo e (b) o fundo (Soille 1998). Por exemplo, na área da patologia, os diagnósticos das doenças são feitos através do processamento das imagens de amostras de tecidos. Através disto, é possível a detecção de células precursoras de câncer de cérebro, câncer cervical, entre outros tipos de câncer (Bankhead et al. 2017, Teodoro et al. 2013). Outros exemplos de objetos de imagens são: os vasos sanguíneos detectados em angiografias de raios-X, os núcleos e paredes celulares em citologia, poros representados por pontos que sobressaltam do fundo da imagem em petrografia, partículas com vários níveis em escala de cinza em mineralogia, entre outros tipos de imagens (Heijmans e Ronse 1990).

Dentre as operações morfológicas básicas para imagens binárias e/ou em níveis de cinza tem-se: dilatação, erosão, abertura e fechamento (Haralick, Sternberg e Zhuang 1987). Essas operações transformam a imagem original em outra imagem, através de iterações sucessivas com outra imagem (de forma e tamanho específicas), onde essa imagem de referência é nomeada de *elemento estruturante* (*Structuring Element*-SE) (Soille 1998). O resultado das operações morfológicas depende da topologia do pixel que for tratado. Por exemplo, um pixel pode possuir quatro vizinhos (*4-connectivity*), seis vizinhos (*6-connectivity*) ou oito vizinhos (*8-connectivity*) (Vincent 1992).

Assim, os objetos dentro de imagens binárias podem ser expressados como regiões que são conexas, sendo nomeadas de *componentes conexos* da imagem (Vincent 1992). Exemplos de transformações morfológicas são: transformada *hit-or-miss*, transformada de distância, esqueletização, reconstrução morfológica, transformada por divisor de águas, entre outras (Serra 1984, Vincent 1992). No entanto, a aplicação de algumas destas transformações apresenta problemas em alguns casos, gerando assim perda de informação e modificação em algumas estruturas dos objetos da imagem (Serra 1984). Dentre as técnicas usadas para evitar a perda de informação encontram-se a *reconstrução morfológica* e a *transformada por divisor de águas*.

Neste contexto, a *reconstrução morfológica* e a *transformada por divisor de águas* são parte de um conjunto de operadores conhecidos como *geodésicos*, sendo baseados no *principio de dilatação* (Vincent 1992). Estas transformações usam imagens de referência (*markers images*) a fim de marcar os objetos desejados em primeiro plano (Vincent 1992). Outra técnica para identificação de objetos é o algoritmo de componentes conexos (*Connected Component Labeling*-CCL) (He et al. 2017).

Atualmente, na área da biomedicina, com os recentes avanços nos *scanners*, é frequente ter imagens que possuem cerca de um bilhão de *células* (Bankhead et al. 2017, Teodoro et al. 2013). Com o objetivo de se obter uma automatização do processo de detecção das células, as quais podem apresentar mudanças significativas, tem se gerado uma grande motivação para desenvolver algoritmos que sejam eficientes e rápidos. Assim, o algoritmo de reconstrução morfológica pode ser descrito mediante o uso de um padrão computacional, nomeado padrão de propagação de frente de onda (*Irregular Wavefront Propagation Pattern*-IWPP) (Teodoro et al. 2013).

No IWPP, um elemento da imagem (pixel) pode ser tratado como uma fonte independente de ondas que propaga seu valor para os elementos da vizinhança sempre que uma determinada condição de propagação é alcançada. A composição dessas frentes de onda pode ser dinâmica, tendo uma dependência de dados que são calculados durante a execução na medida que as frentes de onda são propagadas. Dessa forma, o algoritmo de reconstrução morfológica permite uma rápida convergência da solução (Vincent 1992).

Dentro dos algoritmos para reconstrução morfológica (baseados no IWPP), o algoritmo *Sequential Reconstruction* (SR) é amplamente usado no processamento de imagens, destacando-se por ser simples e de fácil implementação (Vincent 1992). O algoritmo SR segue dois princípios: (a) os pixels da imagem são escaneados em uma determinada ordem, *raster* (*forward*) e *anti-raster* (*backward*) e (b) o novo valor do pixel que está sendo processado é determinado pela sua vizinhança (Vincent 1992). No entanto, o algoritmo SR converge lentamente quando possui poucas fontes de onda, gerando um alto custo computacional.

Neste cenário, o algoritmo *Fast Hybrid Reconstruction* (FH) foi desenvolvido para acelerar a reconstrução morfológica da imagem. No mesmo sentido, o algoritmo FH segue dois princípios: (a) são feitas algumas varreduras usando o algoritmo SR e durante as varreduras são armazenadas dentro de uma fila novas fontes de propagação, e (b) as fontes de propagação armazenadas em uma fila são propagadas e, por sua vez, novas fontes de propagação são calculadas (Karas 2011).

Assim como um pixel pode propagar seu valor para a sua vizinhança no algoritmo FH, o mesmo comportamento acontece nos algoritmos da transformada por divisor por águas e CCL (Vincent 1992, He et al. 2017). Neste contexto, o elevado custo computacional destes algoritmos tem provocado uma grande motivação para desenvolver técnicas de operações morfológicas usando algoritmos que sejam eficientes do ponto de vista de desempenho, facilmente paralelizáveis e implementáveis em plataformas computacionais.

Com relação ao processamento de imagens, podem ser usados três tipos de plataformas computacionais: (a) programáveis, (b) reconfiguráveis e (c) de aplicações específicas. As arquiteturas

programáveis incluem os processadores de propósito geral (*General Purpose Processors*-GPPs), Unidades de Processamento Gráfico (*Graphics Processing Units*-GPUs) e a Unidade de Processamento Gráfico de Propósito Geral (*General Purpose Graphics Processing Unit*-GPGPU) (Kornaros 2010). As arquiteturas reconfiguráveis permitem programar *hardware* (*hardware* reconfigurável), tal como o caso dos FPGAs (*Field Programmable Gate Arrays*) e os Processadores Integrados de Aplicações Específicas Reconfiguráveis (*Reconfigurable Application-Specific Integrated Processors*-RASIPs) (Kornaros 2010, Klaiber et al. 2016).

Dentro das arquiteturas de aplicações específicas encontram-se os Processadores Integrados de Aplicações Específicas (*Application-Specific integrated Processors*-ASIPs) e os Processadores Digitais de Sinais (*Digital Signal Processors*-DSPs) (Kornaros 2010). Assim, uma abordagem para a implementação dos algoritmos de processamento de imagens é mapeá-los diretamente em *hardware*.

Atualmente, pode-se encontrar plataformas FPGA que possuem mais de mil somadores, multiplicadores, DSPs, capacidade de memória (*On-chip memory*), processadores *soft-core* e *hard-core* tais como: NIOS II, ARM, *MicroBlaze* ou *Picoblaze* (Klaiber et al. 2016, Muttillo et al. 2016).Portanto, pode-se aproveitar o uso das plataformas reconfiguráveis para tirar proveito do co-projeto *hardware*/*software* (*Hw*/*Sw co-design*) a fim de implementar a reconstrução morfológica visando melhorar o desempenho no quesito de consumo de potência, espaço ocupado e tempo de processamento (Muttillo et al. 2016).

1.1 MOTIVAÇÃO DO TRABALHO

Os algoritmos baseados em morfologia matemática, especialmente os algoritmos de reconstrução morfológica, são amplamente usados para segmentação de imagens em plataformas *software*, e pouco explorados em *hardware*. Geralmente, os algoritmos de reconstrução são iterativos, sendo limitados a ser executados em duas ou várias varreduras definidas pelo programador. A fixação das iterações pode gerar um aumento no tempo de execução e a perda de eficiência do algoritmo devido a redundância de operações sobre as imagens. A metodologia do algoritmo de reconstrução morfológica é abordada mostrando sempre duas etapas: (a) pré-processamento usando varreduras e (b) processamento que segue um caminho irregular sobre a imagem usando estrutura de dados como *filas*. Essa abordagem permite um ganho de velocidade para alcançar a convergência da solução.

A literatura apresenta métricas que mostram a dependência entre o tipo de plataforma escolhida e o desempenho dos algoritmos, por exemplo: cálculo de densidade e entropia das imagens (Baniani e Chalechale 2013, He et al. 2017). A inserção dessas métricas incrementaria o custo computacional do algoritmo na etapa de pré-processamento, além dos requisitos de memória e latência. Neste contexto, busca-se uma métrica que permita analisar o comportamento do algoritmo durante o processamento da reconstrução até a convergência da solução. A alternativa

mais acessível é a medição da taxa variação dos pixels efetivos por iteração que estão sendo modificados, com o intuito de estabelecer o momento em que o algoritmo apresenta seu melhor desempenho (menor tempo de processamento). De este modo, o problema se converte em um problema de otimização, com características complexas (multimodal e multidimensional), onde podem ser utilizadas técnicas avançadas de aprendizado de maquina (*machine learning)*.

Atualmente, o uso de algoritmos de aprendizagem de máquina combinados com meta-heurísticas são usados para resolver problemas de otimização conseguindo um balanço entre desempenho e tempo de execução. As máquinas de vetores de suporte (*Support Vector Machine*-SVM) tem se mostrado como um algoritmo de aprendizagem de máquina robusto para classificação e/ou regressão de sistemas lineares ou não lineares. Por outro lado, as meta-heurísticas inspiradas no comportamento da natureza oferecem um grande campo de soluções para otimização de problemas.

1.2 PROPOSTA DE HIPÓTESIS PARA ESTE TRABALHO

Uma revisão da literatura mostra uma lacuna na área de arquiteturas específicas (*ad-hoc*) para aceleração de algoritmos para reconstrução morfológica (vide capítulo 4), especificamente para os operadores de reconstrução morfológica. Igualmente, a literatura mostra uma dificuldade na paralelização dos algoritmos que processam as imagens usando *filas* devido à dependência de dados e ao caminho irregular que seguem.

Neste sentido, a combinação de técnicas de Máquinas de Vetores de Suporte (SVMs) e meta-heurísticas podem originar soluções visando otimizar o desempenho de algoritmos de reconstrução morfológica, visando também sua implementação em sistemas embarcados (baseados em FPGAs). Assim, pode-se criar critérios de convergência para minimizar o efeito do processamento irregular da fila. Assim, as hipóteses básicas deste projeto de pesquisa seriam: (a) existe a possibilidade de se desenvolver arquiteturas de *hardware* eficientes para a reconstrução morfológica via o algoritmo FH, (b) existe uma lacuna que pode ser preenchida no quesito de abordagens de co-projeto *hardware*/*software* para algoritmos de reconstrução morfológica, (c) existe a possibilidade de se desenvolver uma arquitetura heterogênea eficiente usando uma estrutura de co-projeto de *hardware-software* onde uma SVM possa prever o ponto ótimo de chaveamento entre as etapas de execução do raster/anti-raster e propagação da fila, envolvidas no algoritmo FH.

1.3 OBJETIVOS

1.3.1 Objetivo Geral

Este trabalho se propõe a desenvolver arquiteturas de *hardware* reconfigurável para a aceleração de algoritmos de reconstrução morfológica, capazes de serem aplicadas a sistemas embarcados com aplicações para segmentação de imagens.

1.3.2 Objetivos Específicos

1. Realizar um perfilamento (*profile*) de algoritmos baseados em matemática morfológica com o objetivo de compreender seu funcionamento, seus gargalos de desempenho e as possíveis implementações em *hardware* dedicado.

2. Desenvolver arquiteturas *ad-hoc* em *hardware* para implementação dos algoritmos de reconstrução morfológica explorando os métodos para estabelecer um critério de parada e generalização de algoritmos com similar metodologia.

3. Desenvolver co-projetos *hardware*/*software* para a implementação do preditor para o algoritmo FH.

4. Analisar, verificar e testar as arquiteturas obtendo as métricas de desempenho e precisão das soluções em relação à soluções equivalentes em *software*.

5. Particionar as imagens em várias sub-imagens processando cada uma de elas para demostrar a dependencia entre o desempenho e o nível de paralelização.

1.4 METODOLOGIA

No desenvolvimento deste trabalho, a seguinte metodologia foi aplicada:

1. *Revisão Bibliográfica*: Foi realizada uma revisão bibliográfica do estado da arte a respeito de algoritmos baseados em matemática morfológica procurando ressaltar os principais trabalhos que tentam implementar estes algoritmos de modo efetivo em *hardware* e *software*.

2. *Implementação por software*: Uma solução do SR e FH foi inicialmente desenvolvida em software em linguagem C e no Matlab procurando testar os melhores aspectos para mapear as características desses algoritmos em *hardware*.

3. *Arquitetura dedicada em hardware para o SR e FH*: A implementação em *hardware* foi desenvolvida em VHDL e testada tanto de maneira independente na ferramenta QuestaSim, Quartus II e ISE onde seu desempenho em termos de velocidade e de precisão foram analisados para vários tamanhos e conteúdos de imagens.

4. *Implementação por software do SVR para o FH*: A solução do preditor por uma SVR em software no linguagem C implementado no processador embarcado ARM.

5. *Arquitetura dedicada em hardware do FH com SVR*: A implementação em *hardware* para o FH foi desenvolvida utilizando os mesmos métodos descritos no item 3.

6. *Arquitetura dedicada em hardware do FH particionado*: Desenvolvimento de uma arquitetura com quatro co-processadores *hardware*/*software* trabalhando em paralelo.

1.5 CONTRIBUIÇÕES DAS PROPOSTAS

O estudo tem como principal contribuição a implementação do co-processador *hardware*/*software* para reconstrução morfológica de imagens de modo a definir um compromisso entre o desempenho e tamanho das imagens, tendo em conta o consumo de recursos lógicos e consumo de memória da plataforma FPGA. As propostas desenvolvidas neste trabalho foram os dois algoritmos sequenciais (*Sequential Reconstruction* e *Fast Hybrid reconstruction algorithm*).

Como principais contribuições deste trabalho, citam-se: (a) uma arquitetura *hardware*/*software* usando o algoritmo sequencial SR para duas abordagens, a primeira mapeia o algoritmo em *hardware*, e a segunda abordagem usa como base a arquitetura SR para implementar em *software* o algoritmo FH; (b) a arquitetura do algoritmo FH usando uma máquina de vetores de suporte como criterio de convergência da solução do algoritmo;(c) uma arquitetura *hardware*/*software* para implementar o algoritmo FH totalmente em *hardware*; e (d) a paralelização do algoritmo FH em quatro co-processadores *hardware*/*software*.

As arquiteturas desenvolvidas neste estudo podem ser expandidos para aplicações específicas, como, por exemplo, algoritmo de componentes conexos usando imagens binárias, algoritmo de divisor por águas, ou algoritmos que sigam uma estrutura parecida com o algoritmos de reconstrução morfológica.

Basicamente, as arquiteturas desenvolvidas são compostas pelos seguintes módulos: (a) comunicação entre o processador, seja o NIOS II ou o ARM; (b) controle da arquitetura através de uma máquina de estados; (c) transfêrencia das imagens do processador para a unidade *hardware*; (d) leitura e armazenamento das imagens no *hardware*; (e) escaneamento e processamento das imagens; e (f) transfêrencia das imagens do *hardware* para o processador.

No escaneamento e processamento das imagens, são usados diferentes circuitos para a reconstrução da imagem, entre eles temos: (a) escaneamento raster/ anti-raster, (b) fila normal (*queue*), e/ou (c) uma máquina de vetores de suporte (SVM).

1.6 ORGANIZAÇÃO DO TRABALHO

O presente trabalho está organizado da seguinte forma:

No Capítulo 2, uma introdução à teoria de matemática morfológica é descrita.

No Capítulo 3, realiza-se uma revisão do estado da arte focado nos algoritmos de componentes conexos, divisor por águas e reconstrução morfológica em *hardware* e *software*.

Nos Capítulos 4 e 5, são apresentados os aspectos gerais de *Hardware* reconfigurável e a implementação das arquiteturas.

No Capítulo 6, são apresentadas cinco soluções em *Hardware* relativas ao SR, FH, FH com preditor, e FH paralelizado.

Finalmente, no capítulo 7, são apresentadas as conclusões obtidas a partir dos resultados e dos testes realizados no Capítulo 6.

2 ASPECTOS DE PROCESSAMENTO DE IMAGENS E RECONSTRUÇÃO MORFOLÓGICA

Na área de processamento de imagens, aplicações como filtragem, reconhecimento de padrões, extração de características, e/ou restauração de imagens, demandam uma grande quantidade de recursos computacionais (Yang e Papa 2016). O passo antecessor ao processo de reconhecimento de padrões, etiquetamento ou extração de características é conhecido como *segmentação*. No processo de segmentação, uma imagem é dividida em regiões ou *texturas* e elementos nomeados como *partículas*. Historicamente, foram desenvolvidas vários algoritmos de segmentação para diversas arquiteturas sendo a técnica mais popular de segmentação conhecida como algoritmo de divisor de águas ou *watershed algorithm* (Vincent 1991). Os algoritmos de segmentação junto com a morfologia matemática são ferramentas de uso frequente no processamento de imagens para evitar a sobre-segmentação das imagens.

2.1 CONSIDERAÇÕES INICIAIS

O processamento de uma imagem pode ser computacionalmente caro, dependendo do tipo de finalidade, dimensões e conteúdo da imagem. Na área da visão computacional, destacam-se quatro níveis de abstração que o ser humano pode perceber, entre eles temos (Sonka, Hlavac e Boyle 2014):

- Pré-processamento de imagens: são as imagens que não foram alteradas ainda ou que passam por um processo de filtragem, reconstrução ou restauração. Comumente, a base e o gargalo dos algoritmos (incluídos nesta etapa) são o processo de convolução e deconvolução.
- Segmentação: partes de uma imagem são organizadas em grupos. Neste processo, o efeito indesejado é a quando acontece uma sobre-segmentação da imagem por causa do ruído ou da abordagem utilizada.
- Descrição de objetos: etiquetamento dos objetos ou regiões encontradas na etapa de segmentação.
- Modelo relacional: uso de aprendizagem de máquina, comumente associado a reconhecimento ou classificação dos objetos.

A Figura 2.1 mostra a representação de cada nível de abstração para um sistema de adquisição de imagens. Inicialmente a imagem é obtida de um sensor CMOS em escala de cinza e segmentada para detecção de movimento. Observa-se que as etapas de segmentação, morfológica matemática, e componentes conexos podem ser implementadas em *hardware* para a aceleração do processamento.

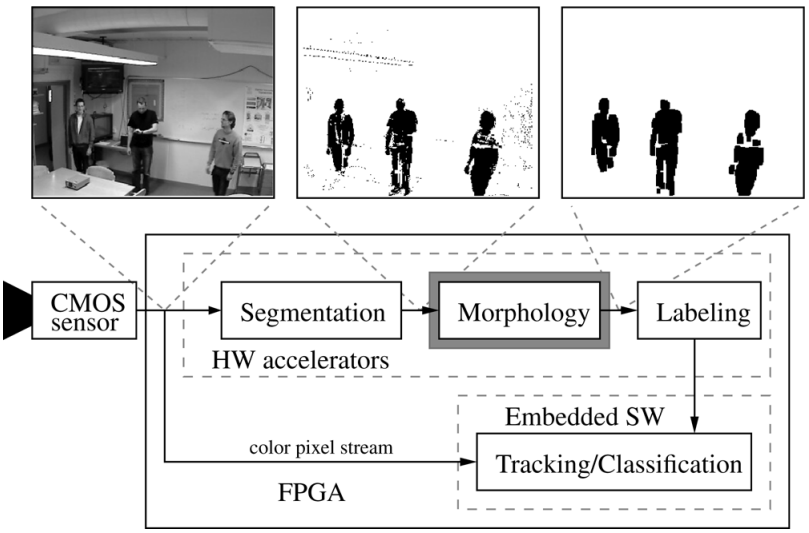


Figura 2.1: Níveis de abstração para o processamento de uma imagem (Hedberg, Kristensen e Owall 2008).

A seguir serão discutidos os três primeiros níveis de abstração, principalmente o foco de este trabalho centra-se na reconstrução morfológica que se encontra no nível mais baixo de abstração, funcionando como uma pré-filtragem para a etapa de segmentação de imagem.

2.2 FORMAÇÃO DE UMA IMAGEM

A formação de uma imagem é o resultado do processo de convolução entre uma imagem real (f) e uma função onde estão as informações físicas do sistema conhecida como *Função de Espalhamento de Ponto* ou *Point Spread Function* (PSF), adicionada a uma função de ruído (n) produto do sistema de adquisição n. A Equação 2.1 representa o modelo de formação de uma imagem.

$$g(\mathbf{x}) = h(\mathbf{x}) * PSF(\mathbf{x}) + n(\mathbf{x}), \tag{2.1}$$

onde $*$ é o operador de convolução.

O processo de convolução utiliza uma imagem h com dimensões $M \times N$ pixels, uma máscara PSF ou *kernel* com dimensões $m \times n$ pixels, e uma imagem de saída h com dimensões $M \times N$ pixels. A Equação 2.2 representa a operação de convolução (Pedrini e Schwartz 2008).

$$g(\mathbf{x}) = h(\mathbf{x}) * PSF(\mathbf{x}) = \sum_{i=-m/2}^{m/2} \sum_{j=-n/2}^{n/2} h(i,j) PSF(x-i, y-j). \tag{2.2}$$

A Figura 2.2 mostra o processo de convolução entre a imagem h e o *kernel* PSF. Neste processo, a posição do pixel $g(i,j)$ do resultado da convolução é a mesma posição do pixel que

está sendo tratado na imagem $h(i, j)$.

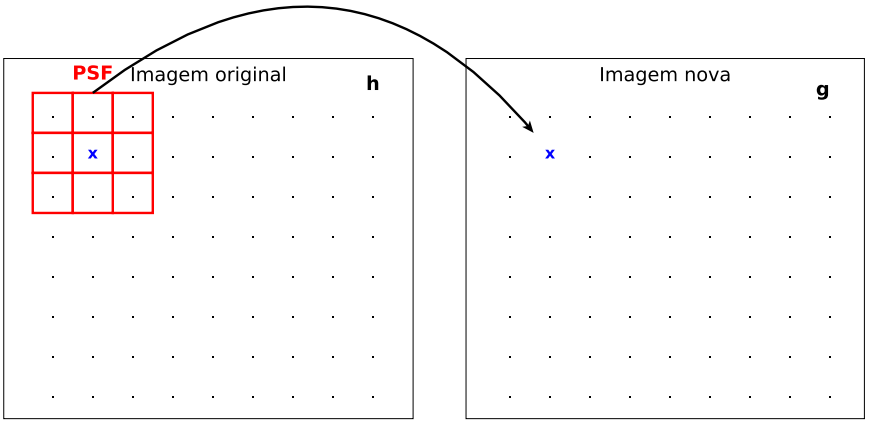


Figura 2.2: Processo de convolução (adaptado de (Nixon e Aguado 2012).

2.3 PRÉ-PROCESSAMENTO DE IMAGENS

O pré-processamento de imagens é o nível mais baixo de abstração, sendo usado para melhorar a qualidade das imagens para o processo de segmentação. Neste passo, a entrada é uma imagem em níveis de intensidade que possuem uma faixa entre [0, 255], e a imagem de saída possui a mesma característica (Sonka, Hlavac e Boyle 2014). As operações de pré-processamento podem ser divididas em três classes: (a) operações pixel a pixel, (b) operações em grupo de pixels e (c) operações baseadas em conectividade. Exemplos de operações pixel a pixel temos o cálculo do histograma, normalização, limiarização, equalização do histograma, entre outras (Nixon e Aguado 2012).

Por outro lado, nas operações baseadas em vizinhança temos a convolução no domínio da frequência e do espaço, deconvolução, filtros, a morfologia matemática, entre outras (Nixon e Aguado 2012). Basicamente, as operações baseadas em conectividade são aquelas onde os pixels de uma vizinhança têm uma relação entre si. Durante a etapa de adquisição das imagens, informação não desejada (ruído ou distorções) pode-se somar ao resultado da formação da imagem. O objetivo do processo de pré-processamento é a filtragem da imagem para o caso de ruído, ou restaurar a imagem para o caso de distorções.

Geralmente, o pré-processamento pode ser feito no domínio da frequência usando a *transformada de Fourier*, ou no domínio do espaço usando os pixels da imagem (vide Figura 2.3). Em ambos casos os resultados são equivalentes (Sonka, Hlavac e Boyle 2014).

No caso da filtragem, os métodos são divididos em dois grupos: (a) suavizado (*smoothing*) e (b) operadores de gradiente. Suavização é a atenuação de pequenas flutuações dentro da imagem (ex. ruído sal e pimenta). Por outro lado, os operadores de gradientes são baseados nas mudanças abruptas de intensidade entre duas regiões distintas (Pedrini e Schwartz 2008). Exemplos de

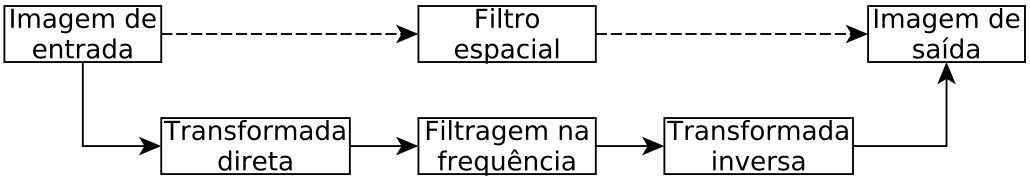


Figura 2.3: Filtragem no domínio do espaço ou no domínio da frequência (adaptado de (Sonka, Hlavac e Boyle 2014).

implementações em *hardware* para filtragem e processamento de imagens podem ser encontrados em (Mori, Llanos e Berger 2012, Sánchez-Ferreira, Mori e Llanos 2012, Sánchez-Ferreira et al. 2013).

Na matemática morfológica, o *kernel* usado para aplicar alguma operação é conhecido como *elemento estruturante*. A seguir são apresentadas as definições de vizinhança de um pixel e o tipo de notação matemática usada para sua descrição.

2.3.1 Vizinhança de um pixel

Segundo a definição de *vizinhança* dada em Pedrini (Pedrini e Schwartz 2008), temos:

Definição 2.3.1 *Um pixel f de coordenadas (x,y) possui quatro vizinhos horizontais e verticais, cujas coordenadas são $(x+1,y)$, $(x-1,y)$, $(x,y+1)$ e $(x,y-1)$. Esses pixels formam a vizinhança-4 de f, denotada $N_4(f)$.*

Definição 2.3.2 *Os quatro vizinhos diagonais de f são os pixels de coordenadas $(x-1,y-1)$, $(x-1,y+1)$, $(x+1,y-1)$ e $(x+1,y+1)$, formando a vizinhança-d ou $N_d(f)$.*

Definição 2.3.3 *A vizinhança-8 de f ou $N_8(f)$ é definida como:*

$$N_8(f) = N_8(f) \cup N_d(f). \tag{2.3}$$

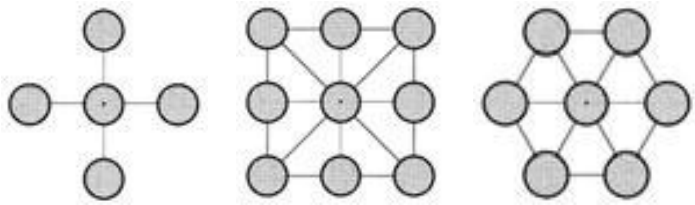
A notação adotada neste trabalho para expressar as coordenadas (x,y) dos pixels de uma imagem será $\mathbf{p} = (x, y)$. Assim, a notação usada para determinar se dois pixels são vizinhos é dada pela seguinte definição:

Definição 2.3.4 *O quadrado discreto $G \subset \mathbb{Z}^2 \times \mathbb{Z}^2$ fornece as relações de vizinhança entre os pixels $\mathbf{p}$ e $\mathbf{q}$, onde $\mathbf{p}$ é um vizinho de $\mathbf{q}$ se e somente se $(\mathbf{p}, \mathbf{q}) \in G$.*

2.3.2 Elemento estruturante

Um elemento estruturante (*Structuring Element*-SE) é um conjunto pequeno de valores usado para investigar a morfologia dos objetos de uma imagem n-dimensional (vide Figuras 2.4). O resultado da imagem que está sendo transformada depende da topologia do elemento estruturante,

as topologias para imagens 2-D mais comuns são: (a) diamante (*vizinhança-4*), (b) quadrado (*vizinhança-8*) e (c) hexágono (*vizinhança-6*) (vide Figuras 2.4a, 2.4b e 2.4c).



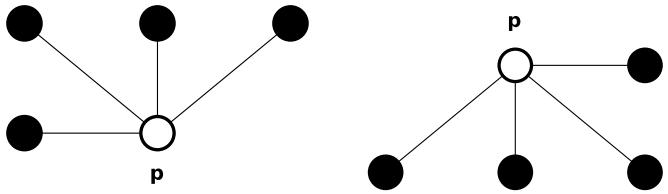
(a) 4-conectividade (b) 8-conectividade (c) 6-conectividade

Figura 2.4: Tipos de conectividade (Soille 1998).

De acordo à escolha do *elemento estruturante*, para topologias de diamante e quadrado, temos a seguinte definição:

Definição 2.3.5 *Considere-se $N_G^+(\mathbf{p})$ e $N_G^-(\mathbf{p})$ como os sub-conjuntos de $N_G(\mathbf{p})$. $N_G^+(\mathbf{p})$ é usado durante a varredura da imagem no sentido raster (esquerda até a direita e de acima para baixo) e $N_G^-(\mathbf{p})$ é usada durante a varredura no sentido anti-raster (direta até esquerda e de abaixo para acima).*

As Figuras 2.5a e 2.5b apresentam os sentidos para varredura raster e anti-raster para um pixel $\mathbf{p}$.



(a) (b)

Figura 2.5: $N_G^+(\mathbf{p})$ e $N_G^-(\mathbf{p})$ em 8-conectividade (Vincent 1992).

Outros tipos de SEs são: discos, segmento de linha, pares de pontos, octágonos, retângulos ou de formas arbitrarias (Vincent 1992). O formato do SE deve ser escolhido dependendo das propriedades geométricas dos objetos que são processados. Por exemplo, segmentos de linha são utilizados para extração de objetos lineares.

2.3.3 Relacionamentos básicos entre elementos de uma imagem

Além das operações mencionadas anteriormente, em (Pedrini e Schwartz 2008) tem-se as definições das relações entre os elementos dentro de uma imagem como conectividade, adjacência, caminho, e componentes conexos.

Definição 2.3.6 *Em imagens, dois pixels são considerados conexos se e somente se são vizinhos e possuem o mesmo valor, isto é, existe uma conectividade entre os pixels.*

Definição 2.3.7 *Um elemento f_1 é adjacente a um elemento f_2 se eles forem conexos de acordo com o tipo de vizinhança.*

Definição 2.3.8 *Um caminho na imagem do pixel (x_1,y_1) a um pixel (x_n,y_n) é uma sequência de pixels distintos com coordenadas (x_1,y_1), (x_2,y_2), ..., (x_n,y_n), onde n é o comprimento do caminho e (x_i,y_i) e (x_{i+1},y_{i+1}) são adjacentes, tal que $i = 1, 2, ..., n - 1$.*

Definição 2.3.9 *Um subconjunto de elementos C da imagem que são conexos entre si é chamado de componente conexo. Dois elementos f_1 e f_2 são conexos se existir um caminho de f_1 a f_2 contido em C.*

2.3.4 Filtragem no domínio espacial

A filtragem espacial refere-se ao processamento da imagem modificando um pixel baseado nas informações da dua vizinhança. O processo da filtragem é feito usando a operação de convolução, onde os valores dentro do *kernel* são chamados de *coeficientes de filtro* (Sonka, Hlavac e Boyle 2014). Um efeito não desejado da filtragem de uma imagem é o borramento das bordas da imagem conhecido como *padding* (Sonka, Hlavac e Boyle 2014).

Alguns exemplos de filtros são o filtro de mediana, filtro de meia, filtro de moda, filtro gaussiano, e filtragem com preservação das bordas, e os operadores de gradiente como Roberts, Laplace, Prewitt, Sobel, e Kirsch (Sonka, Hlavac e Boyle 2014). Na Figura 2.6 observa-se alguns exemplos de filtragem, entre eles tem-se filtro gaussiano (Figura 2.6b) e filtro da mediana (Figura 2.6c).

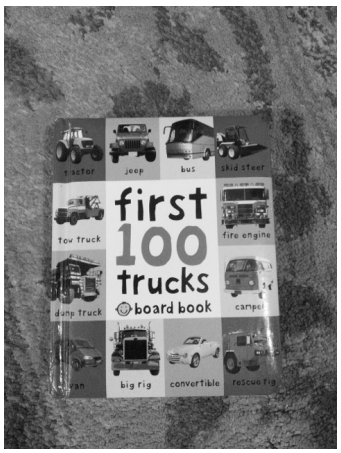
2.3.5 Morfologia matemática

As operações morfológicas podem ser realizadas utilizando a álgebra de *Minkowski* baseada na teoria de conjuntos: *união*, *intersecção* e *complementação*. Um operador morfológico é uma operação de mapeamento entre a imagem A e uma imagem B, onde B é o *elemento estruturante* (Pedrini e Schwartz 2008).

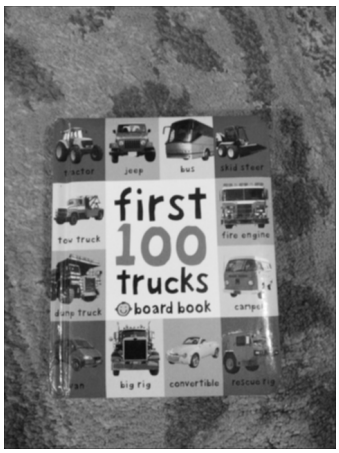
Uma imagem em $\mathbb{Z}^d$ é definida como a versão discreta do espaço Euclideano $\mathbb{R}^d$. O elemento fundamental de uma imagem bidimensional é o pixel. Em imagens em escala de cinza o pixel pode tomar valores entre 0 a $n - 1$ em níveis de intensidade, onde n representa o limite máximo dos níveis. No caso de imagens binárias, o pixel pode tomar o valor de 0 ou 1.

A morfologia matemática utiliza a teoria de conjuntos para representar a forma dos objetos, por exemplo, uma imagem A de coordenadas de pixels $\mathbf{p} = (x_n, y_n)$ que satisfazem uma determinada condição é expressado como A={$\mathbf{p}$| condição}.

Sejam A e B duas imagens binárias em $\mathbb{Z}^d$. As operações definidas entre as imagens A e B são: *união*, *interseção*, *traslação*, *reflexão*, *complemento* e *diferença* (Shih 2009, Pedrini e Schwartz 2008). As Equações (2.4a)–(2.4f) mostram as operações matemáticas básicas entre os dois conjuntos.



(a) Imagem original



(b) Filtro gaussiano

(c) Imagem com ruído sal e pimenta, e filtro da mediana

Figura 2.6: Exemplos de filtragem.

$$A \cup B = \{\mathbf{c} \in \mathbb{Z}^d | \mathbf{c} \in A \ \vee \mathbf{c} \in B\} \qquad (\textit{união}) \tag{2.4a}$$

$$A \cap B = \{\mathbf{c} \in \mathbb{Z}^d | \mathbf{c} \in A \ \wedge \mathbf{c} \in B\} \qquad (\textit{interseção}) \tag{2.4b}$$

$$(A)_c = \{\mathbf{c} \in \mathbb{Z}^d | \mathbf{c} = \mathbf{a} + \mathbf{b} \ \exists \ \mathbf{a} \in A\} \qquad (\textit{traslação}) \tag{2.4c}$$

$$\widehat{A} = \{\mathbf{w} | \mathbf{w} = -\mathbf{a} \ \forall \ \mathbf{a} \in A\} \qquad (\textit{reflexão}) \tag{2.4d}$$

$$\bar{A} = \{\mathbf{a} | \mathbf{a} \notin A\} \qquad (\textit{complemento}) \tag{2.4e}$$

$$A - B = \{\mathbf{c} \in \mathbb{Z}^d | \mathbf{c} \in (A \cap \bar{B})\} \qquad (\textit{diferença}) \tag{2.4f}$$

2.3.6 Operadores morfológicos

Os operadores morfológicos básicos da matemática morfológica são: dilatação, erosão, abertura e fechamento.

Definição 2.3.10 *A dilatação binária de J por B, denotada como $\delta_B(J)$ ou $J \oplus B$, é definida como a adição de Minkowski e mostrada na Equação 2.5.*

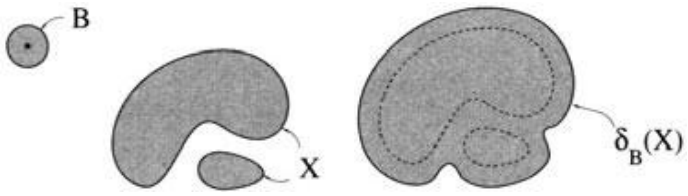
$$\delta_B(J) = \bigcup_{\mathbf{b} \in B} J_b \tag{2.5}$$

A dilatação de J por B define-se como o conjunto de todos os deslocamentos por um elemento $\mathbf{c}$ tais que J e B sobrepõem-se na origem de B como visto nas Equações 2.6 e 2.7.

$$\delta_B(J) = \{\mathbf{c} \in \mathbb{Z}^d | \mathbf{c} = \mathbf{j} + \mathbf{b}\, \exists\, \mathbf{j} \in J\ \wedge \mathbf{b} \in B\} \tag{2.6}$$

$$\delta_B(J) = \{c | \overline{J}_c \cap J \neq \emptyset\} \tag{2.7}$$

Isto é, existe pelo menos um pixel de B com valor 1 em comum com a imagem J. O processo de traslação de B sobre a imagem J é semelhante a operação de convolução. A Figura 2.7 ilustra a definição de dilatação, onde a Figura 2.7a apresenta o SE em forma de disco e a imagem original, enquanto que a Figura 2.7b é o resultado da dilatação.



(a) Imagem X e B (b) Erosão ϵ de X por um disco

Figura 2.7: Interpretação geométrica da operação de dilatação $\delta_B(X)$ (Soille 1998).

Definição 2.3.11 *A erosão binária de J por B, denotada por $\epsilon_B(J)$, ou $J \ominus B$, é definida como a substração de Minkowski, e mostrada na Equação 2.8.*

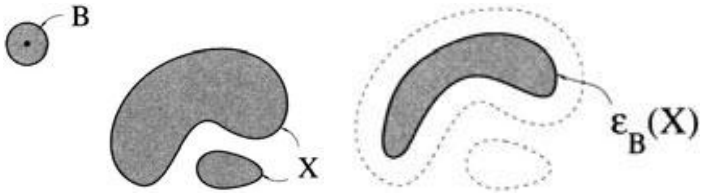
$$\delta_B(J) = \bigcap_{\mathbf{b} \in B} J_{-b} \tag{2.8}$$

A erosão de J por B define-se como o conjunto de todos os pontos $\mathbf{c}$ tais que B, trasladado por $\mathbf{c}$, é um subconjunto de J. Definições alternativas para a erosão são apresentadas nas Equações 2.9 e 2.10.

$$\epsilon_B(J) = \{\mathbf{c} \in \mathbb{Z}^d | B_c \subseteq J\} \tag{2.9}$$

$$\epsilon_B(J) = \{\mathbf{c} \in \mathbb{Z}^d | \mathbf{c} + \mathbf{b}\ \forall\ \mathbf{b} \in B\} \tag{2.10}$$

Na Figura 2.8 mostra-se a definição de erosão, onde a Figura 2.8a apresenta o SE em forma de disco e a imagem original, enquanto que a Figura 2.8b é o resultado da erosão. Observa-se uma perda de informação da imagem original relacionada a imagem B.



(a) Imagem X e B (b) Erosão ϵ de X por um disco

Figura 2.8: Interpretação geométrica da operação de erosão $\epsilon_B(X)$ (Soille 1998).

A operação de *erosão* pode gerar efeitos não desejados durante a transformação da imagem que esteja sendo tratada. Por exemplo, a erosão não somente remove estruturas assim como também encolhe outras. Aplicando uma operação de dilatação, após uma erosão, pode-se reverter o encolhimento das estruturas. Esta sequencia de operações conhece-se como *abertura*. A operação que segue a sequencia de dilatação e erosão é nomeada como *fechamento*.

Definição 2.3.12 *A abertura de uma imagem J por B, denotada por $\gamma_B(J)$, ou $J \circ B$, e definida na Equação 2.11.*

$$\gamma_B(J) = \delta_B[\epsilon_B(J)] \tag{2.11}$$

Definição 2.3.13 *O fechamento de uma imagem J por B, denotada por $\phi_B(J)$, ou $J \bullet B$, e definida na Equação 2.12.*

$$\phi_B(J) = \epsilon_B[\delta_B(J)] \tag{2.12}$$

Definição 2.3.14 *A transformada de acerto-ou-erro é usada para detecção de características de objetos. Seja J uma imagem binária, e B_1 e B_2 os elementos estruturantes, onde $B_1 \subseteq J$ e $B_2 \subseteq \bar{J}$, a transformada acerto-ou-erro $J \circledast (B_1, B_2)$ é definida na Equação 2.13.*

$$J \circledast (B_1, B_2) = \epsilon_{B_1}(J) \cap \epsilon_{B_2}(\bar{J}) \tag{2.13}$$

Isto significa que $J \circledast (B_1, B_2)$ contem todos os pontos que simultaneamente B_1 e B_2 coincidem com J e $\bar{J}$, respetivamente.

2.3.7 Morfologia matemática em escala de cinza

As operações morfológicas de *dilatação*, *erosão*, *abertura* e *fechamento* para imagens binárias possuem extensões para imagens em escala de cinza com valores entre $[I_L, I_H]$. Igualmente, o elemento estruturante também pode assumir valores em escala de cinza. Imagens em escala de cinza podem ser representadas como uma superfície topográfica; ou seja, a imagem é considerada como se fosse em 3-D (vide Figura 2.9).

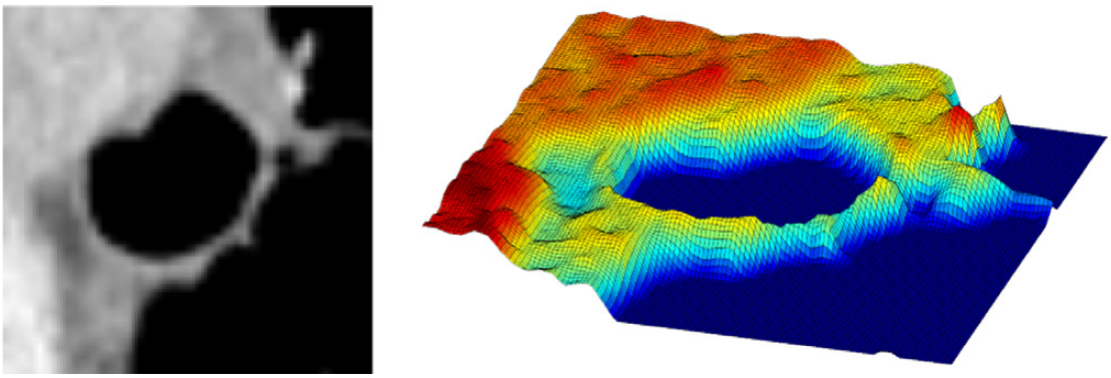


Figura 2.9: Topografia de uma imagem em escala de cinza (Smistad et al. 2015).

Onde a altura da superfície corresponde a intensidade da imagem em 2D.

Definição 2.3.15 *Seja J uma imagem em escala de cinza em $\mathbb{Z}^2$. A dilatação de uma imagem J por B, denotada por $\delta_B(J)$, é definida pela Equação 2.14.*

$$[\delta_B(J)](\mathbf{x}) = \max_{\mathbf{b}\in B} J(\mathbf{x}+\mathbf{b}), \tag{2.14}$$

ou seja, o pixel dilatado é o máximo valor de uma região definida pelo elemento estruturante com origem em $\mathbf{x}$.

Definição 2.3.16 *A erosão de uma imagem J por B, denotada por $\epsilon_B(J)$, é definida pela Equação 2.15.*

$$[\epsilon_B(J)](\mathbf{x}) = \min_{\mathbf{b}\in B} J(\mathbf{x}+\mathbf{b}), \tag{2.15}$$

ou seja, o pixel dilatado é o mínimo valor de uma região definida pelo elemento estruturante com origem em $\mathbf{x}$.

2.4 TRANSFORMAÇÕES GEODÉSICAS

Todas as operações morfológicas discutidas anteriormente usam uma imagem de entrada e um SE especifico. O objetivo das transformações geodésicas é transformar uma imagem conhecida como máscara (*mask*) através de uma imagem conhecida como marcador (*marker*) usando dilatações e erosões (Soille 1998). As transformações geodésicas são processos iterativos até estabilizar a imagem que está sendo processada; ou seja, até que nenhum pixel seja modificado (Vincent 1992).

A reconstrução morfológica é parte de um conjunto de operadores geodésicos usada em matemática morfológica para extrair componentes conexos de um objeto que foram marcados por

a imagem "marcadora". A reconstrução morfológica pode ser usada tanto para o caso binário como para imagens em escala de cinza (Vincent 1992).

2.4.1 Reconstrução morfológica por dilatação

Seja I a imagem máscara com $I \in \mathbb{Z}^2$, J a imagem marcador com $J \in \mathbb{Z}^2$, e $J \leq I$. Inicialmente, o marcador é dilatado e forçado para permanecer embaixo da máscara; ou seja, a máscara é o limite para as propagações do marcador. A dilatação geodésica para uma iteração do marcador J com respeito a máscara I é denotada por $\delta_I^1(J)$, e definida como o valor mínimo pixel a pixel entre a máscara e a dilatação $\delta_B^1(J)$, descrita na Equação 2.16.

$$\delta_I^{(1)}(J) = \delta_B^{(1)}(J) \wedge I \tag{2.16}$$

O processo iterativo da dilatação geodésica para $i-$vezes pode ser expressada pela Equação 2.17.

$$\delta_I^{(i)}(J) = \delta_I^{(1)}[\delta_I^{(i-1)}(J)], \tag{2.17}$$

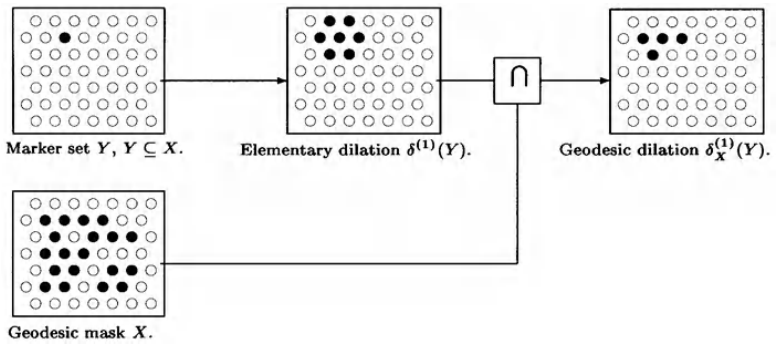
onde $\delta_I^{(0)}(J) = J$. A equação 2.17 é a reconstrução morfológica por dilatação da imagem I sob J. Expressado de uma forma alternativa na Equação 2.18.

$$R_I^{\delta}(J) = \delta_I^{(i)}(J), \tag{2.18}$$

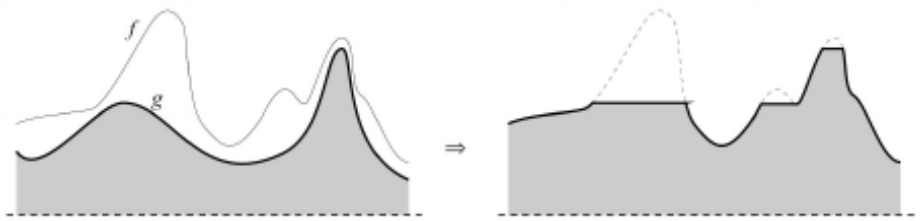
onde i é tal que $\delta_I^{(i)}(J) = \delta_I^{(i+1)}(J)$.

Na reconstrução morfológica por abertura, o marcador J é normalmente obtido por abertura geodésica γ_B. A Figura 2.17 mostra o processo de dilatação geodésica e reconstrução morfológica usando imagens binárias e em escala de cinza. A princípio, na imagem 2.10a, o pixel na imagem marcadora (Y) é propagado até o próximo vizinho e depois é feito o processo de dilatação geodésica usando a imagem máscara (X), onde o resultado são os pixels que são comuns tanto a X como a Y.

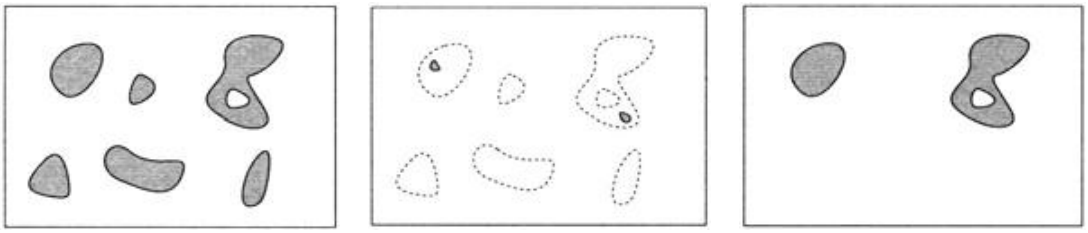
Na Figura 2.10b, o algoritmo de dilatação geodésica é aplicado para uma imagem unidimensional, onde a máscara f limita a propagação do marcador g. O processo de reconstrução morfológica por dilatação é apresentado nas Figuras 2.10c, 2.10d e 2.10e. Um exemplo de uma imagem marcadora pode ser encontrado na Figura 2.10d são escolhidos os objetos que serão preservados da imagem 2.10c. Aplicando sucessivas dilatações usando as imagens I e J obtêm-se a imagem 2.10e.



(a) Dilatação geodésica de uma imagem binária (Soille 1998).



(b) Dilatação geodésica de uma imagem 1-D em escala de cinza (Vincent 1992).



(c) Particulas I (d) Sementes J, $J \subseteq I$ (e) $R_I^{\delta}(J)$

Figura 2.10: Reconstrução morfológica (Soille 1998)

2.4.2 Reconstrução morfológica por erosão

A reconstrução por erosão geodésica, semelhante a reconstrução por dilatação geodésica com $J \leq I$, é um processo iterativo de sucessivas erosões geodésicas da imagem J usando a imagem I como limite, dada pela Equação 2.19.

$$\epsilon_I^{(1)}(J) = \epsilon^{(1)}(J) \wedge I, \tag{2.19}$$

O processo iterativo da Equação 2.19 da erosão geodésica para $n-$vezes pode ser expressada como:

$$\epsilon_I^{(n)}(J) = \epsilon_I^{(1)}[\epsilon_I^{(n-1)}(J)], \tag{2.20}$$

onde $\epsilon_I^{(0)}(J) = J$. A equação 2.20 é a reconstrução morfológica por erosão da imagem I sob J. Expressado de uma forma alternativa pode ser definido usando a Equação 2.21.

$$R_I^{\epsilon}(J) = \epsilon_I^{(n)}(J), \tag{2.21}$$

onde n é tal que $\epsilon_I^{(n)}(J) = \epsilon_I^{(n+1)}(J)$. Na reconstrução morfológica por fechamento, o marcador J é normalmente obtido por fechamento geodésico ϕ_B.

As operações morfológicas são amplamente implementadas para diversas plataformas *software* e *hardware*, exemplos podem ser encontrados em (Heijmans e Ronse 1990, Vincent 1995, Dokládal e Dokládalová 2011, Bartovskỳ et al. 2015, Youkana et al. 2017). No entanto, a maioria das implementações *software* e *hardware* visam encontrar um balanço entre complexidade computacional e latência, e nos casos iterativos usando estruturas de dados como a tabela ou fila, os algoritmos são rodados numa iteração.

2.5 SEGMENTAÇÃO DE IMAGENS

O principal objetivo do processo de segmentação de uma imagem é dividi-la em áreas ou objetos separados do fundo. A segmentação de imagens pode ser dividida em três grupos: conhecimento global da imagem (histograma de características), baseada em bordas, e baseada em regiões (ex. brilho e textura) (Sonka, Hlavac e Boyle 2014).

2.5.1 Limiarização

A limiarização é um processo simples de segmentação, onde muitos objetos ou regiões da imagem são caracterizados por valores constantes de nível de intensidade. Neste processo, a escolha de uma valor de intensidade ou valor limiar é suficiente para diferenciar os objetos do fundo da imagem. Limiarização é um algoritmo de baixa complexidade computacional e rápido. No entanto, esse método possui algumas desvantagens como conhecimento *a priori* do valor do limiar e a limiarização global (Sonka, Hlavac e Boyle 2014). A função de limiarização de uma imagem é apresentada na Equação 2.22.

$$g(x,y) = \begin{cases} 1 & \text{Se } f(x,y) \geq T, \\ 0 & \text{Se} f(x,y) < T, \end{cases} \tag{2.22}$$

$$\tag{2.23}$$

onde T é o valor limiar, f é a imagem original e g é a imagem com representação binária. A Figura 2.11 apresenta o processo de limiarização, onde a Figura 2.11a é a imagem original e a Figura 2.11b é a imagem binarizada usando o *algoritmo de Otsu* (Nixon e Aguado 2012).

(a) Imagem original (b) Imagem binária

Figura 2.11: Processo de limiarização.

2.5.2 Segmentação baseada em bordas

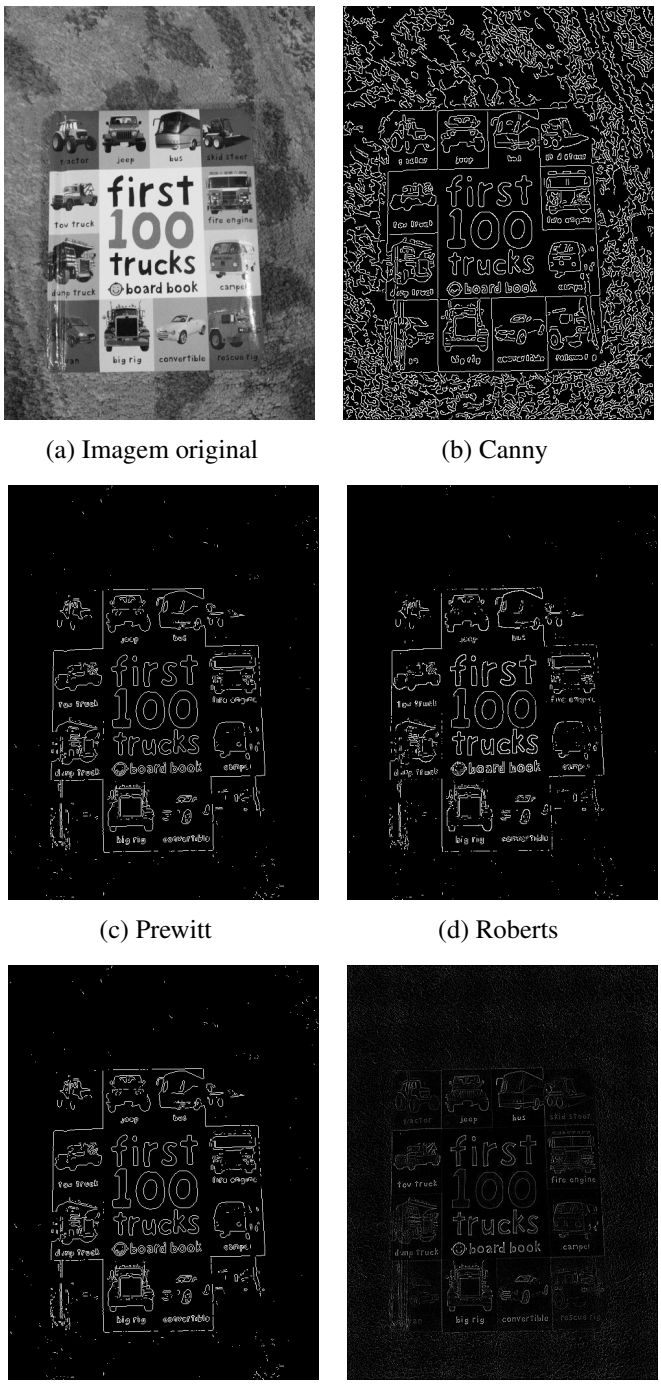
Durante o processamento de uma imagem, a importância de determinar a forma geométrica dos objetos ou regiões vai determinar a acurácia do resultado final (etapa do modelo racional). A segmentação baseada em bordas detecta descontinuidades ou similaridades dos objetos (Pedrini e Schwartz 2008). O problema mais comum da segmentação é causada por ruído ou informação inadequada que afeta a imagem processada, originando detecção de bordas onde não há presencia destas e originando uma *sobre-segmentação* da imagem (Sonka, Hlavac e Boyle 2014). Exemplos de algoritmos baseados em segmentação de bordas são apresentados a seguir:

- *Operadores de gradiente:* são classificados em detectores de primeira ordem (Roberts, Prewitt, Sobel, Canny) e segunda ordem (laplaciano) (Nixon e Aguado 2012). A Figura 2.12 apresenta os detectores de bordas como canny 2.12b, prewitt 2.12c, sobel 2.12d, roberts 2.12e, e laplaciano 2.12f.

- *Contornos ativos ou contornos deformáveis:* são os algoritmos que detectam curvas correspondentes a objetos. Nesta categoria, alguns exemplos são o algoritmo de *snake*, transformada da distância, esqueletização, onde o resultado do algoritmo é uma curva que encerra o objeto de interesse (Nixon e Aguado 2012).

- *Transformada Hough:* esse algoritmo serve para detectar figuras geométricas como segmentos de reta, circunferências e elipses dentro de uma imagem (Szeliski 2010).

- *Detecção de bordas como busca de grafos:* neste algoritmo, os pixels da imagem são considerados como vértices e as arestas são consideradas como funções custo. O objetivo deste algoritmo é detectar a borda de um objeto com o menor valor da função custo.



(a) Imagem original (b) Canny

(c) Prewitt (d) Roberts

(e) Sobel (f) Laplaciano

Figura 2.12: Detectores de bordas.

2.5.3 Segmentação baseada em regiões

Os algoritmos baseados em regiões conseguem detectar os objetos sem precisar encontrar as bordas. Nesta abordagem, é importante que as regiões apresentem homogeneidade baseadas no nível de intensidade em escala de cinza, color, textura, forma, modelo, etc (Sonka, Hlavac e Boyle 2014). O objetivo dessa classe de algoritmos é a detecção dos objetos e/ou regiões quando não são facilmente detectáveis de objetos diferentes ou do fundo da imagem (Sonka, Hlavac e Boyle

2014, Pedrini e Schwartz 2008).

- *Crescimento de regiões:* a abordagem neste algoritmo é a suposição de que cada pixel da imagem é uma região. Os pixels que representam regiões de interesse são selecionados e chamados de sementes. A partir dessas sementes, os pixels com certa similaridade são agregados para essa região.
- *Divisão de regiões:* é a técnica oposta ao crescimento de regiões, onde o processo inicia-se com regiões já formadas e divide as regiões não homogêneas de áreas menores.
- *Divisão e união regiões:* consiste na combinação do crescimento e divisão de regiões. Neste algoritmo, a divisão de uma imagem em regiões é feita usando uma representação piramidal. As sub-regiões são representadas em forma de quadros, onde cada quadro pode ser interligado com outros quadros usando uma arvore de hierarquia (vide Figura 2.13).

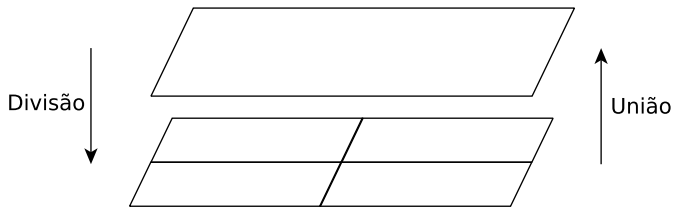


Figura 2.13: União e divisão de regiões (adaptado de (Sonka, Hlavac e Boyle 2014).

- *Segmentação por divisor de águas:* a transformada está baseada nos conceptos de linhas divisoras de águas e bacias hidrográficas ou reservatórios usados em topografia. Nesta abordagem, a topografia é definida pelo nível de intensidade presentes na imagem. Os reservatórios são homogêneos e os pixels que os compõem são conexos com um vale. Cada reservatório representa uma região ou um objeto da imagem, diferente dos métodos mostrados anteriormente (Sonka, Hlavac e Boyle 2014). A transformada por divisor de águas pode ser apresentada no contexto da matemática morfológica (Sonka, Hlavac e Boyle 2014).

2.5.3.1 Transformada por divisor de regiões

As propostas para segmentação de imagens baseadas em bordas são suscetíveis a sofrer sobre-segmentação por causa da dificuldade de separar os objetos do fundo, ruído presente na imagem e descontinuidade nas bordas. Uma proposta eficiente para plataformas *software* do algoritmo divisor por águas usando estruturas de dados foi implementada por Vincent e Soille (1991) (Vincent e Soille 1991).

O algoritmo está baseado em quatro passos: (a) considerar a imagem como uma topografia, (b) ordenamento dos pixels em ordem crescente do nível de intensidade, (c) uma inundação da vizinhança de um pixel considerado como mínimo regional, e (d) etiquetamento de cada reservatório encontrado (Vincent e Soille 1991). A proposta de Vincent conseguiu melhorar o

desempenho na área da segmentação de imagens reduzindo a ordem do tempo de processamento de horas para segundos (Sonka, Hlavac e Boyle 2014).

A transformada por divisor de águas, ou segmentação por divisor de águas, possui o mesmo conceito da Figura 2.9, ou seja, a imagem é vista como uma imagem topográfica com vários mínimos regionais (vide 2.14). Os mínimos regionais criam reservatórios (*catchment basins*), que podem ser preenchidos por simulação de chuva ou por simulação de inundação, separados por linhas divisoras chamadas de divisões de água (*watersheds*) (Vincent e Soille 1991).

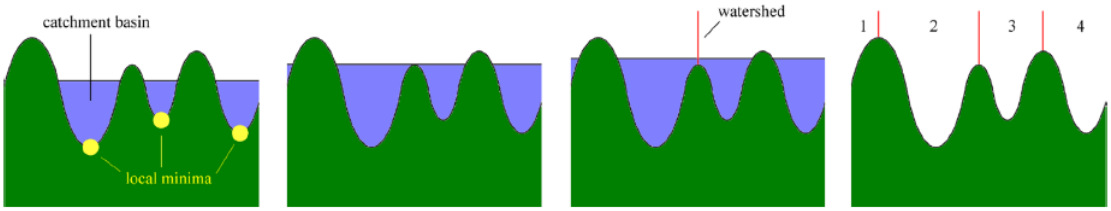


Figura 2.14: Representação topografica de uma imagem em escala de cinza (Soille 1998)

Segundo (Vincent e Soille 1991) e (Soille e Vincent 1990), algumas definições do algoritmo por divisor de águas são:

Definição 2.5.1 *O* máximo regional M *de uma imagem em escala de cinza é um componente conexo de pixels com um valor* h *(planície de altitude* h*), assim que todo pixel na vizinhança de M possui um valor estritamente menor que* h*.*

Definição 2.5.2 *O* máximo local *é definido como o pixel* $\mathbf{p}$ *de uma imagem* I *em escala de cinza se e somente se para todo pixel* q *vizinho de* p*,* $I(\mathbf{p}) \geq I(\mathbf{q})$*.*

Definição 2.5.3 *Um mínimo* M *de altitude* h *da imagem* I *é um planalto de pixels conexos com valor* h*, no qual pixels dos limites externos possuem uma altura maior que* h*.*

Definição 2.5.4 *O reservatório* C *associado com um mínimo* M *é o conjunto de pixels da vizinhança de* $\mathbf{p}$ *tal que uma gota de água caindo desde* $\mathbf{p}$ *desce ao longo do relevo, seguindo um caminho desde* $\mathbf{p}$*, e eventualmente alcança* M*. A notação do reservatório pode ser observada na Equação 2.24.*

$$C_h(M) = \{\mathbf{p} \in C(M) | I(\mathbf{p}) \leq h\} \tag{2.24}$$

A inundação de um reservatório possui a analogia de inundação de uma lagoa, começando desde uma mínima altitude até uma altura chamada como barragem (Vincent e Soille 1991). A definição formal é apresentada a seguir:

Definição 2.5.5 *Seja uma imagem* I *em escala de cinza, e* h_{min} *é um valor de* I *em* $\mathbb{Z}^2$*. Similarmente, seja* h_{max} *o maior valor de* I *em* $\mathbb{Z}^2$*. O processo de inundação é similar ao processo de limiarização de* I *por um valor* h *mostrado na Equação 2.25:*

$$T_h(I) = \{\mathbf{p} \in \mathbb{Z}^d | I(\mathbf{p}) \leq h\} \tag{2.25}$$

O algoritmo proposto por Vincent e Soille resume-se na seguinte descrição:

- Varredura da imagem em uma ordem determinada, *raster* e *anti-raster*.
- O algoritmo não possui um número fixo de iterações. No entanto, quando poucos pixels são modificados, os pixels candidatos para gerar mudanças na imagem são armazenados em um fila (FIFO). Neste processo, são satisfeitas duas condições: (a) acesso randômico aos pixels da imagem e (b) acesso direto à vizinhança de um pixel.
- Os pixels dentro da fila são os mínimos regionais ordenados de forma crescente.
- Etiquetamento (números ou cores) dos reservatórios e dos pixels que são conexos a eles.

2.6 DESCRIÇÃO DE OBJETOS

Um passo posterior ao processo de detecção de bordas ou figuras geométricas, e à segmentação de imagens, é a *extração e analise de características*. A extração de características é referenciado a detecção da quantidade de elementos (pixels) que compõem um objeto, à descrição das formas e os objetos encontrados. Por exemplo, a detecção de características faciais do ser humano como olhos, ouvidos, nariz, etiquetamento dos pixels do um objeto dentro de uma imagem, etc (Sonka, Hlavac e Boyle 2014).

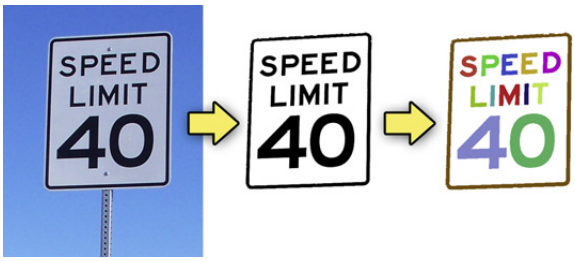


Figura 2.15: Exemplo de segmentação e detecção de componentes conexos de uma imagem(Št, Beneš et al. 2011).

Outra forma de descrição de um objeto é através de seu esqueleto (Rosenfeld e Pfaltz 1966). Diferente da anatomia do ser humano, um esqueleto de um objeto está localizado no eixo central do objeto. O esqueleto é equidistante desde o eixo até a borda do objeto, esse processo de transformação pode ser calculado usando a *transformada da distância* (vide Figura 2.16).

2.6.1 Componentes conexos

O algoritmo de componentes conexos (*Connected Component Labeling-CCL*) é usado para detectar regiões (objetos) que possuem uma conexão entre si. O algoritmo CCL é usado como pré-processamento para reconhecimento de padrões, visão computacional, e processamento de

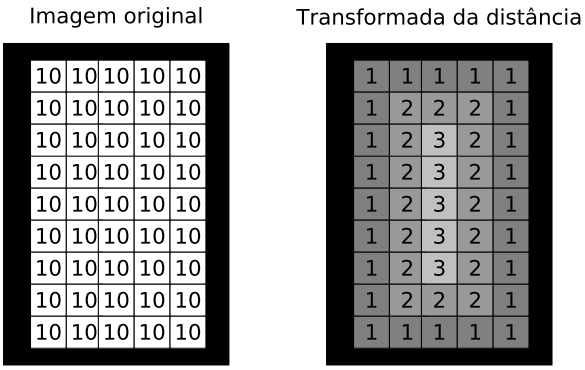


Figura 2.16: Exemplo do cálculo da transformada da distância de um objeto (adaptado de (Nixon e Aguado 2012)).

imagens (Št, Beneš et al. 2011) (vide Figura 2.15. O algoritmo CCL pode ser usado para descrever propriedades dos objetos como área, centro de masa, perímetro, etc.

Inicialmente propostos para imagens binárias, essa classe de algoritmos foram estendidos para imagens em escala de cinza. O algoritmo de componentes conexos foi proposto por *Rosenfeld* e *Pfaltz* (1966) para processamento de imagens que envolvam operações locais usando uma vizinhança de pixels (Rosenfeld e Pfaltz 1966). Semelhante ao processo de convolução, o processo de escaneamento do CCL sobre a imagem é semelhante ao processo de filtragem de uma imagem (Rosenfeld e Pfaltz 1966).

Na Figura, 2.15 pode-se observar o processo de segmentação de uma imagem (imagem binária), e o processo de componentes conexos onde cada letra é etiquetada com uma cor diferente. O algoritmo CCL percorre a imagem procurando conexões entre elementos, e os objetos são diferenciados usando etiquetas (números ou cores) (Št, Beneš et al. 2011). Uma modificação do algoritmo CCL baseada em uma arvores de busca para cada objeto, é o algoritmo *union-find* (Št, Beneš et al. 2011). Os algoritmos de componentes conexos usam estruturas de dados como tabelas ou filas para armazenar as etiquetas de cada objeto encontrado.

Esse algoritmo foi implementado para processar a imagem em duas varreduras: (a) uma varredura para a identificação das rotulações, onde cada rotulação de um pixel pode ser propagada para um pixel vizinho, e (b) outra varredura onde os conflitos entre rotulações vizinhas são resolvidos (Soille 1998). O algoritmo de componentes conexos pode ser divididos em duas classes: sequenciais e paralelos.

No caso sequencial, o algoritmo em cada passo usa os resultados obtidos no passo anterior (realimentação) gerando uma dependência de dados até que o processo de escaneamento finaliza. Dependendo do sentido de escaneamento da imagem, tem-se varreduras no sentido *raster* e *anti-raster* (Rosenfeld e Pfaltz 1966).

Um exemplo de componentes conexos para regiões uniformes em imagens em escala de cinza é apresentado no Algoritmo 1. Seja uma imagem f em escala de cinza com $f \in \mathbb{Z}^2$, o algoritmo de etiquetamento começa com uma imagem f e uma imagem L que é a imagem que possui os

Algoritmo 1: Componentes conexos para escala de cinza baseado no algoritmo Rosenfeld (Soille 1998).

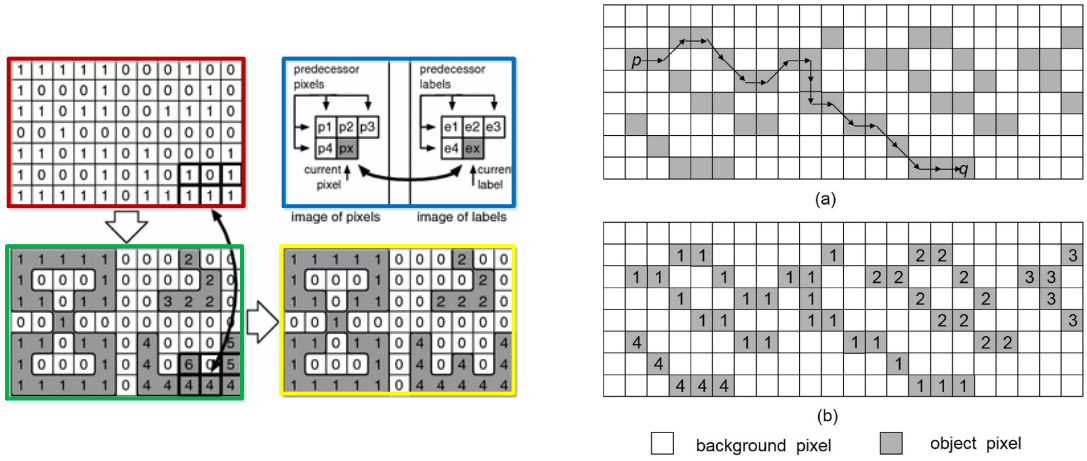
Entrada: `f`: imagem em escala de cinza, `L`: imagem de etiquetas
Saída: `L`: imagem de etiquetas

```
1  início
2      para todo p ∈ ℤ² faça
3          se L(p) = 0 então
4              t ← f(p)
5              lval ← lval + 1
6              L(p) ← lval
7              para todo p′ ∈ N_G(p) faça
8                  se f(p′) = t então
9                      fifo_add(p′)
10                     L(p′) ← lval
11                 fim
12             fim
13             enquanto fifo_empty()=false faça
14                 q ← fifo.first()
15                 para todo q′ ∈ N_G(q) faça
16                     se f(q′) = t ∧ L(q′) = 0 então
17                         fifo_add(q′)
18                         L(q′) ← lval
19                     fim
20                 fim
21             fim
22         fim
23     fim
24 fim
```

componentes conexos da imagem f.

O algoritmo começa usando a vizinhança N_G do pixel $\mathbf{p}$ (ou as vizinhanças N_G^+ ou N_G^-). Durante o primeiro escaneamento, cada pixel de um objeto recebe uma rotulação ($lval$) e ao mesmo tempo sua vizinhança é examinada para calcular pixels sementes ($\mathbf{p}'$) para propagar essa rotulação e resolver o conflito de um mesmo objeto possuir diversas rotulações. Na linha 13 do Algoritmo 1, através de uma fila e seguindo uma condição, os pixels sementes propagam suas rotulações para os pixels que pertencem a sua vizinhança.

As Figuras 2.17a e 2.17b apresentam exemplos de componentes conexos usando imagens binárias. Para imagens em escala de cinza obtêm-se resultados semelhantes. Na Figura 2.17a parte superior esquerda (bloco vermelho), tem-se a imagem binária que vai ser processada para contabilizar o número de elementos conectados (*componentes conexos*) presentes na imagem; do lado dessa imagem, tem-se o elemento estruturante (bloco azul). O resultado da primeira varredura são as rotulações parciais (bloco verde), e a figura da parte inferior direita (bloco amarelo) mostra o resultado das rotulações finais. Neste exemplo são contabilizados o número de elementos presentes na imagem, gerando três elementos, embora apareça uma rotulação com valor de 4.



(a) Algoritmo *Rosenfeld* (Cabaret, Lacassagne e Etiemble 2016).

(b) *Caminho* do pixel $\mathbf{p}$ até o pixel $\mathbf{q}$ (He et al. 2017)

Figura 2.17: Componentes conexos para imagens binárias

A Figura 2.17b apresenta o *caminho* da propagação do pixel p até o pixel q, essa *propagação* depende do elemento estruturante escolhido. Geralmente, os elementos estruturantes usados são: diamante (vizinhança-4) e quadrado (vizinhança-8).

2.7 DISCUSSÃO GERAL SOBRE OS ALGORITMOS DE PROCESSAMENTO DE IMAGENS

Geralmente, o processamento de imagens é aplicado para o analise de imagens em 2-D e em 3-D. Dependendo do problema a resolver podem ser usados os três níveis de abstração apresentados. Por exemplo, em (Wang, Lin e Miller 2016) e (Wang, Lin e Miller 2015), Wang, Lin e Miller são analisadas imagens tomográficas em 3-D para analise dos minerais que são agrupados em forma de partículas. As imagens em (Wang, Lin e Miller 2016) são segmentadas usando a reconstrução morfológica como ferramenta auxiliar para o algoritmo por divisor de águas. O fluxo de projeto para o analise das imagens é mostrado a seguir:

1. Adquisição das imagens de raios X em três dimensões usando um tomografo de raios X de alta velocidade e um microtomógrafo de alta resolução para raios X.
2. Filtragem do ruído presente na imagem.
3. Limiarização da imagem para separar as partículas do fundo da imagem.
4. A transformada da distância é aplicada à imagem binária gerada na etapa anterior, neste passo através dessa transformada é gerada uma imagem em escala de cinza.
5. A imagem marcadora é gerada calculando os regionais máximos através da transformada acerto-ou-erro.

6. Finalmente, usando a transformada por divisor de águas seguida por marcador, as partículas são etiquetadas e analisadas.

A Figura 2.18 mostra o processo de caracterização de um mineral em 3-D. A etapa do analise por semivariograma pode ser encontrado em (Journel e Huijbregts 1978). Neste contexto, a combinação da transformada por divisor de águas e a matemática morfológica são uma ferramenta de grande utilidade, não somente para tomografia senão também para analise de imagens medicas, biológicas, páginas de documentos e outras aplicações que envolvam operações com imagens (Vincent 1995).

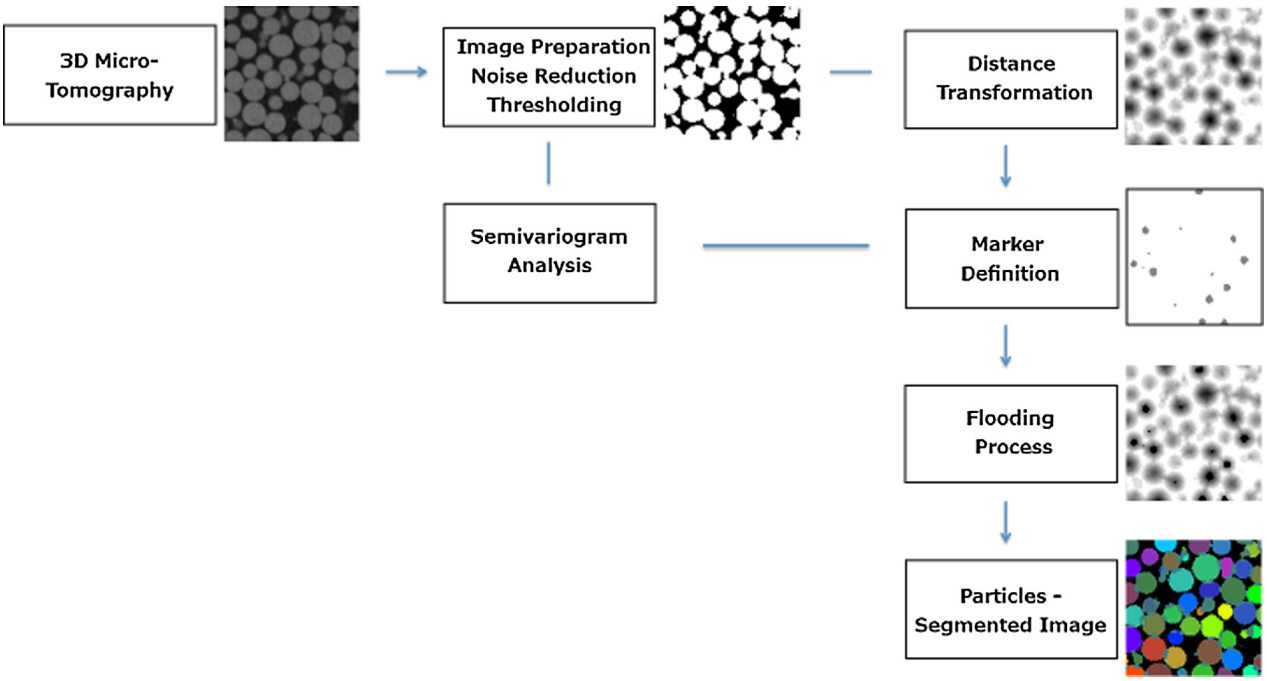


Figura 2.18: Exemplo de segmentação de uma tomografia (Wang, Lin e Miller 2015).

2.8 CONCLUSÕES

Os algoritmos de componentes conexos, transformada por divisor de águas e reconstrução morfológica são amplamente usados para detecção de objetos ou caraterísticas relevantes dentro de uma imagem. O primeiro passo deste processo é isolar as zonas desejadas da imagem, por exemplo ressaltar os pixels de maior ou menor intensidade, isto pode ser conseguido através da reconstrução morfológica.

Seguidamente cada elemento da imagem é identificado usando a transformada por divisor de águas. Dentro das estrategias mais destacadas encontram-se: (a) divisão da imagem em sub-imagens para processar cada uma em paralelo, (b) uso de um pré-processamento eficiente, (c) uso de filas ou tabelas para terminar o processamento do algoritmo e (d) otimização do espaço (no caso das FPGAs) e/tempo de execução. Finalmente, o processo de etiquetamento dos objetos

encontrados pode ser implementado via algoritmo de componentes conexos.

Diante do exposto, os algoritmos que seguem a metodologia de varredura de uma imagem podem ser otimizados usando estruturas de dados como filas ou tabelas.

3 ALGORITMOS MORFOLÓGICOS E TÉCNICAS PARA OTIMIZAR O DESEMPENHO

O desempenho de um algoritmo morfológico pode ser definido usando três critérios: velocidade de processamento, acurácia e flexibilidade. No caso de plataformas *hardware*, a latência e a número de operaçãoes (ciclos de relógio) para o algoritmo chegar na solução devem-se encontrar em faixas que sejam aceitáveis. Sobre a acurácia, o algoritmo deve fornecer resultados com bons valores de qualidade, por exemplo, evitando borramento na imagem por causa da metodologia de varredura da imagem (relatada às bordas da imagem). Neste sentido, uma arquitetura apropriada deve apresentar uma grande flexibilidade para adaptabilidade das características do elemento estruturante, transformações para algoritmos semelhantes, mudanças no tamanho da imagem processada, e permitir o uso de métricas de qualidade.

Adicionalmente, este capítulo expõe o problema de otimização para um tipo de algoritmo a ser implementado em *hardware* (o *Fast Hibrid Algorithm*- FH), que será solucionado mediante o uso de Máquinas de Vetores de Suporte (SVM).

3.1 CONSIDERAÇÕES INICIAIS

Os algoritmos morfológicos como o algoritmo por divisor de águas, reconstrução morfológica, transformada da distância, componentes conexos, podem ser classificados em quatro tipos, eles são: (a) paralelo, (b) sequencial, (c) fila normal ou fila de prioridade, e (d) um algoritmo híbrido entre os algoritmos (b) e (c) (Vincent 1991). Dentre estes, o algoritmo híbrido apresenta uma rápida convergência no processamento da imagem, apresentando-se como o de melhor desempenho em tempo de execução.

Inicialmente, foram propostas várias abordagens de algoritmos híbridos que melhoraram o desempenho dos algoritmos paralelo e sequencial, entre eles temos os trabalhos desenvolvidos para processar diretamente os pixels da imagem e os algoritmos baseados em grafos (Vincent 1989, Vincent 1991, Heijmans e Vincent 1992, Yeong et al. 2009), onde um pixel é representado por um nó. Essas duas abordagens permitiram melhorar o desempenho de algoritmos como o divisor por águas (Soille e Vincent 1990, Trieu e Maruyama 2006, Trieu e Maruyama 2008, Parvati, Rao e Das 2008, Körbes et al. 2009, Osma-Ruiz et al. 2007, Rambabu e Chakrabarti 2007, Trieu e Maruyama 2007, Trieu e Maruyama 2007, Yeong et al. 2009, Körbes et al. 2011, Sameer 2012, Quesada-Barriuso, Heras e Argüello 2013, Sivakumar e Janakiraman 2020), reconstrução morfológica (Vincent 1991, Vincent 1992, Robinson e Whelan 2004, Karas 2011, Teodoro et al. 2013, Sung et al. 2015, Teodoro et al. 2017), e componentes conexos (Klaiber et al. 2016, Rosenfeld e Pfaltz 1966, Cabaret, Lacassagne e Etiemble 2016, Benkrid et al. 2003), entre outras operações

morfológicas. Em (Teodoro et al. 2013), Teodoro *et al* mostra que o algoritmo de reconstrução morfológica possui um melhor desempenho quando a imagem original é particionada em sub-imagens, sendo processadas em paralelo. Neste contexto, o foco deste trabalho é o algoritmo de reconstrução morfológica do tipo sequencial e híbrido. No entanto, além das abordagens sequencial e híbrido, temos outras abordagens: paralelo e baseado em fila.

3.2 ALGORITMO PARALELO

No algoritmo paralelo, os pixels da imagem podem ser acessados sem uma ordem especifica. Neste algoritmo, tem-se como entradas as imagens I e J, onde a imagem reconstruída pode ser processada por sucessivas dilatações em J usando I como limite até alcançar a estabilidade (Vincent 1992). O algoritmo de reconstrução paralela usa a vizinhança N_G de cada pixel $\mathbf{p}$, e uma conectividade oito ou quatro. Quando um novo valor do pixel $J(\mathbf{p})$ é calculado, o novo valor deve ser escrito na mesma posição do pixel $\mathbf{p}$ na imagem J. O pseudocódigo 2 mostra o algoritmo nomeado como algoritmo paralelo.

Neste algoritmo, o escaneamento da imagem pode ser modelado como uma operação de convolução. Na operação de convolução, o resultado final é produto de um somatório e umas multiplicações entre uma vizinhança de um pixel e um *kernel* ou elemento estruturante. Para o caso do algoritmo paralelo, essas operações matemáticas viram operações comparativas (*maior que* e *menor que*).

Geralmente, devido ao conteúdo das imagens, forma de propagação e a baixa taxa de variação dos pixels, os algoritmos paralelos apresentam uma maior quantidade de varreduras para alcançar terminar o processamento da imagem (Vincent 1992). O número de ciclos de relógio necessários para cada varredura é de N*M, onde M e N são as dimensões da imagem, além do tempo de latência da arquitetura. Neste caso, o número necessário de operações (ciclos de relógio) até alcançar a estabilidade pode ser consideravelmente grande.

Algoritmo 2: Reconstrução paralela (Vincent 1992).

Entrada: `J`: marcador, `I`: máscara
Saída: `J`: imagem reconstruída

```
1 início
2     repita
3         para todo p faça
4             J(p) ← max{J(q)|q ∈ N_G(p) ∪ {p}}
5             J(p) ← min{J(p), I(p)}
6         fim
7     até estabilidade;
8 fim
```

Neste algoritmo, para o caso de implementações em *hardware*, os resultados das funções dos valores máximo e mínimo possuem dependência de dados sendo difícil realizar uma quebra que

permita a paralelização dessas operações, isto significa que as duas operações devem ser feitas durante o mesmo ciclo de relógio. Uma implementação alternativa é apresentada em (Bartovskỳ et al. 2015), onde o valor máximo e mínimo são calculados em ciclos de relógio diferentes.

3.3 ALGORITMO SEQUENCIAL

Um algoritmo mais eficiente que o algoritmo paralelo foi nomeado como *Sequential Reconstruction Algorithm* ou denominado com a sigla SR (Vincent 1992). Ao contrario do algoritmo paralelo, o algoritmo sequencial segue dois princípios:

- O algoritmo segue uma ordem de escaneamento específica, fazendo uma decomposição da vizinhança N_G nas vizinhanças N_G^+ (*raster*) e N_G^- (*anti-raster).*
- Semelhante ao algoritmo paralelo, o novo valor do pixel $J(\mathbf{p})$ calculado deve ser substituído na mesma posição do pixel $\mathbf{p}$ na imagem J.

Esses dois princípios conseguem uma convergência mais rápida da solução, isto é, alcançar a estabilidade da imagem reconstruída (Vincent 1992). O valor total das operações em ciclos de relógio (NM) é igual que o algoritmo paralelo. No entanto, uma desvantagem deste algoritmo é a dependência de dados entre varreduras. A estrutura do algoritmo sequencial é semelhante ao Algoritmo 2, onde as varreduras *raster* e *anti-raster* são feitas em sequencia. Semelhante à desvantagem do algoritmo paralelo, conforme são feitas várias varreduras poucos pixels estão contribuindo para a modificação da imagem. Assim, um passo necessário para a rápida convergência da solução é marcar os pixels que podem propagar seus valores para a vizinhança. Isto foi conseguido implementando um algoritmo baseado em uma fila.

3.4 ALGORITMO BASEADO EM FILA

Inicialmente, os pixels que são candidatos a propagação ou considerados como *sementes*, ou seja, os pixels que são máximos regionais são identificados e colocados em uma fila e propagados em J procurando por mais pixels como fontes de propagação. O uso da fila torna-se mais eficiente porque o algoritmo não precisa percorrer a imagem procurando novos pixels para modificar. No entanto, para obter as sementes deve-se efetuar uma varredura dentro da imagem. Neste contexto, foi proposto um algoritmo híbrido entre o algoritmo sequencial e este algoritmo, caracterizando-se por sua eficiência e rápida convergência (Vincent 1991). Semelhante às desvantagens dos algoritmos anteriores, este algoritmo apresenta uma dependência de dados, além de seguir um caminho irregular de propagação, o que torna o algoritmo não paralelizável.

3.5 ALGORITMOS HÍBRIDOS

Da mesma forma que o algoritmo sequencial e o algoritmo baseado em filas, o algoritmo híbrido ou FH funciona aplicando uma decomposição da vizinhança N_G de cada pixel $\mathbf{p}$. O pseudocódigo 3 mostra os dois algoritmos mencionados anteriormente (sequencial e filas). O algoritmo híbrido está composto pelas seguintes fases:

- Algoritmo sequencial (linhas 3 até 15 do Algoritmo 3).
- Algoritmo usando uma fila (linhas 16 até 24 do Algoritmo 3).

Algoritmo 3: Fast Hybrid Grayscale Reconstruction Algorithm (Vincent 1992).

```
   Entrada: J: marcador, I: máscara
   Saída: J: imagem reconstruída
1  início
2      repita
           /* Algoritmo Sequencial                                   */
3          para todo p faça
4              J(p) ← max{J(q)|q ∈ N_G^+(p) ∪ {p}}
5              J(p) ← min{J(p), I(p)}
6          fim
7          para todo p faça
8              J(p) ← max{J(q)|q ∈ N_G^-(p) ∪ {p}}
9              J(p) ← min{J(p, I(p)}
10             para todo q ∈ N_G^-(p) faça
11                 se J(q) < J(p) e J(q) < I(q) então
12                     fifo_add(q)
13                 fim
14             fim
15         fim
           /* Algoritmo usando fila                                  */
16         enquanto fifo_empty()=false faça
17             p ← fifo.first()
18             para todo q ∈ N_G(p) faça
19                 se J(q) < J(p) and I(q) ≠ J(q) então
20                     J(q) ← min{J(p), I(q)}
21                     fifo_add(q)
22                 fim
23             fim
24         fim
25     até estabilidade;
26 fim
```

Na fase *anti-raster* da fase do SR, os pixels candidatos à propagação são armazenados numa fila (linha 12 do Algoritmo 3), que serve como uma entrada para a fase de propagação. Na fase de propagação, os pixels são lidos da fila (linha 17 do Algoritmo 3) e seus vizinhos são examinados

para calcular mais pixels que satisfazem as condições de propagação. O algoritmo termina quando a fila fica vazia.

Este algoritmo herda as mesmas desvantagens dos algoritmos que a compõem: dependência de dados entre varreduras e um caminho irregular de propagação. Dessas três limitações, a dependência de dados e o armazenamento da imagem temporal dependem da plataforma computacional usada, ou seja, para o caso de uma implementação em *hardware* será necessário um FPGA de grande suporte.

A Tabela 3.1 apresenta o resumo das características principais dos algoritmos morfológicos. Nesta tabela, o algoritmo híbrido apresenta-se como o de melhor desempenho por causa da exploração da vizinhança e da implementação da fila. No entanto, o número de propagações da fila é indeterminado devido à dependência do conteúdo da imagem, e converte-se numa desvantagem para este algoritmo para implementações *hardware* devido ao aumento nos requisitos de armazenamento. Deve-se ressaltar que o desempenho de cada algoritmo depende do conteúdo e tamanho de cada imagem. Assim, a escolha do melhor algoritmo de reconstrução depende do tipo de problema para resolver.

Tabela 3.1: Características dos algoritmos morfológicos

Algoritmo	**Número de varreduras**	**Varreduras**			**Propagação por fila**	**Requisitos de armazenamento**	**Velocidade**	**Mapeamento em hardware**	**Ciclos de relógio**
		Raster		**Anti-raster**					
		N_G	**N_G^+**	**N_G^-**					
Paralelo	Indeterminado	Sim	Não	Não	Não	2*M*N	Baixa	Alta	NM
Sequencial	Indeterminado	Não	Sim	Sim	Não	2*M*N	Baixa	Média	NM
Fila	Não aplica	Não	Não	Não	Sim	2*M*N + FIFO	Alta	Baixo	NM + ind.
Híbrido	2	Não	Sim	Sim	Sim	2*M*N + FIFO	Alta	Baixo	NM + ind.

3.6 SEGMENTAÇÃO POR DIVISOR DE ÁGUAS CONTROLADA POR MARCADOR

A reconstrução morfológica é a base para o processo de segmentação usando transformada por divisor de águas controlada por marcador (*marker-controlled segmentation*) (Wu et al. 2017). A transformada por divisor de águas, ou segmentação por divisor de águas, possui o mesmo conceito da Figura 2.9, ou seja, a imagem é vista como uma imagem topográfica com vários mínimos regionais (vide Figura 2.14).

O Algoritmo 4 apresenta a transformada por divisor de águas baseado em componentes conexos. O algoritmo identifica e rotula cada região que possui um mínimo local. O algoritmo começa com uma imagem $f \in \mathbb{Z}^2$ e termina com uma imagem dos componentes conexos L. O algoritmo consiste em quatro varreduras no sentido *raster* usando a vizinhança N_G do pixel $\mathbf{p}$.

No primeiro passo, é feita uma varredura *raster*, são identificados os pixels que possuem valores mínimos e os pixels que são planaltos. No segundo passo, os pixels $\mathbf{p}'$ com o mesmo valor de intensidade do pixel $\mathbf{p}$ e que também possuem um vizinho de menor intensidade, são colocados dentro de uma fila. Esses pixels dentro da fila são sementes usadas para propagar suas etiquetas

Algoritmo 4: Transformada *watershed* baseada em componentes conexos (Bieniek e Moga 2000, Körbes 2010).

Entrada: `f`: imagem em escala de cinza
Saída: `l`: imagem de etiquetas=mask

início
  /* Varredura no sentido raster (1) */
  para todo $\mathbf{p}$ **faça**
    $\mathbf{q} \leftarrow \mathbf{p}$
    para todo $\mathbf{p}' \in N_G(\mathbf{p}) \wedge f(\mathbf{p}') < f(\mathbf{p})$ **faça**
      se $f(\mathbf{p}') < f(\mathbf{q})$ **então** $\mathbf{q} \leftarrow \mathbf{p}'$
    fim
    Se $\mathbf{q} \neq \mathbf{p}$ **então** $L(\mathbf{p}) \leftarrow \mathbf{q}$
    senão
      $L(\mathbf{p}) \leftarrow \mathbf{plateau}$
    fim
  fim
  /* Varredura no sentido raster (2) */
  para todo $\mathbf{p}$ **faça**
    se $L(\mathbf{p}) = \mathbf{plateau}$ **então**
      para todo $\mathbf{p}' \in N_G(\mathbf{p})$ **faça**
        se $L(\mathbf{p}') \neq \mathbf{plateau} \wedge f(\mathbf{p}) = f(\mathbf{p}')$ **então** fifo_add($\mathbf{p}'$)
      fim
    fim
  fim
  /* Fase de propagação (3) */
  enquanto *fifo_empty()=false* **faça**
    $\mathbf{p} \leftarrow$ fifo.first()
    para todo $\mathbf{p}' \in N_G(\mathbf{p}) \wedge L(\mathbf{p}') = \mathbf{plateau}$ **faça**
      $L(\mathbf{p}') \leftarrow \mathbf{p}$
      fifo_add($\mathbf{p}'$)
    fim
  fim
  /* Varredura no sentido raster (4) */
  para todo $\mathbf{p}$ **faça**
    se $L(\mathbf{p}) = \mathbf{plateau}$ **então**
      $L(\mathbf{p}) \leftarrow \mathbf{p}$
      para todo $\mathbf{p}' \in N_{prev}(\mathbf{p}) \wedge f(\mathbf{p}') = f(\mathbf{p})$ **faça**
        $\mathbf{r} \leftarrow FIND(L, \mathbf{p})$
        $\mathbf{r}' \leftarrow FIND(L, \mathbf{p}')$
        $L(\mathbf{r}) \leftarrow L(\mathbf{r}') \leftarrow min(\mathbf{r}, \mathbf{r}')$
      fim
    fim
  fim
  /* Varredura no sentido raster (5) */
  para todo $\mathbf{p}$ **faça**
    $lval \leftarrow 1$
    $\mathbf{r} \leftarrow FIND(L, \mathbf{p})$
    $L(\mathbf{p}) \leftarrow \mathbf{r}$
    se $L(\mathbf{r}) = mask$ **então**
      $l(\mathbf{r}) \leftarrow lval$
      $lval \leftarrow lval + 1$
    fim
    $l(\mathbf{p}) \leftarrow l(\mathbf{r})$
  fim
fim

Função FIND(L,**p**)

Entrada: L: endereços, **p**: pixel
Saída: q: pixel

```
1  q ← p
2  enquanto L(q) ≠ q faça
3  |  q ← L(q)
4  fim
5  u ← u
6  enquanto L(u) ≠ q faça
7  |  tmp ← L(u)
8  |  L(u) ← q
9  |  u ← tmp
10 fim
11 return q
```

para os vizinhos com igual intensidade (passo 3). No passo quatro, as planícies são conetados usando o operador de componentes conexos. Finalmente, no passo 5 são colocadas as etiquetas finais para cada componente conexo.

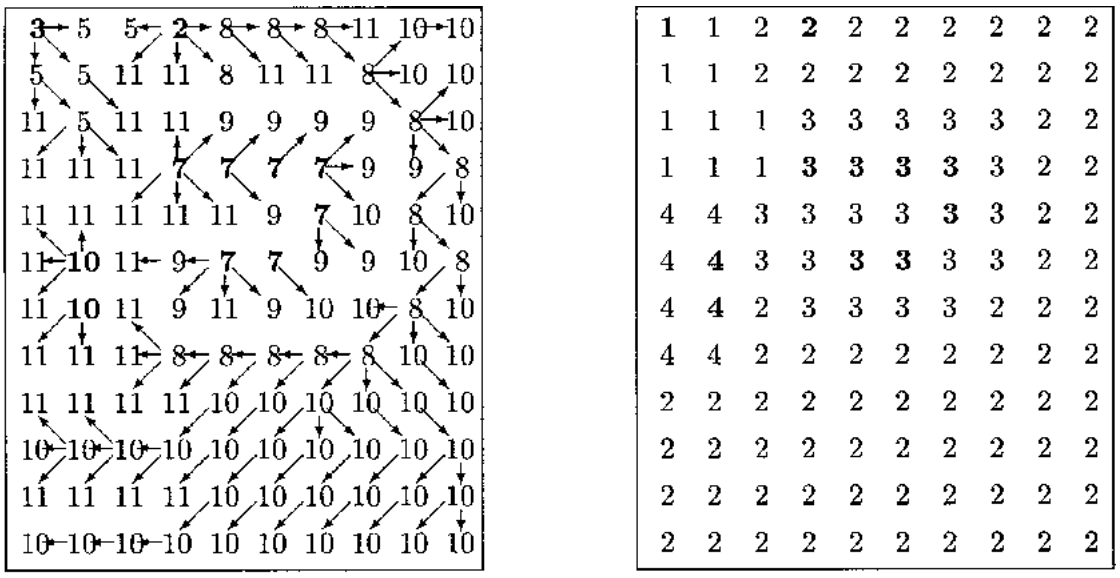


Figura 3.1: Transformada por divisor de águas baseada em componentes conexos (Bieniek e Moga 2000).

A Figura 3.1 mostra o resultado de segmentação de uma imagem usando transformada por divisor de águas e componentes conexos. Nesta figura, cada mínimo regional propaga o valor da sua etiqueta para dividir a imagem em várias regiões.

3.7 DISCUSSÃO GERAL SOBRE OS ALGORITMOS MORFOLÓGICOS

Diante do exposto, o algoritmo híbrido apresenta-se como o algoritmo morfológico de melhor desempenho para plataformas *software* e *hardware*. No entanto, o uso do algoritmo baseado em

uma fila tem implicação direta no uso de uma grande quantidade de recursos computacionais, especificamente no caso de uso de espaço de memória. Neste caso, não é possível determinar *a priori* o número de fontes de propagação para a alocação de espaço de memória. Assim, é necessário encontrar uma relação entre a quantidade de fontes de propagação e os recursos computacionais.

Neste contexto, dependendo da aplicação, os algoritmos morfológicos podem ser divididos em três tipos: (a) multi-varredura, (b) mono-varreduras e (d) uma combinação de ambas. As caraterísticas dos algoritmos são apresentadas a seguir.

1. **Multi-varredura**:

   No escaneamento de uma imagem, considera-se uma varredura ao escaneamento no sentido *raster* (N_G^+) seguido do escaneamento no sentido *anti-raster* (N_G^-), ou um escaneamento somente no sentido *raster* (N_G). No caso de usar a vizinhança N_G, o algoritmo pode ser implementado num *pipeline* com um número fixo de estágios, no entanto, essa implementação apresenta-se como ineficiente devido ao desconhecimento do número de varreduras até a estabilização da imagem o que exigiria uma grande quantidade de recursos da plataforma.

   Por outro lado, o uso das vizinhanças N_G^+ e N_G^- apresenta uma limitação devido à dependência de dados entre os escaneamentos. Neste sentido, para começar o escaneamento *anti-raster* é necessário terminar o escaneamento *raster*. Em consequência, os requisitos de armazenamento incrementariam mas se apresentaria um ganho na velocidade de convergência da solução.

2. **Mono-varredura**:

   Nesta classificação, os algoritmos mais eficientes são baseados nas vizinhanças N_G^+ e N_G^-, e no uso de uma fila. Essa classe de algoritmos, exigem mais capacidade de armazenamento por causa da implementação da fila. Em plataformas *hardware*, a configuração da fila não pode ser modelada em *dataflow* senão que deve ser modelada como uma *máquina de estados finita* (FSM). Além disto, o desconhecimento do número de fontes de propagação apresenta-se como uma desvantagem porque não poderia aloca-se uma quantidade infinita de elementos dentro da fila.

3. **Combinação entre multi-varredura e mono-varredura**:

   Este tipo de algoritmos apresenta-se como uma solução ao problema das limitações para plataformas *hardware*. O algoritmo herda as desvantagens no quesito de espaço de armazenamento e modelado usando uma FSM, mas permite executar múltiplas varreduras procurando um balanço entre os recursos utilizados e o desempenho do algoritmo.

A Figura 3.2 mostra os diagramas de blocos dos algoritmos apresentados. Observa-se que cada algoritmo possui uma etapa de escaneamento da imagem usando as vizinhanças N_G^+, N_G^- e N_G. Deve-se ressaltar que no caso das implementações em plataformas *software* (GPUs) o gargalo apresenta-se na etapa de escaneamento (Teodoro et al. 2013). Neste contexto, a estrategia de

solução para os algoritmos de reconstrução pode ser extrapolada para algoritmo de divisor por águas e de componentes conexos, além de outros algoritmos morfológicos como a transformada da distância e esqueletização (Vincent 1989).

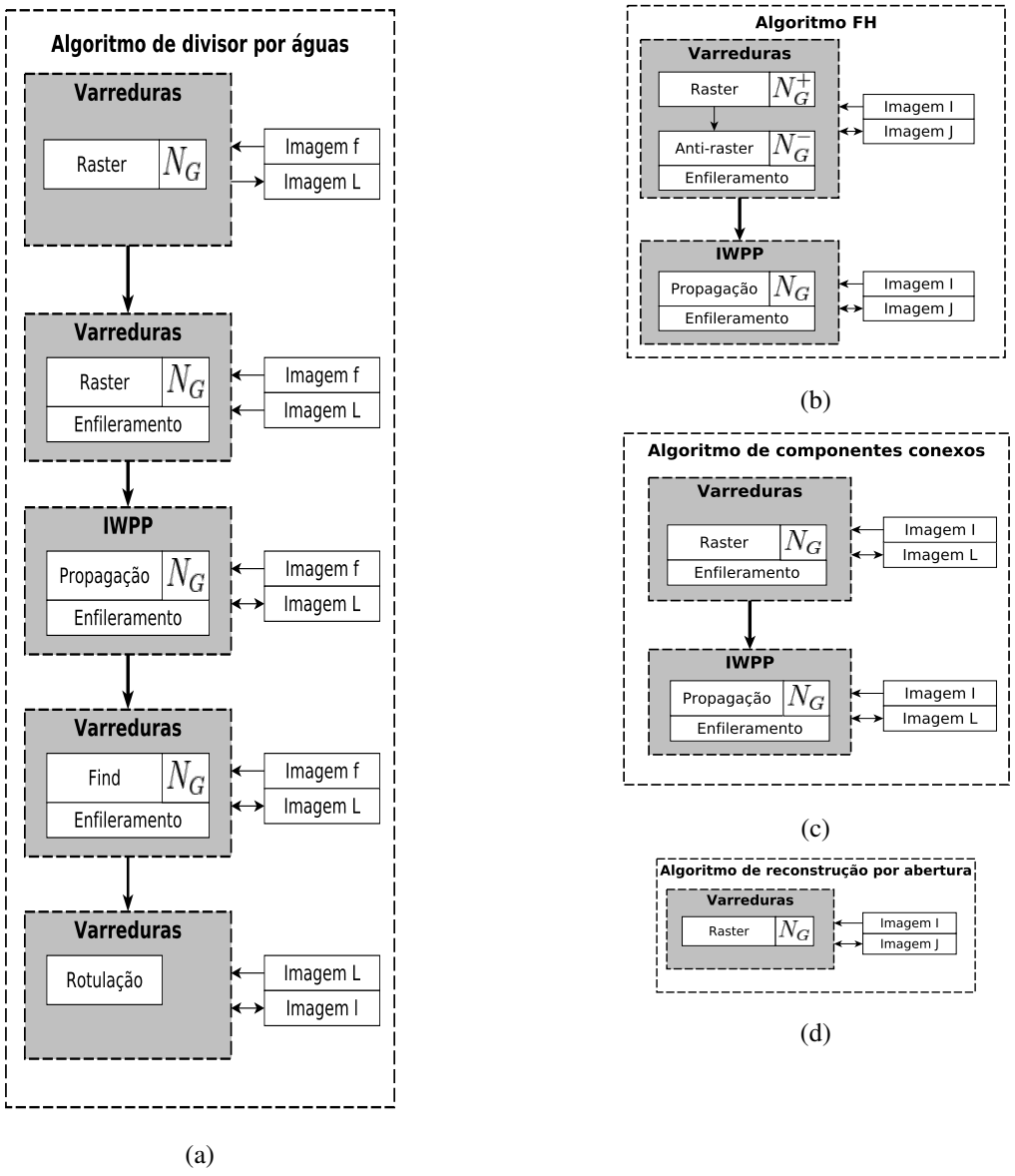


Figura 3.2: Fluxograma dos algoritmos apresentados

Os algoritmos morfológicos apresentados são computacionalmente caros, principalmente a transformada por divisor de águas. As otimizações apresentadas em (Vincent 1995) são muito usadas e otimizadas em plataformas *software*. No entanto, para implementações em plataformas *hardware*, apresenta-se uma limitação devido à etapa da implementação da fila por causa do caminho irregular que é seguido. Para os critérios de estabilização de uma imagem, precisam-se métricas que sejam calculadas e avaliadas durante o processo de reconstrução. Além disso, o processo de reconstrução morfológica, por ser um método iterativo, incrementando o custo computacional em número de ciclos de relógio (iMN, onde i é o número de iterações).

No algoritmo FH, a fase de propagação é não-linear, dependendo fortemente do conteúdo da imagem a fim de alcançar a estabilização, causando um aumento no número de operações do algoritmo; o que permite abrir possibilidades para a exploração de técnicas que permitam otimizar o custo computacional desse algoritmo. Uma estratégia para encontrar uma solução aceitável para o problema de reconstrução morfológica é mediante o uso de métodos baseados em *heurísticas*, as quais não garantem a solução ótima. No contexto das *heurísticas* encontram-se um conjunto de algoritmos heurísticos que permitem conseguir soluções consideradas aceptáveis e que podem ser generalizadas para serem aplicadas a vários tipos de problemas. Este conjunto de metodos-soluções tem sido denominado pela comunidade de *meta-heurísticas*.

As meta-heurísticas mais populares para resolver problemas de otimização são baseadas em processos naturais ou biológicos, por exemplo: colônia de formigas (*Ant Colony Optimization*-ACO), otimização por exame de partículas (*Particle Swarm Optimization*-PSO), otimização de colônia de abelhas (*Bee Colony Optimization*-ABC), borboletas, gravidade e interações da massa, filtro de partículas, algoritmo genético, método de reprodução dos pássaros cuco (*cuckoo search*), e eco localização dos morcegos (*Bath Algorithm*) (Yang e Papa 2016, Karaboga et al. 2014). Neste trabalho, o algoritmo PSO foi usado para o processo de treinamento de uma máquina de vetores de suporte (SVM) a fim de estabelecer um critério de parada dentro do algoritmo FH.

3.8 MÁQUINAS DE VETORES DE SUPORTE

Técnicas de processamento de imagens são comumente combinadas com meta-heurísticas e algoritmos de aprendizagem de máquina para fornecer soluções que possam ter uma viabilidade na área da computação (Norvig e Russell 2014, Giagkiozis, Purshouse e Fleming 2015, Yang e Papa 2016, Maggiani, Luca et al. 2018). Essas combinações são empregadas com o intuito de aumentar o desempenho do algoritmo e/ou plataforma computacional procurando um balanço entre o esforço computacional e/ou o consumo de potência. Na área de reconhecimento de padrões, as amostras podem ser classificadas em várias categorias chamadas de *classes*. Cada conjunto de amostras têm características (*features*) que são representativas de cada classe, permitindo sua classificação em subconjuntos. De forma geral, no processo de classificação pode-se usar informação *a priori* (aproveitamento paramétrico) e informação não *a priori* (aproveitamento não-paramétrico) sobre o conjunto de dados. Exemplos de classificadores não-paramétricos são as redes neurais (RNA), sistemas difusos, e as máquinas de vetores de suporte (SVM) (Shigeo 2005).

Entretanto, as SVMs são comumente usadas para classificação, como uma técnica eficiente de aprendizagem supervisionada, baseada em um modelo matemático (Maggiani, Luca et al. 2018). Este modelo matemático está baseado em uso de *kernels*, os quais são funções responsáveis por levar os vetores de treinamento do espaço original a um espaço de dimensão alta o suficiente para que os dados sejam linearmente separáveis (Santos et al. 2021). Os kernels podem ser quaisquer função que atenda o teorema Merce (Hamdani, T.M., Alimi, A.M. e Khabou, M.A 2011). Por exemplo, o uso de uma SVM com kernel polinomial possui baixo custo computacional pois este

possui apenas uma soma e multiplicação em sua equação. Enquanto que as RNA, o uso da função *sigmóide* ou de *tanh* na função de transferência é obrigatório funções que envolvem operações mais complexas que o kernel polinomial das SVMs.

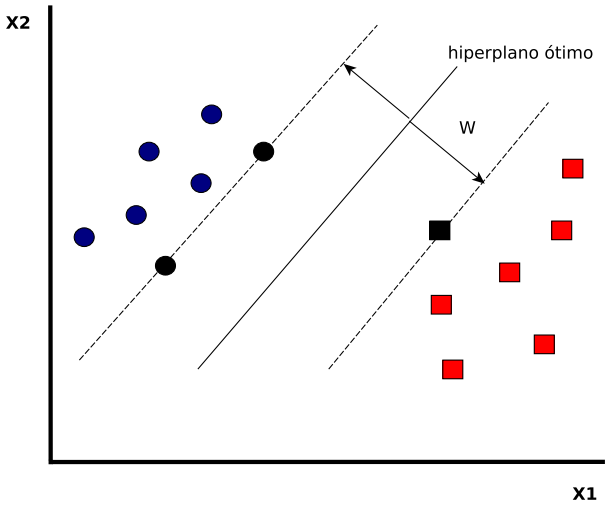


Figura 3.3: Hiperplano de separação ideal no espaço de 2-D.

Na pratica, uma SVM é parcialmente não-paramétrica, isto é, uma SVM armazena uma fração do número de amostras do processo de treinamento mas por outro lado possui uma grande capacidade de flexibilização para classificar novas amostras. Durante o processo de treinamento, os pares entrada-saída são usados para gerar soluções ótimas chamadas de *funções de decisão* que vão rotular os dados de entrada (Norvig e Russell 2014).

Por exemplo, a Figura 3.3 apresenta amostras divididas em duas distintas classes separadas por um hiperplano, onde o hiperplano é considerado como a função de decisão. Os pontos que estão sobre as margens (retas tracejadas) são nomeadas como *vetores de suporte*, e o hiperplano é a reta classificadora. Durante o processo de treinamento, o classificador pode apresentar sobreajuste (*overfitting*), ou seja, perde a capacidade de generalização.

Uma SVM é treinada para maximizar as margens e assim possuir uma maior capacidade de generalização e robustez. Os pontos que estão sobre as margens satisfazem as equações 3.1 e 3.2 (Shigeo 2005).

$$\left|\mathbf{w}^T\mathbf{x} + b\right| = 1, \tag{3.1}$$

onde $\mathbf{w}$ e $\mathbf{x}$ são vetores m-dimensionais e b é o bias.

$$\begin{cases} \mathbf{w}^T\mathbf{x} + b \geqslant +1 & \text{se} \quad y_i = +1 \\ \mathbf{w}^T\mathbf{x} + b \leqslant -1 & \text{se} \quad y_i = -1 \end{cases} \tag{3.2}$$

A equação 3.2 representa os dois hiperplanos que separam as classes do conjunto de amostras

separados por uma distância d (vide equação 3.3).

$$d = \frac{2}{\|\mathbf{w}\|} \tag{3.3}$$

O principal objetivo é maximar a distância, ou as margens, entre os dois planos. A maximização de d é equivalente a minimização de $\|\mathbf{w}\|$. No entanto, a maioria dos problemas de classificação possuem dados que não são linearmente separáveis, o que pode gerar uma função de decisão incapaz de classificar os dados corretamente, mesmo no conjunto de treinamento (Santos et al. 2017). Nestes casos, para alcançar a separabilidade linear deve-se aplicar uma transformada do espaço original (linearmente não separável) para um espaço de características de dimensão superior onde seja linearmente separável.

Deve-se encontrar uma transformação não linear ϕ que eleve os dados a um espaço de dimensão maior chamado de espaço de características em que os dados sejam linearmente separáveis (Santos et al. 2017). A transformação segue a forma $K(\vec{x}_i, \vec{x}_j) = \Phi(\mathbf{x}) \cdot \dot{\Phi}(\mathbf{x})$, denominada função núcleo (*kernel*). Existem vários tipos de *kernel*, entre eles temos: (a) polinomial, (b) Gaussiano (c) sigmoidal e/ou (d) rede neural artificial-RNA com função de ativação radial (RBF). Na literatura, o kernel mais usado é o tipo RBF, conforme a equação 3.4.

$$K(\mathbf{x}, \mathbf{x_i}) = \exp\left(-\frac{\|(\vec{x} - \vec{x}_i)\|^2}{2\sigma^2}\right), \tag{3.4}$$

onde σ^2 é a variânça (Shigeo 2005). De formal geral, as vantagens das máquinas de suporte sobre uma rede neuronal multi-camada são:

1. **Máxima capacidade de generalização**. No processo de treinamento das máquinas de suporte, as margens são maximizadas com o objetivo de dar à máquina uma maior capacidade de generalização.
2. **Não possui mínimos locais**. As máquinas de suporte possuem um mínimo global.
3. **Robustez a ruído**. Redes neurais são susceptíveis a valores que são atípicos (*outliers*), gerando erros na classificação. Por outro lado, a correta sintonização dos hiperparâmetros das SVMs permite que a máquina suprima esses valores atípicos.

Por outro lado, as desvantagens das máquinas de suporte são:

1. **Extensão para problemas multi-classe**. A extensão para problemas multi-classe não é um processo simples devido a que as máquinas de suporte usam funções de decisão diretas.
2. **Tempo de treinamento muito longo**. O tempo de treinamento das máquinas de suporte pode ser longo.

3. **Seleção de parâmetros**. O processo de treinamento de uma SVM usa um parâmetro de regularização (C) e o *kernel*, ambos chamados de hiperparâmetros. Esses hiperparâmetros controlam o balanço entre capacidade de generalização e a robustez da SVM. As métricas de avaliação de cada hiperparâmetro são: quantidade de vetores de suporte e o erro empírico (*Mean Squared Error*-MSE). A escolha do tipo de *kernel* é fundamental para o desempenho da SVM. Sintonizando bem esses hiperparâmetros na etapa de treinamento, a SVM controla os erros de classificação melhor que uma rede neuronal.

Os algoritmos bio-inspirados são meta-heurísticas que usam como fonte de inspiração os comportamentos de formigas, abelhas, vespas, cupins, vaga lumens, lagartos, morcegos, entre outros tipos de animais (Giagkiozis, Purshouse e Fleming 2015). Estas espécies organizadas em colônias, enxames ou bandos procuram uma solução ótima para encontrar: fontes de alimentos, formas de reprodução e/ou desenvolvimento dos integrantes da colônia, onde cada integrante dessa sociedade é chamado de agente (Marler e Arora 2004, Giagkiozis, Purshouse e Fleming 2015).

Nesse sentido, cada sociedade considera-se como sistemas multi-agentes com interações e regras locais, esses sistemas podem agir de forma decentralizada, e podem apresentar características de uma inteligencia coletiva (Yang e Papa 2016). Dependendo da aplicação, se o objetivo for maximizar as fontes de alimento ou minimizar o uso de esforço busca dessas fontes, esses instintos podem ser modelados matematicamente como funções dentro de um espaço dimensional. Essas funções são chamadas de funções objetivo. O mesmo raciocínio aplica-se para maximizar o uso de elementos físicos ou minimizar o esforço computacional de uma plataforma computacional (vide Figura 3.4).

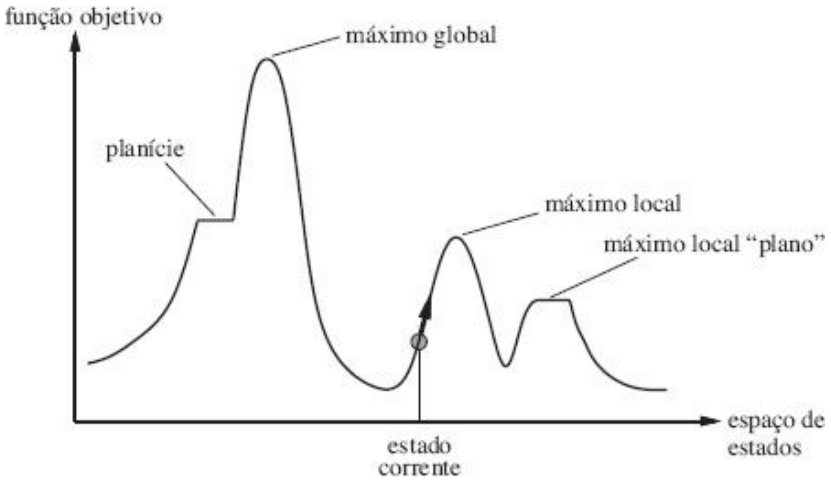


Figura 3.4: Função objetivo unidimensional (Norvig e Russell 2014).

A Figura 3.4 mostra o comportamento da função objetivo ou função custo. Pode-se observar a presença de valores mínimos e máximos, o valor mais baixo é nomeado como *mínimo global* e o valor mais alto é o *máximo global*. O objetivo das meta-heurísticas é a exploração desse espaço dimensional para encontrar os valores ótimos que são considerados como máximos/mínimos globais (Norvig e Russell 2014).

Os problemas de otimização que visam obter o valor mínimo/máximo de uma única função objetivo são nomeados como problemas *mono-objetivo*. No entanto, quando um problema é composto de duas ou mais funções objetivo (para guiar a meta-heurística) o mesmo é chamado de problema *multi-objetivo*. No caso de problemas *multi-objetivo*, várias soluções desse conjunto podem dominar as outras soluções procurando um balanço entre os objetivos do problema. Esse conjunto de soluções são nomeadas como *fronteira de Pareto*.

Além dos algoritmos inspirados no comportamento da natureza, tem-se algoritmos de inteligência artificial como as redes neurais artificiais (*Artificial Neural Nework*-ANN) e as máquinas de suporte (*Support Vector Machines*-SVM) (Yang e Papa 2016). Esses dois algoritmos são exemplos de algoritmos *multi-objetivo*. As SVMs foram propostas para classificação, regressão ou identificação de sistemas lineares e não-lineares (Yang e Papa 2016).

3.8.1 Otimização por exame de partículas-PSO

O método de otimização por enxame de partículas (*Particle Swarm Optimization*-PSO) foi proposto por Eberhart & Kennedy (1995), e modificado em diversos trabalhos (Marler e Arora 2004). O PSO é baseado no comportamento dinâmico individual e social de grupos de pássaros e cardumes. Na inteligência de enxame, vários agentes de comunicação trocam informações para proteger-se de predadores e obter comida com maior eficiência (Marler e Arora 2004, França 2005). No PSO cada agente pode ser considerado como uma partícula que possui informações de vetores de posição num determinado espaço de busca $D \subseteq \mathbb{Z}^n$, e a cada momento este vetor é atualizado com as informações de outras partículas.

O algoritmo do PSO utiliza matrizes $M \times N$ para armazenar as informações de cada partícula, onde M é o número de partículas e N são as dimensões do espaço de busca. As posições das partículas são armazenadas na matriz $\mathbf{X}= (x_{ij})$ onde $1 \leq i \leq M$ e $1 \leq j \leq N$. Durante o processo de busca são incorporadas as ideias de competição e colaboração simultaneamente. Cada partícula procura por uma solução melhor que as outras partículas mas que ao mesmo tempo beneficie as outras usando memória individual e coletiva. A direção, sentido e distância percorrida por cada partícula é determinada por sua velocidade $\mathbf{V}= (v_{ij})$, onde $\mathbf{V}$ em outra matriz $M \times N$. A equação que relaciona a posição e velocidade das partículas é dada por:

$$\mathbf{X}_{updated} = \mathbf{X}_{previous} + \mathbf{V}\dot{\Delta}t, \tag{3.5}$$

onde Δt é um passo de tempo, e $\mathbf{V}$ é dado por

$$\mathbf{V}_{updated} = \mathbf{V}_{previous} + c_1\eta_1(\mathbf{P} - \mathbf{X}) + c_2\eta_2(\mathbf{G} - \mathbf{X}), \tag{3.6}$$

onde $c_{1,2}$ são constantes de aceleração associados ao conhecimento individual e coletivo, e $\eta_{1,2}$ são valores aleatórios no intervalo [0,1], $\mathbf{P}$ é a melhor posição da partícula $\mathbf{P}= (p_{ij})$, e $\mathbf{G}$ é a melhor posição global do enxame $\mathbf{G}= (g_{ij})$ avaliada por uma função custo f. O Algoritmo 5

apresenta o pseudocódigo do PSO.

Para iniciar o PSO são necessários os parâmetros M, N, c_{12}, I_{max} e thr que são respectivamente o número de partículas, número de dimensões, os coeficientes de aceleração individual e social, o número máximo de iterações e o erro aceitável para o caso de estudo. O algoritmo segue os seguintes passos:

- As partículas são inicializadas em posições e velocidades aleatórias no espaço de busca (linha 2).
- Avaliação da função custo e identificação da melhor partícula do enxame (linhas 6 até 13).
- Atualização da velocidade e da posição de cada partícula usando as informações das memorias locais e globais (linhas 17 e 18).
- O algoritmo termina depois de atingir o limite máximo de iterações ou se atender o critério de parada (linha 22).

No caso do PSO, o algoritmo *Multi-Objective Particle Swarm Algorithm* (MOPSO) tem como objetivo trabalhar com várias funções custo para encontrar valores ótimos de cada múltiplas métricas usadas para avaliar o desempenho de um algoritmo (Santos et al. 2017).

3.8.2 Aplicações dos algoritmos bio-inspirados e SVMs

Como mencionado anteriormente, as aplicações em processamento de imagens envolvem operações como detecção de rostros, objetos, figuras ou características, limiarização, filtragem e restauração de imagens. Estas técnicas podem ser combinadas com os algoritmos bio-inspirados e/ou máquinas de suporte para melhorar o desempenho dessas técnicas e encontrar soluções que sejam ótimas e possam ser mapeadas em plataformas computacionais como os GPUs ou FPGAs. Um exemplo de restauração de imagens é o algoritmo de deconvolução *Richardson-Lucy*, esse algoritmo é iterativo derivado do teorema de Bayes que busca minimizar o error estimado entre a imagem real e a imagem borrada da Equação 3.7 (Gunturk e Li 2012).

$$\arg\max_{I} n\|(I_b - I \otimes K)\|^2 \tag{3.7}$$

O algoritmo Richardson-Lucy usa uma imagem desborrada I, um kernel que possui informação do tipo de borramento K, a imagem observada I_b , e $n(\cdot)$ como a distribuição do ruído da imagem. A solução pode ser obtida usando a equação 3.8 .

$$I^{t+1} = I^t \times K * \frac{I_b}{I \otimes K}, \tag{3.8}$$

onde $*$ representa a operação de correlação, e $\otimes$ é o operador convolução. O objetivo deste algoritmo é recuperar através de iterações sucessivas a imagem original. Neste algoritmo, a

Algoritmo 5: Algoritmo de otimização por exame de partículas-PSO (França 2005)

Entrada: `M`: número de partículas, `N`: número de dimensões, `thr`: threshold
Saída: `f`: função objetivo

início
  /* Inicialização das partículas */
  X=inicia_particulas(M,N)
  V=inicia_velocidade(M,N)
  It=0
  repita
    /* Avaliação da função objetivo */
    para $i = 1$ **to** *M* **faça**
      fx=avalia($\mathbf{x}_{ij}$)
      se $fx \leq f(p_{ij})$ **então**
        $\mathbf{p}_{ij} = \mathbf{x}_i j$
        fG_i=fx
      fim
    fim
    /* Deteção global */
    [$GBest_{It}$ k] = minimo(fG)
    g_{1j}=p_{kj}
    para $j = 1$ **to** *N* **faça**
      para $i = 1$ **to** *M* **faça**
        /* Atualização da velocidade */
        $\mathbf{v}_{ij} = \mathbf{v}_{ij} + c_1\eta_1(\mathbf{p}_{ij} - \mathbf{x}_{ij}) + c_2\eta_2(\mathbf{g}_1 j - \mathbf{x}_{ij})$
        /* Atualização da posição */
        $\mathbf{x}_{ij} = \mathbf{x}_{ij} + \mathbf{v}_{ij}$
      fim
    fim
    $It = It + 1$
  até $GBest_{It} \leq thr | \ It \leq I_{max}$
fim

máscara K é conhecida como função de espalhamento de ponto (*Point Spread Function*-PSF). Tendo toda informação da PSF pode-se restaurar uma imagem com baixo erro, no entanto, a PSF possuir nenhuma ou parte da informação do processo de borramento da imagem. Neste caso, o uso de algoritmos bio-inpirados em combinação com uma SVMS podem dar uma solução para calcular os parâmetros de restauração do algoritmo *Richardson-Lucy* (número de iterações e a PSF).

No caso do processo de treinamento de uma rede neuronal, os algoritmos bio-inspirados como o ABC ou o PSO podem ser usados para calcular os valores dos pesos iniciais dos neurônios (Karaboga et al. 2014). Uma técnica normalmente usado na segmentação de imagens é a limiarização, limiarização é baseada no histograma e pode ser mono-objetivo (único *threshold*) ou multi-objetivo (vários *thresholds*) (Baniani e Chalechale 2013). Alguns trabalhos já foram implementados para resolver esse tipo de otimização usando o algoritmo ABC, algoritmo PSO e o algoritmo genético (Baniani e Chalechale 2013, Karaboga et al. 2014).

O algoritmo MOPSO combinado com uma máquina se suporte foram empregados para a

implementação de uma solução aproximada de um modelo baseado em controle preditivo (*Model based predictive control*-MPC). O MOPSO foi usado para o treinamento da máquina de suporte na sintonização dos hiperparâmetros para buscar uma solução ótima que permita alcançar um balaço entre desempenho e consumo de potência (Santos et al. 2017) relacionado à relação entre o erro empírico e número de vetores de suporte.

3.9 CONCLUSÕES

Neste capítulo foram apresentados os conceitos básicos dos algoritmos da morfologia matemática. Embora os algoritmos de reconstrução por erosão e dilatação sejam simples, em termos de custo computacional (latência e memória), são ineficientes por causa da exploração exaustiva da imagem que está sendo reconstruída até sua estabilização. Por outro lado, o algoritmo de reconstrução baseado em uma fila apresenta-se como uma solução eficiente porque não explora a imagem senão que os pixels de regiões máximas são escolhidos e carregados dentro de uma fila na busca de mais fontes de propagação.

No entanto, um passo antecessor para escolher os pixels que serão armazenados na fila é obtido através da exploração da imagem. Assim, com a combinação de um algoritmo sequencial e um baseado em fila chegou-se a uma rápida convergência da solução do processo de reconstrução. A eficiência dos algoritmos de reconstrução morfológica usando a versão híbrida depende do elemento estruturante escolhido, do tipo de vizinhança adotado, e do conteúdo da imagem.

Neste último aspecto, pode-se usar métricas como a entropia para determinar a dependência entre a velocidade de convergência da solução e a variação dos níveis de intensidade da imagem. Dependendo da plataforma computacional definida, o desempenho do algoritmo no quesito de latência, consumo de memória, acessos à memória, e potência, pode indicar um balanço entre o número de iterações feitas antes de terminar a reconstrução usando uma fila.

Desta forma, a relação entre o processo de chaveamento, os pixels que estão sendo modificados (*pixels efetivos*) e o menor custo computacional pode-se tratar como um problema de otimização. Neste contexto, as máquinas de vetores de suporte podem ser usadas para estabelecer o ponto de melhor desempenho do algoritmo. Em contrapartida, a sintonização dos hiperparâmetros das SVMs é um processo complexo que pode levar um tempo considerável de processamento. Em compensação da desvantagem das SVMs, as meta-heurísticas apresentam-se como ferramentas para encontrar hiperparâmetros de boa qualidade que produzam SVMs com alta capacidade de generalização e baixa complexidade.

Observa-se uma similaridade na estrutura dos algoritmos morfológicos, embora existam divergências em formas de implementação, conteúdo da imagem, otimização de funções, quantidade de imagens usadas (máscara e/ou marcador), forma de percorrer a imagem, número de operações, etc. Essa similaridade entre o comportamento dos algoritmos permite inferir que a mesma abordagem de otimização para algum algoritmo possa ser expandida para algoritmos com estrutura

semelhantes.

4 TRABALHOS CORRELATOS EM ALGORITMOS SEQUENCIAIS VISANDO IMPLEMENTAÇÕES EM HARDWARE

Vários trabalhos reportam implementações em operações morfológicas usando dilatação e erosão em diversas plataformas computacionais (He et al. 2017). O número de operações dos algoritmos depende do tamanho, forma, e origem do elemento estruturante (Dokládal e Dokládalová 2011). O principal objetivo das propostas para esse tipo de algoritmos foca-se na otimização das métricas tais como como frequência de operação, latência, quadros por segundo, vazamento, uso de recursos computacionais, tempo de processamento e potência (Dokládal e Dokládalová 2011, Bartovskỳ et al. 2015, ELLOUMI, KRID e SELLAMI 2018).

O desempenho de cada algoritmo depende do tipo da abordagem com a qual vai ser implementado. Normalmente, os algoritmos podem ser classificadas em quatro tipos: (a) paralelo, (b) sequencial, (c) fila normal ou fila de prioridade, e (d) algoritmo híbrido juntando os algoritmos (b) e (c) (Vincent 1991). Dentre de estas categorias, os algoritmos que são implementados de forma híbrida oferecem um melhor desempenho. Inicialmente, foram propostos várias abordagens de algoritmos híbridos que melhoraram o desempenho dos algoritmos paralelo e sequencial. Essa abordagem permitiu melhorar o desempenho de algoritmos tais como o divisor por águas, reconstrução morfológica, transformada da distância, esqueletização, componentes conexos, divisor por águas dirigido por marcador, entre outras operações morfológicas (Rosenfeld e Pfaltz 1966, Verwer, Verbeek e Dekker 1989, Vincent 1989). Estes algoritmos possuem características similares dentro da sua estrutura de funcionamento, por exemplo: propagação do valor de um pixel para sua vizinhança (Vincent 1991).

O foco deste trabalho é o algoritmo de reconstrução morfológica do tipo sequencial e híbrido. No entanto, devido á familiaridade na implementação desse grupo de algoritmos no uso de varreduras *raster/anti-raster* e propagação usando uma fila, também serão discutidos alguns aspectos dos algoritmos de dilatação/erosão, segmentação por divisor de águas e o algoritmo de componentes conexos.

4.1 CONSIDERAÇÕES INICIAIS

Neste trabalho, todas as implementações são focadas para o processamento de imagens em tempo real, e propostas para executar as operações morfológicas no menor número de varreduras possíveis, sem relacionar a dependência entre o número de escaneamentos e o conteúdo da imagem. Na literatura encontram-se três abordagens para o processamento de uma imagem como: (a) a arquitetura de linha de atraso (*Delay Line Architecture*-DLA), (b) arquitetura de reutilização de

resultados parciais, (*Partial-Result-Reuse Architecture*-PRRA), e (c) arquitetura paralela (*Parallel Architecture*-PA).

A arquitetura DLA, dependendo do tamanho do elemento estruturante, permite armazenar uma ou mais linhas da imagem. Com isto, podem-se usar elementos estruturantes de vários tamanhos e formas. Nesta classe de abordagem, o tempo de processamento é proporcional ao tamanho da imagem e ao número de ciclos de relógio da arquitetura. No entanto, o consumo de elementos de memória (blocos internos de memória BRAM ou registradores) é elevado e depende do tamanho do elemento estruturante. Exemplos dessas implementações são encontradas em (Hedberg, Kristensen e Owall 2008), (Gibson et al. 2013), e (Holzer et al. 2012).

Por outro lado, a arquitetura PRRA otimiza e reduz o número de comparações que são redundantes para o processamento de um pixel. Nesta estratégia, a diminuição do número de comparações permite uma redução do consumo de elementos lógicos. Um exemplo dessa implementação pode ser encontrado em (Chien, Ma e Chen 2005). No caso da arquitetura PA, a imagem é dividida em vários blocos horizontais e verticais que são processados em *pipeline* e em paralelo devido ao particionamento da imagem, permitindo assim um ganho no desempenho, mas incrementando os recursos de memória que dependem do tamanho do elemento estruturante. Um exemplo dessa implementação pode ser encontrado em (Bartovský et al. 2014).

O tempo de processamento para uma varredura em *hardware* é constante. No entanto, para uma plataforma baseada em *software*, o tempo de processamento depende dos acessos a memória, tamanho de palavra, número de comparações, latência, etc. Com respeito às implementações em *hardware* dos algoritmos baseados em ASFs, em (Chien, Ma e Chen 2005), (Clienti, Beucher e Bilodeau 2008), (Holzer et al. 2012),(Déforges, Normand e Babel 2013), (Gibson et al. 2013), (Bartovský et al. 2014), (Bartovskỳ et al. 2015), (Torres-Huitzil 2016), (Mukherjee, Mukhopadhyay e Biswas 2016), (Mukherjee, Mukhopadhyay e Biswas 2017), e (ELLOUMI, KRID e SELLAMI 2018) são implementadas arquiteturas para a implementação desse tipo de algoritmos.

4.2 ASPECTOS SOBRE HARDWARE RECONFIGURÁVEL

Tendo em conta que este trabalho visa a implementação de algoritmos sequenciais em *hardware*, será feita uma breve descrição de aspectos de arquiteturas de dispositivos do tipo FPGA. Neste sentido, pode ser observado que dispositivos lógicos são divididos em duas categorias: (a) fixos e (b) programáveis. Exemplos de arquiteturas fixas são os ASICs que não podem ter mudanças na sua estrutura. Por outro lado, as Arquiteturas Lógicas Programáveis (PLDs)permitem projetar sistemas que podem ser modificados a cada momento. Os PLDs são baseados em tecnologia de memória regravável; isto é, uma alteração do circuito é possível via programação em *software*. Dentro das arquiteturas reconfiguráveis temos os dispositivos Lógicos Programáveis Complexos (*Complex Programmable Logic Devices*-CPLDs) e as matrizes de portas programáveis em campo (*Field Programmable Gate Arrays*-FPGAs).

FPGAs podem ser intermediários para la fabricação de circuitos integrados, pela rápida implementação de prototipos de sistemas complexos, incrementando o desempenho e a escalabilidade, devido à baixa complexidade de desenvolvimento comparada com a fabricação de ASICs.

Atualmente, apresentam-se vários desafios na área de desenvolvimento de um co-projeto *hardware*/*software* como por exemplo a integração de tecnologias SoC (*System-on-Chip*) envolvendo de 100 a 1000 processadores dentro de um único *chip*, interligados através de redes intra-chip (NoC - *Network on Chip*). Estes sistemas são denominaodos de MPSoCs (*Multi Processor System on Chip*).

Além de isto, procura-se a integração dessas plataformas (tipicamente heterogêneas) seguindo um processo de verificação funcional dos componentes, envolvendo a integração de módulos desenvolvidos por companhias diferentes (Teich 2012, Llanos, Hurtado e Alfaro 2016).

Hoje em dia, devem-se atualizar os sistemas embarcados baseados nas exigências de cada necessidade (referido-se ao ciclo útil), o que gera uma maior complexidade no fluxo de desenvolvimento do produto final ou *chip*. Esse processo temporal que vai da geração da ideia até o produto final é conhecido como *Time-to-market* (TTM) (Teich 2012). Tipicamente, os sistemas embarcados são compostos por GPPs, DSPs, GPUs, FPGAs, e circuitos específicos como os ASICs, os quais podem ser implementados em modulos complexos assim como os observados nos SoCs (*System-on-Chip*) e os MPSoCs (Llanos, Hurtado e Alfaro 2016).

O desempenho dos algoritmos que são implementados em plataformas *software* depende fortemente do tipo de arquitetura da plataforma, por exemplo GPPs, GPUs, e DSPs (Teich 2012). Em outro caso, apresentam-se outras opções de plataformas com potencial melhor desempenho, tais como os FPGAs e os ASICs. No entanto, o custo de desenvolvimento de um ASIC, relacionado ao custo não recorrente (*Non Recurring Expenses*-NRE), podem ser de alto custo para TTMs curtos (Knapp 1995).

Das plataformas mencionadas anteriormente, os FPGAs (devido a sua flexibilidade e facilidade de prototipagem) apresentam-se como candidatos para procurar um balanço entre o TTM e o NRE (Llanos, Hurtado e Alfaro 2016).

Outra das vantagens dos FPGAs é a integração com outras plataformas computacionais, tais como os processadores *hard-core* e a a possibilidade do uso de processadores do tipo *soft-core*. Isto possibilitou o estudo e prototipação de sistemas com particionamento de tarefas que poderiam ser feitas em *software* e em *hardware*, acelerando assim o desempenho dos algoritmos que tomam vantagem das duas abordagens (Teich 2012). A implementação em *software* pode ser feita via processadores *soft-core* como o Microblaze e/ ou NIOS II, ou via processadores *hard-core* como o ARM (Göhringer et al. 2011).

Exemplos de aceleração de algoritmos em *hardware* usando operações em ponto flutuante, comparados com o desempenho em *software* (Microblaze ou PC) podem ser encontrados em (Muñoz et al. 2009, Munoz et al. 2010, Muñoz et al. 2013, Santos et al. 2017, Sampaio et al. 2017). Nessas trabalhos, a implementação em FPGA apresenta melhor desempenho que

as implementações em um processador do tipo *soft-core* (como o Microblaze/NIOS II) e em PC, mostrando as vantagens da exploração do paralelismo que pode ser realizada em algoritmos mapeados diretamente em *hardware*.

4.2.1 FPGAs - Field Programmable Gate Arrays

Os FPGAs são circuitos integrados compostos por um arranjo de blocos lógicos configuráveis (CLBs) que podem ser programadas para executar funções lógicas e armazenar dados. Igualmente, são compostos de interconexões programáveis (PIs) para efetuar roteamento de sinais para outros CLBs. Outros componentes dos FPGAs são:

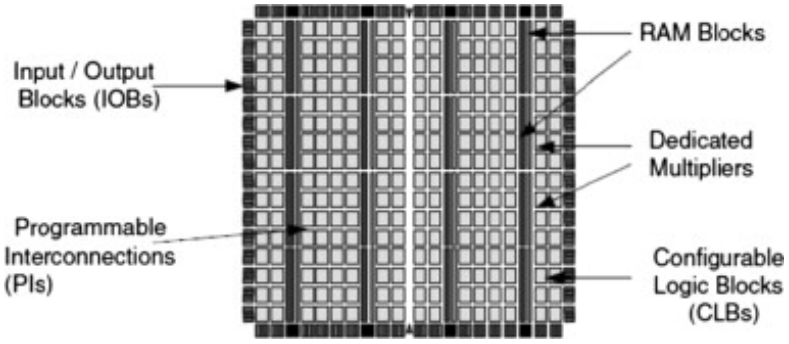


Figura 4.1: Exemplo de distribuição dos CLBs, IOBs, PIs, blocos RAM e multiplicadores dentro de um FPGA (Deschamps, Bioul e Sutter 2006).

- **Blocos I/O:** são a interface entre o FPGA e o mundo exterior.
- **Blocos de lógica configurável:** possuem a lógica combinacional e elementos de armazenamento. Os CLBS estão compostos de *look-up tables* (LUTs) podendo ser programadas com memórias RAMs síncronas.
- **Blocos RAM:** um FPGA contem uma grande quantidade de blocos de memória RAM organizados em colunas ao longo do *chip*. Dependendo da família, o número de blocos começa desde 8 até mais de 100 blocos.
- **Recursos aritméticos:** um FPGA pode possuir circuitos específicos desenvolvidos para operações matemáticas como somadores, multiplicadores, além de outras operações.

Assim, o FPGA oferece a possibilidade implementar qualquer algoritmo desejado devido ao fato de possuir somadores, multiplicadores, DSPs, capacidade de memória (*On-chip memory*), processadores *soft-core* e *hard-core* tais como: NIOS II e ARM. Neste contexto, o FPGA fornece uma grande capacidade de paralelismo elaborando dois ou mais copias do modulo que vai ser implementado. Sua programação é realizada através de linguagens de descrição de *hardware* HDL (*Hardware Description Language*) sendo as mais comuns o VHDL (*VHSIC Hardware Description Language*) e o Verilog.

Atualmente, os FPGAs oferecem arquiteturas do tipo SoC (*System-on-Chip*) assim como do tipo MPSoC (*Multiprocessor System-on-Chip*) com a finalidade de acelerar aplicações que requerem processamento intensivo de dados (Göhringer et al. 2009). A Altera (Hall e Hamblen 2004) oferece em seus FPGAs a possibilidade de configurar o processador NIOS II que é um processador *Soft-core* (processador reconfigurável que utiliza os elementos lógicos do FPGA). Também estão disponíveis processadores *hard-core* (SoC) como o ARM na família Cyclone V da Altera, o que torna soluções de co-design mais eficientes.

Recentemente, a Altera disponibilizou FPGAs da família Stratix 10 com tecnologia de fabricação com transistores de 14 nm, 70% de redução de consumo de potência comparado com versões anteriores, 5.510.000 de elementos lógicos, processador ARM Quad-core de 64 bits rodando a 1,5 GHz, incremento de unidades de memória, entre outras características (Altera, Altera 2018). Estas características, tornam estes dispositivos cada vez mais atrativos para uso em larga escala (Lewis et al. 2016).

4.2.2 Etapas de um projeto com FPGA

O fluxo de projeto de um FPGA possui muitos passos em comum com o fluxo dos ASIC. As etapas necessárias são as seguintes:

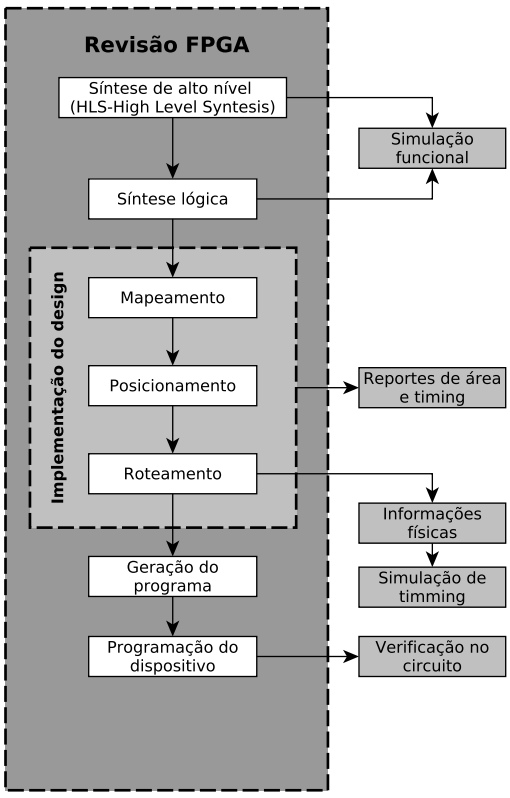


Figura 4.2: Fluxo de projeto em FPGAs (adaptado de (Deschamps, Bioul e Sutter 2006, Pistorius et al. 2007)).

- *Entrada do projeto*: descrição funcional do projeto junto com as restrições de área e desempenho. Neste passo, o esquemático do circuito é projetado usando linguagem de descrição de *hardware* VHDL ou verilog.
- *Síntese lógica*: cria-se a partir do HDL ou esquemático, a lista de elementos lógicos (*netlist*) disponíveis e suas conexões.
- *Verificação comportamental*: verifica o correto funcionamento do circuito a partir do *netlist* gerado.
- *Mapeamento*: determina as funções lógicas dentro dos elementos configuráveis do dispositivo.
- *Posicionamento*: mapeamento das funções lógicas em posições específicas do FPGA.
- *Roteamento*: neste passo é feita a conexão entre os blocos lógicos.
- *Geração do programa*: um arquivo *bit-stream* é gerado com as informações dos componentes do dispositivo.
- *Programação do dispositivo*: carregamento do *bit-stream* no FPGA.
- *Simulação funcional*: inclui a simulação comportamental do design e a simulação de *timing* com as informações físicas dos componentes do circuito. A verificação do projeto é feita usando um arquivo chamado de *testbench* escrito em VHDL. Neste arquivo estão especificados os vetores de teste para comprovar o funcionamento do circuito.
- *Verificação no circuito*: a verificação é feita depois da programação do dispositivo.

4.3 IMPLEMENTAÇÕES EM *HARDWARE* PARA DILATAÇÃO E EROSÃO

Em (Bartovskỳ et al. 2015), Bartovsky implementa um co-projeto *hardware*/*software* para várias operações morfológicas (vide Figura 4.3). Bartovskỳ, Dokládal, Dokladalova, Faessel, e Bilodeau implementam soluções *hardware* para as operações básicas da matemática morfológica como dilatação, erosão, a concatenação delas, e um operador de reconstrução morfológica. A arquitetura foi implementada numa FPGA Virtex-5 com Microblaze usando uma memória externa DDR2 de 512 MB e controlada por computador externo usando linguagem C/C++ e Python. Nesta implementação, as imagens de entrada e saída são controladas por uma unidade nomeada como *Multi-Port Memory Controller*-MPMC que sincroniza a comunicação entre a unidade *hardware* e a memória externa.

Na Figura 4.3, observa-se que as imagens de entrada e saída são armazenadas em *buffers* internos do FPGA. O desempenho da arquitetura depende da velocidade de comunicação entre a unidade co-processadora e a unidade MPMC, assim como o tamanho de armazenamento das unidades de memória internas.

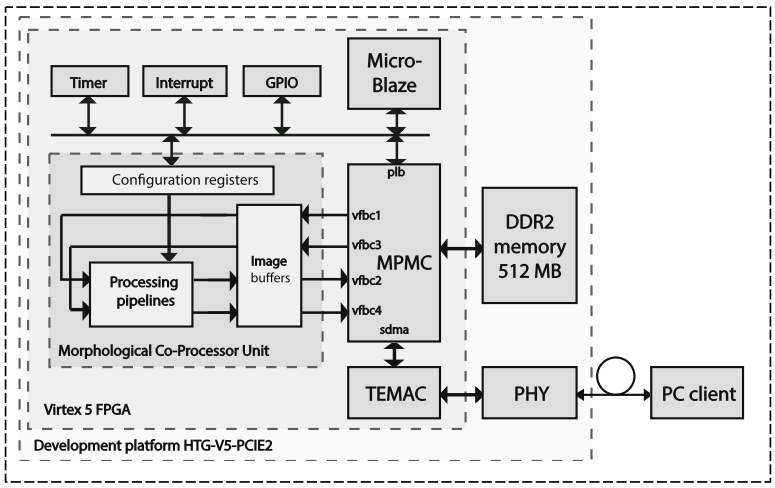


Figura 4.3: co-projeto da unidade coprocessadora morfológica implementada em (Bartovskỳ et al. 2015).

Bartovskỳ, Dokládal, Faessel, Dokladalova e Bilodeau implementam um *pipeline* de sucessivas dilatações e erosões que podem ser chaveadas para a implementação de filtros sequenciais alternados (*Alternate Sequential Filter-ASF* ou operações geodésicas (vide Figura 4.10a). Na Figura 4.10a é mostrada um elemento de processamento da proposta de (Bartovskỳ et al. 2015), pode-se observar que as operações de dilação e erosão são executadas em sequencia e em *pipeline*, conseguindo processar múltiplas imagens em paralelo. As unidades morfológicas apresentam uma grande escabilidade variando o tamanho do elemento estruturante desde 3×3 até octágonos com 43 pixels de diâmetro.

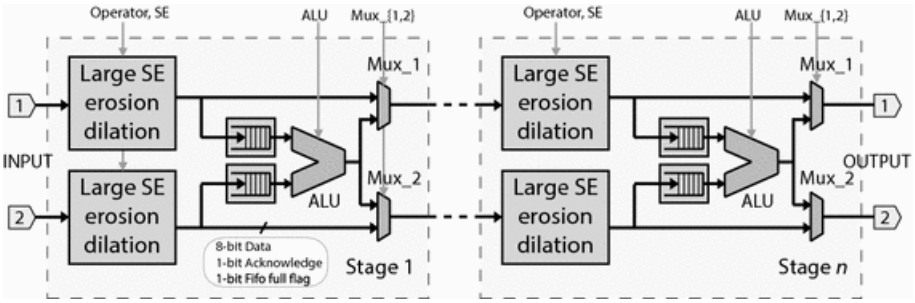


Figura 4.4: Arquitetura em *pipeline* dos operadores erosão/dilatação (Bartovskỳ et al. 2015).

Bartovskỳ, Dokládal, Faessel, Dokladalova e Bilodeau mostram comparações de desempenho entre a arquitetura proposta e plataformas computacionais como um processador ARM A9 rodando a 1 GHz com 1 GB de DDR3 e rodando um Linux embarcado, e PC Intel Xeon E5620 rodando a 2,40GHz. O resultado para uma operação de dilatação mostra que possui um tempo de processamento menor que as plataformas *software*. No entanto, para operações onde são necessárias operações rodando em paralelo como o filtro ASF, a arquitetura em FPGA apresenta um melhor desempenho no quesito de tempo de processamento.

Adicionalmente, o co-projeto foi comparado com outras arquiteturas em *hardware* comparando métricas como vazamento (Mpx/s), frequência máxima (MHz), tamanho de imagem, quadros por segundo (*frames per second*-FPS) e tempo de latência. A implementação apresentou um tempo

de latência menor ($84N$), tamanho de imagem de até 1024×1024 pixels, processamento de 185 quadros por segundo e frequência máxima de 100 MHz.

A arquitetura de Bartovskỳ, Dokládal, Faessel, Dokladalova e Bilodeau apresenta um co-projeto *hardware/software* que possui uma grande flexibilidade para escolher vários tamanhos do elemento estruturante, escabilidade para escolher o número de estágios que uma operações pode ter e um boa implementação para a adquisição e transmissão das imagens por video usando o processador Microblaze. A diferença de (Dokládal e Dokládalová 2011), Bartovskỳ, Dokládal, Faessel, Dokladalova e Bilodeau não apresentam um analise da dependência do número de estágios do *pipeline* com o conteúdo da imagem.

Uma configuração de um filtro ASF com $n = 6$ é composta de 13 operações morfológicas, assim $ASF^6 = \delta_{13\times13}\epsilon_{25\times25}...\delta_{5\times5}\epsilon_{3\times3}$. Em (Bartovský et al. 2015), Bartovskỳ, Dokládal, Bilodeau, Dokladalova, e Akil conseguem desenvolver uma arquitetura que particiona a imagem em vários blocos de uma dimensão, alcançando executar o filtro ASF^6 em uma iteração. O resultado conseguido é comparado com o desempenho obtido em (Chien, Ma e Chen 2005) com 45 varreduras, (Clienti, Beucher e Bilodeau 2008) com 6 varreduras, e (Déforges, Normand e Babel 2013) com 13 varreduras.

Bartovskỳ, Dokládal, Bilodeau, Dokladalova, e Akil implementam o sistema numa FPGA Virtex-6 para imagens de 1920×1080 pixels. A abordagem de um *pipeline* de dilatação e erosão em sequencia é usada também por Torres-Huitzil em (Torres-Huitzil 2016). Torres-Huitzil mostra várias comparações de desempenho com as placas usadas (Spartan-6 e Virtex-6 FPGAs) com uma GPU NVIDIA GTX 470 com 448 CUDA núcleos, e uma CPU AMD Athlon II rodando a 3067MHz com sistema operacional Ubuntu 10.04 LTS. Torres-Huitzil mostra o ganho no tempo de processamento do FPGA sobre as outras plataformas.

Uma implementação semelhante é apresentada em (Mukherjee, Mukhopadhyay e Biswas 2017), Mukherjee, Mukhopadhyay e Biswas implementam um *pipeline* de dilatações e erosões em sequencia para vários tamanhos de elemento estruturante. Em (Mukherjee, Mukhopadhyay e Biswas 2017), o elemento estruturante é processado usando uma decomposição em três vetores de 1-D. Mukherjee, Mukhopadhyay e Biswas compara o desempenho da implementação proposta com outros trabalhos em *hardware* como as implementações de (Holzer et al. 2012, Gibson et al. 2013, Bartovský et al. 2014, Mukherjee, Mukhopadhyay e Biswas 2016). As principais métricas usadas foram o tamanho e forma do elemento estruturante, frequência máxima de operações, latência, taxa de quadros por segundo, vazamento, potencia dinâmica, e consumo de elementos lógicos. As implementações mencionadas foram implementadas numa FPGA Virtex-5 para processamento em tempo real.

Resumindo, a Tabela 4.1 as diferentes implementações em várias plataformas , pode-se observar que as implementações são visadas para atingir o requerimento de operação em tempo real para diferentes tipos de imagens. As operações de dilatação e/ou erosão são feitas de forma sucessiva.

Tabela 4.1: Comparações de desempenho para as operações morfológicas de dilatação ou erosão

Ref.	Tecnologia	SE	Fmax (MHz)	Tamanho de imagem (pixels)	Throughput (Mpixel/s)	FPS	Latencia (clk)
Chien (Chien, Ma e Chen 2005) (2005)	ASIC	5×5	200	720×480	190	12,2	40.320
Clienti (Clienti, Beucher e Bilodeau 2008) (2008)	Virtex-4	3×3	100	1024×1024	403	66,7	86.016
Déforges (Déforges, Normand e Babel 2013) (2013)	APEX 20KC	7×7	50	512×512	50	14,7	3.078
Gibson (Gibson et al. 2013) (2013)	Virtex-5	7×7	254	1024×768	254	323	4.628
Bartovskỳ (Bartovský et al. 2014) (2014)	Virtex-5	31×31	100	1920×1080	234	113	11.641
Bartovskỳ (Bartovskỳ et al. 2015) (2015)	Virtex-6	Pol. Reg.	100	1024×1024	195	185	14.336
Torres-Huitzil (Torres-Huitzil 2016) (2016)	Virtex-6	51×51	260	512×512	119	-	-
Mukherjee (Mukherjee, Mukhopadhyay e Biswas 2016) (2016)	Virtex-5	7×7	206	1024×768	219	278	4.599
Mukherjee (Mukherjee, Mukhopadhyay e Biswas 2017) (2017)	Virtex-5	7×7	164	1024×768	164	51	32.456

4.4 ARQUITETURAS EM *HARDWARE* PARA COMPONENTES CONEXOS

No caso do algoritmo de componentes conexos, as imagens processadas são binárias, diferente das imagens para o algoritmo de reconstrução morfológica que são em escala de cinza. No entanto, quando um pixel é etiquetado usando o algoritmo de componentes conexos, este valor de etiqueta pode ser propagado para a vizinhança, dependendo da conectividade do pixel. Devido a este fato, algumas implementações do algoritmo de componentes conexos podem ser usadas como base para a implementação do algoritmo de reconstrução morfológica. Neste contexto, vários trabalhos reportam implementações de componentes conexos em plataformas FPGA, entre eles estão as propostas que usam mono-varredura (escaneamento *raster*) (Dae Ro Lee et al. 2007), (Pandey et al. 2014), (Klaiber et al. 2016), (Tsai, Ho e Tsai 2018), e (Spagnolo, Perri e Corsonello 2019), e multi-varredura (escaneamento *raster* e *anti-raster*) como em (Crookes 1999), (Benkrid et al. 2003), (Chan et al. 2013), e (Schwenk e Huber 2015).

Com o intuito de poupar recursos *hardware* e obter um ganho no desempenho das arquiteturas, todos os trabalhos implementam os algoritmos de componentes conexos para evitar conflitos de etiquetamento entre pixels vizinhos, assim como o uso de memórias externas para leitura e escrita dos pixels etiquetados. Por exemplo, em (Spagnolo, Perri e Corsonello 2019) é usada uma plataforma Zynq-7000 da Xilinx, onde através da DMA (*Direct Memory Acces*) é acessada uma memória externa para transferir a imagem para a arquitetura, e para enviar os resultados da arquitetura para a memória externa (vide Figura 4.5. Esta implementação permite minimizar o uso de memória interna do FPGA, especificamente BRAM.

De forma geral, as implementações das arquiteturas *hardware* são baseadas na Figura 4.6. Nesta figura, a arquitetura possui circuitos para o etiquetamento dos pixels (*label selection*), disponibilizar a vizinhança que vai ser processada (*Neighborhood context*), um buffer de pixels que carregam uma fila da imagem (*Row buffer*), e um sistema de controle de varredura (*Scan control*).

Em (Crookes 1999) e (Benkrid et al. 2003) são implementadas uma arquitetura do algoritmo *multi-scan* para componentes conexos em *hardware*. No algoritmo a imagem principal é tratada com a estrategia de particionamento, onde cada bloco é processado por um circuito diferente. As varreduras da imagem são feitas no sentido *raster* e *anti-raster* aproveitando o mesmo circuito. A única diferença entre esses dois tipos de varredura é a posição do pixel de entrada da imagem. No

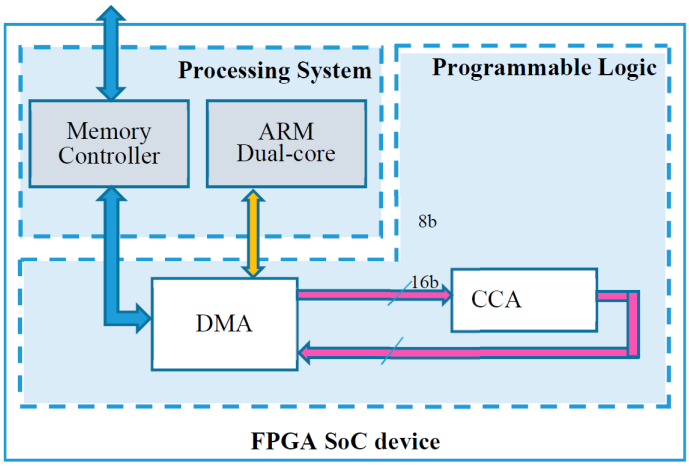


Figura 4.5: Arquitetura proposta por (Spagnolo, Perri e Corsonello 2019) implementada numa plataforma SoC .

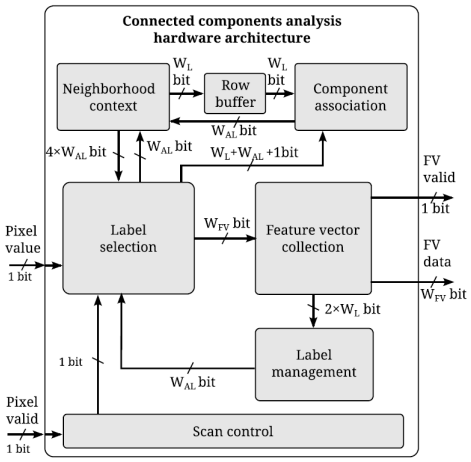


Figura 4.6: Diagrama de blocos da arquitetura proposta para componentes conexos em (Klaiber et al. 2016).

sentido *raster*, o pixel na posição (0,0) é o primeiro pixel de entrada e o pixel na posição (M-1,N-1) é o ultimo pixel em ser processado. Caso contrario acontece na varredura *anti-raster*, o primeiro pixel de entrada é o pixel na posição (M-1,N-1) e o pixel na posição (0,0) é o ultimo pixel. Ambas implementações possuem tratamento nas bordas para evitar distorções na imagem.

No entanto, essa arquitetura possui um número indeterminado de varreduras até estabilizar a imagem, isto é, identificar cada objeto da imagem. Além disso, a arquitetura deve salvar a imagem depois da varredura *raster* em uma memória RAM externa para depois disponibilizar para a etapa *anti-raster*. Em (Chan et al. 2013), Chan *et al.* implementa uma arquitetura para componentes conexos baseada no trabalho de (Crookes 1999). Chan *et al* usa a metodologia de varreduras *raster* e *anti-raster* (vide Figura 4.7). Na varredura *raster*, os pixels usados são A, B, C, D, e E, sendo A o pixel que está sendo tratado. No entanto, para a varredura no sentido *anti-raster*, os pixels usados são A e F. As varreduras são usadas para colocar as etiquetas parciais a cada pixel (vide Figura 4.8).

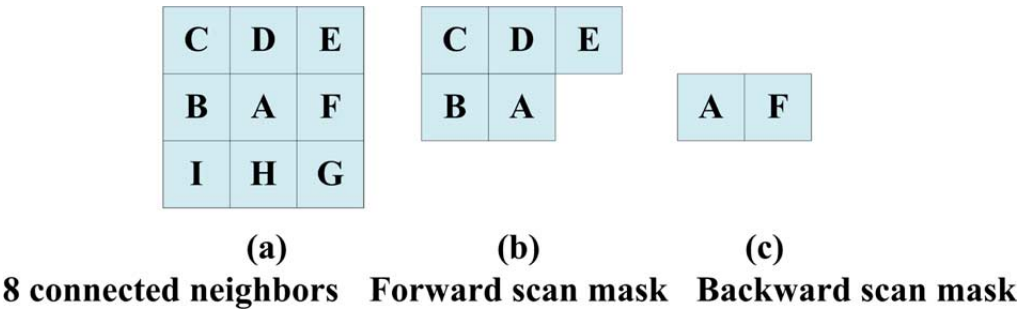
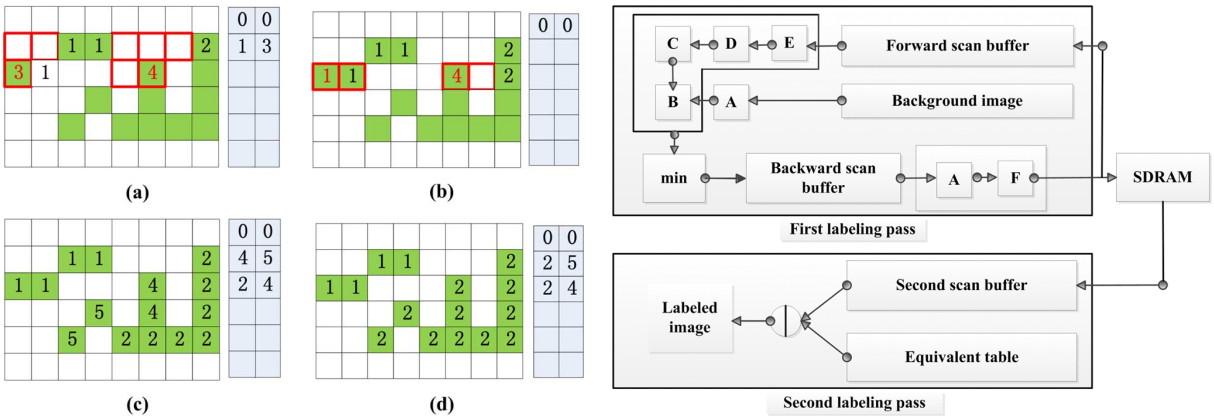


Figura 4.7: Vizinhança usada por Chan *et al.* para componentes conexos (Chan et al. 2013).

Semelhante ao algoritmo de reconstrução híbrido, Chan *et al.* usam o escaneamento *anti-raster* para gerar uma tabela onde são armazenados cada par de pixels que possuem conflito de etiquetamento, isto é, pixels vizinhos que possuem etiquetas diferentes (vide Figura 4.8). A Figura 4.8a mostra um exemplo onde vários pixels possuem conflito de etiquetamento (pixels de cor verde). Por outro lado, a Figura 4.8b mostra a arquitetura para o circuito de componentes conexos onde os valores das etiquetas são armazenados e carregados num memória externa SDRAM.



(a) Exemplos de escaneamento raster e anti-raster junto com a tabela para resolução de etiquetas

(b) Arquitetura do processamento de componentes conexos

Figura 4.8: Metodologia usada para processamento de etiquetamento mostrada em (Chan et al. 2013).

Em (Dae Ro Lee et al. 2007), (Pandey et al. 2014) e (Tsai, Ho e Tsai 2018) são implementadas arquiteturas com escaneamento raster, especificamente duas varreduras raster. Todas as implementações são usadas para vídeo em tempo real, e usam memórias externas para armazenar a imagem de entrada. Em (Tsai, Ho e Tsai 2018), Tsai e Ho usam uma placa Vertex-5 rodando a 128 MHz para processar uma imagem de 640x480 e 1280x720 pixels. No caso de (Pandey et al. 2014), Pandey *et al.* usa a Virtex-5 como sistema de adquisição da imagen e usa um processador PPC440 para rodar o algoritmo de componentes conexos. Na Tabela 4.2, pode-se observar as implementações para componentes conexos que usam a metodologia multi-varredura usando *raster* e *anti-raster*, ou mono-varredura usando só o *raster*.

Tabela 4.2: Comparações de desempenho para as implementações de componentes conexos

Ref.	Tecnologia	SE	Tamanho imagem	Frequencia (MHz)	Recursos lógicos (%)	BRAM (%)
Crookes (Crookes 1999) (1999)	Xilinx XC400	3×3	256×256	76	40	-
Benkrid (Benkrid et al. 2003) (2003)	Virtex-E	3×3	1024×1024	72	3	3
Chan (Chan et al. 2013) (2013)	Spartan-6	3×3	1920×1080	-	8	-
Pandey (Pandey et al. 2014) (2014)	Virtex-5	-	512×512	100	8	5
Li (Li 2015) (2015)	Stratix-IV	2×2	1920×1080	100	-	-
Schwenk (Schwenk e Huber 2015) (2015)	Virtex-6	-	1024×1024	100	1	1
Klaiber (Klaiber et al. 2016) (2016)	Kintex-7	-	7680×4320	170,53	-	3
Tsai (Tsai, Ho e Tsai 2018) (2018)	Virtex-5	3×3	1280×720	128	-	-
Spagnolo (Spagnolo, Perri e Corsonello 2019) (2020)	Zynq-7000	3×3	640×480	140	-	0
Zhao (Spagnolo, Perri e Corsonello 2019) (2020)	Zynq-7000	3×3	2048×1536	316	-	4

4.5 IMPLEMENTAÇÕES *HARDWARE* PARA ALGORITMOS POR DIVISOR DE ÁGUAS

O algoritmo por divisor por águas pertence a outro nível de abstração no processamento de imagens (vide Seção 3.6). Este algoritmo possui o mesmo princípio de funcionamento que o algoritmo de reconstrução morfológica e o de componentes conexos. Neste caso, o pixel que está sendo tratado pode propagar seu valor para sua vizinhança, conhecida como simulação por inundação. O algoritmo por divisor de águas, além de propagar o valor de intensidade de um pixel, o pixel também pode ser etiquetado e propagar o valor dessa etiqueta.

Nesta seção, mostram-se algumas implementações que usam a metodologia de varredura *raster* e/ou *anti-raster* junto com componentes conexos. Por exemplo, em (Sivakumar e Janakiraman 2020) é proposta uma arquitetura para segmentação de imagens médicas usando o algoritmo por divisor de águas (vide Figura 4.9). Neste trabalho, faz-se um pre-processamento da imagem e posteriormente é feita a segmentação da imagem. As fases da implementação são: leitura da imagem de entrada, aplicação de um filtro passa altas, deteção de bordas usando o filtro Canny, e a segmentação usando o algoritmo por divisor de águas. Todas as etapas do trabalho são implementadas em *hardware*. Os resultados da segmentação das imagens, ou seja, a deteção dos tumores no cerebro, são comparados com os resultados com outras propostas do algoritmo por divisor de águas e com o algoritmo *k-means*.

Em (Trieu e Maruyama 2006), (Trieu e Maruyama 2007) e (Trieu e Maruyama 2007) são propostas arquiteturas em *pipeline* da transformada baseada em componentes conexos para FPGA. O algoritmo segue três passos: (a) varredura no sentido *raster*/*anti-raster* para marcar ou atualizar os mínimos locais, (b) varredura no sentido *raster*/*anti-raster* para calcular ou atualizar as distâncias desde os mínimos locais até os plateaus, e (c) varredura no sentido *raster*/*anti-raster* para rotulação dos pixels. Nesta arquitetura, os passos a-c são implementados em *pipeline* para cada varredura e as imagens são lidas/armazenadas desde memórias externas.

Em (Rambabu e Chakrabarti 2007), Rambabu e Chakrabarti implementam uma arquitetura para o algoritmo de transformada por divisor de águas para a segmentação de imagens. A implementação *hardware* de (Rambabu e Chakrabarti 2007) usa uma etapa de escaneamento e processamento de

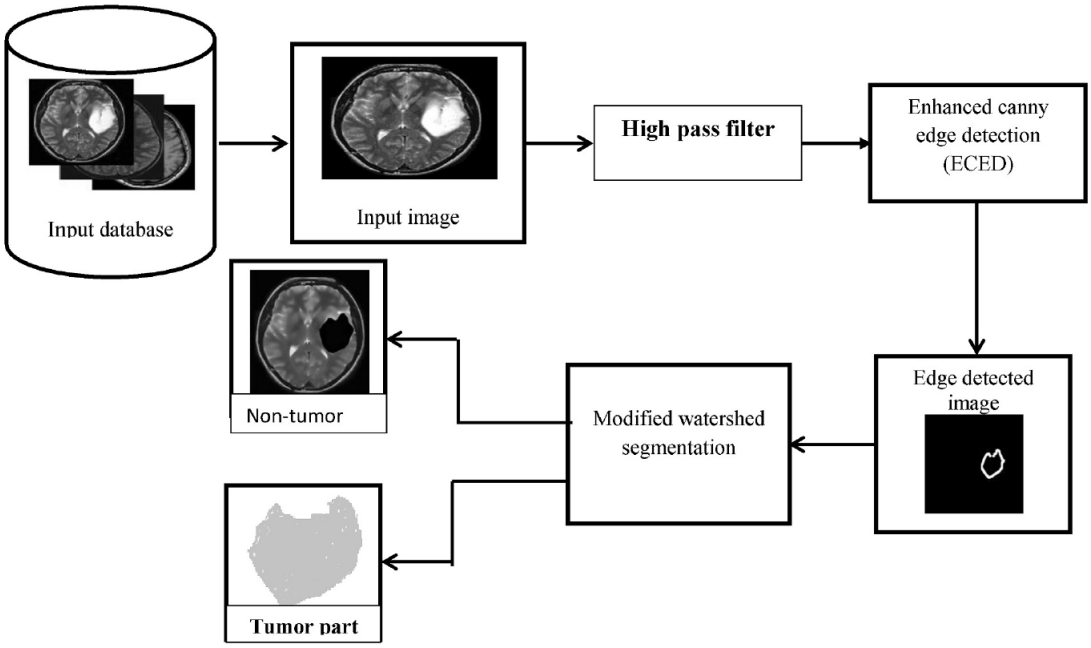


Figura 4.9: Processo de segmentação de uma imagem proposto em (Sivakumar e Janakiraman 2020).

uma fila. De forma semelhante que o algoritmo de reconstrução morfológica, a implementação de (Rambabu e Chakrabarti 2007) começa marcando os pixels de interesse (valores mínimos) e depois processa as regiões de interesse usando uma fila (FIFO). O algoritmo proposto por (Rambabu e Chakrabarti 2007) possui um melhor desempenho que o algoritmo original proposto por Vincent (Vincent e Soille 1991). O algoritmo de Rambabu e Chakrabarti evita que duas bacias diferentes se sobreponham, assim como evita deixar linhas divisoras incompletas e mínimos locais isolados.

Em (Trieu e Maruyama 2008), Trieu e Maruyama implementaram uma arquitetura para divisor por águas usando componentes conexos usando varredura *raster* e fila de prioridade. Durante o escaneamento *raster*, os pixels que são mínimos locais são encontrados e sua vizinhança é armazenada dentro de uma fila para a propagação da etiqueta do mínimo local. Essa implementação segue a metodologia do algoritmo híbrido de reconstrução morfológica, varredura da imagem e processamento por fila. Nesta implementação, são usados vários escaneamentos para a segmentação da imagem. Trieu e Maruyama usam a métrica de distância para propagar as etiquetas devido a que cada etiqueta deve ser atribuída para o mínimo local mais próximo.

Em (Yeong et al. 2009) são propostas três tipos de arquiteturas para implementar a transformada por divisor de águas, assim: sequencial, paralelo e baseado em grafos. Na abordagem sequencial, a leitura dos pixels de interesse (vizinhança) são carregados em 5 ciclos de relógio. No entanto, na abordagem paralela e baseada em grafos, a vizinhança é carregada em um ciclo de relógio. A diferença entre as duas abordagens é o consumo de memória, a abordagem paralela necesita de cinco blocos de memória para carregar todos os pixels da vizinhança. A arquitetura possui três etapas de funcionamento: escaneamento raster, enfileirsamento, processamento da fila e etiquetamento.

Em (Sameer 2012), foi implementado uma arquitetura em *hardware* do algoritmo da transformada baseada em componentes conexos para processar imagens em tempo real. Ruparelia (Sameer 2012) usou um algoritmo híbrido de segmentação baseado em limiarização (*threshold based*), bordas (*edge*) e regiões usando a transformada. A arquitetura possui múltiplas unidades de segmentação em *pipeline* junto com uma memória externa (*DDR RAM*).

Na Tabela 4.3, pode-se observar as implementações para transformada por divisor de águas que usam escaneamento *raster* e/ou *anti-raster*. Pode-se observar que dependendo da metodologia do algoritmo como o uso de varreduras raster/anti-raster, fila simples ou de prioridade, memória interna (BRAMs) ou externa (SDRM ou DDR), a quantidade no consumo de recursos lógicos pode variar significativamente.

Tabela 4.3: Comparações de desempenho para as operações morfológicas de dilatação ou erosão

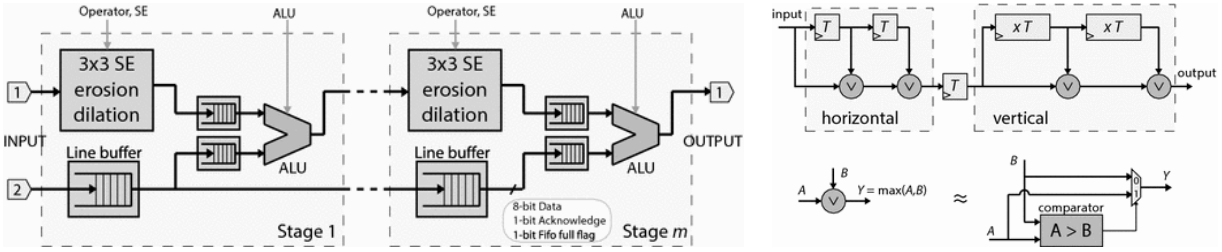
Ref.	**Tecnologia**	**SE**	**Tamanho imagem**	**Frequencia (MHz)**	**Recursos lógicos (%)**	**RAM (%)**
Trieu (Trieu e Maruyama 2006) (2006)	Virtex-II	3×3	512×512	66	92	95
Trieu (Trieu e Maruyama 2007) (2007)				61	93	60
Trieu (Trieu e Maruyama 2007) (2007)				56	13	13
Trieu (Trieu e Maruyama 2008) (2008)				86	10	82
Ruparelia (Sameer 2012) (2012)	Virtex-4			65	70	10
Sivakumar (Sivakumar e Janakiraman 2020) (2020)	Virtex-5			-	1	-
Rambabu (Rambabu e Chakrabarti 2007) (2007)	Virtex	3×3	16×16	48,52	52	30
Yeong (Yeong et al. 2009) (2009)	Spartan-3	3×3	64×64	-	20	5

4.6 ARQUITETURAS *HARDWARE* PARA RECONSTRUÇÃO MORFOLÓGICA

A literatura apresenta poucos trabalhos implementados para reconstrução morfológica usando as versões sequencial e híbrida. No entanto, em (Bartovskỳ et al. 2015) e (Jivet, Brindusescu e Bogdanov 2008) foram implementadas propostas usando dilatações e erosões geodésicas. Esses algoritmos executam sucessivas operações morfológicas de dilatação e erosão. As implementações citadas usam a varredura no sentido *raster* junto com toda a vizinhança. Devido ao processo de varredura da imagem e ao uso da redundância no tratamento de cada vizinhança, essas implementações levam muitas iterações para alcançar a estabilidade (convergência da solução).

Como mencionado anteriormente, Bartovskỳ, Dokládal, Faessel e Dokladalova (Bartovskỳ et al. 2015) implementam soluções *hardware* para as operações básicas da matemática morfológica como dilatação, erosão, a concatenação delas, e um operador de reconstrução morfológica. As arquiteturas foram implementadas em forma de *pipeline* de *n* estágios devido á estrutura sequencial dos algoritmos. Assim, pode-se ter vários circuitos executando as operações morfológicas em paralelo.

A Figura 4.10 mostra as arquiteturas em *pipeline* das unidades geodésicas para a reconstrução morfológica. O *pipeline* permite processar várias imagens em paralelo com tamanho de elemento estruturante de 3×3 pixels. Na Figura 4.10a, são apresentados os módulos de erosão e dilatação



(a) Operador geodésico (b) Módulo de comparação

Figura 4.10: Arquitetura em *pipeline* do operador geodésico (Bartovskỳ et al. 2015).

assim como uma FIFO e uma unidade ALU. A FIFO é usada para sincronizar os valores da imagem marcadora e máscara para calcular o valor máximo/mínimo na ALU (vide Figura 4.10b). Por outro lado, a convergência da solução para reconstrução é lenta, levando muitas iterações. Segundo (Teodoro et al. 2013), esse tipo de arquiteturas não exploram eficientemente a vizinhança de cada pixel, causando que a reconstrução perca desempenho.

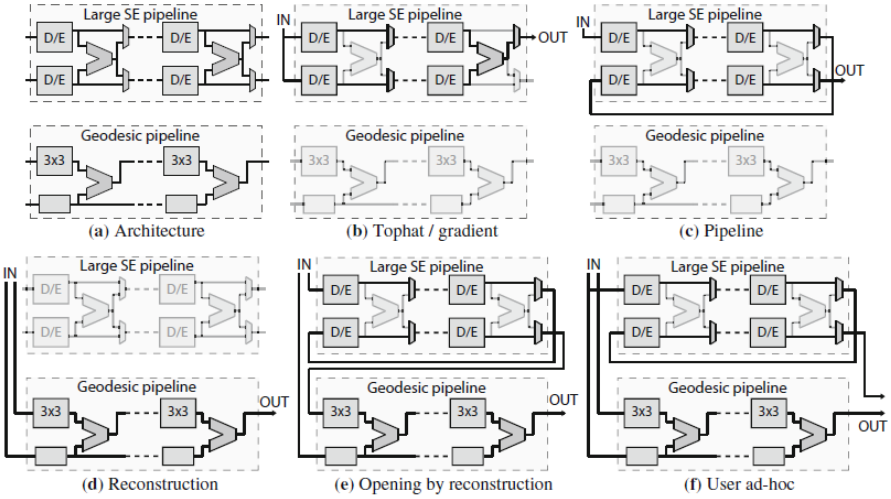


Figura 4.11: Unidade de dilatação do operador geodésico (Bartovskỳ et al. 2015).

A Figura 4.10a apresenta o circuito para a propagação do pixel que está sendo tratado. Neste caso, o cálculo do valor máximo da vizinhança é feito em forma de decomposição da vizinhança, isto é, calcular individualmente o máximo de cada fila e compará-lo com o resultado da fila anterior (vide Figura 4.10b). Finalmente, o resultado é armazenado no banco de registradores e carregado no pixel que esta sendo tratado. Na Figura 4.11,são apresentadas as configurações para as operações morfológicas usando as arquiteturas das Figuras 4.10a e 4.10. As operações morfológicas são *top-hat*, gradiente, erosão/dilatação, reconstrução, reconstrução por abertura e *ad-hoc*. Nesta arquitetura, a escolha de cada operação é programada pelo usuário.

A arquitetura de Bartovskỳ, Dokládal, Faessel e Dokladalova apresenta um co-projeto que possui uma grande flexibilidade para escolher vários tamanhos do elemento estruturante, escabilidade para escolher o numero de estágios que uma operações pode ter e um boa implementação para a adquisição e transmissão das imagens por video usando o processador Microblaze (vide

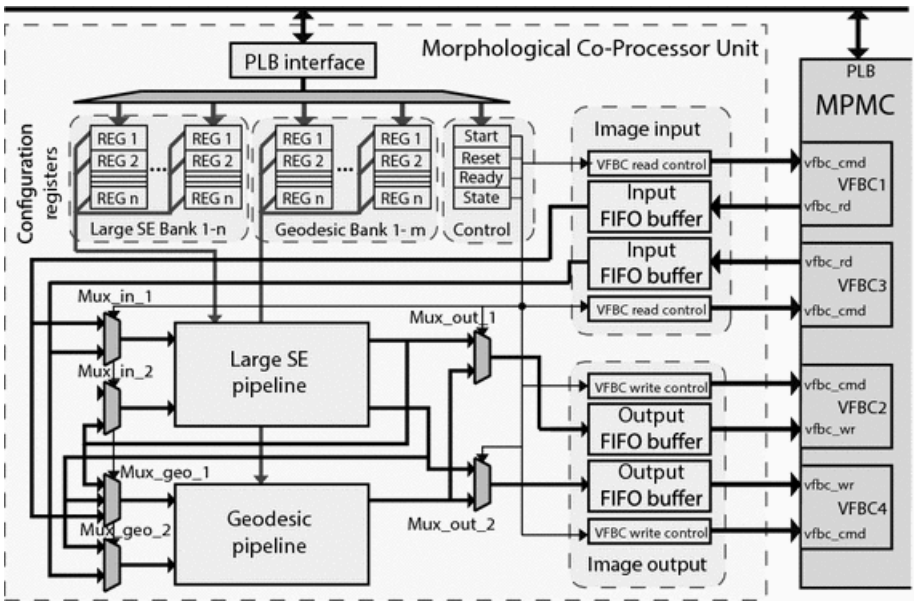


Figura 4.12: Arquitetura do co-projeto *hardware*/*software* (Deschamps, Bioul e Sutter 2006).

Figura 4.12). A Figura 4.12 apresenta o co-projeto *hardware*/*software* da unidade co-processadora, onde observa-se que o usuário pode escolher entre processar a imagem usando as operações morfológicas básicas ou as operações geodésicas.

No entanto, no quesito de desempenho da reconstrução morfológica, essa implementação é muito ineficiente, além de não possuir uma implementação como o uso de filas para acelerar a convergência da solução. A inclusão das arquiteturas de reconstrução usando o algoritmo híbrido pode contribuir para um co-projeto de melhor qualidade e desempenho para o trabalho apresentado anteriormente.

Na Tabela 4.4, pode-se observar as implementações para reconstrução morfológica. Na literatura, encontram-se implementações *hardware* para o algoritmo paralelo em (Jivet, Brindusescu e Bogdanov 2008) e (Bartovskỳ et al. 2015). Recentemente, foi publicado a versão sequencial implementada pelo autor desde trabalho em (Anacona-Mosquera et al. 2018).

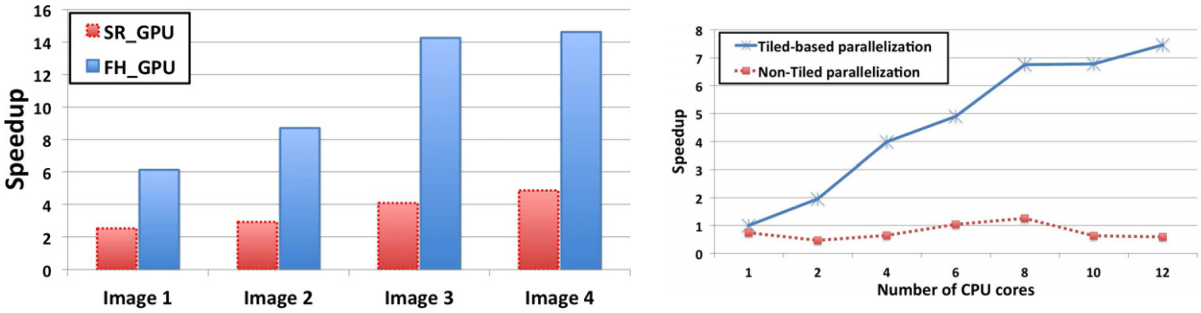
Tabela 4.4: Comparações de desempenho para as operações morfológicas de dilatação ou erosão

Ref.	**Tecnologia**	**SE**	**Tamanho imagem**	**Frequencia (MHz)**	**Recursos lógicos (%)**	**RAM (%)**
Jivet (Jivet, Brindusescu e Bogdanov 2008) (2008)	Cyclone-II	33×33	256×256	-	40	45
	Virtex-E		512×512	-	74	80
Bartovskỳ (Bartovskỳ et al. 2015) (2015)	Virtex-5	3×3	1024×1024	125	92	95

4.6.1 Implementações em *software* para reconstrução morfológica

Implementações da versão sequencial e híbrida do algoritmo de reconstrução são encontradas em plataformas computacionais como CPU e GPUs. Em (Robinson e Whelan 2004), (Karas 2011), (Teodoro et al. 2013), e (Teodoro et al. 2017) os algoritmos SR e FH foram implementados.

Teodoro *et al.* (Teodoro et al. 2017) mostra que o desempenho dos algoritmos depende da plataforma computacional e da escolha do algoritmo de reconstrução morfológica. O algoritmo FH apresenta um melhor desempenho que o algoritmo sequencial devido ao processo de pré-processamento que marca os pixels que são considerados sementes e no processo final são tratados usando uma fila (vide Figura 4.13). Na Figura 4.13a, pode-se observar o ganho em desempenho do algoritmo FH sobre o algoritmo SR. Por outro lado, a Figura 4.13a mostra que o desempenho do algoritmo FH é diretamente proporcional à nível de paralelização do algoritmo. Semelhante aos algoritmos de componentes conexos e transformada por divisor de águas, a referência (Teodoro et al. 2013) divide a imagem em várias sub-imagens e num processamento posterior resolve as bordas.



(a) Comparação de desempenho dos algoritmos SR e FH

(b) Desempenho da paralelização dos algoritmos

Figura 4.13: Desempenho dos algoritmos de reconstrução morfológica (Teodoro et al. 2013).

Em (Robinson e Whelan 2004) e (Sung et al. 2015) foram propostos algoritmos baseados nas direções de propagação dos pixels para evitar uma re-avaliação dos pixels que são considerados candidatos para propagar seus valores para a vizinhança. De forma semelhante para GPUs no caso da transformada por divisor de águas, (Smistad et al. 2015) apresenta as caraterísticas mais relevantes para a implementação do algoritmo de reconstrução morfológica. Smistad *et al.* apresenta que o algoritmo de reconstrução morfológica poderia ter um bom desempenho em plataformas GPU.

Na Tabela 4.5, pode-se observar as implementações para reconstrução morfológica em plataformas *software*. A literatura mostra que as implementações do algoritmo híbrido em plataformas como a GPU apresentam um melhor desempenho que as implementações em plataformas como a CPU. Teodoro *et al.* (Teodoro et al. 2013) mostra o analise de desempenho da fase da propagação do algoritmo híbrido, onde o tempo de processamento depende da implementação da fila, tamanho da imagem, conteúdo da imagem, e plataforma usada.

4.7 CONCLUSÕES

Os algoritmos de reconstrução morfológica são amplamente usados como pré-filtragem para o processo de segmentação de imagens. O primeiro passo deste processo é isolar as zonas desejadas da imagem, por exemplo ressaltar os pixels de maior intensidade, isto pode ser conseguido através

Tabela 4.5: Implementações em plataformas *software* para o algoritmo de reconstrução morfológica

Ref.	**Tecnologia**	**SE**	**Tamanho imagem**	**Frequencia**	**Memória**
Robinson (Robinson e Whelan 2004) (2014)	Pentium 4	3x3	256×256	1,8 GHz	512 MB
Pavel (Karas 2011) (2011)	Intel Core2 Quad		$512 \times 512 \times 5$	2,4 GHz	8 GB
	GeForce GTX 470 GPU			608 MHz	1280 MB
Teodoro (Teodoro et al. 2013) (2013)	Intel Xeon X5660		$4K \times 4K$ $8K \times 8K$ $16K \times 16K$ $32K \times 32K$	2,8 GHz	24 GB
	NVIDIA Tesla M2090			1,8 GHz	6 GB
Sung (Sung et al. 2015) (2015)	Intel i7		1300×864	3,4 GHz	4 GB
Teodoro (Teodoro et al. 2017) (2017)	Intel Xeon		$4K \times 4K$	1,09 GHz	8 GB
	NVIDIA K20			706 MHz	5 GB

da reconstrução morfológica. Dentro das estrategias mais destacadas anteriormente encontram-se: (a) divisão da imagem em sub-imagens para processar cada uma em paralelo, (b) uso de um pré-processamento eficiente (varreduras *raster* e *anti-raster)*, (c) uso de filas ou tabelas para terminar o processamento do algoritmo, e (d) otimização do espaço (no caso das FPGAs) e/tempo de execução. Diante do exposto, os algoritmos que seguem esta técnica (pré-processamento e fila) são os algoritmos ótimos para restauração de imagens.

Observando a revisão da estado da arte, nenhuma implementação (*software* e *hardware*) estuda o processamento de pixels efetivos (taxa de pixels modificados por varredura) para escolher em qual iteração deve passar ao processamento da fila. Identificando as limitações dos algoritmos, por exemplo arquiteturas com controle de dados (*control flow*) e fluxo de dados (*data flow*), os FPGAs poderiam fornecer uma vantagem sobre as outras plataformas. Neste contexto, uma proposta promissora para melhorar o desempenho desta classe de algoritmos é o desenvolvimento e implementação desse algoritmo num co-projeto *hardware*/*software* usando um preditor (*Support Vector Regression*-SVR).

5 ARQUITETURAS HARDWARE PARA ALGORITMOS DE RECONSTRUÇÃO MORFOLÓGICA

Neste trabalho, foram implementadas cinco propostas para os dois algoritmos sequenciais (*Sequential Reconstruction* e *Fast Hybrid reconstruction algorithm*). As Cinco abordagens são mencionadas a seguir:

1. **Arquitetura do algoritmo SR em *hardware*:** Implementação do algoritmo SR num co-projeto *hardware/software*. Divide-se em três partes: (a) as operações realizadas pelo processador (ARM ou NIOS II) consistem em escrever e enviar os comandos para o módulo de *hardware*, (b) as operações realizadas pelo *hardware* consistem em receber e processar as imagens usando o algoritmo SR, e (c) o processador requisita as imagens processadas pelo co-processador (*hardware*). Essa implementação gerou os artigos (Anacona-Mosquera et al. 2017) e (Cabral et al. 2020). O artigo (Anacona-Mosquera et al. 2017) foi apresentado em 2017 na conferencia *30th Symposium on Integrated Circuits and Systems Design* (SBCCI) nomeado como "*Efficient Hardware Implementation of Morphological Reconstruction based on Sequential Reconstruction Algorithm*", em Fortaleza (Brasil). Por outro lado, o artigo (Cabral et al. 2020) foi publicado no *Journal of Real-Time Image Processing*, entitulado como "*Optimized execution of morphological reconstruction in large medical images on embedded devices*".

2. **Arquitetura do algoritmo FH em *hardware* e *software*:** Implementação do algoritmo FH usando uma estratégia de coprojeto *hardware/software*. Semelhante à primeira abordagem, o algoritmo FH divide-se em três partes: (a) o processador (ARM) envia os comandos para o co-processador *hardware*, (b) o co-processador *hardware* processa a imagem usando o algoritmo SR para n iterações (fixadas pelo usuário), e (c) o processador (ARM) recebe a imagem do co-processador e executa a fase de propagação (IWPP).

3. **Arquitetura do algoritmo FH em *hardware*:** Implementação do algoritmo FH num co-projeto *hardware/software*. Semelhante às abordagens anteriores, o algoritmo FH dividese em quatro partes: (a) inicialmente, o processador (ARM) envia os comandos para o co-processador *hardware*, (b) o co-processador *hardware* processa a imagem usando o algoritmo SR para várias iterações, (c) o co-processador *hardware* executa a fase de propagação (IWPP), e (d) o processador requisita as imagens processadas pelo co-processador *hardware*. Essa implementação gerou o trabalho nomeado "*Efficient hardware Implementation of the Fast Hybrid Morphological Reconstruction Algorithm*"que foi apresentado em 2018 na conferencia *31th Symposium on Integrated Circuits and Systems Design* (SBCCI), em Bento Gonçalves (Brasil).

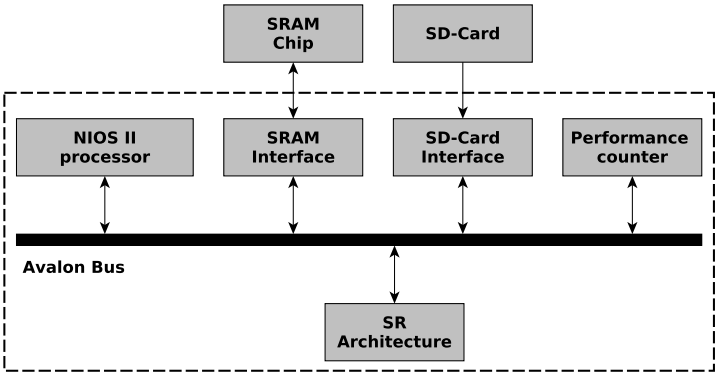
4. **Arquitetura do algoritmo FH em hardware com preditor em *software*:** Implementação do algoritmo FH em *hardware* com um preditor em *software*. Semelhante às abordagens anteriores, o algoritmo FH divide-se em cinco partes: (a) inicialmente, o processador (ARM) envia os comandos para o co-processador *hardware*, (b) o co-processador *hardware* processa a imagem usando o algoritmo SR para várias iterações, (c) um preditor implementado no processador é usado para calcular o valor da iteração ótima, usando uma estratégia proposta por (Santos et al. 2017), (d) o co-processador (*hardware*) executa a fase de propagação (IWPP), e (e) o processador requisita as imagens processadas pelo co-processador *hardware*. Esse trabalho gerou o artigo nomeado como "*Hardware-Based Fast Hybrid Morphological Reconstruction*" publicado no journal IEEE Design Test (Anacona-Mosquera et al. 2020).

5. **Arquitetura do algoritmo FH paralelo em *hardware*:** Implementação do algoritmo FH paralelizado. Nesta implementação, dividiu-se a imagem em sub-imagens para processar cada uma em paralelo e, posteriormente, processou-se as bordas das sub-imagens que são adjacentes. Semelhante à implementação anterior, o algoritmo FH paralelo consiste em: (a) carregamento das imagens através do processador ARM para a memória de cada co-processador (*hardware*) de cada sub-imagem, (b) cada sub-imagem é processada usando o algoritmo FH normal para uma iteração, (c) um módulo específico é usado para analisar as propagações entre as bordas de cada sub-imagem, e (d) depois do processamento, a imagem é enviada para a memória do processador ARM.

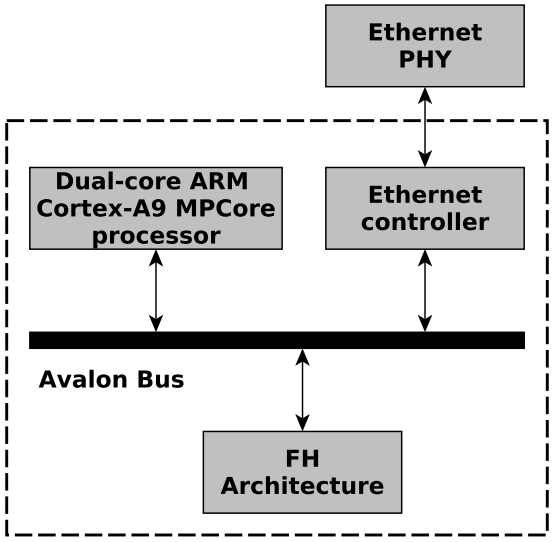
As arquiteturas foram implementadas nas seguintes plataformas: (a) Altera Cyclone-IV E-FPGA, usando o processador do tipo soft-processor NIOS II, rodando a 100 Mhz, (b) Cyclone V SoCKit-FPGA com processador do tipo hard-processor ARM Cortex A9 Dual Core, rodando a 925 Mhz e (c) placa Arria 10-FPGA, ARM Cortex A9 Dual Core, rodando a 925 Mhz. A descrição das abordagens 2, 3, 4 e 5 será feito na seção 5.2, por tratar-se de arquiteturas com implementações similares. Além do trabalho aqui apresentado, a arquitetura SR serviu como base para o trabalho de dissertação de mestrado do aluno Felipe Cabral. Assim, neste capitulo será apresentada em detalhe a versão otimizada da arquitetura para reconstrução morfológica.

As arquiteturas da Figura 5.1 são baseadas no algoritmo FH apresentado em 3, usando como elemento estruturante uma janela de 3×3 pixels (8-connected) para definir a vizinhança. O algoritmo começa usando uma imagem marcadora (J) e uma imagem máscara (I) em escala de cinza de tamanho $m \times n$ pixels. A propagação dos pixels é feito durante várias varreduras *raster* e *anti-raster*. Em cada varredura, os valores de pixels, na parte superior esquerda ou na metade inferior da vizinhança, são propagados para o pixel que está sendo processado durante a varredura *raster* ou *anti-raster*. Adicionalmente, durante a varredura *anti-raster* são calculados os pixels que são candidatos para propagação, e seus endereços são armazenados numa fila normal (vide Algoritmo 3).

A arquitetura da Figura 5.1a representa o coprojeto *hardware*/*software* da arquitetura para o algoritmo SR proposto em (Anacona-Mosquera et al. 2017). Pode ser observado o processador NIOS II, o SRAM (usado como memória interna do processador NIOS II), e o SD-Card (arma-



(a) Particionamento *hardware*/*software* do algoritmo SR na Cyclone-IV(Anacona-Mosquera et al. 2017).



(b) Particionamento *hardware*/*software* das arquiteturas baseadas no algoritmo FH na Cyclone-V e Arria-10.

Figura 5.1: Arquiteturas gerais *hardware*/*software* dos algoritmos SR e FH

zenamento das imagens), todos ligados através do barramento Avalon. Esta plataforma possui limitações que afetam o desempenho do algoritmo, por exemplo: (a) quantidade de elementos lógicos (LEs), (b) tamanho de memória, e (c) funcionamento do processador NIOS II. Por outro lado, as plataformas Cyclone-V e Arria 10 oferecem a possibilidade de implementar arquiteturas com melhor desempenho que a Cyclone-IV devido ao processador ARM (Cyclone-V e Arria 10), que possui um melhor desempenho que o processador NIOS II (Cyclone-IV). Neste contexto, a Figura 5.1b apresenta o coprojeto *hardware/software* do algoritmo usado neste trabalho, implementado na Cyclone-IV, Cyclone-V e Arria 10; onde o processador ARM e o módulo ethernet, usado para carregar as imagens de teste, são ligados através do barramento Avalon.

5.1 ARQUITETURA DO ALGORITMO SR EM HARDWARE

Na implementação usada em (Anacona-Mosquera et al. 2017), a arquitetura do algoritmo SR possui quatro memórias BRAM internas do FPGA: (a) para o marcador (J), (b) para a máscara (I), (c) imagem intermediaria para o módulo *raster*, e (d) imagem intermediaria para para o módulo *anti-raster* (vide Figura 5.2). Neste circuito, usou-se o módulo *raster* para executar a mesma varredura *anti-raster*. Semelhante ao trabalho proposto por (Crookes 1999), a diferença entre as etapas *raster* e *anti-raster* é a ordem de entrada dos pixels da imagem que está sendo processada. Na etapa *raster*, o primeiro pixel a ser processado é o pixel na posição (0,0), e o último pixel é da posição (m$-$ 1,n$-$ 1), onde m e n são as dimensões da imagem. Por outro lado, o primeiro pixel que é processado na etapa *anti-raster* é o pixel da posição (m$-$ 1,n $-$ 1), e o último pixel é da posição (0,0). O controlador dos módulos da arquitetura (*main control module*), que tem a interface Avalon, a escrita/leitura nas memórias (RAM1, RAM2 e I/J RAM), e as varreduras *raster/anti-raster*, está baseado em uma Máquina de Estados Finitos (FSM - *Finite State Machine*). Os seguintes módulos foram implementados para os circuitos das varreduras *raster/anti-raster* (vide Figura 5.3):

1. **Neighborhood marker:** neste módulo são disponibilizados os pixels da imagem marcadora que são usados para processar o pixel de interesse (vide Figura 5.3a). O principio de funcionamento desde módulo está baseado na filtragem por convolução. Este módulo está composto pelos seguintes sub-módulos:

   (a) **J-image RAM:** bloco de memória BRAM para armazenar a imagem marcadora.

   (b) **J-neighborhood loader:** bloco projetado para carregar o elemento estruturante da imagem marcador. Esse circuito está composto por três FIFOs e dois *buffers* (vide Figura 5.3b).

   (c) **Neighborhood padding:** bloco para evitar a distorção na imagem relacionada às bordas da mesma.

   (d) **Stop criteria:** bloco para fazer a contagem dos pixels que estão sendo propagados dentro da imagem. O valor desta contagem gera duas métricas: (a) *thr1*, a diferença

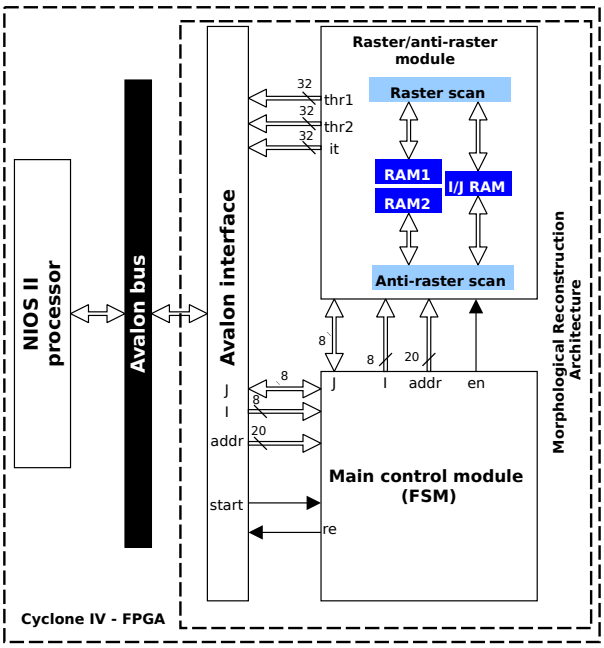


Figura 5.2: Arquitetura do algoritmo SR

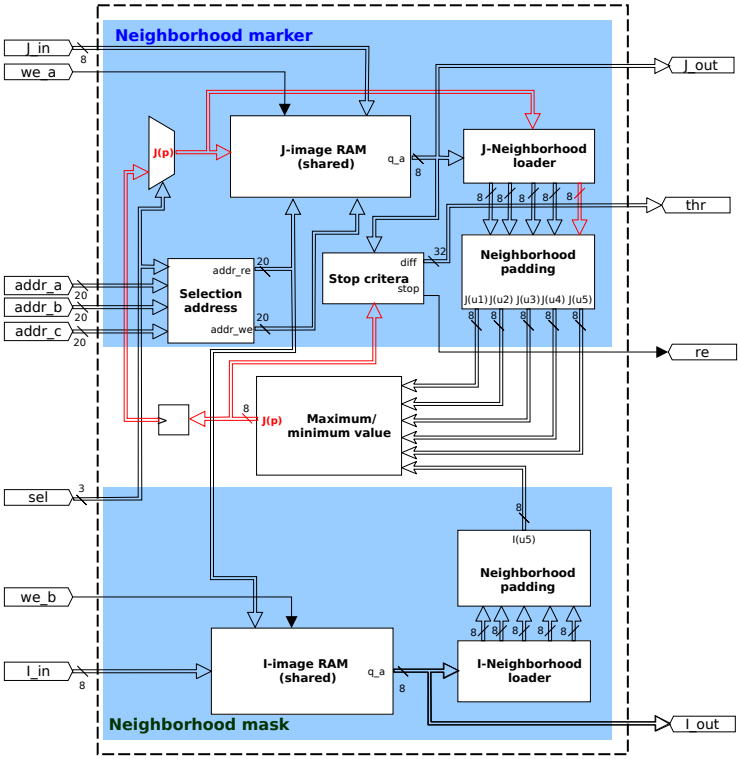
entre a imagem original e o resultado da varredura *raster*, e (b) *thr2*, a diferença entre a imagem resultante da varredura *raster* e o resultado da varredura *anti-raster*. O critério de parada foi a estabilização da imagem que está sendo reconstruída, ou seja, quando os pixels da imagem marcador não mudem de valor (vide Figura 5.3c).

(e) **Selection address:** indica as posições (i, j) dos pixels dentro da imagem para a leitura/escrita nos blocos de memória RAM.

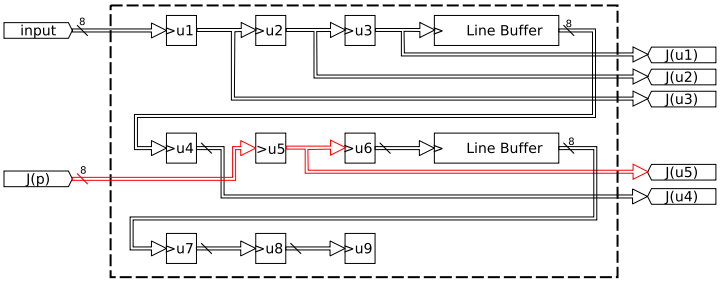
2. **Maximum/minimum value:** calcula o pixel de maior valor dentro da vizinhança e define o menor valor entre o resultado anterior com o pixel da máscara. Esse processo é feito durante um ciclo de relógio, e o resultado da comparação é carregado no pixel marcador.

3. **Neighborhood mask:** neste módulo são disponibilizados os pixels da imagem marcadora que são usados para processar o pixel de interesse. Os módulos (a), (b), e (c) do *neighborhood marker* são adaptados para disponibilizar a vizinhança da imagem máscara.

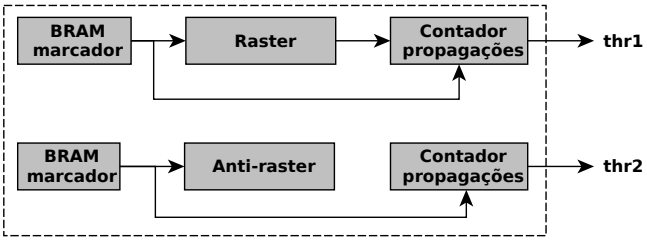
A métrica de qualidade escolhida para avaliar as imagens reconstruídas obtidas da arquitetura foi a comparação com o resultado da implementação do algoritmo em *software* (Matlab ou linguagem-C). Depois do processo finalizar, a imagem marcadora armazenada na memória BRAM é disponibilizada para o processador através do barramento Avalon para o processador NIOS II. Com o objetivo de acrescentar o tamanho das imagens de entrada, melhorar o desempenho dos algoritmo SR e implementar o algoritmo FH, a versão do algoritmo SR apresentada foi otimizada para uma plataforma com maior desempenho e recursos de *hardware*.



(a) Circuito das varreduras *raster*/*anti-raster*.



(b) Circuito da operação de janelamento.



(c) Circuito estabilizador usado como critério de parada do algoritmo.

Figura 5.3: Arquiteturas gerais *hardware*/*software* dos algoritmos SR e FH

5.2 ARQUITETURA DO ALGORITMO FH EM HARDWARE COM IWPP EM HARDWARE

Esta seção apresenta a arquitetura do coprojeto *hardware*/*software* do algoritmo FH (vide Figura 5.4). O processador ARM é usado para armazenar, tanto na memória interno do processador assim como do FPGA, as imagens I e J que são usadas para o processo de reconstrução morfológica. Nesta abordagem, a arquitetura baseia-se no Algoritmo 1, sendo composta por três fases principais: (a) etapas das varreduras *raster*/*anti-raster*, (b) etapa de enfileiramento dos pixels, e (c) etapa de propagação e geração de pixels fontes de propagação (IWPP). A fase IWPP é executada dentro do *hardware*, evitando assim a transferência da fila inicial via barramento Avalon para o processador ARM. A arquitetura base do algoritmo FH para as quatro abordagens é apresentada na Figura 5.4. A diferença entre as quatro abordagens do algoritmo FH, está no circuito de estabilização da imagem e no circuito do processamento da fila, usado na fase IWPP (*Wavefront propagation module*), podendo ser feito tanto em *software* como em *hardware*.

As arquiteturas das Figuras 5.5 e 5.6 mostram as implementações das três fases em *hardware*. O algoritmo FH usa o módulo de varredura *raster*, e três memórias BRAM compartilhadas (vide Figura 5.5) conseguindo-se eliminar as duas memórias intermediárias da arquitetura base (algoritmo SR). Segue-se usando duas memórias BRAM para armazenar as imagens I e J, e se incrementa um bloco de memória para os pixels considerados fontes de propagação para a última fase. O algoritmo FH não específica o tamanho que deve ser alocado para armazenar os pixels fontes, deixando-se um valor fixo para armazenar esses pixels.

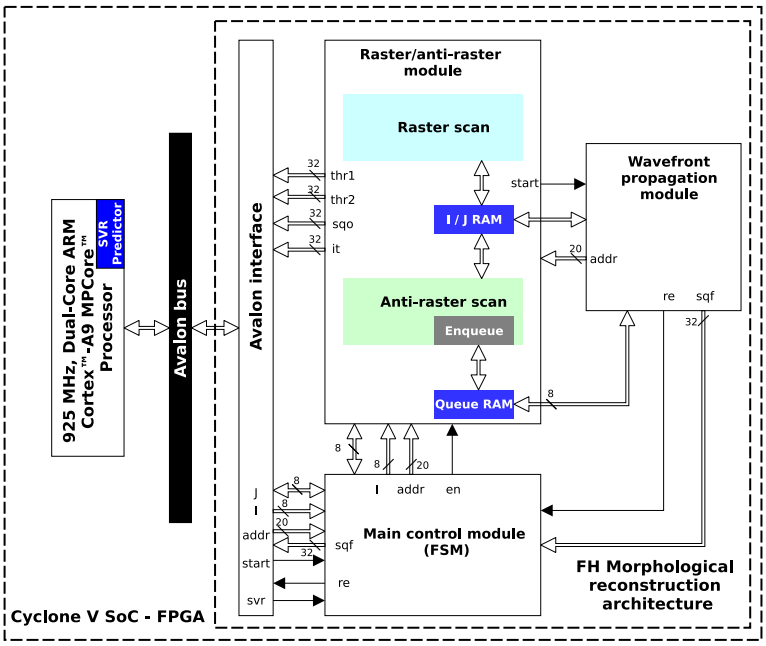


Figura 5.4: Arquitetura geral do algoritmo de reconstrução morfológica FH.

Os principais módulos da arquitetura FH implementada em *hardware* são descritos a seguir.

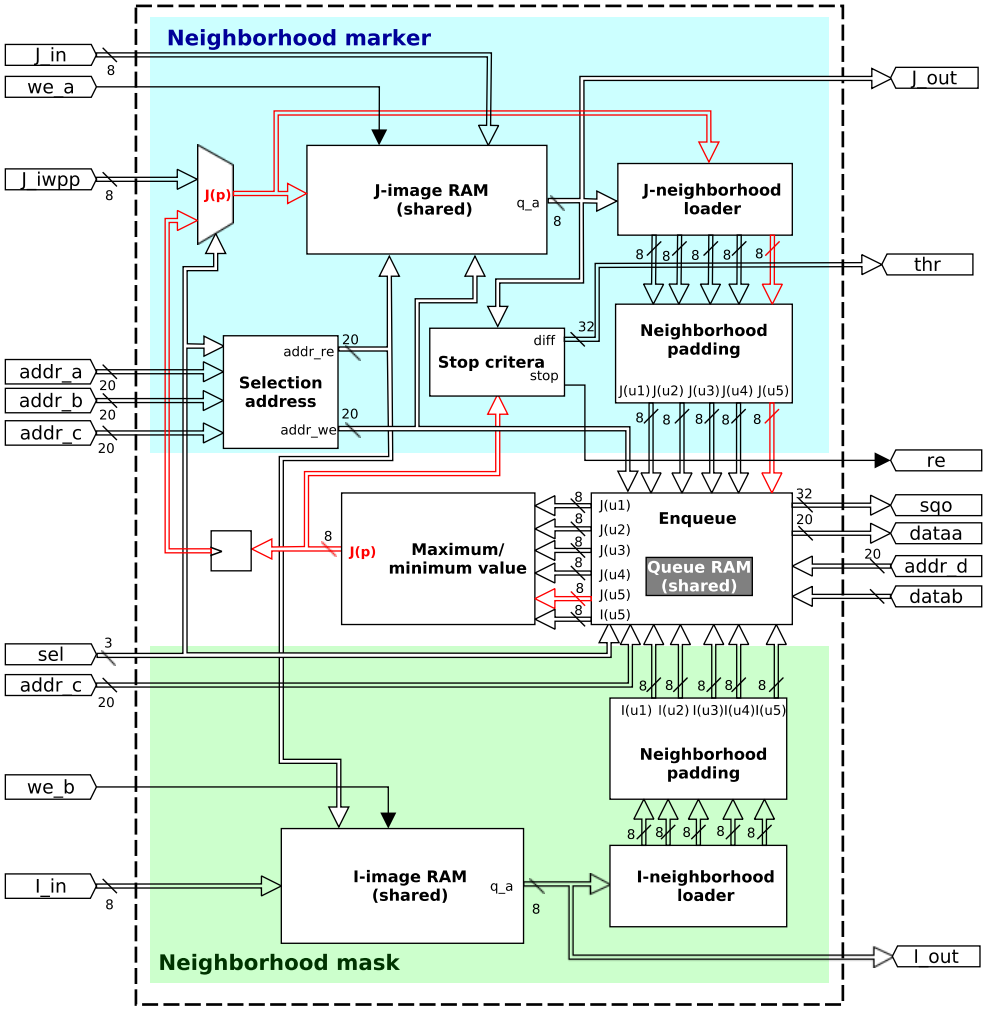
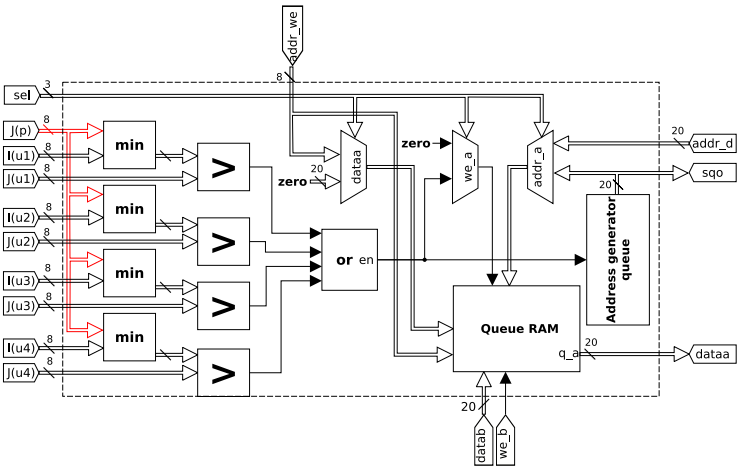
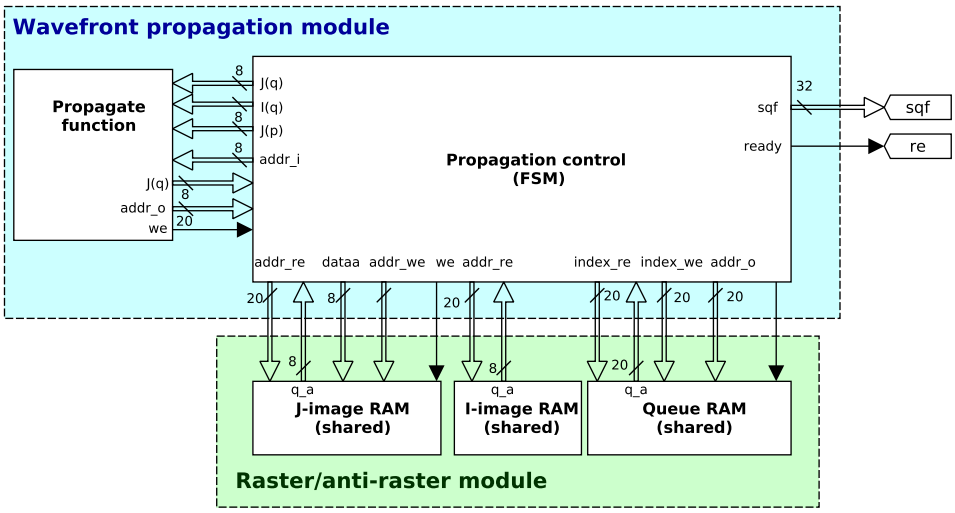


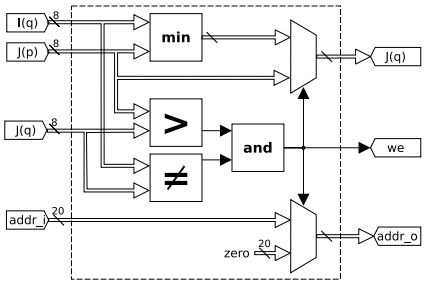
Figura 5.5: Arquitetura geral do algoritmo de reconstrução morfológica FH.



(a) Arquitetura da função de enfileiramento.



(b) Wavefront propagation module.



(c) Propagation function architecture.

Figura 5.6: Arquitetura geral do módulo de propagação

1. **Main control module (FSM):** a estrutura de controle da arquitetura é apresentada como uma FSM. Este módulo sincroniza as etapas das varreduras *raster/anti-raster*, assim como o gerenciamento da escrita das imagens para as memórias BRAM, os pixels que são candidatos para a fase de propagação, e as novas fontes de propagação na fase IWPP.

2. **Raster/anti-raster module:** semelhante ao módulo implementado na arquitetura do algoritmo SR, as imagens são processadas pelos módulos *raster/anti-raster*. Os seguintes módulos são adicionados ao módulo *raster* e executados durante a varredura *anti-raster*:

   (a) **Enqueue:** neste módulo, são calculados os pixels que vão ser usados como novas fontes de propagação na fase IWPP (vide Figura 5.6a). A quantidade de fontes iniciais calculadas por este módulo é usada como métrica para avaliar o desempenho da arquitetura na etapa de verificação comportamental e funcional, sendo atribuída à variável sqi. Neste circuito, a vizinhança do pixel de interesse do marcador e da máscara são usados para calcular os pixels candidatos para propagação na fase IWPP.

   (b) **Queue RAM:** armazena em uma memória BRAM os endereços (i,j) dos pixels que cumprem as condições paras ser novas fontes de propagação.

   (c) **Stop criteria:** o módulo decide se o algoritmo continua executando o SR, ou se passa para a fase de propagação, ou se a imagem reconstruída estabilizou sem precisar entrar na fase IWPP.

3. **Wavefront propagation module:** os pixels candidatos a propagação na fase IWPP são processados por este modulo (vide Figuras 5.6b), assim como são calculadas novas fontes de propagações usando o circuito da Figura 5.6c.

4. **Propagation control (FSM):** este circuito sincroniza o circuito que permite verificar se um pixel pode ser uma nova fonte de propagação, escrita/leitura na imagem marcador e na memória da fila (Queue RAM), e disponibilizar a métrica de contagem de pixels propagados como critério de parada *sqf* .

5. **Propagation function:** este circuito avalia se um pixel pode propagar seu valor para sua vizinhança (vide Figura 5.6c).

5.2.1 Critério de parada e tratamento da fila (IWPP)

No algoritmo FH , devido ao conteúdo de cada imagem, não é possível prever em qual momento deve ser executada a fase IWPP. Assim, usou-se as métricas *thr1*, *thr2*, sqi, e o número de varreduras para estabelecer os critérios de parada do algoritmo (vide Figura 5.4). Nesta abordagem, dois critérios de parada são usados: (a) a estabilização da reconstrução do algoritmo SR, ou seja, o valor da métrica *thr2* deve ser zero, e (b) um limitado número de varreduras para depois passar para a fase IWPP. No segundo critério, a imagem parcialmente reconstruída armazenada na BRAM e os pixels candidatos (armazenados na BRAM) são disponibilizados para completar o processo

de reconstrução da imagem através do módulo *wavefront propagation module*. No entanto, se o critério de parada for o primeiro, a memória BRAM está vazia e a imagem reconstruída é enviada para o processador ARM via barramento Avalon.

5.3 ARQUITETURA DO ALGORITMO FH EM HARDWARE COM IWPP EM SOFTWARE

A Figura 5.7 apresenta a arquitetura do algoritmo FH híbrido: (a) etapa das varreduras *raster/anti-raster*, e enfileiramento implementadas em *hardware*, e (b) fase IWPP implementada em *software*. As etapas do processo de reconstrução morfológica são descritas a seguir:

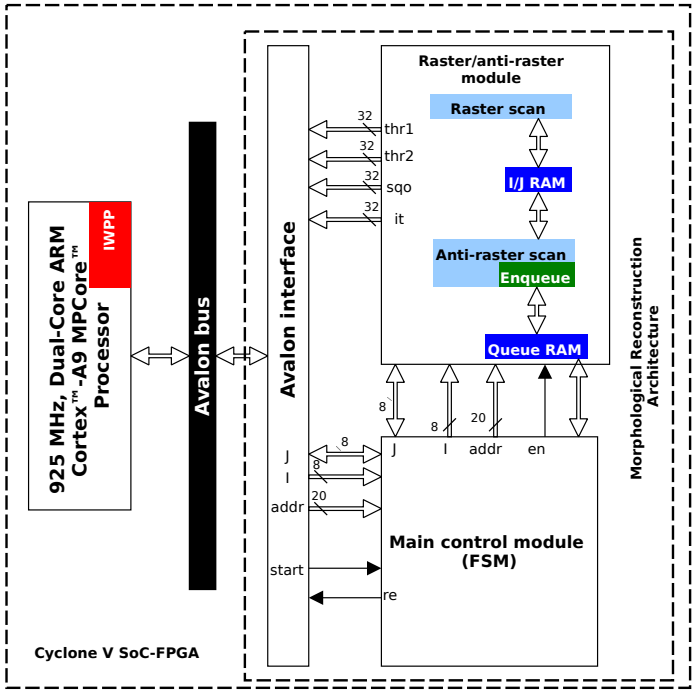


Figura 5.7: Arquitetura geral do algoritmo de reconstrução morfológica FH.

1. **Varreduras raster/anti-raster:** são implementadas os circuitos usados pelo algoritmo FH sem usar o módulo wavefront propagation, ou seja, os circuitos principais: *raster/anti-raster* e Main control module.
2. **Enfileiramento:** o módulo de enfileiramento é o mesmo módulo usado na implementação mostrada na seção anterior.
3. **Upload da imagem marcador e fila:** via barramento Avalon, a imagem marcadora e os pixels armazenados na BRAM são enviados para o processador ARM.

Uma vez carregada a imagem marcadora e a fila no processador ARM, a fase IWPP implementada em linguagem-C é executada pelo processador. Nesta etapa, o critério de parada é o mesmo

usado na arquitetura do algoritmo implementado totalmente em *hardware*, ou seja, quando não sejam geradas novas fontes de propagação.

5.4 ARQUITETURA DO ALGORITMO FH EM HARDWARE COM PREDITOR EM SOFTWARE

Semelhante à implementação anterior, o algoritmo FH é mapeado em *hardware*, ou seja, as varreduras *raster/anti-raster* e o tratamento da fila (IWPP) são ambos implementados uma arquitetura ad-hoc de *hardware*. Devido a que as imagens possuem diferentes conteúdos, o número de varreduras fixas podem ser diferente para cada imagem e portanto, o processo de reconstrução morfológica apresenta desempenho baixo. Com o intuito de evitar usar um número de iterações fixas, e assim melhorar o desempenho do processo de reconstrução morfológica, desenvolveu-se uma ferramenta para arbitrar o critério de parada, visando uma otimização no tempo de processamento. A arquitetura da Figura 5.8 apresenta a arquitetura do algoritmo FH com o preditor em *software*. Toma-se como base a implementação do algoritmo FH da seção 1.2 (vide Figura 5.4). Depois de cada iteração (*raster* e *anti-raster*), as métricas *thr1*, *thr2*, sqi, e *it* são enviadas para o processador ARM. Usando essas métricas, o algoritmo segue os seguintes passos:

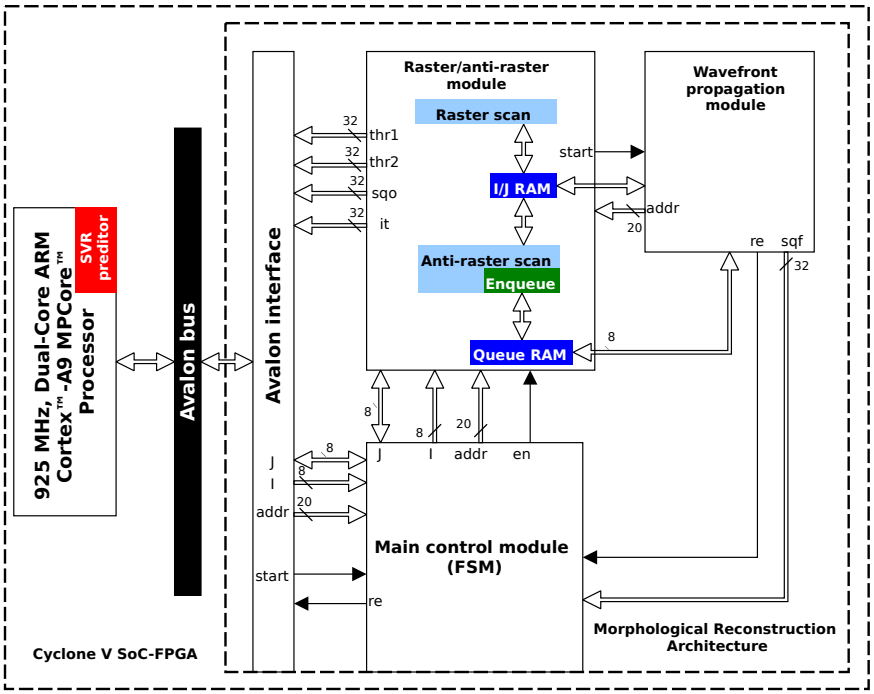


Figura 5.8: Arquitetura geral do algoritmo de reconstrução morfológica FH.

1. **Varreduras raster/anti-raster:** são gerados valores para as métricas thr1, thr2, sqi, e *it*.
2. **Interface Avalon:** o controlador envia os valores das métricas para o processador ARM e espera a avaliação do preditor. As condições que são avaliadas para entrar na fase IWPP são:

(a) **Condição 1:** se o valor da iteração calculada pelo preditor é maior que zero e *thr2* é maior que zero. Neste caso, as varreduras *raster/anti-raster* devem continuar.

(b) **Condição 2:** se o valor da iteração de saída é zero e *thr2* é maior que zero. Neste caso, as varreduras *raster/anti-raster* terminam e entra-se na fase IWPP.

(c) **Condição 3:** se *thr2* é igual a zero. Se esta condição for verdadeira o processo de reconstrução termina.

3. **Calculo do preditor:** o preditor implementado no processador ARM usa os valores das métricas para estimar se o algoritmo deve passar da fase de varreduras para a fase IWPP. A equação 1.1 foi implementada no processador ARM para calcular o número da iteração de saída para passar à fase de propagação.

$$f(\mathbf{x}) = \sum_{i=1}^{\#SV} (\alpha_i - \alpha_i^*) \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}\|^2}{\gamma}\right) - b, \tag{5.1}$$

onde $\mathbf{x}$ são os vetores de entrada e $\mathbf{x}_i$ são as constantes de treinamento, α, α^* e γ são parâmetros gerados no treinamento, $\#SV$ é o número de vetores de suporte, e b é o *bias*.

5.5 ARQUITETURA DO ALGORITMO FH PARALELIZADO EM HARDWARE

Como descrito na seção 4.6.1, o desempenho do algoritmo FH aumenta dependendo do nível de paralelização do algoritmo, isto é, depende do número de partições das imagens originais. Baseado nesta abordagem, as imagens originais foram particionadas em várias sub-imagens, sendo cada sub-imagem processada por uma arquitetura implementada totalmente em *hardware*. Cada sub-imagem é processada em paralelo usando como base a arquitetura descrita na seção 5.2. Adicionalmente, um controlador mestre (FSM) administra a leitura/escrita de cada co-processador, assim como o processamento das bordas verticais/horizontais (vide Figura 5.9).

Finalizada a etapa de armazenamento para cada sub-imagem, cada processador executa o algoritmo de reconstrução da imagem até a estabilização de cada sub-imagem. O número de iterações antes de entrar para a fase IWPP foi fixado em uma iteração. Com isto, o módulo do preditor perde funcionalidade e é descartado da arquitetura. As imagens (marcadora e máscara) foram particionadas em quatro sub-imagens e carregadas em forma de streamming para cada co-processador FH *hardware* da arquitetura. Os passos do processamento do algoritmo paralelo são:

Passo 1: as imagens originais (marcador e máscara) são particionadas em tamanhos de 1024×1024 pixels no ARM, e armazenadas dentro das memórias BRAM de cada co-processador (FH1, FH2, FH3, e FH4).

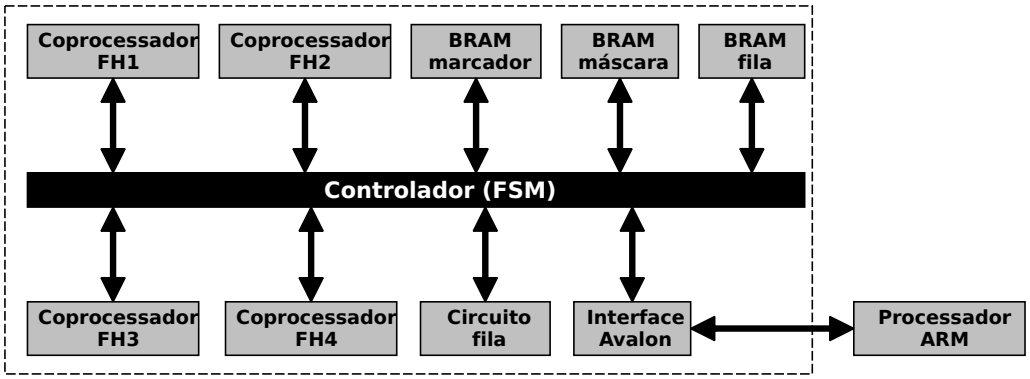


Figura 5.9: Arquitetura do algoritmo FH paralelizado

Passo 2: depois que as imagens são armazenadas nas memórias de cada co-processador, cada co-processador inicia o processo de reconstrução morfológica da sua sub-imagem correspondente.

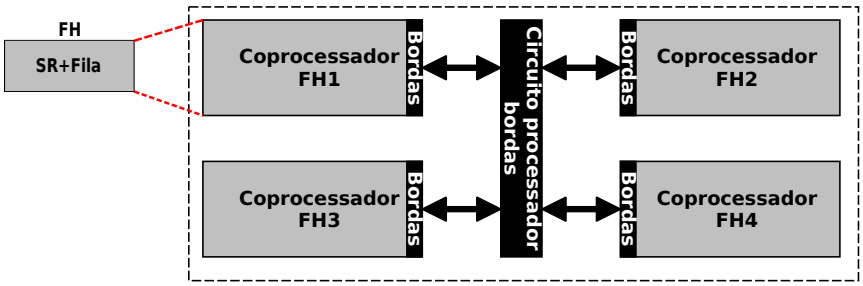
Passo 3: os co-processadores FH1 e FH2 são sincronizados para o processamento das bordas de ambas sub-imagens (vide Figura 5.10a). Neste passo, cada co-processador espera a finalização do processo de reconstrução do co-processador adjacente. De forma semelhante, o mesmo procedimento é feito para os processadores FH3 e FH4, onde o processo de reconstrução acontece de forma independente, ou paralela, dos processadores FH1 e FH2.

Passo 4: depois que cada co-processador, FH1 e FH2, terminaram de executar o algoritmo de reconstrução, as bordas das duas sub-imagens são avaliadas para calcular possíveis propagações entre as duas imagens (bordas verticais). Esse processo de propagação é feito usando o circuito de enfileiramento (*Enqueue*) e o módulo *Wavefront propagation module*. Igual que no passo anterior, faz-se uma avaliação de propagação entre as fronteiras das sub-imagens de FH3 e FH4.

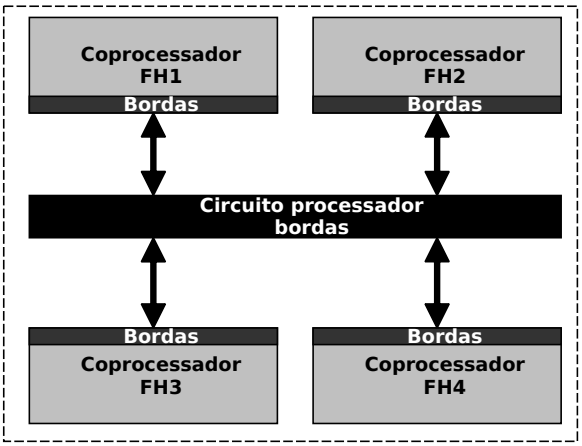
Passo 5: as imagens resultantes do processo de reconstrução dos co-processadores FH1 e FH2 formam uma sub-imagem, chamada de FH1-FH2. De modo igual, é criada uma sub-imagem da reconstrução dos co-processadores FH3 e FH4, denominada de FH3-FH4.

Passo 6: as bordas das sub-imagens FH1-FH2 e FH3-FH4 são analisadas para calcular as propagações entre ambas sub-imagens usando os mesmos módulos do passo 4 (vide Figura 5.10b).

Passo 7: se o processo de propagação das sub-imagens FH1-FH2 e FH3-FH4 é finalizado, a imagem marcadora reconstruída é armazenada na BRAM e é transferida via barramento Avalon para o processador ARM.



(a) Bordas verticais.



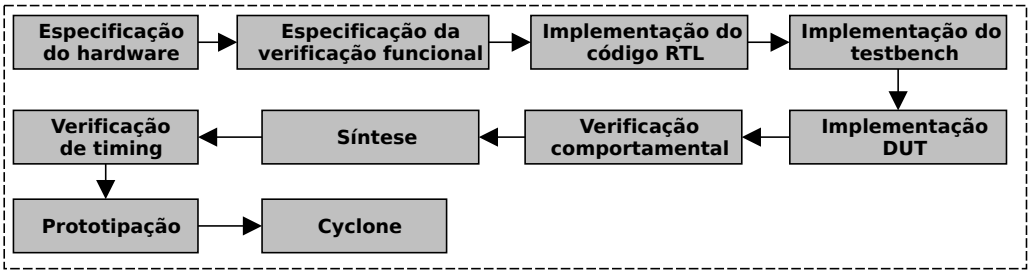
(b) Bordas horizontais.

Figura 5.10: Processamento das bordas para uma imagem dividida em co-processadores

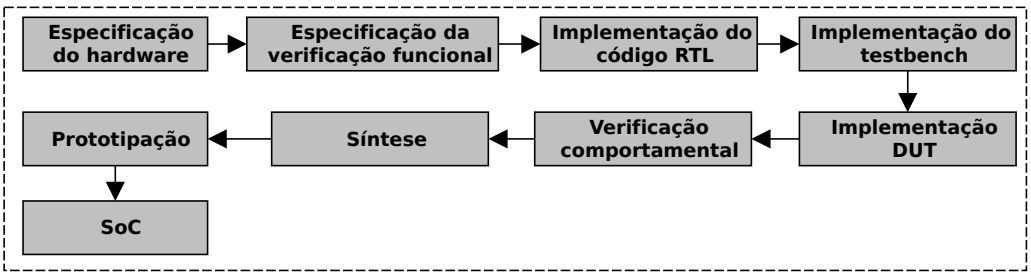
5.6 VERIFICAÇÃO FUNCIONAL, TREINAMENTO, E VALIDAÇÃO DA ARQUITETURA

Neste processo, demostrou-se a funcionalidade do coprojeto *hardware*/*software* seguindo o fluxo de projeto mostrado na Figura 5.11 sintetizados nas placas Cyclone-IV (5.11a) e SoC (Cyclone-V e Arria-10) (5.11b). Para a verificação funcional, criou-se um arquivo testbench como ambiente de simulação usando a ferramenta de simulação QuestaSim para simular o código em nível RTL (Register Transfer Level) ou comportamental, e em nível de portas (Gate level) ou de timing. Os resultados das imagens reconstruídas após as verificações da arquitetura em *hardware* são comparados com os resultados da solução implementada em *software* usando Matlab com a função imreconstruct.

Para a fase de testes foram usadas oito imagens de 4096×4096 pixels (quatro imagens como marcador e quatro imagens como máscara e duas imagens de 8192×8192 pixels (uma imagem como marcador e uma imagem como máscara). As imagens foram particionadas em sub-imagens de 512×512 (Cyclone-V), 1024×1024 pixels (Arria-10), e 288×288 pixels (Cyclone-IV). O processo verificação para cada implementação é apresentado a seguir:



(a) Cyclone-IV.



(b) Cyclone-V e Arria-10.

Figura 5.11: Fluxo de projeto do algoritmo de reconstrução morfológica.

1. **Requerimentos de desenho e desenho RTL:** Os algoritmos de reconstrução morfológica: SR, FH e FH-4 foram mapeados usando linguagem de descrição de *hardware* VHDL.

2. **Síntese e verificação:** As arquiteturas foram simuladas em nível de portas usando um testbench (vide Figura 5.12). O testbench possui cinco elementos: (1) imagens de teste, (2)

reconstrução da imagem em *software*,(3) avaliação dos resultados, e (4) a arquitetura que está sendo verificada ou DUT (Design Under Test). Dentro do DUT, a comunicação do barramento Avalon é simulada usando uma FSM. A Figura 5.11 apresenta o fluxo de projeto do algoritmo SR para a placa Cyclone-IV e SoC. O processo de verificação de timing depende de arquivos com a informação dos componentes presentes dentro da placa. Neste caso, o fabricante somente disponibiliza as informações físicas da placa Cyclone-IV permitindo executar ambas verificações. Para o caso da placa SoC, a verificação de timing converte-se na prototipagem e comparação das métricas com o algoritmo implementado em *software*.

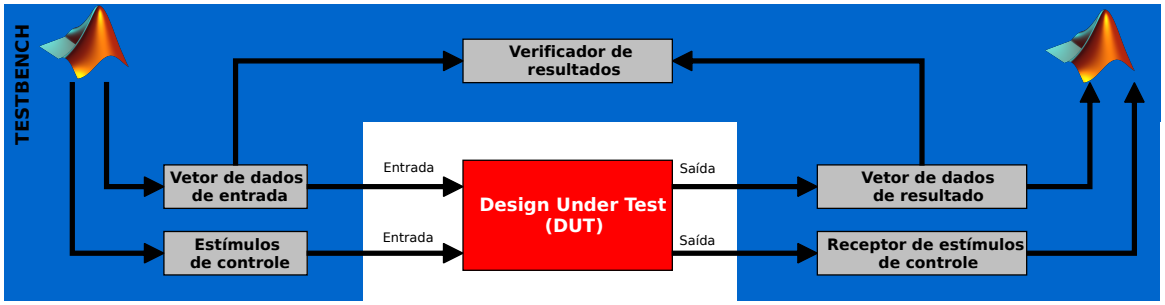


Figura 5.12: Estrutura do testbench usado para o processo de verificação

3. **Prototipação e SoC:** teste da arquitetura usando imagens reais.

Igualmente, o processo de treinamento da máquina SVR é descrito a seguir:

1. **Coleta de dados:** os resultados das métricas *thr1*,*thr2*, *it*, e tempo de processamento produto da reconstrução morfológica das imagens de 512×512 pixels foram armazenados para criar um banco de testes para o treinamento da máquina SVR. Neste processo, executou-se o algoritmo FH aumentando o número da iterações antes de entrar na fase IWPP, ou antes da estabilização da imagem. Por exemplo, uma varredura *raster/anti-raster* e depois o processamento da fila; duas varreduras *raster/anti-raster* e depois o processamento da fila; e assim por diante, até o algoritmo finalizar. Com isto, criou-se um novo banco de testes onde são armazenadas as métricas *thr1*, *thr2*, *sqi*, *sqf* , *it* e o tempo de processamento para cada iteração (varredura *raster/anti-raster* e IWPP) (vide Figura 5.13). Como descrito anteriormente, o número de iterações do algoritmo FH depende do conteúdo das imagens que estão sendo processadas, originando que os valores das métricas (*thr1*, *thr2*, *sqi*, *sqf* , e *it*) sejam diferentes para cada imagem.

2. **Treinamento da máquina SVR:** na Figura 5.13, apresenta-se o processo de treinamento da máquina de vetores de suporte (do tipo *SVR- Support Vector Regression*). O processo de treinamento usa os valores de tempo de processamento, e das métricas *thr1*, *thr2*, *sqi*, e *sqf* para cada iteração em cada imagem de teste. Com tudo isto, conhece-se o número da iteração onde o algoritmo FH possui o menor tempo de processamento e pode-se fornecer esses valores para o treinamento. O módulo da SVR da Figura 5.13 compõe-se de três sub-módulos:

(a) **Training data:** neste sub-módulo são usados 20.000 imagens de treinamento. As imagens de teste foram escolhidas aleatoriamente baseadas no cálculo de uma combinação simples entre o grupo de imagens marcador e máscara. Durante o processo de treinamento, o 70% das imagens de teste são usadas para trinar a máquina e 30% é usado para a validação da máquina treinada.

(b) **Otimizer:** é usado o MOPSO para o treinamento da SVR.

(c) **Reports:** o resultado do treinamento da SVR gera uma solução (função matemática) que pode ser implementada em *software* ou em *hardware*.

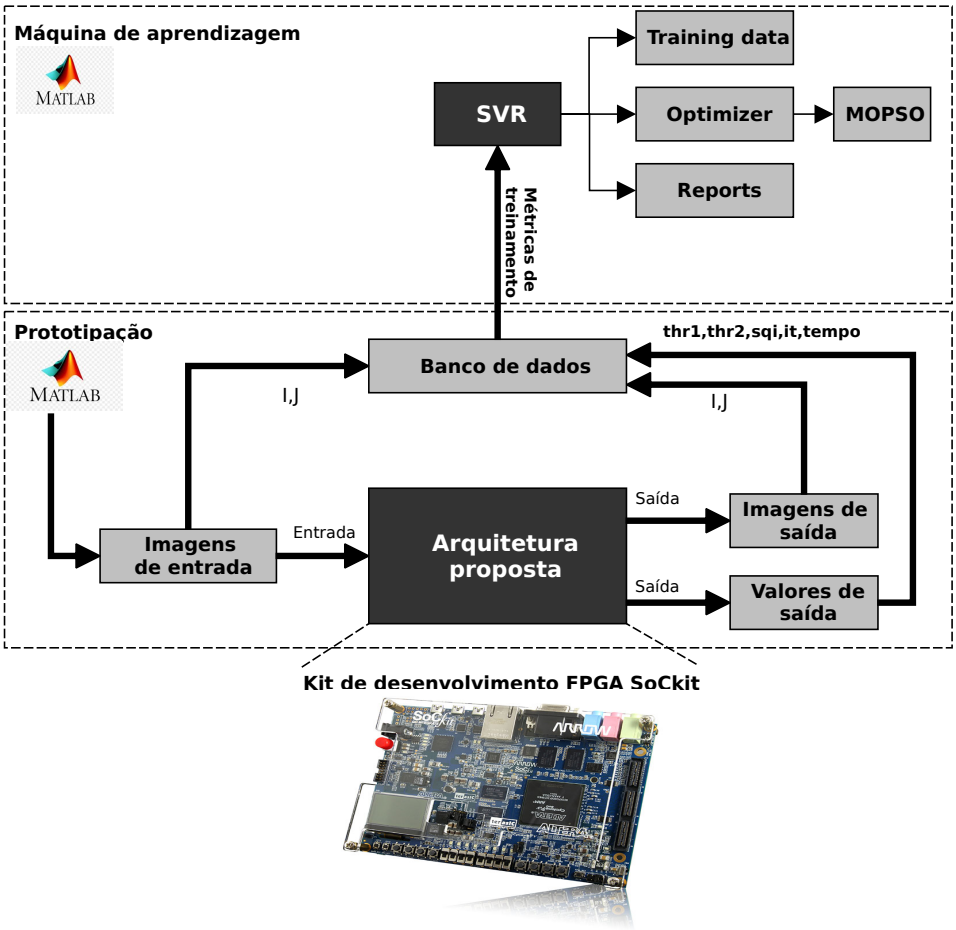


Figura 5.13: Diagrama do treinamento da máquina SVR.

6 RESULTADOS DE DESEMPENHO

As arquiteturas propostas foram implementadas nas placas da Terasic DE2-115, SoCKit, e Arria-10. Os resultados da síntese são apresentados nas Tabelas 6.1 e 6.2 para as arquiteturas *hardware* descritas nas Figuras 5.2 e 5.4. Os resultados sobre o consumo de recursos *hardware* para o FPGA Cyclone-IV (EPC4C115F29C7), a Cyclone-V (5CSXFC6DF31C6N) SoCKit-FPGA, e a Altera Arria-10 (10AS066N3F40E2SG) são apresentados nas Tabelas 6.1, 6.2a, e 6.2b, respetivamente. As Tabelas detalham o consumo de elementos lógicos (LEs) ou *Adaptive Logic Modules* (ALMs), consumo de memória (BRAM), máxima frequência para a arquitetura geral, e consumo estimado de potência.

Tabela 6.1: Consumo de recursos de *hardware* para a Altera Cyclone-IV EPC4C115F29C7

Parâmetro	**Arq SR**
Tamanho (pixels)	288×288
Elementos lógicos	15.941 (14%)
FPGA BRAM	2.752.256 (69%)
Frequência máxima (MHz)	63,83

Parâmetro	**Arq SR**	**Arq FH**
Tamanho (pixels)	512×512	512×512
Elementos lógicos (em ALMs)	4.057 (10%)	4.077 (10%)
FPGA BRAM	4.204.464 (74%)	4.474.482 (79%)
Consumo de potência (mW)	731,30	751,36
Frequência máxima (MHz)	64,94	64,52

(a) Altera Cyclone-V 5CSXFC6DF31C6N

Parâmetro	**Arq FH**	**Arq FH-4**
Tamanho (pixels)	1024×1024	1024×1024
Elementos lógicos (em ALMs)	29.281 (12%)	29.319 (12%)
FPGA BRAM	26.188.914 (60%)	27.648.914 (63%)
Consumo de potência (mW)	7.332,02	7.374,62
Frequência máxima (MHz)	120,53	115,74

(b) Altera Arria-10 10AS066N3F40E2SG

Tabela 6.2: Consumo de recursos de *hardware* para as plataformas SoC.

Como se pode observar os resultados de síntese das arquiteturas, apresentados nas tabelas 6.1, 6.2a e 6.2b, mostram frequências abaixo de 65 MHz para as arquiteturas implementadas nas placas Cyclone-IV e SoCkit, e frequências abaixo de 121 MHz para as arquiteturas implementadas na Arria-10. Estes resultados obtidos de frequência dependem do caminho crítico (*critical path*) da arquitetura. Como visto na seção, o pixel que está sendo reconstruído deve ser calculado e realimentado, todo isto durante um ciclo de relógio.

No caso da plataforma SoCkit, os pixels são armazenados em duas imagens (I e J), e as posições dos pixels que são candidatos a novas fontes de propagação são guardados dentro de uma fila. Neste sentido para armazenar as informações, a arquitetura está composta por três blocos de

memória RAM, ou BRAMs. Dois blocos BRAM de tamanho 512×512 para carregar as imagens *I* e *J*, e um bloco BRAM de tamanho predefinido (não calculado). Nesta plataforma, 21% da memória BRAM é reservado para recursos internos do ARM, deixando disponível o restante da memória para a arquitetura. A memória disponível divide-se em 74% para armazenar as imagens *I* e *J*, e 5% para os endereços dos pixels que vão para a fila (evitando o overflow). A frequência de operação implementada na SoCkit foi de 50 MHz.

Por outro lado, para a placa Arria-10, a memória utilizada foi de 13% para armazenar as imagens *I* e *J*, e os endereços dos pixels que vão para a fila. O tamanho máximo de imagens usadas foi 1024×1024 pixels (38%) e o restante da fila foi alocado para a fila (42%). Igualmente que no caso da SoCkit, a plataforma Arria-10 também reserva o 21% da memória. Para a Arria-10, a frequência de operação implementada foi de 100 MHz.

6.1 ESTIMAÇÃO DA TRANSIÇÃO ENTRE A ETAPA RASTER E ANTI-RASTER E PROPAGAÇÃO DA FILA NO ALGORITMO SR

Com o intuito de implementar arquiteturas para reconstrução morfológica de imagens, foram desenvolvidas diferentes arquiteturas. Uma dessas arquiteturas está baseado no algoritmo SR que possui duas etapas: raster e anti-raster. O análise de desempenho desta arquitetura (CRA), com o intuito de calcular o tempo de processamento do algoritmo implementado, em número de ciclos de relógio para uma iteração *raster* e *anti-raster*, pode ser calculado usando a seguinte equação:

$$CRA = 2(m \cdot n) + \ 2l, \tag{6.1}$$

onde l é a latência, m é altura e n largura da imagem. O valor de latência para as varreduras foi de 514 ciclos para a SoCkit, e de 1026 ciclos para a Arria-10. Esse valor de latência depende do tamanho de largura da imagem e do tamanho do elemento estruturante.

6.2 ESTIMAÇÃO DA TRANSIÇÃO ENTRE A ETAPA RASTER, ANTI-RASTER E PROPAGAÇÃO DA FILA NO ALGORITMO FH

No caso do algoritmo FH, este compõe-se de três etapas: raster, anti-raster e propagação da fila. As etapas raster e anti-raster possuim o mesmo número de ciclos de relógio do algoritmo SR. De forma geral, o número de ciclos do relógio do algoritmo FH (CFH) pode ser calculado usando a seguinte equação:

$$CFH = \Big(\sum_{i=1}^{max} i \cdot CRA \Big) + CQ, \tag{6.2}$$

onde i é o número da varredura, max é o limite de varreduras raster/anti-raster, CRA é o número de ciclos da fase *raster/anti-raster*, e CQ é o número de ciclos da fase de propagação que vem dada por $15p$, com p sendo o número de propagações totais. No caso da fase de propagação, o número de ciclos não pode ser determinado devido a sua dependência com o conteúdo de cada imagem.

6.2.1 Desempenho dos algoritmos de reconstrução morfológica: SR e FH híbrida

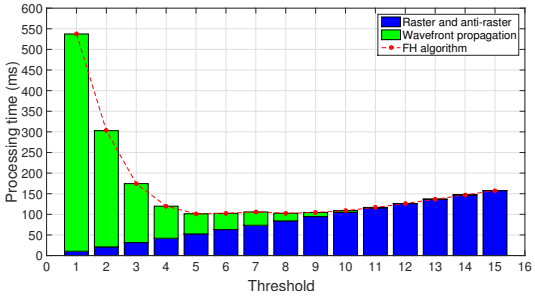
No caso da arquitetura híbrida (raster em *hardware* e IWPP em *software*), apresenta-se um tempo de retardo entre a arquitetura em *hardware* e o processador ARM. Este tempo de retardo é relacionada à transferência de informação (imagens e endereços) para ser armazenado na memória do processador ARM. No caso da SoCkit, o tempo de transferência da imagem parcialmente reconstruída através do barramento Avalon foi de 200 ms. A Figura 6.1a apresenta o tempo de processamento do algoritmo híbrido, ou seja, varreduras *raster/anti-raster* em *hardware* e processamento da fila em *software* (ARM). Pode-se observar que apresenta o mesmo comportamento da Figura 6.2a, a única diferença está no tempo de transferência da imagem parcial e das informações da fila.

A Figura 6.1b mostra o tempo de processamento para diferentes implementações do algoritmo FH: (a) *software* (barra vermelha), (b) *hardware* (barra azul), e (c) hibrido (barra verde). Nesta implementação, a arquitetura em *hardware* apresentou o menor tempo de processamento Comparando o desempenho da arquitetura em *hardware* com as outras implementações: (a) *software* de 35,50x a 48,57x, e (b) hibrida de 2,51x a 3,81x.

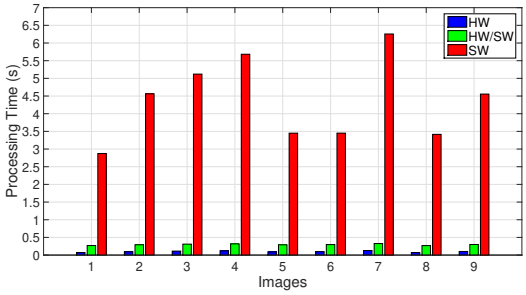
6.2.2 Desempenho dos algoritmos de reconstrução morfológica: ST, SR e FH em *hardware*

A Figura 6.2a mostra o tempo de processamento para a reconstrução de uma imagem. Nesta figura, cada barra representa uma possível solução usando o algoritmo FH. As barras são subdivididas em duas cores: azul, que representa as etapas das varreduras *raster/anti-raster*, e verde, que representa a etapa da fase de propagação. Por outro lado, se o algoritmo para a reconstrução fosse o SR, o número de iterações até a estabilização da imagem seria de dez iterações (somente a barra que possui a cor inteiramente azul). Nota-se que a partir da iteração cinco poucos pixels estão sendo modificados e o tamanho da fila é pequeno. Neste caso, pode ser determinado que a iteração cinco é um ponto critico para passar à fase de propagação. Isto é explicado tendo em conta que o tempo de processamento da fila é menor que o algoritmo SR leva para estabilizar a imagem.

A Figura 6.2b mostra os tempos para várias imagens usando o algoritmo ST (barra azul), SR (barra verde) e FH (barra vermelha). Nesta Figura, o eixo das abscisas mostra o número do conjunto de imagens de teste (imagens aleatórias), e o eixo das ordenadas o tempo de processamento para cada conjunto. Nesta figura, pode-se observar as diferenças de tempos de processamento para cada conjunto de imagens, isto é devido a diferença de conteúdo entre as imagens. O algoritmo FH

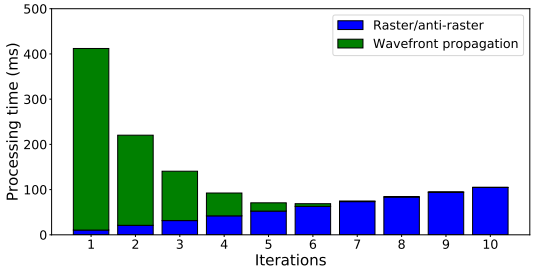


(a) Tempo de processamento para uma imagem com a solução em *hardware*.

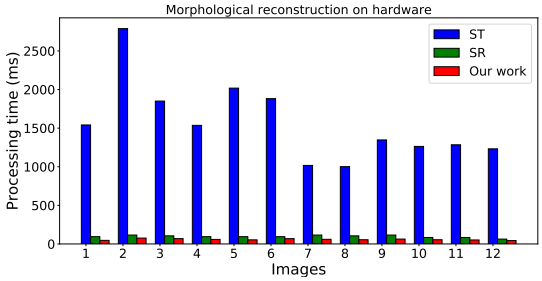


(b) Comparação de tempos de processamento do algoritmo FH em *hardware*, *software* e o co-projeto *hardware/software*.

Figura 6.1: Resultados do algoritmo FH na plataforma SoCkit.



(a) Tempo de processamento para uma imagem com a solução em *hardware*.



(b) Comparação do desempenho entre as soluções dos algoritmos ST, SR, e FH.

Figura 6.2: Resultados do algoritmo FH na plataforma SoCkit.

devido a fase de propagação apresenta um melhor desempenho comparado com os algoritmos SR (1.37x a 2.02x), e ST (21.34x a 38.03x) (vide Figura 6.2b).

6.2.3 Desempenho dos algoritmos de reconstrução morfológica: ST, SR e FH com preditor e sem preditor

A Tabela 6.3 mostra o desempenho para diferentes algoritmos de reconstrução morfológica: ST, SR, e FH com preditor em *software*. Como visto no algoritmo 3, uma fase de varredura (*raster* e *anti-raster*) é executada antes de entrar na fase de propagação, o preditor SVR calcula o número da iteração para que a arquitetura passe da fase da varredura para a fase de propagação. As métricas que o preditor SVR usa para calcular o número da iteração são: iteração atual, treshold 1, treshold 2 e o número de sementes iniciais. Para o processo de reconstrução morfológica, as imagens originais (4096×4096 pixels ou maior tamanho) foram particionadas em sub-imagens de 512×512 pixels que são processadas de forma independente. Imagens com diferentes porcentagens de tecido são usadas, porque imagens com conteúdo menor de tecido tende a necessitar um menor número de propagações na reconstrução morfológica e isto pode afetar o desempenho e tempos de execução das arquiteturas.

Tabela 6.3: Desempenho da arquitetura de reconstrução morfológica para diferentes implementações: ST, SR; e FH com o preditor SVR.

Image size(% of tissue)	**Number of 512×512 blocks**	**Processing time of all blocks (s)**								
						FH-fixed iteration				
		ST	**SR**	**FH-SVR**	**FH-Oracle**	**It3**	**It4**	**It5**	**It6**	**It7**
4096×4096 (25%)	64	27,226	2,391	1,376	1,257	2,353	2,087	2,025	2,017	2,043
4096×4096 (50%)	64	46,835	3,974	2,314	2,028	3,082	2,851	2,881	3,024	3,208
4096×4096 (75%)	64	100,899	6,837	4,341	3,865	6,433	4,826	4,369	4,486	4,756
4096×4096 (100%)	64	110,908	6,239	4,250	3,697	5,794	4,392	4,115	4,294	4,688
8192×5132 (50%)	176	189,844	12,247	8,089	7,191	13,775	11,362	10,238	9,945	9,994

Na Tabela 6.3, para fins de comparação, temos as seguintes versões de algoritmos de reconstrução: ST, SR, FH-SVR que usa o SVR para migrar de fase de forma experimental, FH-Oracle onde foi medido o tempo mínimo de cada processo de reconstrução, ou seja, calcula-se a iteração ótima com o melhor desempenho, e FH-Fixed no qual a iteração para migrar de fase é de valor fixo.

A Tabela 6.3 apresenta os tempos de execução em segundos (s) para o conjunto de sub-imagens para cada implementação. Primeiramente, compara-se a arquitetura proposta (FH-SVR) com o ST, observa-se um ganho superior a 26x. Diferenças de desempenho semelhantes que foi relatada no revisão bibliográfica ao comparar as abordagens em *software* (Vincent 1992, Teodoro et al. 2013), sendo atribuída à baixa eficiência na propagação por dilatação que requer um grande número de varreduras sobre a imagem.

6.2.4 Desempenho dos algoritmos de reconstrução morfológica FH em *hardware* com partições

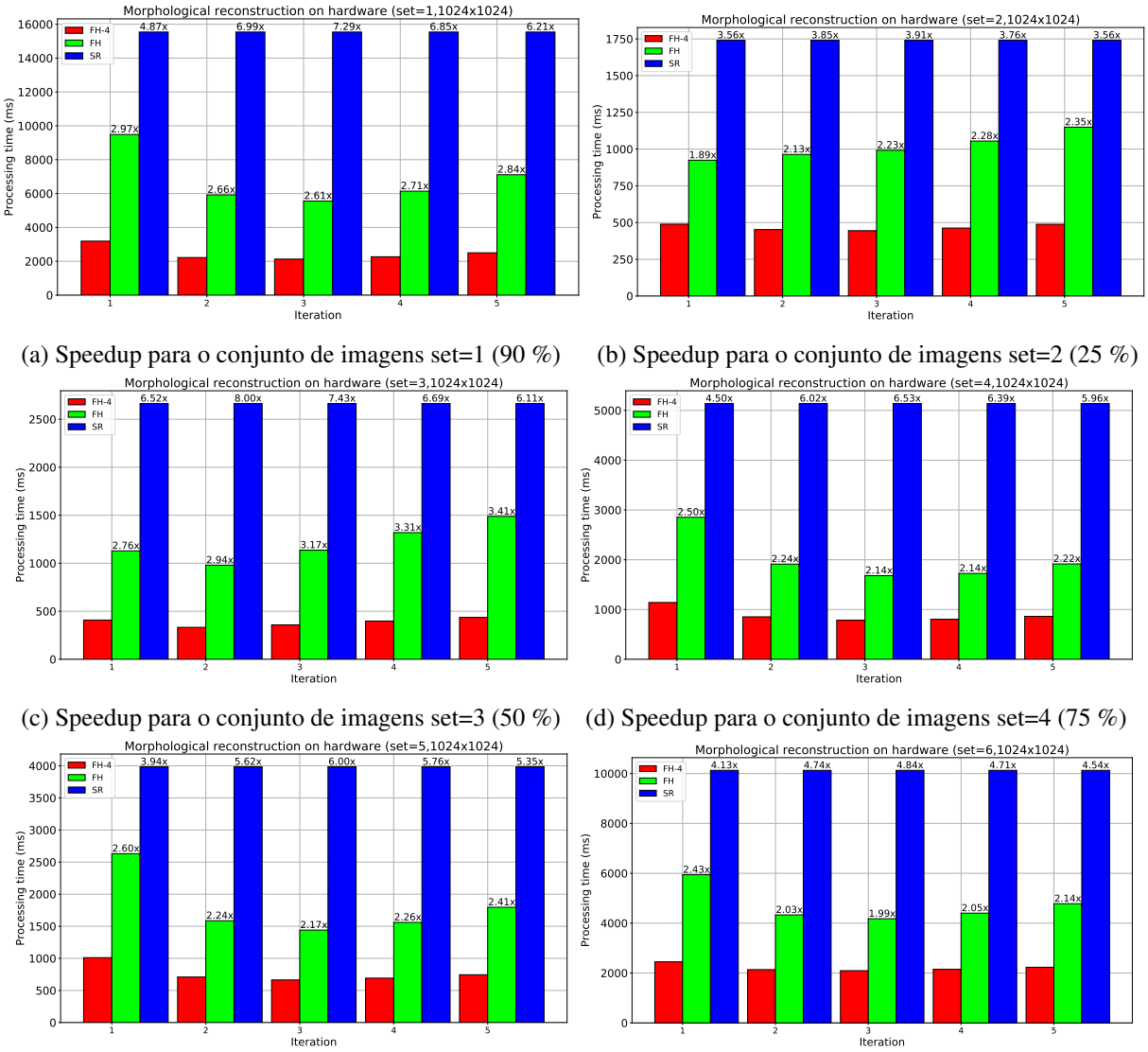
A Figura 6.3 mostra o desempenho da arquitetura paralela FH-4, FH e SR para vários conjuntos de imagens de tamanho de 1024×1024 pixels. Nesta figura, encontra-se a comparação de desempenho entre o algoritmo SR (barra azul), o algoritmo FH (barra verde), e o algoritmo paralelo FH-4 (barra vermelha). Os resultados mostram que a arquitetura do algoritmo FH-4 possui um ganho entre 3,56x a 8,00x comparado com o algoritmo SR, e entre 1,89x a 3,41x comparado com o algoritmo FH. Como mencionado anteriormente, o algoritmo FH com preditor SVR estima o número da iteração para o chaveamento da fase (varreduras para propagação). Essa arquitetura começa a calcular a troca de fase depois da primeira iteração e continua calculando até a troca da fase. A arquitetura FH-4 foi implementada realizando uma otimização da arquitetura FH-SVR para prescindir do preditor, pois a FH-4 realiza a reconstrução morfológica usando uma varredura e a fase de propagação, ou seja, o processo é feito usando uma iteração e depois passa para a fase de propagação. No processo de reconstrução morfológica da arquitetura FH-4, o desempenho e tempo de execução depende do conteúdo de tecido de cada sub-imagem. A quantidade de recursos de *hardware*, especificamente as memórias BRAM, depende do conteúdo de cada coprocessador, uma vez que o algoritmo de reconstrução troca para a fase de propagação, a quantidade de memória BRAM que consume a fila não pode ser estimada *a-priori*.

Nesta implementação, são usadas 4 sub-imagens onde são tratadas por cada coprocessador, onde todos os coprocessadores são sincronizados para obter a reconstrução morfológica de cada sub-imagem correspondente. O desempenho e tempo de execução da arquitetura FH-4 depende do desempenho de cada coprocessador. A Figura 6.3 mostra que o tempo de processamento depende do conteudo de cada imagem, em porcentagem de tecido. Além do tempo de execução de cada sub-imagem, o tempo de processamento das bordas (horizontais e verticais) influencia no desempenho da arquitetura.

6.3 CONSIDERAÇÕES FINAIS E COMPARAÇÃO DAS ARQUITETURAS

As diferentes implementações do algoritmo de reconstrução morfológica usando como base o algoritmo sequencial (SR) proposto nos trabalhos, encontram-se apenas em *software*. Neste sentido, o uso de *hardware* reconfigurável permite desenvolver sistemas eficientes gerando um balanço entre o *software* e o *hardware* dedicado preenchendo a lacuna que existe para implementações em *hardware*. Verifica-se que existe uma limitação de desempenho do sistema *hardware* nos parâmetros de consumo de memória RAM e tempo de execução devido à dependência de dados e ao caminho irregular, dado que a proposta original do algoritmo SR é sequencial impedindo a paralelização do mesmo.

A tabela 6.4 mostra as características das diferentes implementações mostrando que é possível aumentar o nível de paralelização do algoritmo FH. Nesta tabela, a fase 1 corresponde a etapa das



(a) Speedup para o conjunto de imagens set=1 (90 %)

(b) Speedup para o conjunto de imagens set=2 (25 %)

(c) Speedup para o conjunto de imagens set=3 (50 %)

(d) Speedup para o conjunto de imagens set=4 (75 %)

(e) Speedup para o conjunto de imagens set=5 (100 %)

(f) Speedup para o conjunto de imagens set=6 (50 %)

Figura 6.3: Resultados do tempo de execução para as implementações: FH-4, FH e SR.

varreduras, e a fase 2 a fila. Embora o caminho crítico do algoritmo seja a atualização do pixel que está sendo reconstruído, tornando-se difícil a paralelização do mesmo. As versões dos algoritmo SR e FH implementadas no *hardware* reconfigurável, permitiu fornecer uma contribuição ao estado da arte para o processo de reconstrução morfológica de imagens. Entre as contribuições deste podem-se citar as seguintes:

1. O desenvolvimento de *hardware* reconfigurável de uma arquitetura escalável que permite a reconstrução morfológica de uma imagem usando os algoritmos SR e FH. Dependendo do tamanho da imagem, este processo pode ser executado usando partições da imagem original e tratadas de forma independente. Estas arquiteturas estão, principalmente, condicionadas à memória interna RAM disponível no dispositivo selecionado.

2. O processo de verificação funcional foi desenvolvido usando o processador ARM embarcado

Tabela 6.4: Resumo das diferentes implementações dos algoritmos de reconstrução SR e FH.

Implementação	Fases			Tamanho imagem (pixels)	Processador	Frequencia máxima (MHz)	FPGA BRAM
	Fase 1	Fase 2	Preditor				
SR	HW	Não	Não	288x288	NIOS	63,83	2.752.256
	HW	Não	Não	512x512	ARM	64,94	4.204.464
FH	HW	SW	Não	512x512	ARM	64,52	4.204.464
FH	HW	HW	Não	512x512	ARM	79,15	4.474.482
FH-SVR	HW	HW	Sim	512x512	ARM	64,52	4.474.482
FH-4	HW	HW	Não	1024x1024	ARM	115,74	27.648.914

na plataforma FPGA SoCkit e Arria-10, usando o Matlab, C ou python como modelo de referencia para validar os resultados.

3. Demostrou-se que combinação de técnicas de Máquinas de Vetores de Suporte (SVMs) e meta-heurísticas originam soluções visando otimizar o desempenho de algoritmos de reconstrução morfológica.

4. A proposta de uma combinação de um algoritmo de aprendizagem de máquina (SVR), e *hardware* reconfigurável como solução do algoritmo de reconstrução morfológica baseado no algoritmo FH. Os resultados mostram reduções no tempo de processamento da reconstrução morfológica.

5. Durante o processo de treinamento da SVR foi desenvolvido uma metodologia que envolveu a verificação funcional, validação em *hardware* e treinamento da máquina.

6. O resultado principal deste trabalho, a implementação em *hardware* reconfigurável da arquitetura paralelizada (FH-4). Em todos os processos internos do algoritmo de reconstrução e da paralelização do mesmo, foram usadas maquinas de estados finitas (FSMs) para sincronizar os processadores de cada sub-imagem tratada.

7. Finalmente, neste trabalho foi provada a possibilidade de desenvolver arquiteturas em *hardware* reconfigurável eficientes via algoritmo FH usando um coprojeto *hardware*/*software*. Demostrando que segundo o estado da arte, o desempenho do algoritmo FH aumenta conforme o tamanho da imagem processada também aumenta.

7 CONCLUSÕES E PERSPECTIVAS DE TRABALHOS FUTUROS

Neste trabalho foi apresentada uma arquitetura *hardware* reconfigurável para a reconstrução morfológica de imagens baseada no algoritmo *Fast Hybrid* (FH). Como base para o *hardware* reconfigurável do algoritmo FH, foi desenvolvida uma arquitetura dedicada do algoritmo sequencial (SR). Igualmente, foi implementado um algoritmo de aprendizagem de máquina para predizer a convergência do algoritmo FH.

Durante o trabalho foi desenvolvido um método de verificação, treinamento e teste da máquina de aprendizagem SVR. Para avaliar o desempenho das implementações, os resultados dos processos de reconstrução morfológica foram comparados com as implementações em *software*. Os resultados mostram que o ganho em tempo de processamento incrementa conforme o tamanho das imagens também aumenta. Em consequência, os recursos *hardware*, especificamente as memórias internas, tendem a aumentar a demanda de estas.

A convergência do algoritmo FH, fase de varreduras e fase de propagação, depende do conteúdo da imagem que está sendo reconstruída. Para a aceleração da convergência foi implementado um co-projeto *hardware*/*software* do algoritmo FH e um preditor baseado no SVR.

Inicialmente, o sistema foi projetado para trabalhar com imagens de 288×288 pixels (Cyclone-IV), fornecendo um pixel a cada ciclo de relógio (50 MHz) depois de um período de latência; e finalmente, o sistema final consiste em quatro co-processadores *hardware* em paralelo processando uma imagem de 1024×1024 pixels (Arria-10), fornecendo um pixel a cada 100 MHz depois do período de latência. O ganho em tempo de execução foi a partir de 2.02x (algoritmo SR) até 8x (algoritmo FH sem paralelizar).

7.1 PROPOSTAS DE TRABALHOS FUTUROS

A arquitetura foi implementada para a reconstrução de imagens usando como base o algoritmo FH para imagens pequenas.

Assim, como trabalhos futuros sugere-se:

1. O circuito que atualiza o pixel que está sendo reconstruído é a parte critica do projeto, dado que durante o um ciclo de relógio, o pixel deve ser usado, processado e atualizado. Isto reduz o desempenho da arquitetura no quesito de frequência máxima de operação.

2. Implementar a comunicação *Burst* do barramento Avalon para o armazenamento da imagem em uma memória externa (memória RAM DD3), isto permitiria reduzir o consumo de

recursos de memória interna do FPGA.

3. Com auxilio de uma memória externa, um melhor gerenciamento dos recursos *hardware*, a arquitetura pode ser escalável para processar imagens maiores de 1024×1024 pixels.

REFERÊNCIAS BIBLIOGRÁFICAS

Altera ALTERA. *The Breakthrough Advantage for FPGAs with Tri-Gate Technology*.

Altera 2018 ALTERA. *Stratix 10 TX Product Family Overview Table (PDF)*. 2018. Disponível em: <https://www.altera.com/products/fpga/stratix-series/stratix-10/support.html>.

Anacona-Mosquera et al. 2018 ANACONA-MOSQUERA, O.; CABRAL, F.; SAMPAIO, R. C.; TEODORO, G.; JACOBI, R. P.; LLANOS, C. H. Efficient hardware implementation of the fast hybrid morphological reconstruction algorithm. In: *2018 31th Symposium on Integrated Circuits and Systems Design (SBCCI)*. [S.l.: s.n.], 2018. p. 162–167.

Anacona-Mosquera et al. 2020 Anacona-Mosquera, O.; Santos, C. E. d.; Cabral, F. R. G.; Sampaio, R. C.; Teodoro, G.; Jacobi, R. P.; Llanos, C. H. Hardware-based fast hybrid morphological reconstruction. *IEEE Design Test*, v. 37, n. 3, p. 30–39, 2020.

Anacona-Mosquera et al. 2017 ANACONA-MOSQUERA, O.; VINHAL, G.; SAMPAIO, R. C.; TEODORO, G.; JACOBI, R. P.; LLANOS, C. H. Efficient hardware implementation of morphological reconstruction based on sequential reconstruction algorithm. In: *2017 30th Symposium on Integrated Circuits and Systems Design (SBCCI)*. [S.l.: s.n.], 2017. p. 162–167.

Baniani e Chalechale 2013 BANIANI, E. A.; CHALECHALE, A. Hybrid pso and genetic algorithm for multilevel maximum entropy criterion threshold selection. *International journal of hybrid information technology*, v. 6, n. 5, p. 131–140, 2013.

Bankhead et al. 2017 BANKHEAD, P.; LOUGHREY, M. B.; FERNÁNDEZ, J.; DOMBROWSKI, Y.; MCART, D. G.; DUNNE, P. D.; MCQUAID, S.; GRAY, R. T.; MURRAY, L. J.; COLEMAN, H. G.; JAMES, J. A.; SALTO-TELLEZ, M.; HAMILTON, P. W. QuPath: Open source software for digital pathology image analysis. *Scientific Reports*, v. 7, n. 1, p. 16878, 2017. Disponível em: <https://doi.org/10.1038/s41598-017-17204-5>.

Bartovský et al. 2014 BARTOVSKÝ, J.; DOKLÁDAL, P.; DOKLÁDALOVÁ, E.; GEORGIEV, V. Parallel implementation of sequential morphological filters. *Journal of Real-Time Image Processing*, v. 9, n. 2, p. 315–327, Jun 2014. ISSN 1861-8219. Disponível em: <https://doi.org/10.1007/s11554-011-0226-5>.

Bartovský et al. 2015 BARTOVSKÝ, J.; DOKLÁDAL, P.; DOKLÁDALOVÁ, E.; BILODEAU, M.; AKIL, M. Real-time implementation of morphological filters with polygonal structuring elements. *Journal of Real-Time Image Processing*, v. 10, n. 1, p. 175–187, Mar 2015. ISSN 1861-8219. Disponível em: <https://doi.org/10.1007/s11554-012-0271-8>.

Bartovskỳ et al. 2015 BARTOVSKỲ, J.; DOKLÁDAL, P.; FAESSEL, M.; DOKLADALOVA, E.; BILODEAU, M. Morphological co-processing unit for embedded devices. *Journal of Real-Time Image Processing*, Springer, p. 1–12, 2015.

Benkrid et al. 2003 BENKRID, K.; SUKHSAWAS, S.; CROOKES, D.; BENKRID, A. An fpga-based image connected component labeller. In: CHEUNG, P. Y. K.; CONSTANTINIDES, G. A. (Ed.). *Field Programmable Logic and Application*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003. p. 1012–1015. ISBN 978-3-540-45234-8.

Bieniek e Moga 2000 BIENIEK, A.; MOGA, A. An efficient watershed algorithm based on connected components. *Pattern Recognition*, v. 33, n. 6, p. 907 – 916, 2000. ISSN 0031-3203. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0031320399001545>.

Cabaret, Lacassagne e Etiemble 2016 CABARET, L.; LACASSAGNE, L.; ETIEMBLE, D. Parallel light speed labeling: an efficient connected component algorithm for labeling and analysis on multi-core processors. *Journal of Real-Time Image Processing*, Mar 2016. ISSN 1861-8219. Disponível em: <https://doi.org/10.1007/s11554-016-0574-2>.

Cabral et al. 2020 CABRAL, F.; ANACONA-MOSQUERA, O.; SAMPAIO, R. C.; TEODORO, G.; LLANOS, C. H.; JACOBI, R. P. Optimized execution of morphological reconstruction in large medical images on embedded devices. *Journal of Real-Time Image Processing*, v. 14, p. 907–921, 2020.

Chan et al. 2013 CHAN, S. C.; ZHANG, S.; WU, J.; TAN, H.; NI, J. Q.; HUNG, Y. S. On the hardware/software design and implementation of a high definition multiview video surveillance system. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, v. 3, n. 2, p. 248–262, June 2013. ISSN 2156-3357.

Chien, Ma e Chen 2005 CHIEN, S.-Y.; MA, S.-Y.; CHEN, L.-G. Partial-result-reuse architecture and its design technique for morphological operations with flat structuring elements. *IEEE Transactions on Circuits and Systems for Video Technology*, v. 15, n. 9, p. 1156–1169, Sept 2005. ISSN 1051-8215.

Clienti, Beucher e Bilodeau 2008 CLIENTI, C.; BEUCHER, S.; BILODEAU, M. A system on chip dedicated to pipeline neighborhood processing for mathematical morphology. In: *2008 16th European Signal Processing Conference*. [S.l.: s.n.], 2008. p. 1–5. ISSN 2219-5491.

Crookes 1999 CROOKES, K. B. D. Fpga implementation of image component labeling. *Proc.SPIE*, v. 3844, p. 3844 – 3844 – 7, 1999. Disponível em: <https://doi.org/10.1117/12.359538>.

Dae Ro Lee et al. 2007 Dae Ro Lee; Seung Hun Jin; Pham Cong Thien; Jae Wook Jeon. Fpga based connected component labeling. In: *2007 International Conference on Control, Automation and Systems*. [S.l.: s.n.], 2007. p. 2313–2317.

Déforges, Normand e Babel 2013 DÉFORGES, O.; NORMAND, N.; BABEL, M. Fast recursive grayscale morphology operators: from the algorithm to the pipeline architecture. *Journal of Real-Time Image Processing*, v. 8, n. 2, p. 143–152, Jun 2013. ISSN 1861-8219. Disponível em: <https://doi.org/10.1007/s11554-010-0171-8>.

Deschamps, Bioul e Sutter 2006 DESCHAMPS, J.-P.; BIOUL, G. J.; SUTTER, G. D. *Synthesis of arithmetic circuits: FPGA, ASIC and embedded systems*. [S.l.]: John Wiley & Sons, 2006.

Dokládal e Dokládalová 2011 DOKLáDAL, P.; DOKLáDALOVá, E. Computationally efficient, one-pass algorithm for morphological filters. *Journal of Visual Communication and Image Representation*, v. 22, n. 5, p. 411 – 420, 2011. ISSN 1047-3203. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1047320311000393>.

ELLOUMI, KRID e SELLAMI 2018 ELLOUMI, H.; KRID, M.; SELLAMI, D. 2d parallel architecture for morphological operators supporting multiple shaped structuring elements. *Procedia Computer Science*, v. 126, p. 695 – 702, 2018. ISSN 1877-0509. Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 22nd International Conference, KES-2018, Belgrade, Serbia. Disponível em: <http://www.sciencedirect.com/science/article/pii/S187705091831281X>.

França 2005 FRANÇA, F. O. *Algoritmos bio-inspirados aplicados à otimização dinâmica*. Dissertação (Master's thesis) — Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas, 2005. Disponível em: <http://repositorio.unicamp.br/jspui/handle/REPOSIP/259091>.

Giagkiozis, Purshouse e Fleming 2015 GIAGKIOZIS, I.; PURSHOUSE, R. C.; FLEMING, P. J. An overview of population-based algorithms for multi-objective optimisation. *International Journal of Systems Science*, Taylor & Francis, v. 46, n. 9, p. 1572–1599, 2015.

Gibson et al. 2013 GIBSON, R. M.; AHMADINIA, A.; MCMEEKIN, S. G.; STRANG, N. C.; MORISON, G. A reconfigurable real-time morphological system for augmented vision. *EURASIP Journal on Advances in Signal Processing*, v. 2013, n. 1, p. 134, Aug 2013. ISSN 1687-6180. Disponível em: <https://doi.org/10.1186/1687-6180-2013-134>.

Göhringer et al. 2011 GÖHRINGER, D.; OBIE, J.; BRAGA, A. L.; HÜBNER, M.; LLANOS, C. H.; BECKER, J. Exploration of the power-performance tradeoff through parameterization of fpga-based multiprocessor systems. *International Journal of Reconfigurable Computing*, Hindawi Publishing Corp., v. 2011, p. 7, 2011.

Göhringer et al. 2009 GÖHRINGER, D.; PERSCHKE, T.; HÜBNER, M.; BECKER, J. A taxonomy of reconfigurable single-/multiprocessor systems-on-chip. *International Journal of Reconfigurable Computing*, Hindawi, v. 2009, 2009.

Gunturk e Li 2012 GUNTURK, B. K.; LI, X. *Image restoration: fundamentals and advances*. [S.l.]: CRC Press, 2012.

Hall e Hamblen 2004 HALL, T. S.; HAMBLEN, J. O. System-on-a-programmable-chip development platforms in the classroom. *IEEE Transactions on Education*, v. 47, n. 4, p. 502–507, Nov 2004. ISSN 0018-9359.

Hamdani, T.M., Alimi, A.M. e Khabou, M.A 2011 Hamdani, T.M.; Alimi, A.M.; Khabou, M.A. An iterative method for deciding svm and single layer neural network structures. In: *Neural Processing Letters*. [S.l.: s.n.], 2011. v. 33, p. 171–186.

Haralick, Sternberg e Zhuang 1987 HARALICK, R. M.; STERNBERG, S. R.; ZHUANG, X. Image analysis using mathematical morphology. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-9, n. 4, p. 532–550, July 1987. ISSN 0162-8828.

He et al. 2017 HE, L.; REN, X.; GAO, Q.; ZHAO, X.; YAO, B.; CHAO, Y. The connected-component labeling problem: A review of state-of-the-art algorithms. *Pattern Recognition*, v. 70, p. 25 – 43, 2017. ISSN 0031-3203. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0031320317301693>.

Hedberg, Kristensen e Owall 2008 HEDBERG, H.; KRISTENSEN, F.; OWALL, V. Low-complexity binary morphology architectures with flat rectangular structuring elements. *IEEE Transactions on Circuits and Systems I: Regular Papers*, v. 55, n. 8, p. 2216–2225, Sept 2008. ISSN 1549-8328.

Heijmans e Ronse 1990 HEIJMANS, H.; RONSE, C. The algebraic basis of mathematical morphology i. dilations and erosions. *Computer Vision, Graphics, and Image Processing*, v. 50, n. 3, p. 245 – 295, 1990. ISSN 0734-189X. Disponível em: <http://www.sciencedirect.com/science/article/pii/0734189X9090148O>.

Heijmans e Vincent 1992 HEIJMANS, H.; VINCENT, L. Graph morphology in image analysis. *Optical Engineering-New York-Marcel Dekker Incorporated-*, Marcel Dekker AG, v. 34, p. 171–203, 1992.

Holzer et al. 2012 HOLZER, M.; SCHUMACHER, F.; GREINER, T.; ROSENSTIEL, W. Optimized hardware architecture of a smart camera with novel cyclic image line storage structures for morphological raster scan image processing. In: *2012 IEEE International Conference on Emerging Signal Processing Applications*. [S.l.: s.n.], 2012. p. 83–86.

Jivet, Brindusescu e Bogdanov 2008 JIVET, I.; BRINDUSESCU, A.; BOGDANOV, I. Image contrast enhancement using morphological decomposition by reconstruction. *WSEAS Trans. Cir. and Sys.*, World Scientific and Engineering Academy and Society (WSEAS), Stevens Point, Wisconsin, USA, v. 7, n. 8, p. 822–831, ago. 2008. ISSN 1109-2734. Disponível em: <http://dl.acm.org/citation.cfm?id=1482247.1482253>.

Journel e Huijbregts 1978 JOURNEL, A. G.; HUIJBREGTS, C. J. Book. *Mining geostatistics / [by] A. G. Journel and Ch. J. Huijbregts*. [S.l.]: Academic Press London ; New York, 1978. x, 600 p. : p. ISBN 0123910501.

Karaboga et al. 2014 KARABOGA, D.; GORKEMLI, B.; OZTURK, C.; KARABOGA, N. A comprehensive survey: artificial bee colony (abc) algorithm and applications. *Artificial Intelligence Review*, v. 42, n. 1, p. 21–57, Jun 2014. ISSN 1573-7462. Disponível em: <https://doi.org/10.1007/s10462-012-9328-0>.

Karas 2011 KARAS, P. Efficient Computation of Morphological Greyscale Reconstruction. In: *Sixth Doctoral Workshop on Mathematical and Engineering Methods in Computer Science (MEMICS'10) – Selected Papers*. Dagstuhl, Germany: [s.n.], 2011. v. 16, p. 54–61. ISBN 978-3-939897-22-4. ISSN 2190-6807.

Klaiber et al. 2016 KLAIBER, M. J.; BAILEY, D. G.; BAROUD, Y. O.; SIMON, S. A resource-efficient hardware architecture for connected component analysis. *IEEE Transactions on Circuits and Systems for Video Technology*, v. 26, n. 7, p. 1334–1349, July 2016. ISSN 1051-8215.

Knapp 1995 KNAPP, S. K. *Using Programmable Logic to Accelerate DSP Functions*. 1995.

Körbes 2010 KÖRBES, A. *Análise de Algoritmos da Transformada Watershed*. Dissertação (Master's thesis) — Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas, 2010. Disponível em: <http://cutter.unicamp.br/document/?code=000769962>.

Körbes et al. 2009 KÖRBES, A.; VITOR, G. B.; FERREIRA, J. V.; LOTUFO, R. de A.; EINSTEIN, A. A.; MENDELEYEV, R. A proposal for a parallel watershed transform algorithm for real-time segmentation. In: *Proceedings of Workshop de Visao Computacional WVC*. [S.l.: s.n.], 2009.

Körbes et al. 2011 KÖRBES, A.; VITOR, G. B.; LOTUFO, R. de A.; FERREIRA, J. V. Advances on watershed processing on gpu architecture. In: SPRINGER. *International Symposium on Mathematical Morphology and Its Applications to Signal and Image Processing*. [S.l.], 2011. p. 260–271.

Kornaros 2010 KORNAROS, G. *Multi-Core Embedded Systems*. 1st. ed. Boca Raton, FL, USA: CRC Press, Inc., 2010. ISBN 143981161X, 9781439811610.

Lewis et al. 2016 LEWIS, D.; CHIU, G.; CHROMCZAK, J.; GALLOWAY, D.; GAMSA, B.; MANOHARARAJAH, V.; MILTON, I.; VANDERHOEK, T.; DYKEN, J. V. The stratix™10 highly pipelined fpga architecture. In: *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. New York, NY, USA: ACM, 2016. (FPGA '16), p. 159–168. ISBN 978-1-4503-3856-1. Disponível em: <http://doi.acm.org/10.1145/2847263.2847267>.

Li 2015 Li, Y. Fast multi-level connected component labeling for large-scale images. In: *2015 International Conference on Optoelectronics and Microelectronics (ICOM)*. [S.l.: s.n.], 2015. p. 334–337.

Llanos, Hurtado e Alfaro 2016 LLANOS, C. H.; HURTADO, R. H.; ALFARO, S. C. A. Fpga-based approach for change detection in gtaw welding process. *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, v. 38, n. 3, p. 913–929, Mar 2016. ISSN 1806-3691. Disponível em: <https://doi.org/10.1007/s40430-015-0371-z>.

Maggiani, Luca et al. 2018 Maggiani, Luca; Bourrasset, Cédric; Quinton, Jean-Charles; Berry, François; Sérot, Jocelyn. Bio-inspired heterogeneous architecture for real-time pedestrian detection applications. In: *Journal of Real-Time Image Processing*. [S.l.: s.n.], 2018. v. 14, p. 535–548.

Marler e Arora 2004 MARLER, R. T.; ARORA, J. S. Survey of multi-objective optimization methods for engineering. *Structural and multidisciplinary optimization*, Springer, v. 26, n. 6, p. 369–395, 2004.

Mori, Llanos e Berger 2012 MORI, J. Y.; LLANOS, C. H.; BERGER, P. A. Kernel analysis for architecture design trade off in convolution-based image filtering. In: *2012 25th Symposium on Integrated Circuits and Systems Design (SBCCI)*. [S.l.: s.n.], 2012. p. 1–6.

Mukherjee, Mukhopadhyay e Biswas 2016 MUKHERJEE, D.; MUKHOPADHYAY, S.; BISWAS, G. P. Fpga based parallel implementation of morphological filters. In: *2016 International Conference on Microelectronics, Computing and Communications (MicroCom)*. [S.l.: s.n.], 2016. p. 1–6.

Mukherjee, Mukhopadhyay e Biswas 2017 MUKHERJEE, D.; MUKHOPADHYAY, S.; BISWAS, G. P. Fpga-based parallel implementation of morphological operators for 2d gray-level images. *Arabian Journal for Science and Engineering*, v. 42, n. 8, p. 3191–3206, Aug 2017. ISSN 2191-4281. Disponível em: <https://doi.org/10.1007/s13369-017-2429-y>.

Munoz et al. 2010 MUNOZ, D. M.; LLANOS, C. H.; COELHO, L. d. S.; AYALA-RINCON, M. Hardware particle swarm optimization based on the attractive-repulsive scheme for embedded applications. In: *2010 International Conference on Reconfigurable Computing and FPGAs*. [S.l.: s.n.], 2010. p. 55–60. ISSN 2325-6532.

Muttillo et al. 2016 MUTTILLO, V.; VALENTE, G.; FEDERICI, F.; POMANTE, L.; FACCIO, M.; TIERI, C.; FERRI, S. A design methodology for soft-core platforms on fpga with smp linux, openmp support, and distributed hardware profiling system. *EURASIP Journal on Embedded Systems*, v. 2016, n. 1, p. 15, Sep 2016. ISSN 1687-3963. Disponível em: <https://doi.org/10.1186/s13639-016-0051-9>.

Muñoz et al. 2013 MUñOZ, D. M.; LLANOS, C. H.; COELHO, L. dos S.; AYALA-RINCóN, M. Hardware-based parallel firefly algorithm for embedded applications. In: *2013 NASA/ESA Conference on Adaptive Hardware and Systems (AHS-2013)*. [S.l.: s.n.], 2013. p. 39–46.

Muñoz et al. 2009 MUñOZ, D. M.; SáNCHEZ, D. F.; LLANOS, C. H.; AYALA-RINCóN, M. Tradeoff of fpga design of floating-point transcendental functions. In: *2009 17th IFIP International Conference on Very Large Scale Integration (VLSI-SoC)*. [S.l.: s.n.], 2009. p. 239–242. ISSN 2324-8432.

Nixon e Aguado 2012 NIXON, M. S.; AGUADO, A. S. *Feature extraction & image processing for computer vision*. [S.l.]: Academic Press, 2012.

Norvig e Russell 2014 NORVIG, P.; RUSSELL, S. *Inteligência Artificial: Tradução da 3a Edição*. [S.l.]: Elsevier Brasil, 2014. v. 1.

Osma-Ruiz et al. 2007 OSMA-RUIZ, V.; GODINO-LLORENTE, J. I.; SáENZ-LECHóN, N.; GóMEZ-VILDA, P. An improved watershed algorithm based on efficient computation of shortest paths. *Pattern Recognition*, v. 40, n. 3, p. 1078 – 1090, 2007. ISSN 0031-3203. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0031320306003086>.

Pandey et al. 2014 Pandey, J. G.; Karmakar, A.; Mishra, A. K.; Shekhar, C.; Gurunarayanan, S. Implementation of an improved connected component labeling algorithm using fpga-based platform. In: *2014 International Conference on Signal Processing and Communications (SPCOM)*. [S.l.: s.n.], 2014. p. 1–6.

Parvati, Rao e Das 2008 PARVATI, K.; RAO, P.; DAS, M. M. Image segmentation using gray-scale morphology and marker-controlled watershed transformation. *Discrete Dynamics in Nature and Society*, Hindawi, v. 2008, 2008.

Pedrini e Schwartz 2008 PEDRINI, H.; SCHWARTZ, W. R. *Análise de imagens digitais: princípios, algoritmos e aplicações*. [S.l.]: Thomson Learning, 2008.

Pistorius et al. 2007 PISTORIUS, J.; HUTTON, M.; SCHLEICHER, J.; IOTOV, M.; JULIAS, E.; THARMALINGAM, K. Equivalence verification of fpga and structured asic implementations. In: *2007 International Conference on Field Programmable Logic and Applications*. [S.l.: s.n.], 2007. p. 423–428. ISSN 1946-147X.

Quesada-Barriuso, Heras e Argüello 2013 QUESADA-BARRIUSO, P.; HERAS, D. B.; ARGÜELLO, F. Efficient 2d and 3d watershed on graphics processing unit: block-asynchronous approaches based on cellular automata. *Computers & Electrical Engineering*, Elsevier, v. 39, n. 8, p. 2638–2655, 2013.

Rambabu e Chakrabarti 2007 RAMBABU, C.; CHAKRABARTI, I. An efficient immersion-based watershed transform method and its prototype architecture. *Journal of Systems Architecture*, v. 53, n. 4, p. 210–226, 2007. ISSN 1383-7621. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1383762106001226>.

Robinson e Whelan 2004 ROBINSON, K.; WHELAN, P. F. Efficient morphological reconstruction: a downhill filter. *Pattern Recognition Letters*, v. 25, n. 15, p. 1759 – 1767, 2004. ISSN 0167-8655. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0167865504001692>.

Rosenfeld e Pfaltz 1966 ROSENFELD, A.; PFALTZ, J. L. Sequential operations in digital picture processing. *J. ACM*, ACM, New York, NY, USA, v. 13, n. 4, p. 471–494, out. 1966. ISSN 0004-5411. Disponível em: <http://doi.acm.org/10.1145/321356.321357>.

Sameer 2012 SAMEER, R. *Implementation of Watershed Based Image Segmentation Algorithm in FPGA*. Dissertação (Master's thesis) — Institute of Parallel and Distributed Systems, University of Stuttgart, 2012. Disponível em: <http://dx.doi.org/10.18419/opus-2878>.

Sampaio et al. 2017 SAMPAIO, R. C.; SANTOS, C. E.; AYALA, H.; COELHO, L. d. S.; JACOBI, R.; LLANOS, C. H. Support vector regression based nonlinear model predictive control on fpga. In: *24 th ABCM International Congress of Mechanical Engineering (COBEM)*. [S.l.: s.n.], 2017.

Santos et al. 2017 SANTOS, C. E.; SAMPAIO, R. C.; AYALA, H.; COELHO, L. d. S.; JACOBI, R.; LLANOS, C. H. A svm optimization tool and fpga system architecture applied to nmpc. In: *2017 30th Symposium on Integrated Circuits and Systems Design (SBCCI)*. [S.l.: s.n.], 2017. p. 96–102.

Santos et al. 2021 SANTOS, C. E. da S.; SAMPAIO, R. C.; COELHO, L. dos S.; BESTARD, G. A.; LLANOS, C. H. Multi-objective adaptive differential evolution for svm/svr hyperparameters selection. *Pattern Recognition*, v. 110, p. 107649, 2021. ISSN 0031-3203. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0031320320304520>.

Schwenk e Huber 2015 SCHWENK, K.; HUBER, F. Connected Component Labeling algorithm for very complex and high-resolution images on an FPGA platform. In: D.D.S., B. H.; LóPEZ, S.; WU, Z.; NASCIMENTO, J. M.; ALPATOV, B. A.; MORA, J. P. de (Ed.). *High-Performance Computing in Remote Sensing V*. SPIE, 2015. v. 9646, p. 9 – 22. Disponível em: <https://doi.org/10.1117/12.2194101>.

Serra 1984 SERRA, J. *Image Analysis and Mathematical Morphology*. Academic Press, 1984. (Image Analysis and Mathematical Morphology, v. 1). ISBN 9780126372427. Disponível em: <https://books.google.com.br/books?id=kfY-ngEACAAJ>.

Shigeo 2005 SHIGEO, A. Support vector machines for pattern classification. *Advances in Pattern Recognition, Springer, Heidelberg*, 2005.

Shih 2009 SHIH, F. Y. *Image processing and mathematical morphology: fundamentals and applications*. [S.l.]: CRC press, 2009.

Sivakumar e Janakiraman 2020 SIVAKUMAR, V.; JANAKIRAMAN, N. A novel method for segmenting brain tumor using modified watershed algorithm in mri image with fpga. *Biosystems*, v. 198, p. 104226, 2020. ISSN 0303-2647. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0303264720301179>.

Smistad et al. 2015 SMISTAD, E.; FALCH, T. L.; BOZORGI, M.; ELSTER, A. C.; LINDSETH, F. Medical image segmentation on gpus-a comprehensive review. *Medical image analysis*, Elsevier, v. 20, n. 1, p. 1–18, 2015.

Smistad et al. 2015 SMISTAD, E.; FALCH, T. L.; BOZORGI, M.; ELSTER, A. C.; LINDSETH, F. Medical image segmentation on gpus – a comprehensive review. *Medical Image Analysis*, v. 20, n. 1, p. 1–18, 2015. ISSN 1361-8415. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1361841514001819>.

Soille 1998 SOILLE, P. *Morphological Image Analysis : Principles and Applications*. [S.l.]: Springer, 1998.

Soille e Vincent 1990 SOILLE, P.; VINCENT, L. M. Determining watersheds in digital pictures via flooding simulations. In: INTERNATIONAL SOCIETY FOR OPTICS AND PHOTONICS. *Visual Communications and Image Processing'90: Fifth in a Series*. [S.l.], 1990. v. 1360, p. 240–251.

Sonka, Hlavac e Boyle 2014 SONKA, M.; HLAVAC, V.; BOYLE, R. *Image processing, analysis, and machine vision*. [S.l.]: Cengage Learning, 2014.

Spagnolo, Perri e Corsonello 2019 SPAGNOLO, F.; PERRI, S.; CORSONELLO, P. An efficient hardware-oriented single-pass approach for connected component analysis. *Sensors*, v. 19, n. 14, 2019. ISSN 1424-8220. Disponível em: <https://www.mdpi.com/1424-8220/19/14/3055>.

Št, Beneš et al. 2011 ŠT, O.; BENEŠ, B. et al. Connected component labeling in cuda. In: *GPU computing gems emerald edition*. [S.l.]: Elsevier, 2011. p. 569–581.

Sung et al. 2015 SUNG, J.-M.; LEE, C.-H.; HA, Y.-H.; CHOI, B.-Y. Morphological image reconstruction using directional propagation. *Journal of Imaging Science and Technology*, v. 59, n. 5, p. 50503–1–50503–7, 2015. ISSN 1062-3701. Disponível em: <https://www.ingentaconnect.com/content/ist/jist/2015/00000059/00000005/art00004>.

Szeliski 2010 SZELISKI, R. *Computer vision: algorithms and applications*. [S.l.]: Springer Science & Business Media, 2010.

Sánchez-Ferreira, Mori e Llanos 2012 SáNCHEZ-FERREIRA, C.; MORI, J. Y.; LLANOS, C. H. Background subtraction algorithm for moving object detection in fpga. In: *2012 VIII Southern Conference on Programmable Logic*. [S.l.: s.n.], 2012. p. 1–6.

Sánchez-Ferreira et al. 2013 SáNCHEZ-FERREIRA, C.; MORI, J. Y.; LLANOS, C. H.; FORTALEZA, E. Development of a stereo vision measurement architecture for an underwater robot. In: *2013 IEEE 4th Latin American Symposium on Circuits and Systems (LASCAS)*. [S.l.: s.n.], 2013. p. 1–4.

Teich 2012 TEICH, J. Hardware/software codesign: The past, the present, and predicting the future. *Proceedings of the IEEE*, v. 100, n. Special Centennial Issue, p. 1411–1430, May 2012. ISSN 0018-9219.

Teodoro et al. 2017 TEODORO, G.; KURC, T.; ANDRADE, G.; KONG, J.; FERREIRA, R.; SALTZ, J. Application performance analysis and efficient execution on systems with multi-core cpus, gpus and mics: A case study with microscopy image analysis. *Int J High Perform Comput Appl*, v. 31, n. 1, p. 32–51, Jan 2017. ISSN 1094-3420. 28239253[pmid]. Disponível em: <http://www.ncbi.nlm.nih.gov/pmc/articles/PMC5319667/>.

Teodoro et al. 2013 TEODORO, G.; PAN, T.; KURC, T. M.; KONG, J.; COOPER, L. A.; SALTZ, J. H. Efficient irregular wavefront propagation algorithms on hybrid cpu-gpu machines. *Parallel Computing*, v. 39, n. 4-5, p. 189 – 211, 2013. ISSN 0167-8191.

Torres-Huitzil 2016 TORRES-HUITZIL, C. Fpga-based fast computation of gray-level morphological granulometries. *Journal of Real-Time Image Processing*, v. 11, n. 3, p. 547–557, Mar 2016. ISSN 1861-8219. Disponível em: <https://doi.org/10.1007/s11554-013-0355-0>.

Trieu e Maruyama 2006 TRIEU, D. B. K.; MARUYAMA, T. Implementation of a parallel and pipelined watershed algorithm on fpga. In: *2006 International Conference on Field Programmable Logic and Applications*. [S.l.: s.n.], 2006. p. 1–6. ISSN 1946-147X.

Trieu e Maruyama 2007 TRIEU, D. B. K.; MARUYAMA, T. A pipeline implementation of a watershed algorithm on fpga. In: *2007 International Conference on Field Programmable Logic and Applications*. [S.l.: s.n.], 2007. p. 714–717. ISSN 1946-147X.

Trieu e Maruyama 2007 TRIEU, D. B. K.; MARUYAMA, T. Real-time image segmentation based on a parallel and pipelined watershed algorithm. *Journal of Real-Time Image Processing*, v. 2, n. 4, p. 319–329, Dec 2007. ISSN 1861-8219. Disponível em: <https://doi.org/10.1007/s11554-007-0051-z>.

Trieu e Maruyama 2008 TRIEU, D. B. K.; MARUYAMA, T. An implementation of a watershed algorithm based on connected components on fpga. In: *2008 International Conference on Field-Programmable Technology*. [S.l.: s.n.], 2008. p. 253–256.

Tsai, Ho e Tsai 2018 Tsai, T.; Ho, Y.; Tsai, C. Implementation of real-time connected component labeling using fpga. In: *2018 IEEE International Conference on Consumer Electronics-Taiwan (ICCE-TW)*. [S.l.: s.n.], 2018. p. 1–2.

Verwer, Verbeek e Dekker 1989 VERWER, B. J. H.; VERBEEK, P. W.; DEKKER, S. T. An efficient uniform cost algorithm applied to distance transforms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 11, n. 4, p. 425–429, April 1989. ISSN 0162-8828.

Vincent 1989 VINCENT, L. Graphs and mathematical morphology. *Signal Processing*, v. 16, n. 4, p. 365 – 388, 1989. ISSN 0165-1684. Special Issue on Advances in Mathematical Morphology. Disponível em: <http://www.sciencedirect.com/science/article/pii/0165168489900315>.

Vincent 1991 VINCENT, L. Exact euclidean distance function by chain propagations. In: *Proceedings. 1991 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 1991. p. 520–525. ISSN 1063-6919.

Vincent 1992 VINCENT, L. Morphological algorithms. *Optical Engineering-New York-Marcel Dekker Incorporated-*, Marcel Dekker AG, v. 34, p. 255–288, 1992.

Vincent 1992 VINCENT, L. Morphological grayscale reconstruction: definition, efficient algorithm and applications in image analysis. In: *Proceedings 1992 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 1992. p. 633–635. ISSN 1063-6919.

Vincent 1995 VINCENT, L. Lecture notes on granulometries, segmentation, and morphological algorithms. In: ZAKOPANE, POLAND. *Proceedings of the Summer School on Morphological Image and Signal Processing*. [S.l.], 1995. p. 119–216.

Vincent e Soille 1991 VINCENT, L.; SOILLE, P. Watersheds in digital spaces: an efficient algorithm based on immersion simulations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 13, n. 6, p. 583–598, June 1991. ISSN 0162-8828.

Vincent e Soille 1991 Vincent, L.; Soille, P. Watersheds in digital spaces: an efficient algorithm based on immersion simulations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 13, n. 6, p. 583–598, 1991.

Wang, Lin e Miller 2015 WANG, Y.; LIN, C.; MILLER, J. Improved 3d image segmentation for x-ray tomographic analysis of packed particle beds. *Minerals Engineering*, v. 83, p. 185 – 191, 2015. ISSN 0892-6875. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0892687515300777>.

Wang, Lin e Miller 2016 WANG, Y.; LIN, C.; MILLER, J. 3d image segmentation for analysis of multisize particles in a packed particle bed. *Powder Technology*, v. 301, p. 160 – 168, 2016. ISSN 0032-5910. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0032591016302339>.

Wu et al. 2017 WU, Y.; PENG, X.; RUAN, K.; HU, Z. Improved image segmentation method based on morphological reconstruction. *Multimedia Tools and Applications*, v. 76, n. 19, p. 19781–19793, Oct 2017. ISSN 1573-7721. Disponível em: <https://doi.org/10.1007/s11042-015-3192-2>.

Yang e Papa 2016 YANG, X.-S.; PAPA, J. P. *Bio-inspired computation and applications in image processing*. [S.l.]: Academic Press, 2016.

Yeong et al. 2009 YEONG, L. S.; NGAU, C. W. H.; ANG, L.-M.; SENG, K. P. Efficient processing of a rainfall simulation watershed on an fpga-based architecture with fast access to neighbourhood pixels. *EURASIP J. Embedded Syst.*, Hindawi Limited, London, GBR, v. 2009, jan. 2009. ISSN 1687-3955. Disponível em: <https://doi.org/10.1155/2009/318654>.

Youkana et al. 2017 YOUKANA, I.; COUSTY, J.; SAOULI, R.; AKIL, M. Parallelization strategy for elementary morphological operators on graphs: Distance-based algorithms and implementation on multicore shared-memory architecture. *Journal of Mathematical Imaging and Vision*, v. 59, n. 1, p. 136–160, Sep 2017. ISSN 1573-7683. Disponível em: <https://doi.org/10.1007/s10851-017-0737-1>.